МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ АЭРОКОСМИЧЕСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П.КОРОЛЕВА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» (СГАУ)

Технологии сетевого программирования

Электронный образовательный контент

Работа выполнена по мероприятию блока 1 «Совершенствование образовательной деятельности» Программы развития СГАУ на 2009-2018 годы по проекту «Разработка образовательных контентов в рамках мастер-класса по внедрению и использованию СЭДО в реальном учебном процессе»
Соглашение № 1/27 от 03 июня 2013 г.

САМАРА
2013

УДК 004 (075)
ББК 32.973.26-018.2
Т 384

Автор-составитель: Гаврилов Андрей Вадимович

Технологии сетевого программирования [Электронный ресурс] : электронный образовательный контент / М-во образования и науки РФ, Самар. гос. аэрокосм. ун-т им. С. П. Королева (нац. исслед. ун-т). – Авт. сост.: А.В. Гаврилов. – Электрон. и граф. дан. (9,2 Мбайт). – Самара, 2013. – 1 эл. опт. диск (CD-ROM).

Электронный образовательный контент состоит из 16 презентаций, вопросов по курсу и аттестационных педагогических измерительных материалов. В раках контента рассмотрены проблемы и общие принципы создания сетевых приложений и распределённого программного обеспечения. В качестве платформы для реализации программных систем рассматривается Java Enterprise Edition и используемые в ней технологии: Sockets, Remote Method Invocation, Hypertext Markup Language, Cascading Stylesheets, Extensible Markup Language, JavaScript, Servlets, Java Server Pages, Java Server Faces, Java Naming and Directory Interface, Java Database Connectivity, Enterprise Java Beans, AJAX.

Электронный образовательный контент предназначен для подготовки бакалавров направлений 010400.62 «Прикладная математика и информатика» и 010900.62 «Прикладные математика и физика» факультета информатики, специализирующихся на разработке сетевых программных приложений и web-приложений, изучающих дисциплину «Технологии сетевого программирования» в 7 и 8 семестрах.

Электронный образовательный контент разработан на кафедре технической кибернетики.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Распределенные программные системы

Занятие 1

Гаврилов А.В.

Самара
2013

План занятия

- Понятие распределенной системы
- Причины создания
- “Законы” создания
- Принципы построения
- Проблемы распределенности
- Требования к распределенным системам
- Сложности реализации

Определения

- «...система нескольких автономных вычислительных узлов, взаимодействующих для выполнения общей цели.»
- «Система, чьи компоненты размещены на различных узлах, взаимодействующие и управляемые только посредством передачи сообщений.»
- «Система, состоящая из набора двух или более независимых узлов, которые координируют свою работу посредством синхронного или асинхронного обмена сообщениями.»

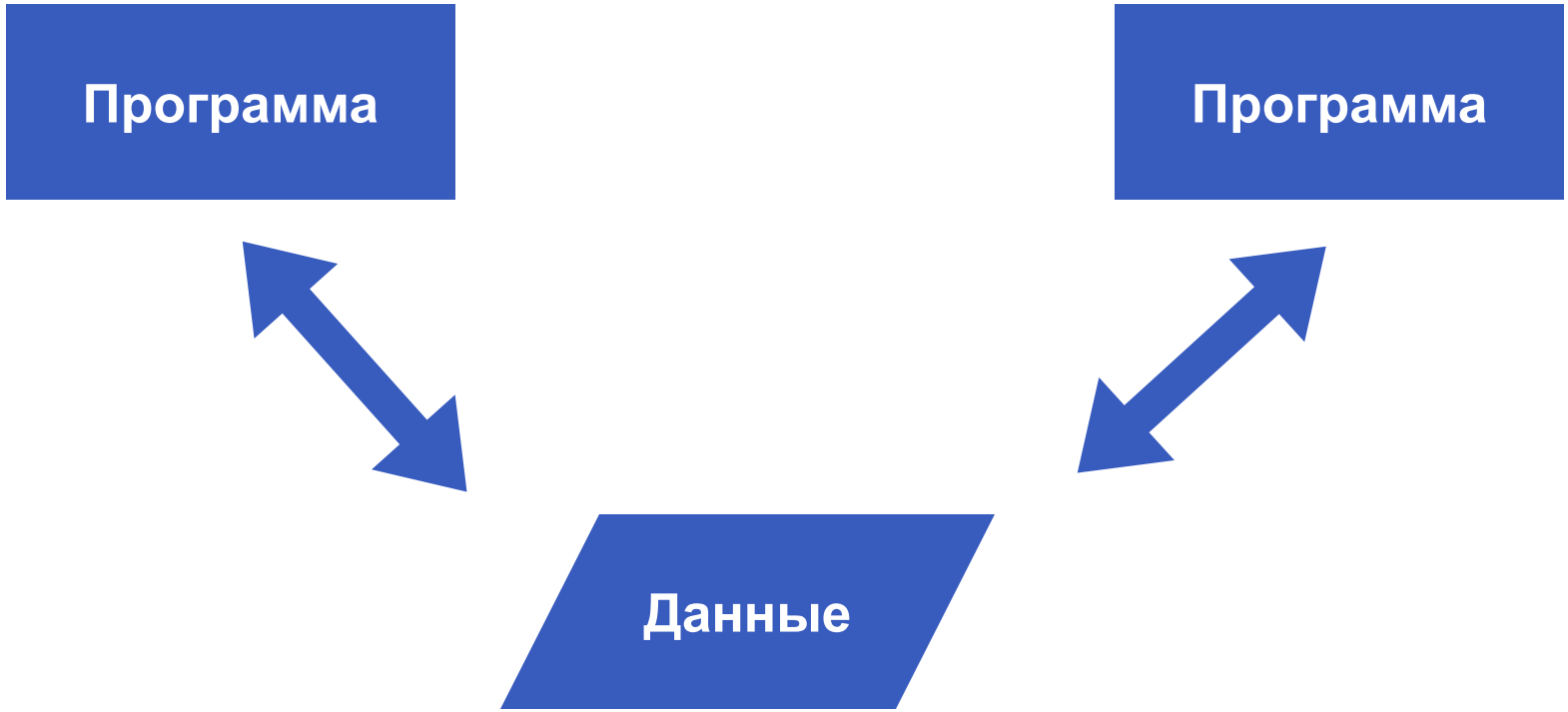
Определения

- «Распределенная система – это набор независимых узлов (компьютеров), которые представляются пользователю как единая система.»
- «Распределенная система – это собрание независимых компьютеров, соединенных сетью и программным обеспечением, обеспечивающим их совместное функционирование.»
- «…я не могу объяснить, что такое распределенная система, но узнаю ее как только мне ее покажут.»

Причины создания распределенных приложений

- Необходимость совместного использования общих ресурсов:
  - Данные
  - Устройства
  - Приложения

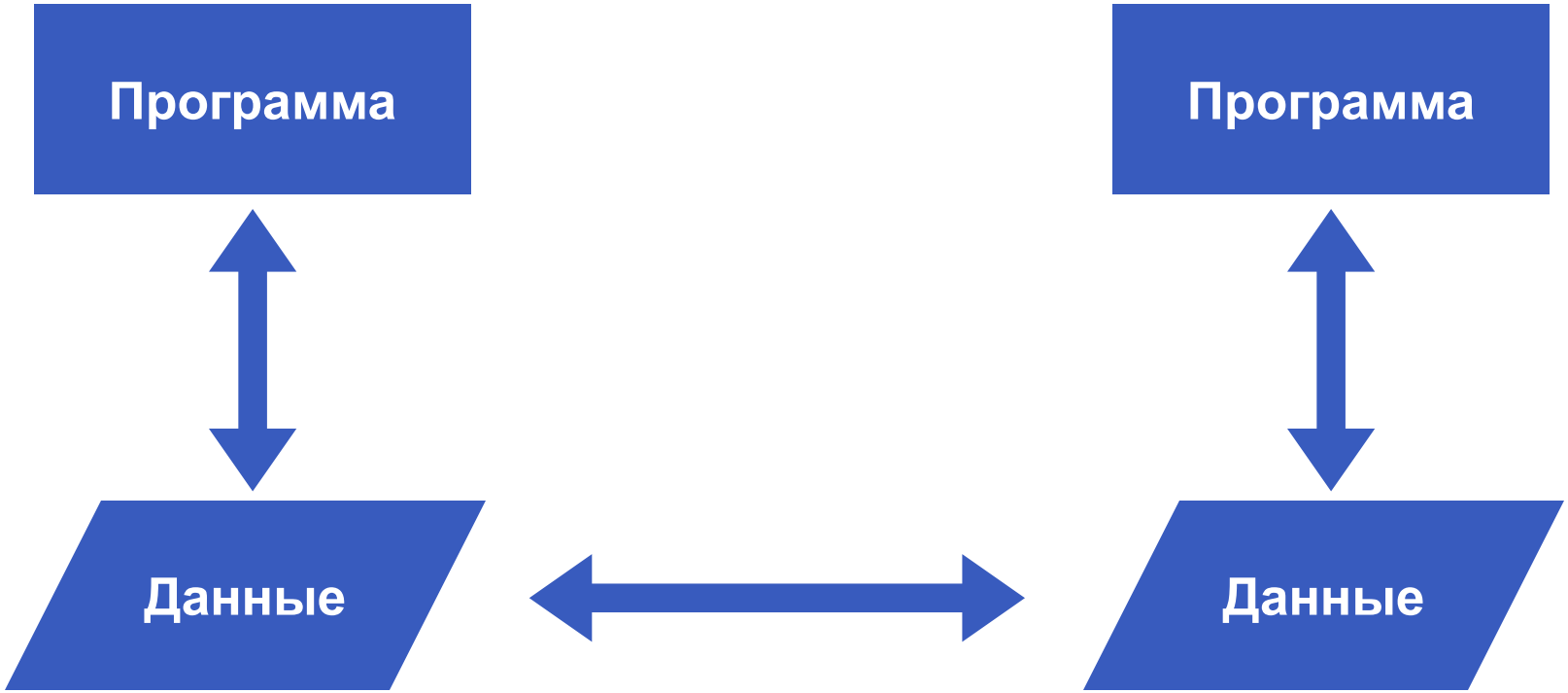
Совместное использование данных



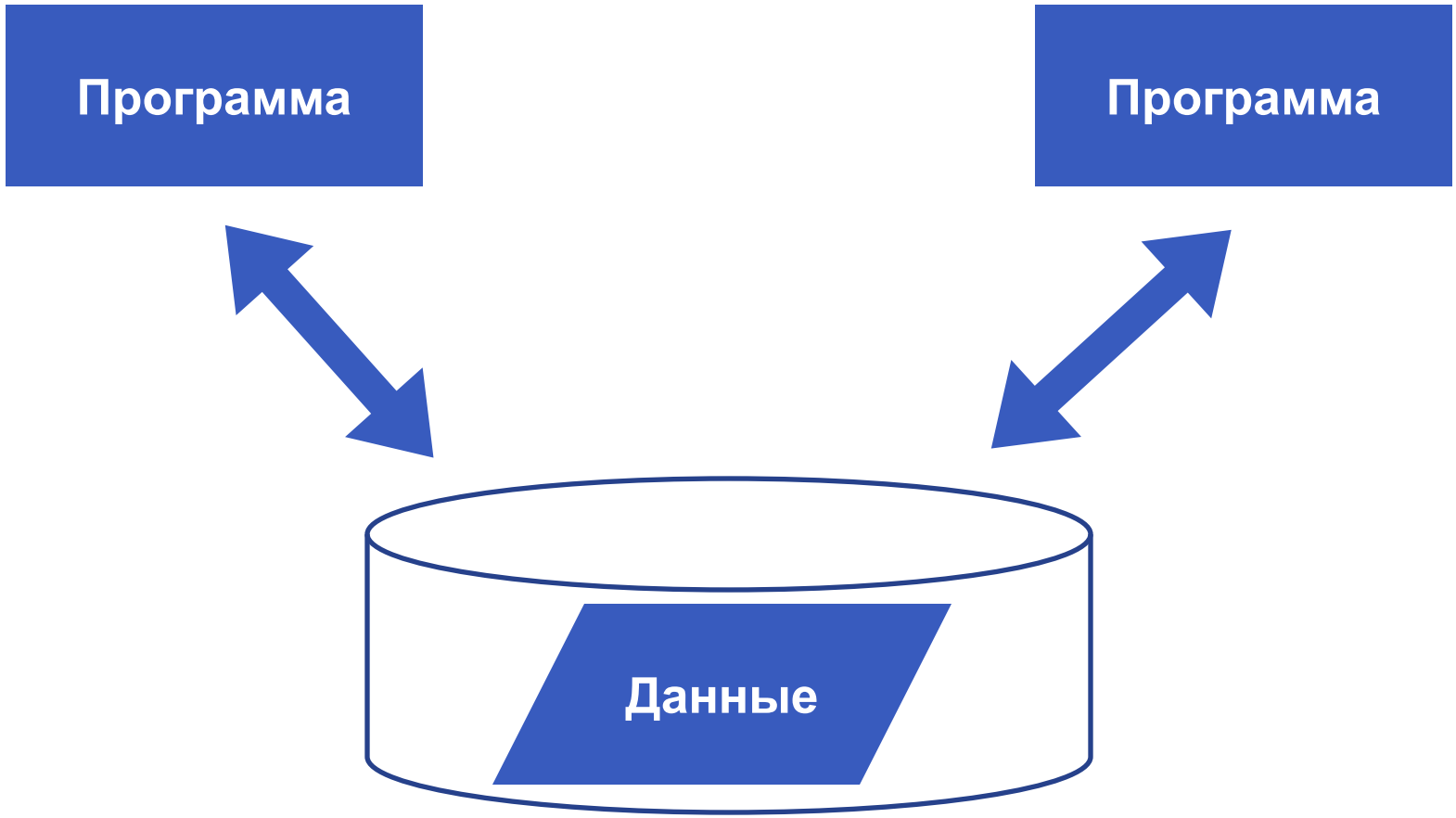
ПЕРВЫЙ ЗАКОН СОЗДАНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ

НЕ СОЗДАВАЙТЕ РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ!

Репликация данных



Совместное использование данных



ВТОРОЙ ЗАКОН СОЗДАНИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ

МИНИМИЗИРУЙТЕ ВЗАИМОДЕЙСТВИЕ РАСПРЕДЕЛЕННЫХ ЧАСТЕЙ!

Причины создания распределенных приложений

- Улучшение функциональных характеристик системы:
  - Производительность
  - Повышение надежности, устойчивости к сбоям
  - Специализация компонентов: упрощение и удешевление
  - Отношение цена/производительность

Принципы построения

- Функциональное разделение
- Естественное разделение
- Балансировка нагрузки

Функциональное разделение

- Узлы выполняют различные задачи
  - Клиент / Сервер
  - Хост / Терминал
  - Сбор данных / Обработка данных
- Решение – создание разделяемых сервисов
- Сервисы
  - Управляют набором ресурсов
  - Предоставляют услуги пользователям

Естественное разделение

- Разделение определяется задачей
  - Система обслуживания сети супермаркетов
  - Сеть для обеспечения коллективной работы
- Функциональная нагрузка одинаковая у однотипных узлов

Балансировка нагрузки

- Функциональность может повторяться
- Задачи назначаются на процессоры таким образом, чтобы нагрузка была равномерной
- Такой подход обеспечивает наибольшую эффективность использования вычислительных ресурсов

Заблуждения относительно компьютерных сетей

1. Безотказность, надежность сетей
2. Нулевая латентность
3. Пропускная способность не ограничена
4. Сеть является защищенной
5. Топология неизменна
6. Есть один администратор
7. Транспортные затраты – нулевые
8. Сеть является однородной

Последствия распределенности

- **Параллельность**
  - Параллельное выполнение (независимые процессы)
    - Вопросы синхронизации
    - «Гонки потоков» (concurrency)
  - Совместное использование ресурсов
    - Коллизии при доступе
    - Данные, сервисы, устройства
  - Типичные проблемы
    - Взаимные блокировки (deadlocks)
    - Ненадежные коммуникации

Последствия распределенности

- **Нет “глобального” времени**
  - Асинхронная передача сообщений
  - Ограниченная точность синхронизации часов
- **Нет состояния системы**
  - В распределенной системе нет ни одного процесса, который бы знал текущее глобальное состояние системы
    - Следствие параллелизма и механизма передачи данных

Последствия распределенности

- **Сбои**
  - Процессы выполняются автономно, изолированно
  - Неудачи отдельных процессов могут остаться необнаруженными
  - Отдельные процессы могут не подозревать об общесистемном сбое
  - Сбои происходят чаще, чем в централизованной системе
  - Новые причины сбоев (которых не было в монолитных системах)
  - Сетевые сбои изолируют процессы и фрагментируют систему на изолированные части

Требования

- Открытость
- Безопасность
- Масштабируемость
- Механизмы обработки ошибок и восстановления после сбоев
- Методы решения проблем параллелизма
- Прозрачность
- Управляемость

Открытость

- Гарантирует расширяемость
- Возможность повторного использования
- Важные факторы:
  - Наличие четких спецификаций
  - Наличие полной документации
  - Опубликованные интерфейсы
  - Тестирование и проверка на многих платформах
- Использование открытых протоколов и стандартов

Безопасность

- Физическая распределенность означает возможность доступа злоумышленников к компонентам
- Три компонента:
  - Защищенность
  - Целостность
  - Доступность
- Задача: посылка важной информации по сети безопасно и эффективно

Безопасность

- Сценарий 1: Доступ к результатам тестирования
  Откуда мы знаем, что пользователь – преподаватель, имеющий доступ к данным?
  - Аутентификация и авторизация

- Сценарий 2: Посылка номера кредитной карты в интернет-магазин
  - Никто кроме получателя не должен прочитать данные
  - Криптография

Безопасность

- Системы распределенного хранения
  - Шифрование данных
  - Обеспечение целостности данных
- Нерешенные проблемы
  - Атаки типа DoS (отказы в обслуживании)
  - Безопасность мобильного кода
    - Непредсказуемые эффекты
    - Может вести себя подобно троянскому коню...

Масштабируемость

- Распределенная система масштабируема, если она остается эффективной при увеличении числа обслуживаемых пользователей или ресурсов

- Проблемы:
  - Контроль стоимости ресурсов
  - Контроль потерь производительности

Масштабируемость

- Стоимость физических ресурсов
  - Растет при увеличении числа пользователей
  - Не должна расти быстрее, чем *O* (n), где n – количество пользователей
- Потери производительности
  - Увеличиваются с ростом размера данных (и количества пользователей)
  - Время поиска не должно расти быстрее, чем *O* (log n), где n – размер данных

Масштабируемость

- Существуют естественные ограничения
  - Некоторые определяются легко
  - Другие труднее
- Обход узких мест
  - Децентрализация алгоритмов
    - Пример – Domain Name Service (DNS)
  - Тиражирование и кэширование данных

Обработка сбоев

- Сбои возникают чаще, чем в централизованных системах, но обычно носят локальный характер

- Обработка сбоев включает в себя:
  - Определение факта сбоя (может быть невозможно)
  - Маскирование
  - Восстановление

Обработка сбоев

- Диагностика
  - В ряде случаев возможна
    - Ошибки передачи могут быть обнаружены с помощью контрольных сумм
  - В ряде случаев невозможна
    - Невозможно определить, удаленный сервер не работает или просто очень загружен?

Обработка сбоев

- Маскирование
  - Многие сбои могут быть скрыты
  - Маскирование может быть невозможно
  - Использование маскирования не всегда приводит к правильным последствиям

Прозрачность

- **Прозрачность** – это сокрытие гетерогенной и распределенной структуры системы таким образом, чтобы пользователю система представлялась монолитной

- Степень прозрачности взаимосвязана с производительностью

Прозрачность

- Прозрачность доступа
  Скрывается разница в представлении данных и доступе к ресурсам
- Прозрачность расположения
  Скрывается местоположения ресурса
- Прозрачность переноса
  Скрывается факт перемещения ресурса в другое место
- Прозрачность смены местоположения
  Скрывается факт перемещения ресурса в другое место в процессе обработки

Прозрачность

- **Прозрачность репликации**
  Скрывается факт и особенности репликации ресурса

- **Прозрачность параллелизма**
  Возможность нескольким процессам параллельно работать с ресурсами, не оказывая влияния друг на друга

- **Прозрачность обработки ошибок**
  Защита программных компонентов от сбоев, произошедших в других программных компонентах; восстановление после сбоев

Прозрачность

- **Прозрачность мобильности**
  Возможность переноса приложения между платформами, без его переделки
- **Прозрачность производительности**
  Возможность конфигурации системы с целью увеличения производительности при изменении состава платформы выполнения
- **Прозрачность масштабируемости**
  Возможность увеличения производительности без изменения структуры программной системы и используемых алгоритмов

Прозрачность

- Критически важными являются:
  - Прозрачность доступа
  - Прозрачность расположения
  - Прозрачность паралеллизма

Управляемость

- Распределенные ресурсы не имеют центральной точки управления
- Локальная оптимизация не всегда означает глобальную оптимизацию
  - Нужно учитывать гетерогенность и другие особенности
  - Нужен глобальный взгляд на проблему
  - Он не всегда возможен (есть системы, никому конкретно не принадлежащие)

Сложности при реализации

- Выбор архитектуры
- Гетерогенность среды
- Сложность развертывания
- Сложность отладки

Архитектура

- Важнейшие характеристики системы очень сильно зависят от выбранной архитектуры
  - Интерфейс модулей системы определяет количество «нелокальных» вызовов
  - Практическая невозможность отказа от принятых в начале проектирования неверных решений
- Решение: использование широко известных шаблонов (паттернов) архитектур

Гетерогенность

- **Гетерогенные** – разнородные
- Различаются:
  - Сетевые инфраструктуры
  - Оборудование (например, Intel и Motorolla)
  - Программное обеспечение (например, UNIX sockets и Winsock calls)
  - Языки программирования
  - Представления данных
- Различные компоненты системы выполняются на различных платформах
- Различия должны быть скрыты

Гетерогенность

- Интерфейсы и реализация могут быть разными, но базовые концепции обычно неизменны
- Средства борьбы с гетерогенностью – стандарты
- Решения:
  - Использование распространенных открытых стандартов и протоколов
  - Использование промежуточного программного обеспечения (middleware)

Гетерогенность

- Middleware: промежуточный программный слой
  - Позволяет гетерогенным узлам взаимодействовать
  - Определяет однородную вычислительную модель
  - Поддерживает один или несколько языков программирования
  - Обеспечивает поддержку распределенных приложений
    - Вызов удаленных объектов
    - Удаленный вызов SQL
    - Распределенная обработка транзакций
- Примеры: CORBA, JavaEE, .Net

Сложность развертывания

- Фрагментация
  - Разделение приложения на модули развертывания
- Конфигурация
  - Связь модулей друг с другом (зависимости)
- Размещение
  - Выгрузка модулей в целевую систему
  - Распределение вычислительных модулей между узлами (статическое или динамическое)
  - Использование специализированных инструментальных средств (deployment tools)

Сложность отладки

- Нет глобального состояния
- Параллельность приводит к неповторяемости (невоспроизводимости) результатов выполнения
- Компоненты распределены по разным узлам

Предварительные итоги

- Распределенная система:
  - Автономные (но соединенные средой передачи данных) узлы
  - Взаимодействие посредством передачи сообщений
- Много доводов в пользу того, что распределенные системы нужны и их нужно уметь строить
- Распределенные системы существуют и их нужно уметь развивать и поддерживать
- При разработке распределенных систем возникают специфические проблемы

Спасибо за внимание!

Дополнительные источники

- Таненбаум, Э. Распределенные системы. Принципы и парадигмы [Текст] / Э. Таненбаум, М. ван Стеем. – СПб. : Питер, 2003. – 877 с.
- Эндрюс, Г.Р. Основы многопоточного, параллельного и распределенного программирования [Текст] / Грегори Р. Эндрюс. – М. : Издательский дом «Вильямс», 2003. – 512 с.
- Фаулер, М. Архитектура корпоративных программных приложений [Текст] / Мартин Фаулер. – М. : Издательский дом «Вильямс», 2007. – 544 с.
- Обзор распределённых систем [Электронный ресурс]. – Режим доступа: http://masters.donntu.edu.ua/2008/fvti/prihodko/library/dist2.htm, дата доступа: 14.10.2013.
- Распределённые вычисления [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/wiki/Распределенные_вычисления, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Модели и архитектура

Занятие 2

Гаврилов А.В.

Самара
2013

План занятия

- Модели распределенных систем
- Архитектура распределенных систем
- Слои и уровни
- Возможные архитектурные решения

Модели распределенных систем

- Модели архитектуры
  - Физическое размещение компонентов между узлами
  - Взаимодействие между компонентами
- Другие модели
  - Формальное описание различных параметров и свойств системы
  - Модели взаимодействия, обработки ошибок, безопасности, …

Модели архитектуры

- Модель архитектуры распределенной системы должна содержать решение двух проблем:
  - физическое размещение компонентов между узлами
  - взаимодействие между компонентами

Модели архитектуры

- Реальные функции отдельных компонентов не указываются
- Указываются:
  - Расположение (размещение по узлам)
  - Шаблоны распределения данных и задач по их обработке
  - Взаимодействие компонентов
    - Роли компонентов
    - Шаблоны взаимодействия

Архитектура

- Определяет разделение системы на наиболее крупные составные части
- Определяет конструктивные решения, которые после их принятия с трудом поддаются изменению
- Отображает общий взгляд разработчиков на результаты проектирования системы: идентификация главных компонентов системы, способов их взаимодействия, выбор основополагающих решений, не подлежащих изменению в будущем

Основные принципы архитектуры

- **Согласованность**
  Частичное знание системы позволяет предсказать остальное
- **Ортогональность**
  Функции независимы и специфицированы по отдельности
- **Соответствие**
  Включаются только функции, соответствующие существенным требованиям к системе, нет ненужных функций
- **Экономичность**
  Отсутствие дублирования

Основные принципы архитектуры

- **Прозрачность**
  Функции должны быть известны пользователю
- **Общность**
  Если функция должна быть введена, ее следует вводить в таком виде, чтобы она отвечала как можно большему числу назначений
- **Открытость**
  Можно использовать функцию иначе, чем это предполагалось при проектировании
- **Полнота**
  Введенные функции должны с учетом экономических и технологических ограничений как можно полнее соответствовать требованиям и пожеланиям пользователя

Преодоление сложности

- Использование паттернов проектирования
- Разделение системы на слои (расслоение)

Типовые решения – паттерны проектирования

Кристофер Александер
(Christopher Alexander):

Каждое типовое решение описывает некую повторяющуюся проблему и ключ к ее разгадке, причем таким образом, что вы можете пользоваться этим ключом многократно, ни разу не придя к одному и тому же результату

Структура типовых решений

- Название решения
- Назначение (аннотация)
- Мотивация, применимость
- Принцип действия (структура, участники, отношения)
- Результаты
- Реализация

ПРАВИЛО ДОБАВЛЕНИЯ КОСВЕННОСТИ

Если есть проблема – введите посредника, т.е. вместо прямого взаимодействия используйте косвенное

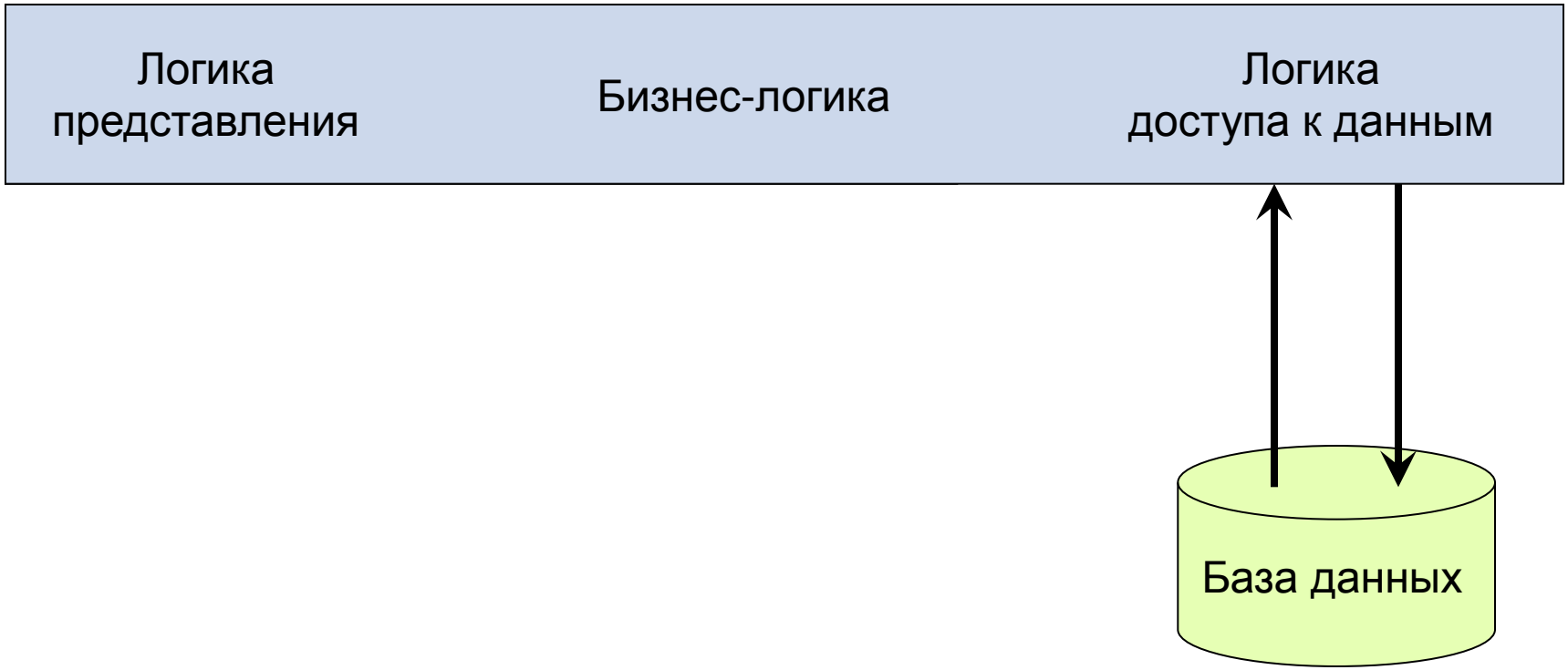
Слои

- Основная идея: независимость нижележащих уровней от вышележащих
- Основная задача: уменьшать сложность систем, разделяя их на слои и сервисы
  - Слой (уровень): группа сильно связанных и закрытых элементов, реализующих одну функциональность
  - Сервис: функциональность, обеспечиваемая для вышестоящего слоя

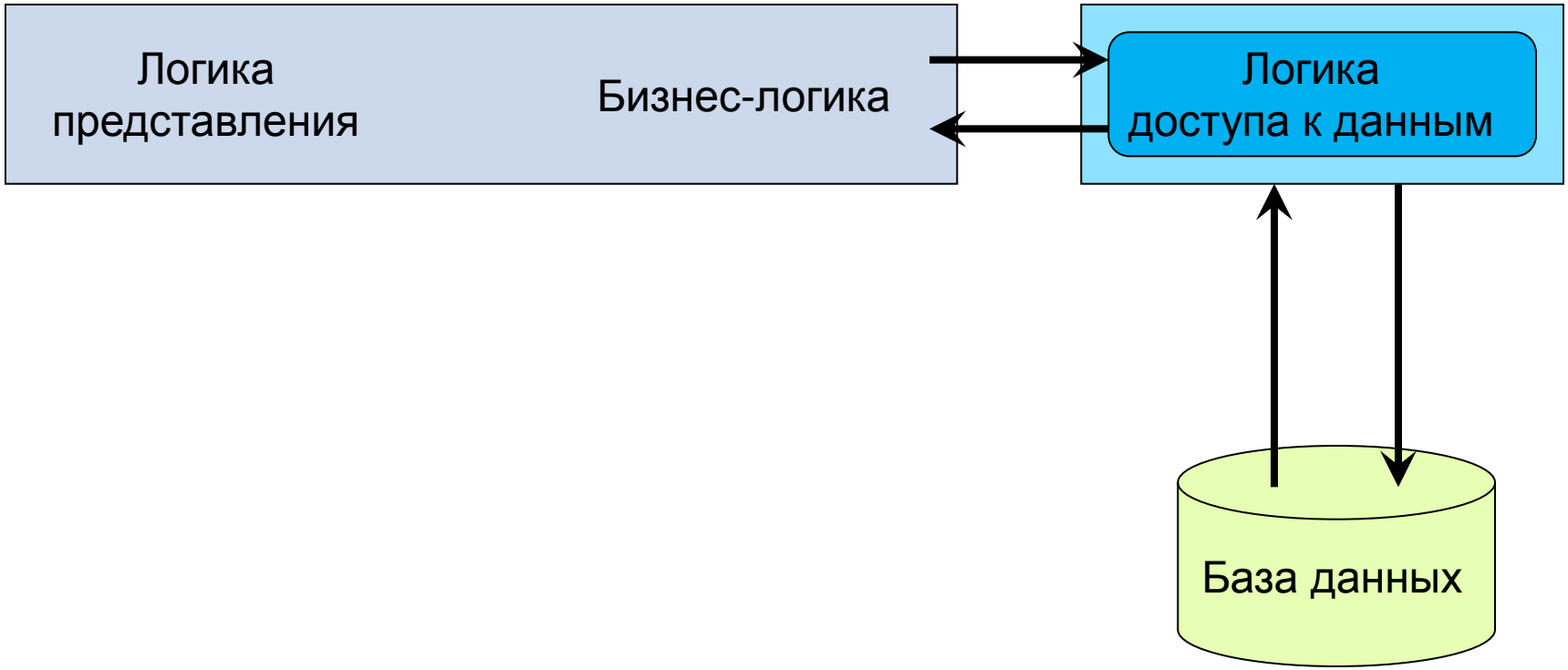
Примеры подхода

- Архитектура сетевых протоколов
  - Физический уровень
  - Уровень соединения
  - Сетевой уровень
  - Транспортный уровень
  - Сеансовый уровень
  - Уровень представления
  - Прикладной уровень
- Распределенные приложения
  - Аппаратура
  - Операционная система } **Платформа**
  - Промежуточное программное обеспечение
  - Приложения, сервисы

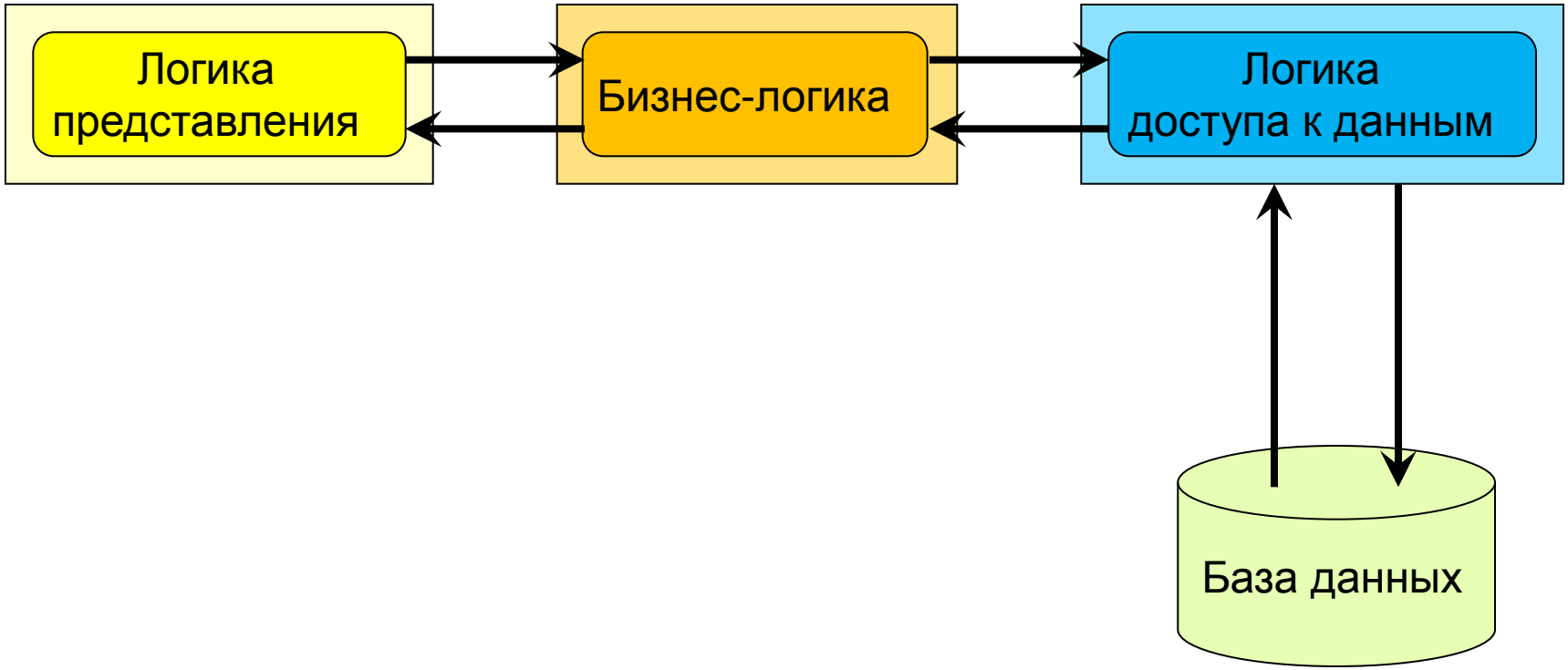
Монолитная система



Двухслойная система



Трехслойная система

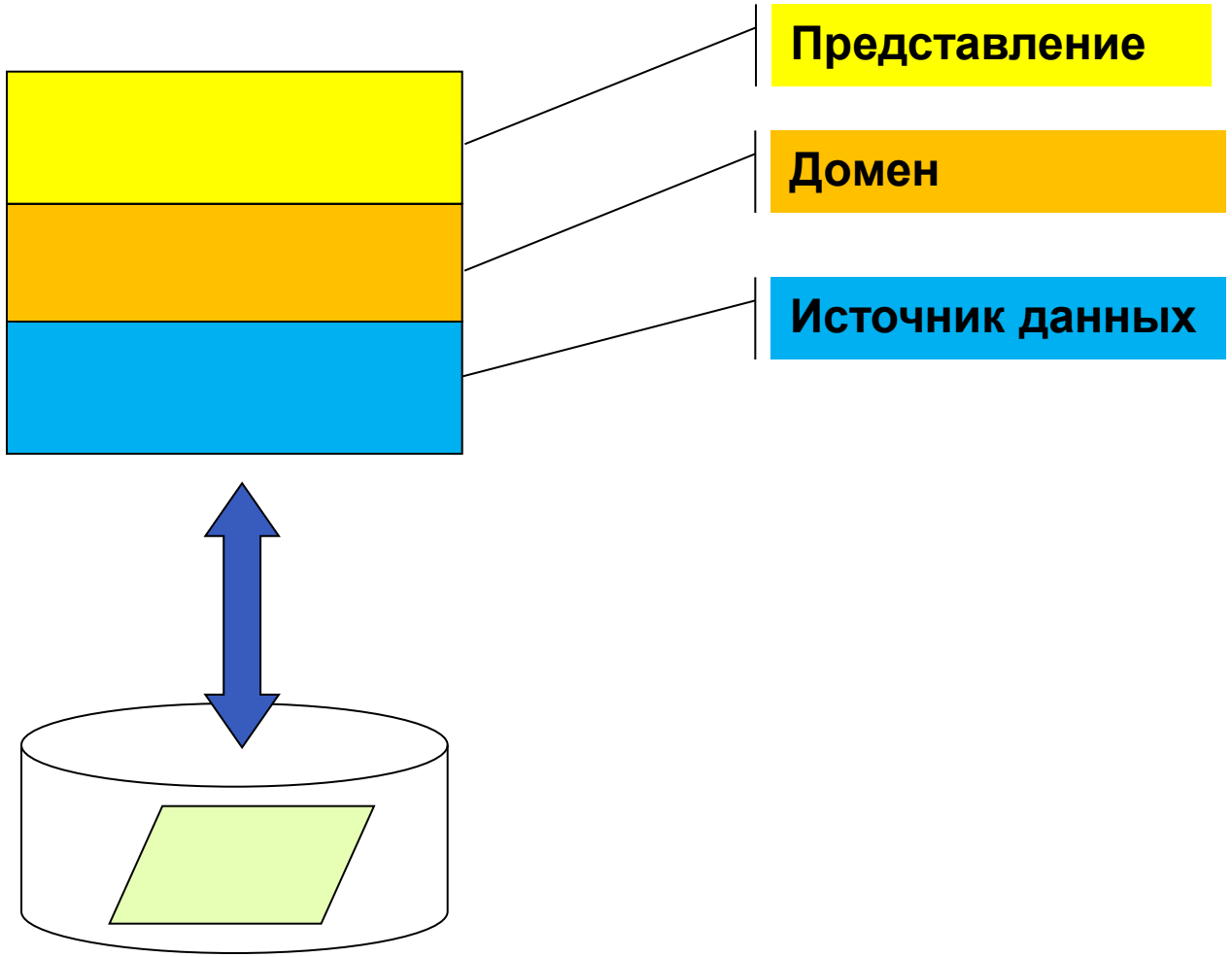


Три основных слоя

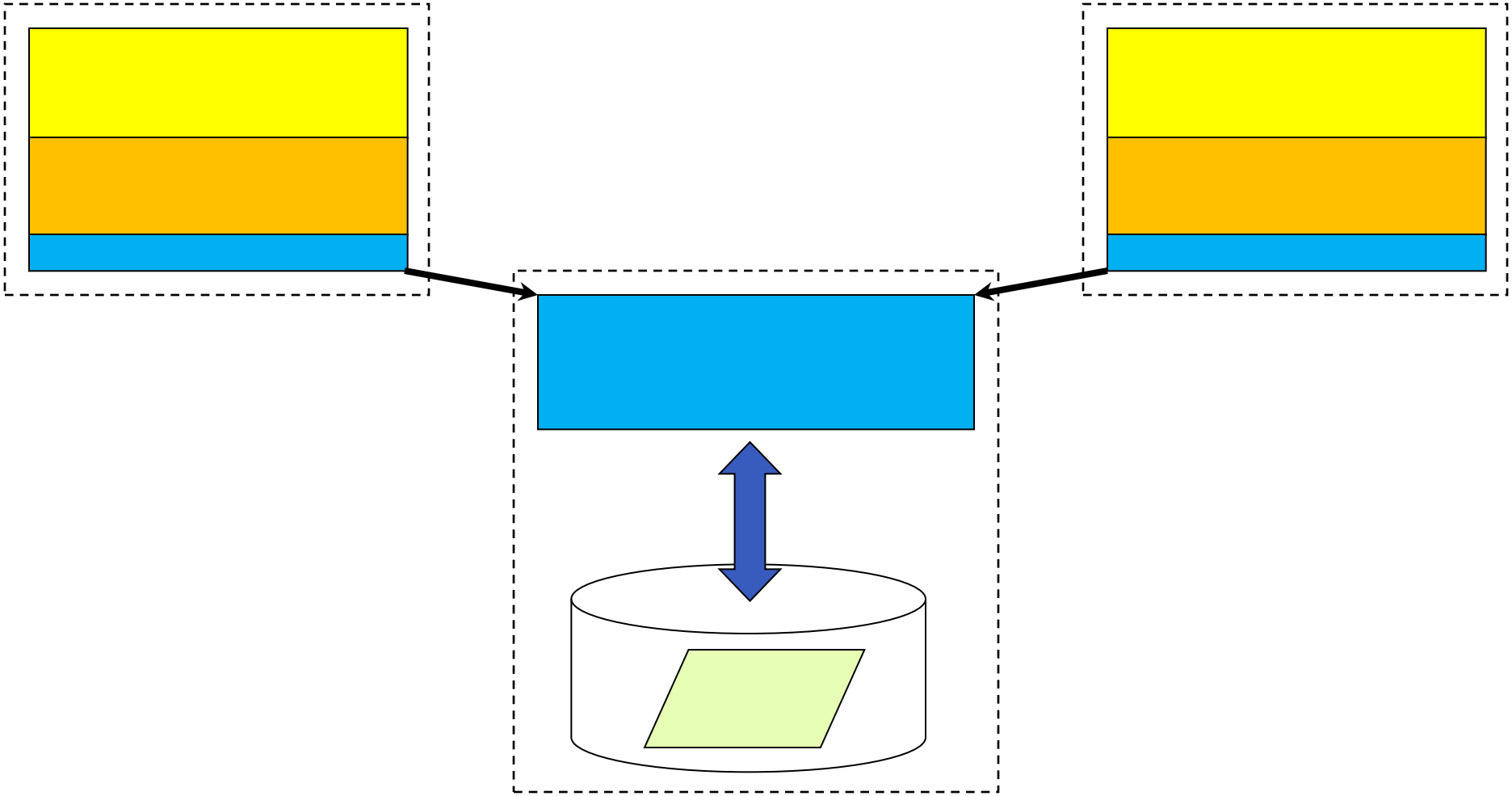
- Слои:
  - представление (presentation)
  - домен (domen) – предметная область, бизнес-логика
  - работа с данными (data source)

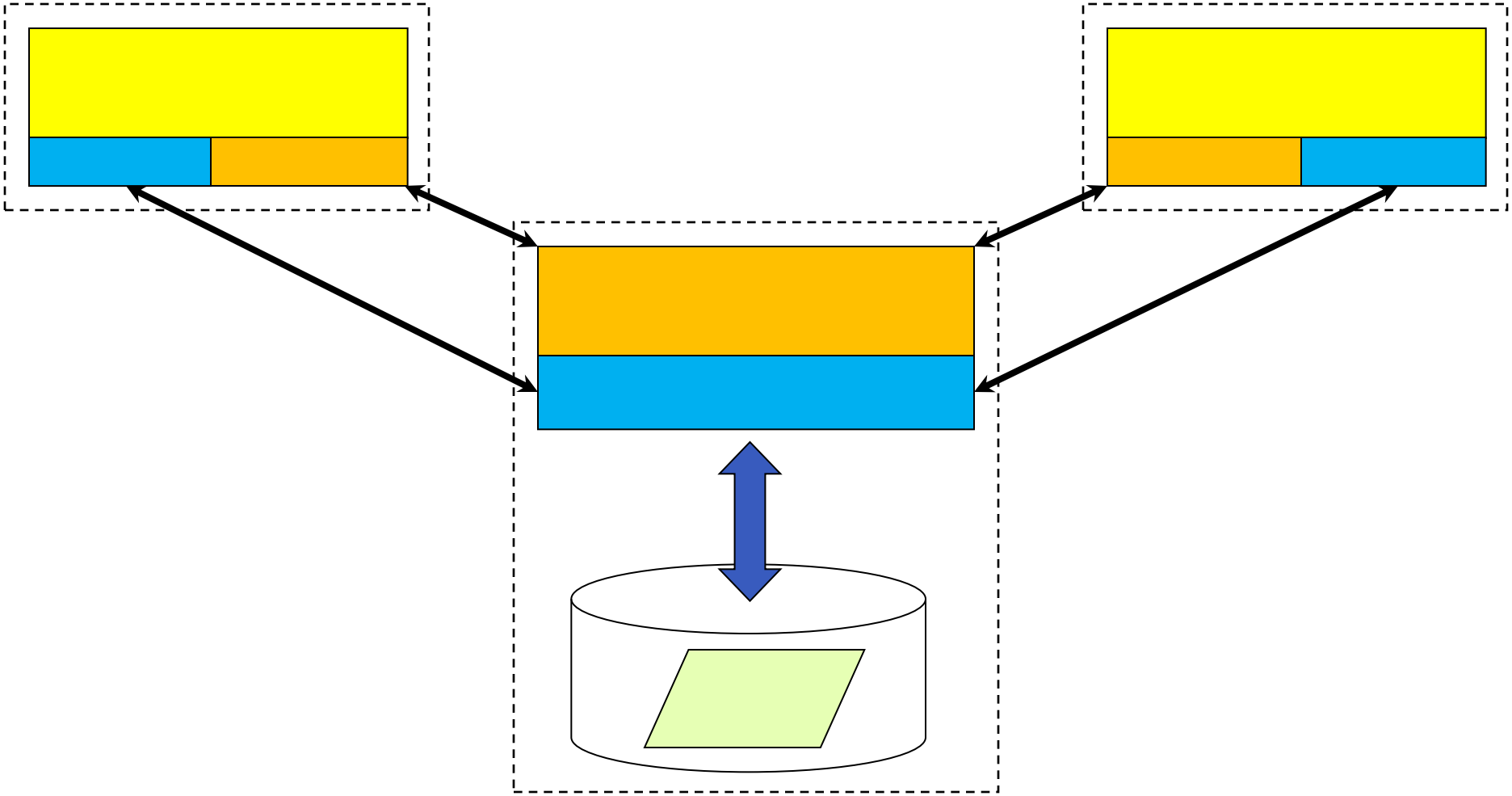
- Рекомендуется различать:
  - **Слой** (layer) – логическое разделение
  - **Уровень** или **Ярус** (tier) – физическое разделение

Одноуровневая система

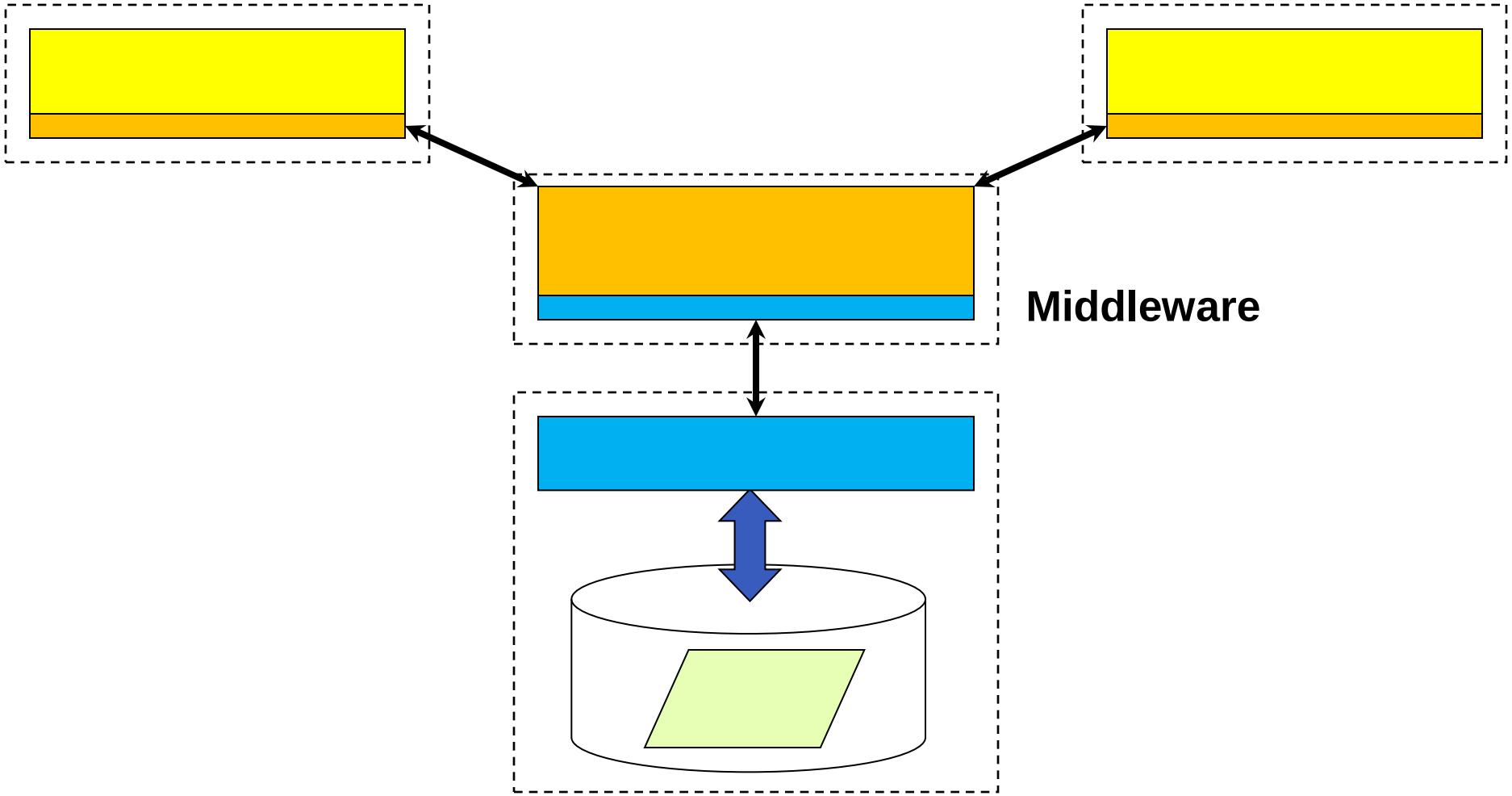


Система клиент/сервер (двухуровневая)





Трехуровневая система



Middleware

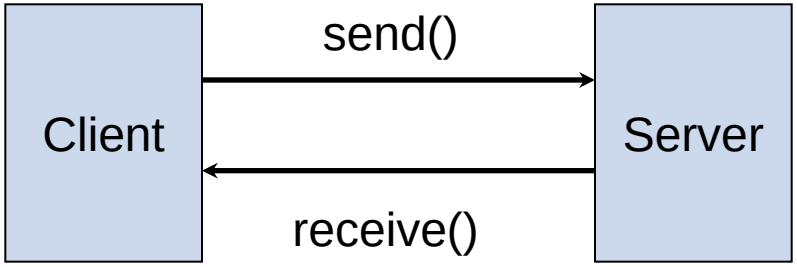
- “Слой программного обеспечения, чья цель состоит в том, чтобы скрывать неоднородность и обеспечивать удобную модель программирования для разработчиков”
  - Предоставляет прикладной интерфейс программирования
- Примеры
  - CORBA (OMG)
  - .Net (Microsoft)
  - Remote Procedure Call (Sun)
  - Java Remote Method Invocation (Sun)
  - Технология EJB (Enterprise JavaBeans, Sun)
- Может также предоставлять услуги (сервисы)

Сервисы Middleware

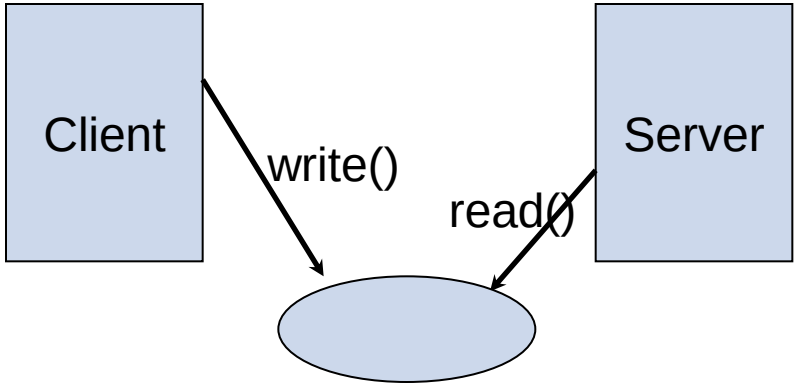
- Сервисы
  - Именования
  - Безопасности
  - Транзакций
  - Долговременного хранения
  - Уведомления о событиях
  - …

Основные парадигмы программирования

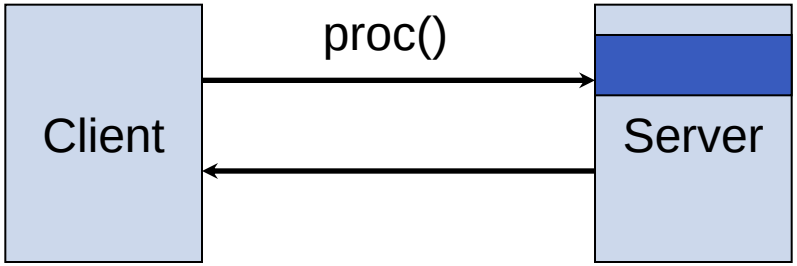
Message Passing



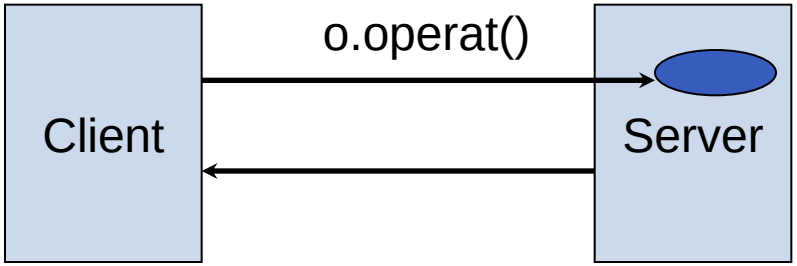
Virtual Shared Memory



Remote Procedure Call



Distributed Object System



Требования к дизайну

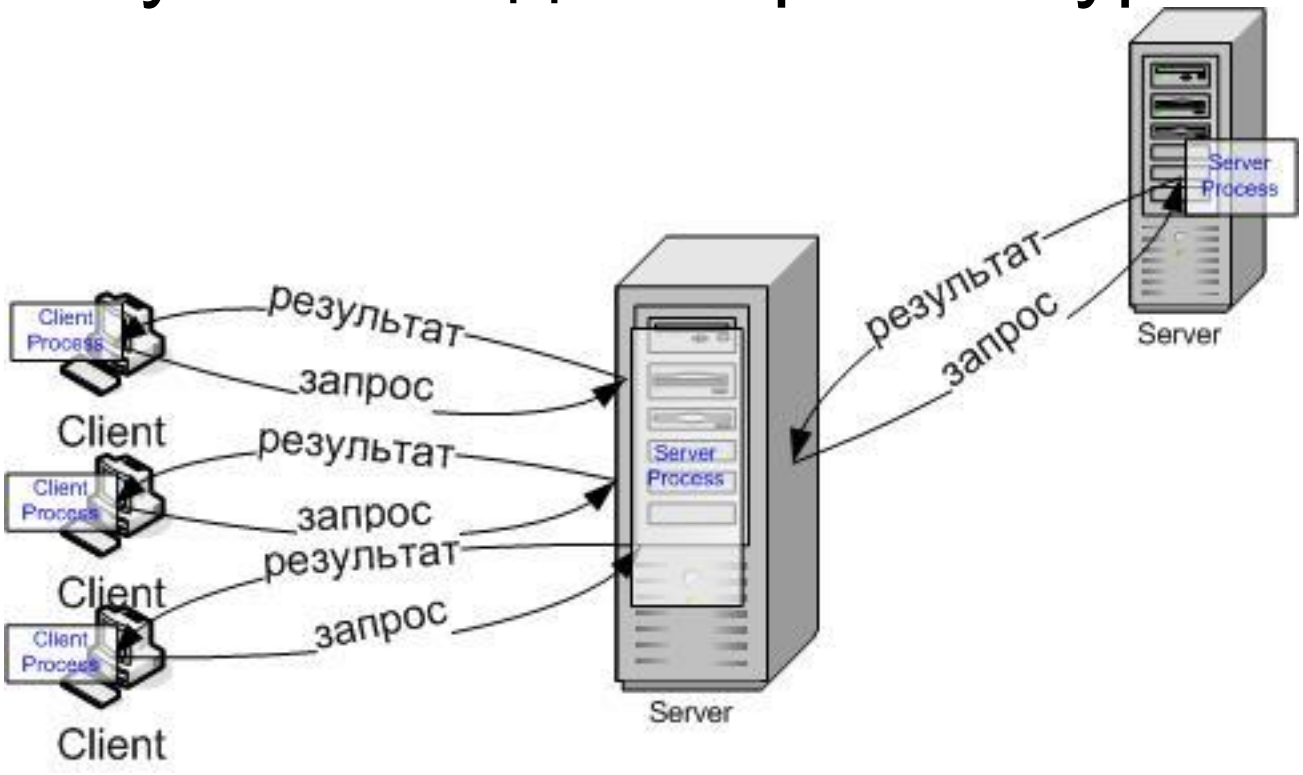
- Требования, накладываемые обеспечением требуемой производительности
  - Время отклика
  - Производительность
  - Балансировка нагрузки
- Использование кэширования и репликации
  - Очень многие проблемы производительности системы могут быть решены путем кэширования данных
- Требование надежности

Возможные архитектуры

- Клиент-сервер
- Модель предоставления услуг пулом серверов
- Модель прокси- и кэш- серверов
- Модель равных процессов

Модель клиент-сервер

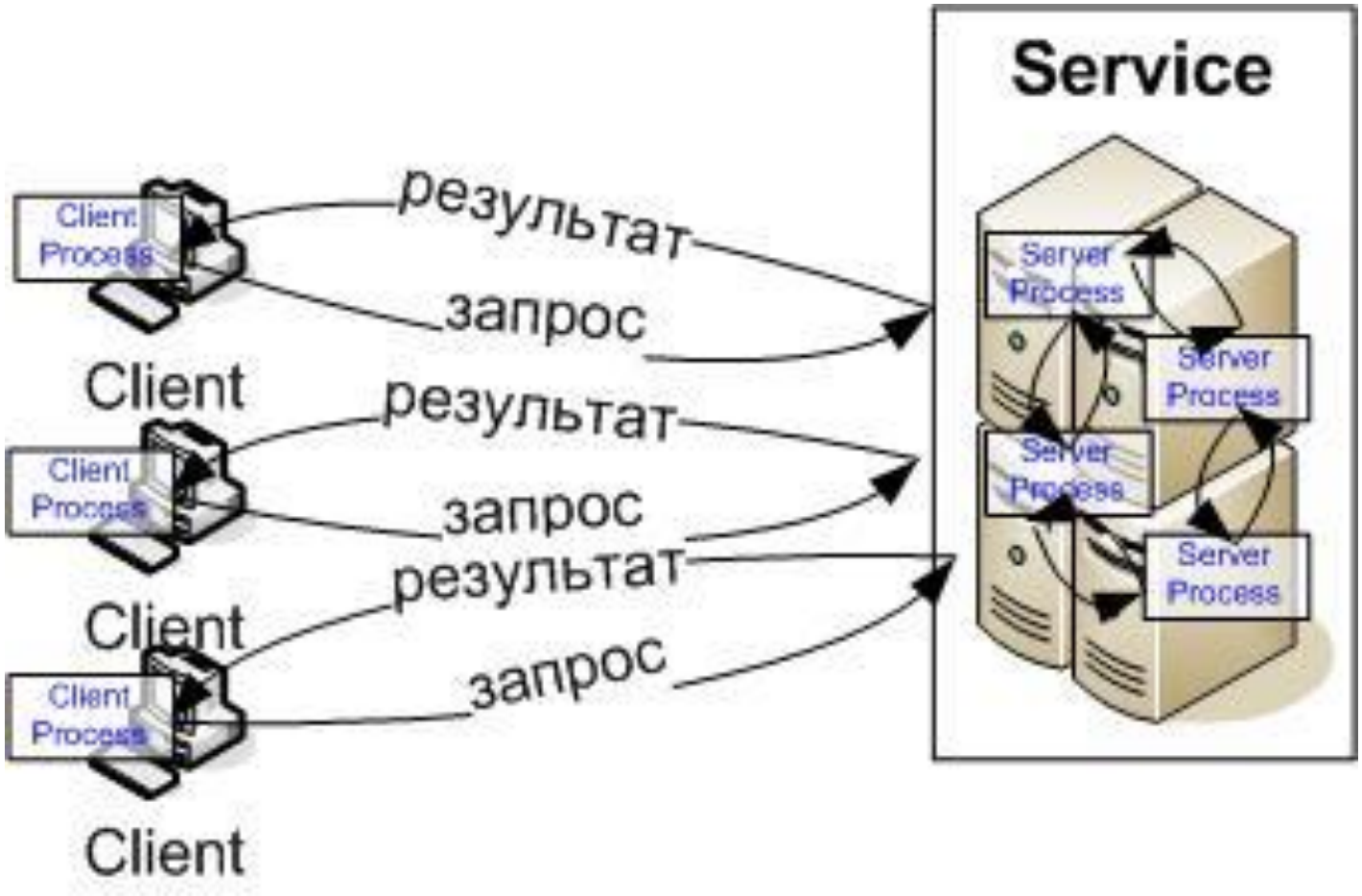
- Наиболее распространенная и часто используемая модель архитектуры



Модель клиент-сервер

- **Клиент**
  Процесс, желающий получить доступ к данным, использует ресурсы и/или выполняет действия на удаленном узле
- **Сервер**
  Процесс, управляющий данными и всеми другими разделяемыми ресурсами, обеспечивающий клиентам доступ к ресурсам и производящий вычисления
- **Взаимодействие**
  Пары запрос / результат
- **Пример**
  http-сервер: клиент (браузер) запрашивает страницу, сервер поставляет страницу

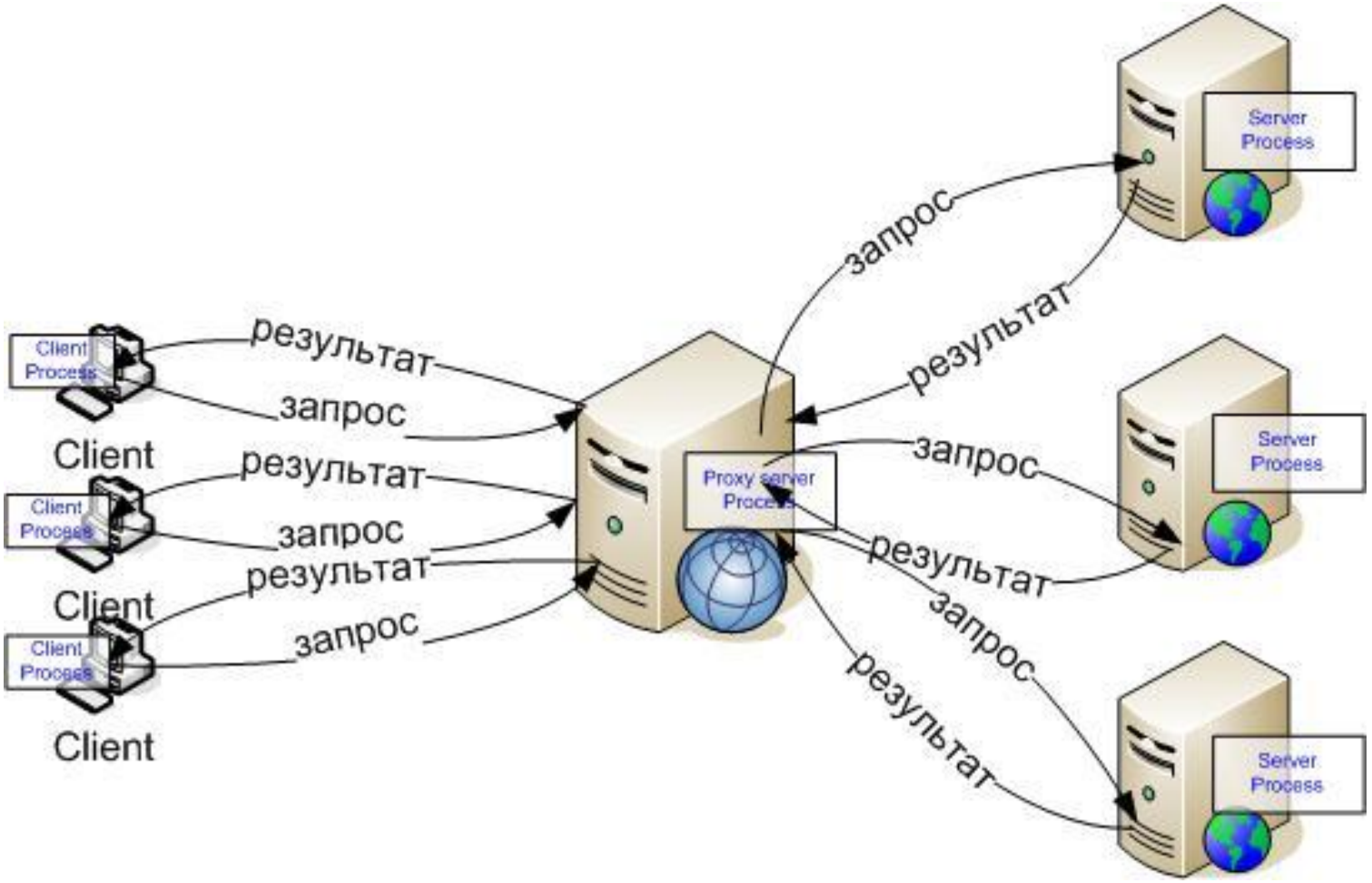
Модель «пул серверов»



Модель «пул серверов»

- Услуги могут обеспечиваться многими серверами
- Распределенные между серверами объекты
- Реплицированные объекты
  - Увеличение производительности, доступности и отказоустойчивости
- Но требуют координации копий / консистентности представления
- Например, высокодоступные серверы (порталы, диллинговые центры), информационные службы
- Серверы, обслуживающие распределенную БД

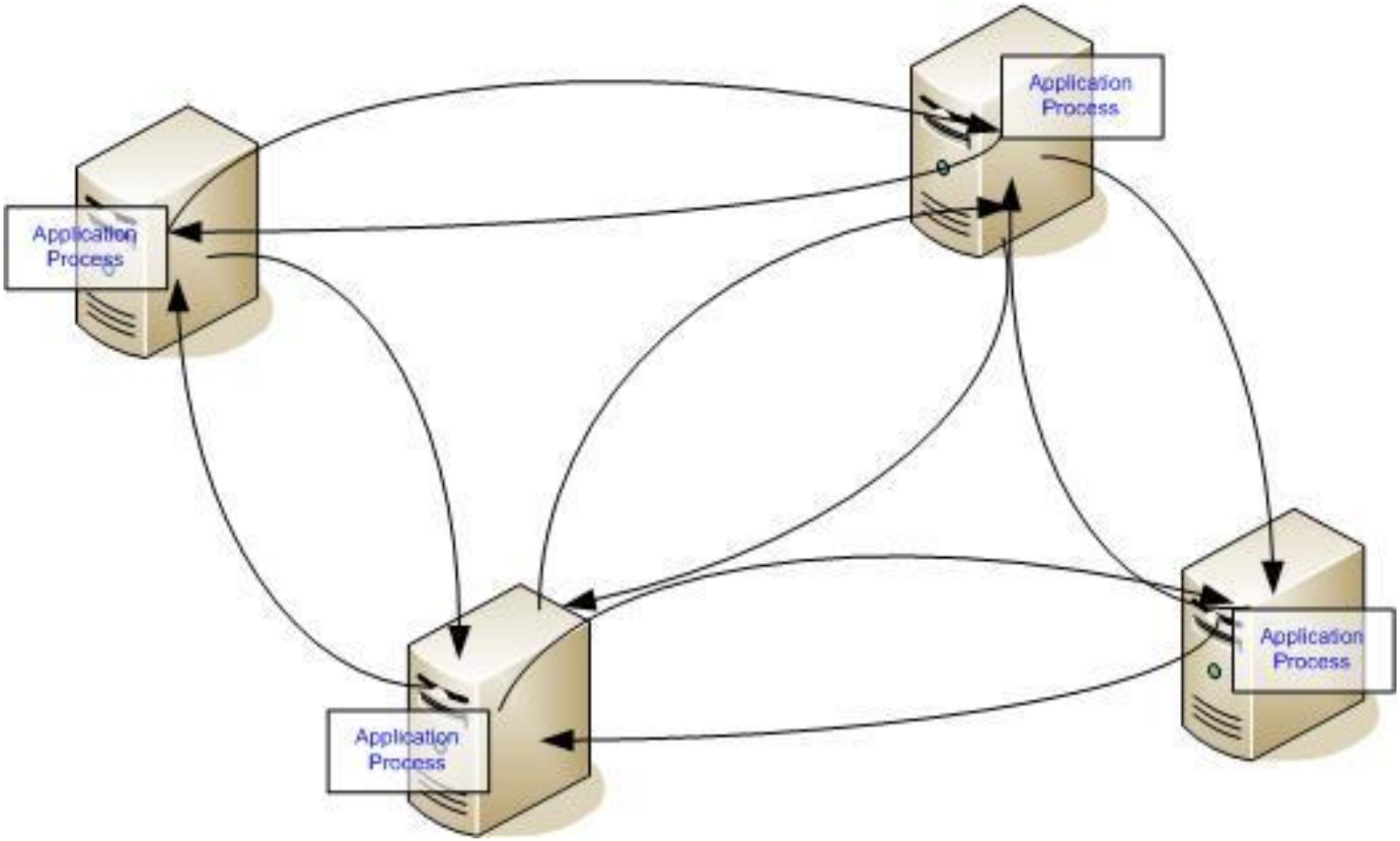
Модель с прокси-сервером



Модель с прокси-сервером

- **Кэш**: «близкая» копия наиболее часто используемых данных
  - **значительно** повышает производительность большинства приложений
  - Но требует усилий по поддержанию когерентности
- **Прокси-сервер**: разделяемый кэш ресурсов
  - Еще сложнее, чем простой кэш…
  - Обычно используется (и хорошо подходит) для доступа к веб-ресурсам

Равноправные процессы (P2P)



Равноправные процессы (P2P)

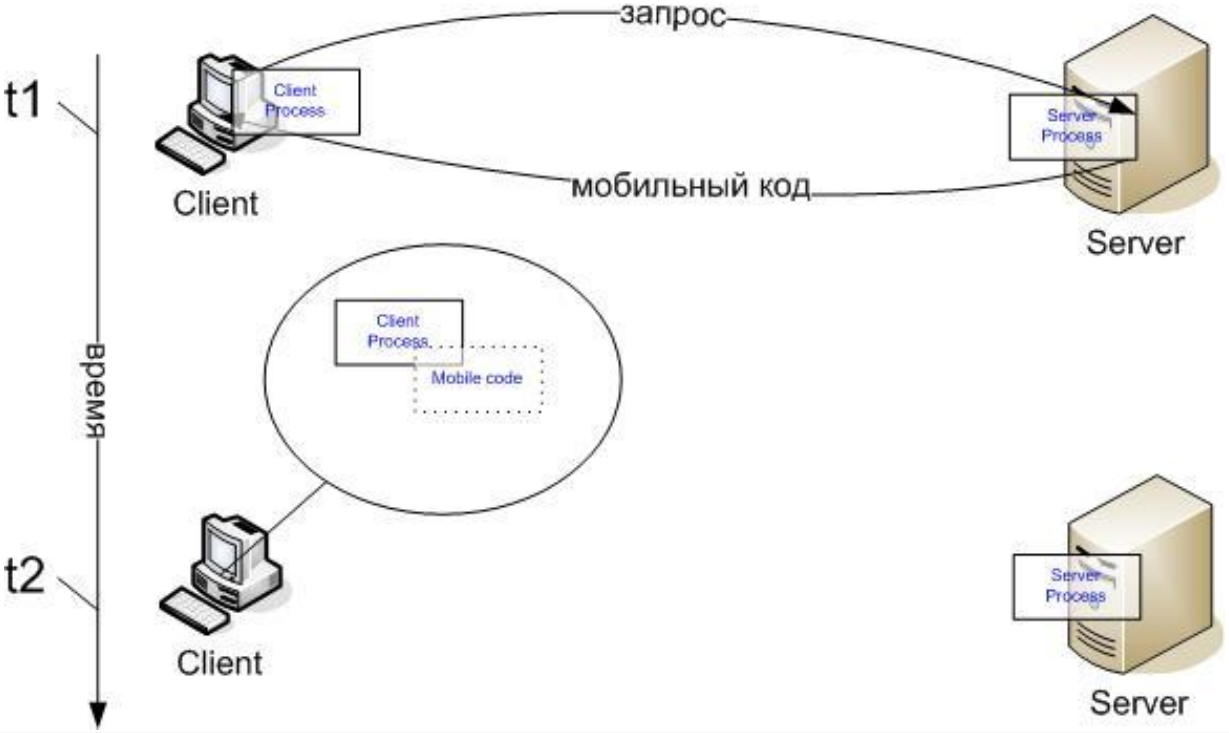
- Равные процессы: процессы, которые играют равные роли
  - Никакого абсолютного различия между клиентом / сервером
  - Роли клиента и сервера различаются от вызова к вызову (или со временем)
- Увеличивает устойчивость к сбоям и масштабируемость
- Трудности с координацией
- Примеры: BitTorrent, распределенное вычисление

Вариации модели клиент-сервер

- Вариации возможны по следующим параметрам:
  - Связь инициируется сервером
  - Использование мобильного кода или мобильных агентов
  - Легкие клиенты базирующиеся на потребности пользователей в дешевых компьютерах и простом управлении (тонкий клиент)

Мобильный код

- Мобильный код: код, посланный процессу клиента, чтобы выполнить его же задачу



Мобильные агенты

- Выполнение программы (код + данные), которая перемещается между узлами в сети
  - Выполняет автономную задачу обычно под управлением некоторого другого процесса
  - Имеет внутреннее знание и цели
- Преимущество: всюду локальный доступ
  - Снижает затраты на коммуникации
- Потенциальная угроза безопасности
  - Ограниченная применимость
- Примеры: сбор данных из многих источников, установка программ, программы типа червей

Тонкие клиенты

Тонкие клиенты

- Аппаратные
  (сетевые компьютеры, network computers)
  - Все файлы сохраняются на удаленном носителе
  - Минимум локального программного обеспечения
  - Любой локальный диск используется только под кэш
- Программные
  - Реализуют только интерфейс пользователя на локальном компьютере
  - Непосредственно программы работают на вычислительном сервере

Итоги

- Использование модели слоев для снижения сложности системы
  - Middleware обеспечивает дополнительное удобство и дополнительные сервисы
- Выбор модели архитектуры в зависимости от особенностей задачи
  - Клиент – сервер
  - Модель предоставления услуг пулом серверов
  - Модель прокси- и кэш-серверов
  - Модель равных процессов

Спасибо за внимание!

Дополнительные источники

- Таненбаум, Э. Распределенные системы. Принципы и парадигмы [Текст] / Э. Таненбаум, М. ван Стеем. – СПб. : Питер, 2003. – 877 с.
- Эндрюс, Г.Р. Основы многопоточного, параллельного и распределенного программирования [Текст] / Грегори Р. Эндрюс. – М. : Издательский дом «Вильямс», 2003. – 512 с.
- Фаулер, М. Архитектура корпоративных программных приложений [Текст] / Мартин Фаулер. – М. : Издательский дом «Вильямс», 2007. – 544 с.
- Обзор распределённых систем [Электронный ресурс]. – Режим доступа: http://masters.donntu.edu.ua/2008/fvti/prihodko/library/dist2.htm, дата доступа: 14.10.2013.
- Распределённые вычисления [Электронный ресурс]. – Режим доступа: http://ru.wikipedia.org/wiki/Распределенные_вычисления, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Основы создания сетевых приложений на Java

Занятие 3

Гаврилов А.В.

Самара
2013

План занятия

- Протоколы транспортного уровня
- Понятие сокета
- Пакет java.net
- Классы Socket и ServerSocket
- Классы DatagramPacket и DatagramSocket
- Класс URL

Модель OSI

- Прикладной уровень
- Уровень представления
- Сеансовый уровень
- Транспортный уровень
- Сетевой уровень
- Уровень соединения
- Физический уровень

Transmission Control Protocol

- TCP – основанный на соединениях протокол, обеспечивающий надежную передачу данных между двумя компьютерами с сохранением порядка данных

- Используется в: HTTP, FTP, Telnet и др.

User Datagram Protocol

- UDP – не основанный на соединениях протокол, реализующий пересылку независимых пакетов данных, называемых дейтаграммами, от одного компьютера к другому без гарантии их доставки

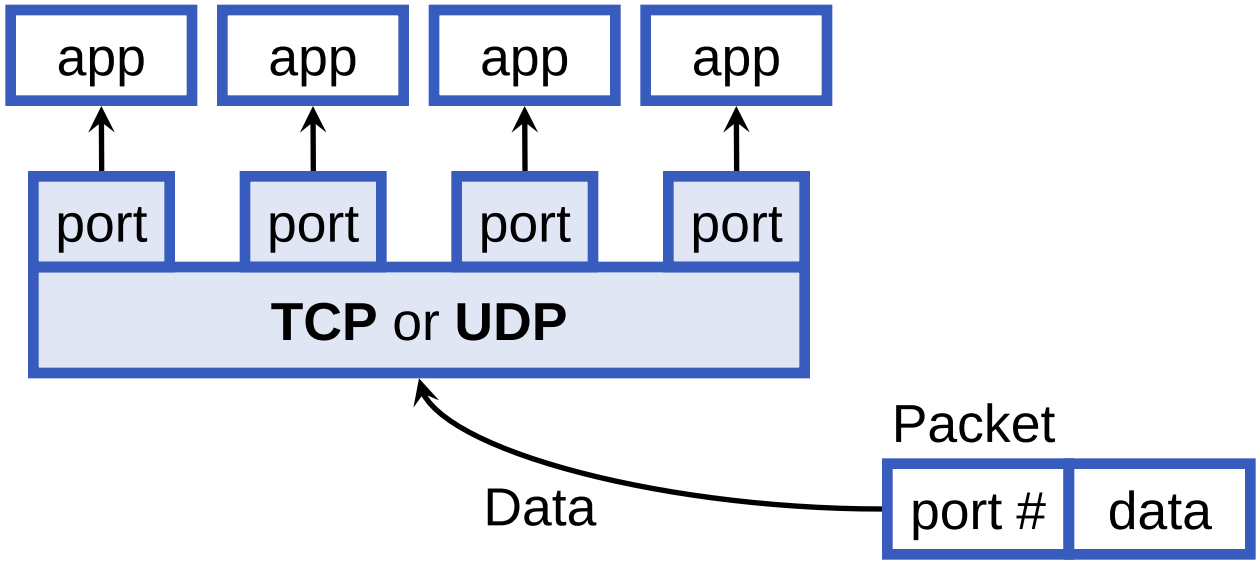
Модель «Клиент-сервер»

- Порядок работы
  - Каждая из сторон виртуального соединения называется «сокет» (socket)
  - Процесс-сервер инициализируется при запуске и далее бездействует, ожидая поступления запроса от клиента
  - Процесс-клиент посылает запрос на установление соединения с сервером, требуя выполнить для него определенную функцию
- Виды приложений-серверов
  - Сервер последовательной обработки запросов
  - Сервер параллельной обработки запросов

Понятие порта

- Компьютер (обычно) имеет только одно физическое соединение с сетью
- Соединение описывается, например, IP-адресом (32 или 128 бит на нынешний момент)
- Как различать информацию для различных приложений?

Понятие порта



- Сокет привязывается к порту
- Порт описывается 16-битным числом
- Порты 0-1023 зарезервированы

Интерфейс сокетов

- В 80-ых годах американское правительственное агентство по поддержке исследовательских проектов (ARPA), финансировало реализацию протоколов TCP/IP для UNIX в Калифорнийском университете в г. Беркли
- Разработан интерфейс прикладного программирования для сетевых приложений TCP/IP (TCP/IP API)
- TCP/IP sockets или **Berkeley sockets**

Связь с файловой системой

- TCP/IP в рамках UNIX
- Интерфейс сокетов – через системные вызовы UNIX
- Системные вызовы ввода-вывода UNIX выглядят как последовательный цикл:
  - открыть
  - считать/записать
  - закрыть
- Нет различий между файлами и внешними устройствами

Проблемы сетевого ввода/вывода

- Модель клиент-сервер не соответствует системе ввода-вывода UNIX
  - Нет пассивных операций ввода-вывода
  - Не умеют устанавливать соединения
  - Используется фиксированный адрес файла
  - Соединение с файлом доступно на протяжении всего цикла запись-считывание
- Для не ориентированных на соединение протоколов фиксированный адрес – проблема: при передаче дейтаграммы адрес есть, а соединения нет

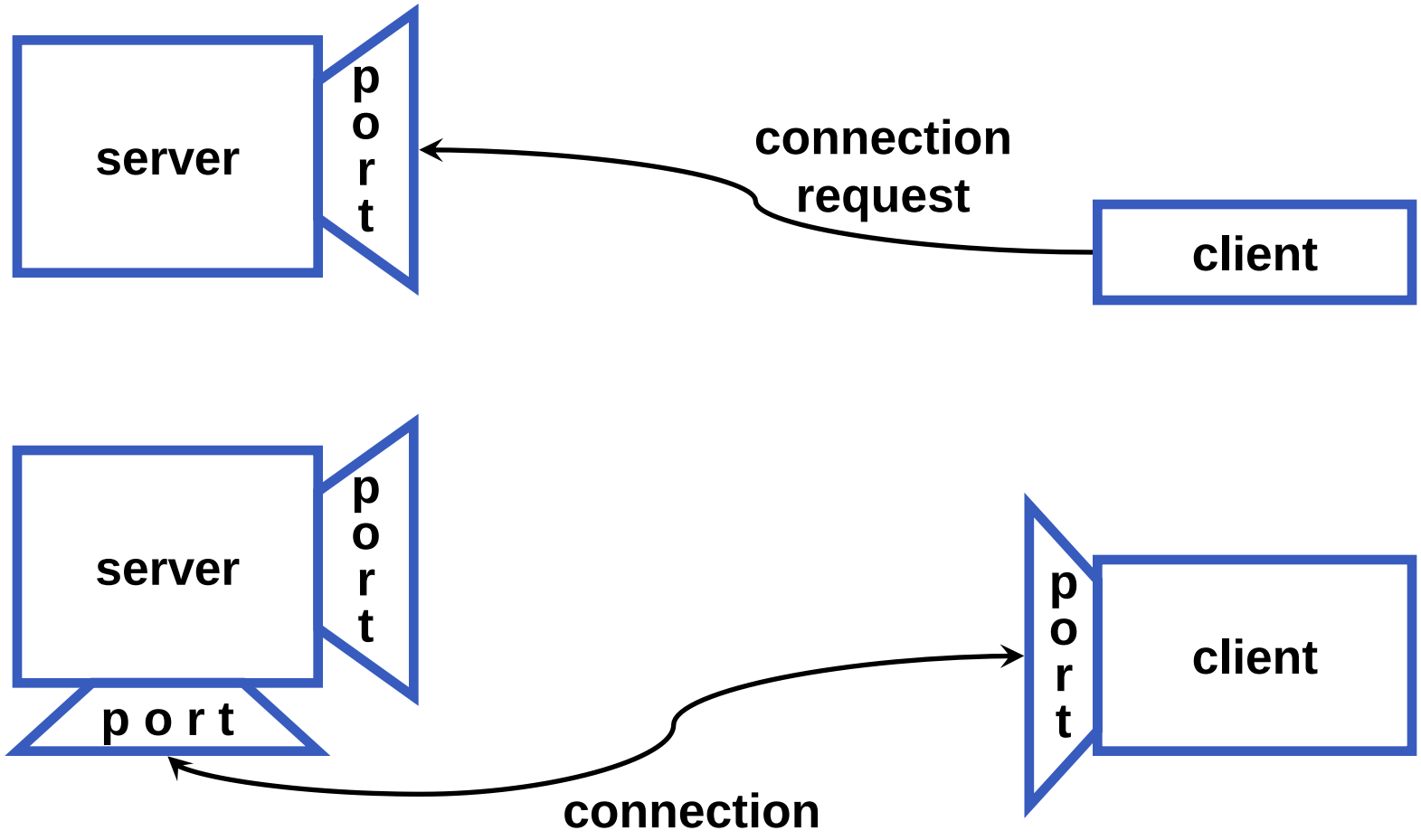
Абстракция сокета

- Сетевое соединение – это процесс передачи данных по сети между двумя компьютерами или процессами
- Сокет – конечный пункт передачи данных
- Для программ сокет – одно из окончаний сетевого соединения
- Для установления соединения каждая из сетевых программ должна иметь свой собственный сокет

Абстракция сокета

- Связь между двумя сокетами может быть ориентированной на соединение
- Связь между двумя сокетами может быть не ориентированной на соединение
- Сокет связан с номером порта

Абстракция сокета



А что же на Java?

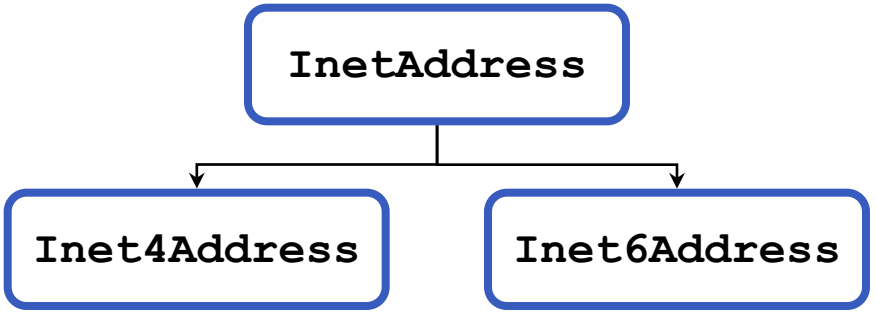
- Сокеты инкапсулированы в экземпляры специальных классов
- Все низкоуровневое взаимодействие скрыто от пользователя
- Существует семейство классов, обеспечивающих настройку сокетов и работу с ними

Пакет java.net

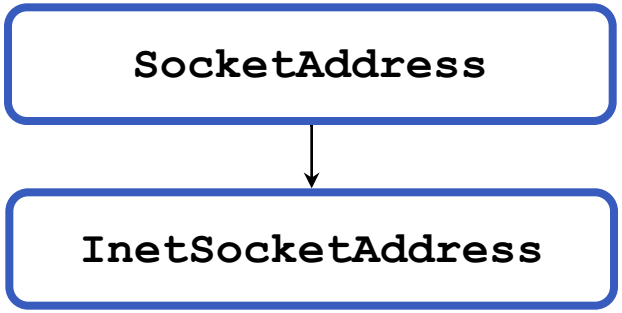
- Адресация
- Установление TCP-соединения
- Передача/прием дейтаграмм через UDP
- Обнаружение/идентификация сетевых ресурсов
- Безопасность: авторизация / права доступа

Адресация

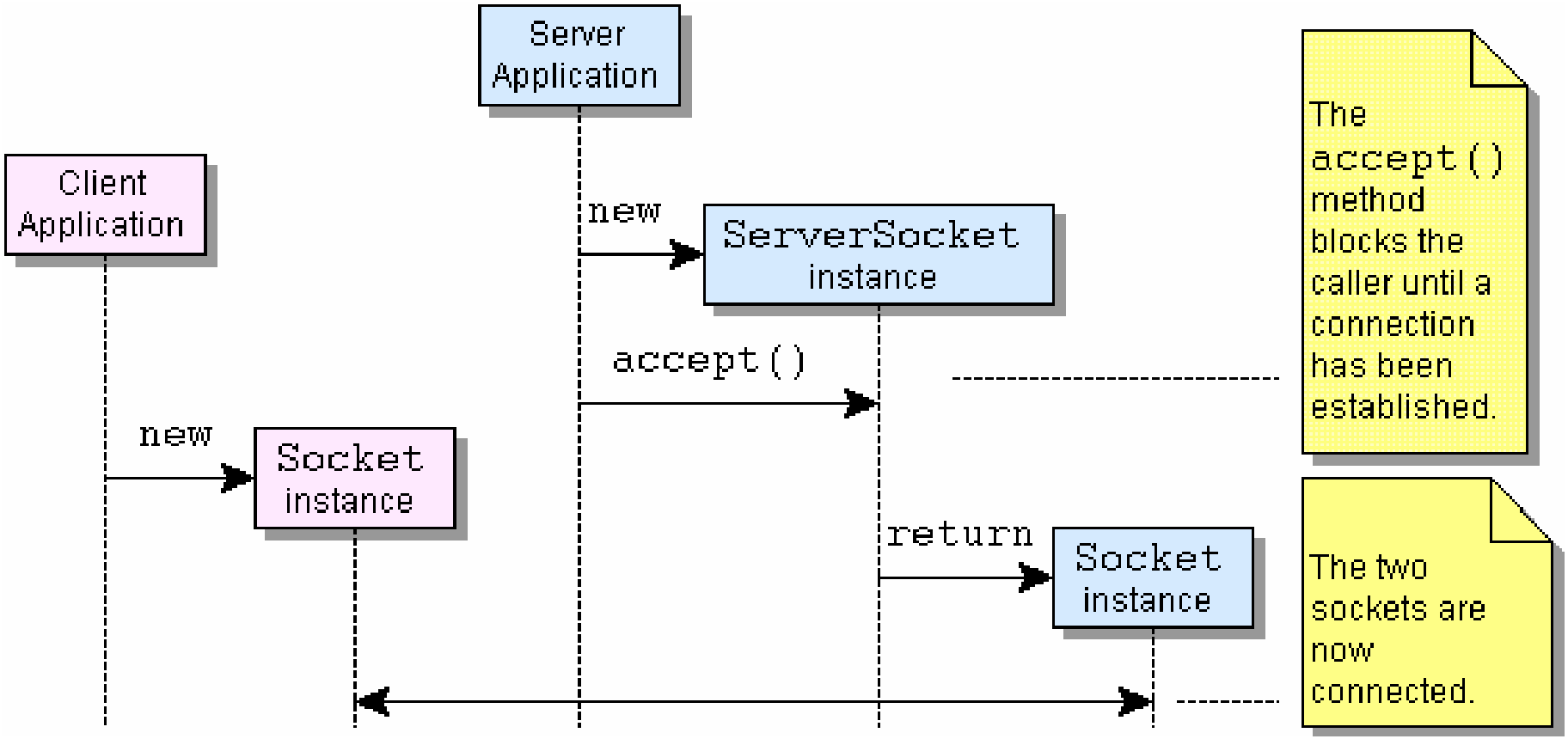
- IP-адресация



- Адрес сокета



Общая схема соединения



Класс Socket

- Реализует клиентский сокет и его функции
- Конструкторы
  - `Socket()`
  - `Socket(InetAddress address, int port)`
  - `Socket(InetAddress address, int port, InetAddress localAddr, int localPort)`
  - `Socket(String host, int port)`
  - `Socket(String host, int port, InetAddress localAddr, int localPort)`
- Методы
  - `void close()`
  - `InetAddress getLocalAddress()`
  - `InputStream getInputStream()`
  - `OutputStream getOutputStream()`
  - `static void setSocketImplFactory(SocketImplFactory fac)`
  - И прочие…

Порядок работы с клиентским сокетом

- Открытие сокета
- Открытие потока ввода и/или потока вывода для сокета
- Чтение и запись в потоки согласно установленному протоколу общения с сервером
- Закрытие потоков ввода-вывода
- Закрытие сокета

Пример клиента

```
import java.io.*;
import java.net.*;

public class EchoClient {
  public static void main(String[] args) throws IOException {
    Socket echoSocket = null;
    PrintWriter out = null;
    BufferedReader in = null;
    try {
      echoSocket = new Socket("taranis", 7);
      out = new PrintWriter(echoSocket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(
                                echoSocket.getInputStream()));
    } catch (UnknownHostException e) {
      System.err.println("Don't know about host: taranis.");
      System.exit(1);
    } catch (IOException e) {
      System.err.println("Couldn't get I/O for the connection to:" +
                         "taranis.");
      System.exit(1);
    }
```

Пример клиента

```
    BufferedReader stdIn = new BufferedReader(
                                new InputStreamReader(System.in));
    String userInput;

    while ((userInput = stdIn.readLine()) != null) {
      out.println(userInput);
      System.out.println("echo: " + in.readLine());
    }

    out.close();
    in.close();
    stdIn.close();
    echoSocket.close();
  }
}
```

Класс ServerSocket

- Реализует серверный сокет и его функции
- Конструкторы
  - `ServerSocket()`
  - `ServerSocket(int port)`
  - `ServerSocket(int port, int backlog)`
- Методы
  - `void close()`
  - `Socket accept()`
  - `void bind(SocketAddress endpoint)`
  - И прочие…

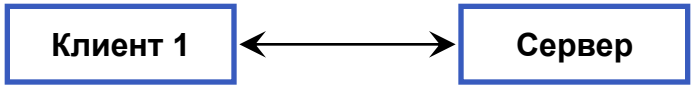
Создание серверного сокета

```
try {
        serverSocket = new ServerSocket(4444);
} catch (IOException e) {
        System.out.println(
                "Could not listen on port: 4444");
        System.exit(-1);
}
```

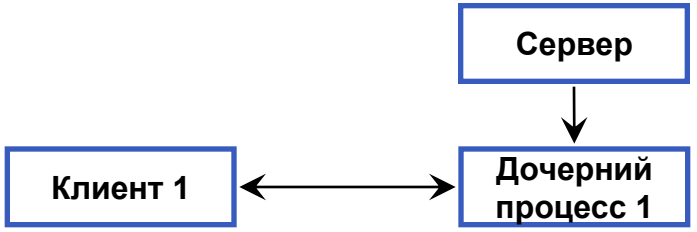
```
Socket clientSocket = null;
try {
        clientSocket = serverSocket.accept();
} catch (IOException e) {
        System.out.println("Accept failed: 4444");
        System.exit(-1);
}
```

Сервер параллельной обработки запросов

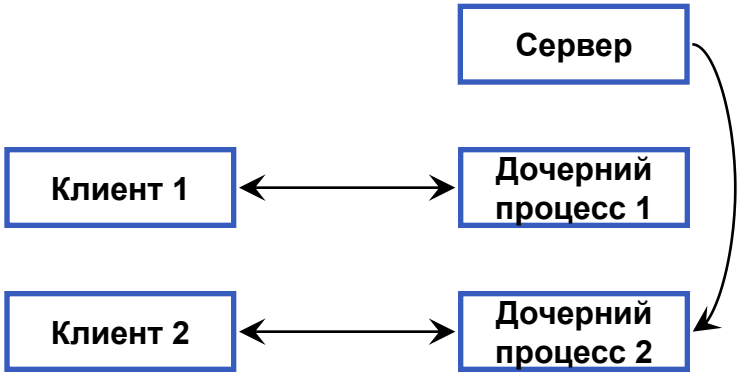
- **Стадия 1**
  Установление соединения клиент-сервер



- **Стадия 2**
  Сервер параллельной обработки передает управление дочернему процессу



- **Стадия 3**
  Если во время обработки запроса поступает запрос от другого клиента, сервер параллельной обработки передает управление новому дочернему процессу



Дейтаграммы

- **Дейтаграмма** – независимое, самодостаточное сообщение, посылаемое по сети, чья доставка, время (порядок) доставки и содержимое не гарантируются

- Могут использоваться как для адресной, так и для широковещательной рассылки

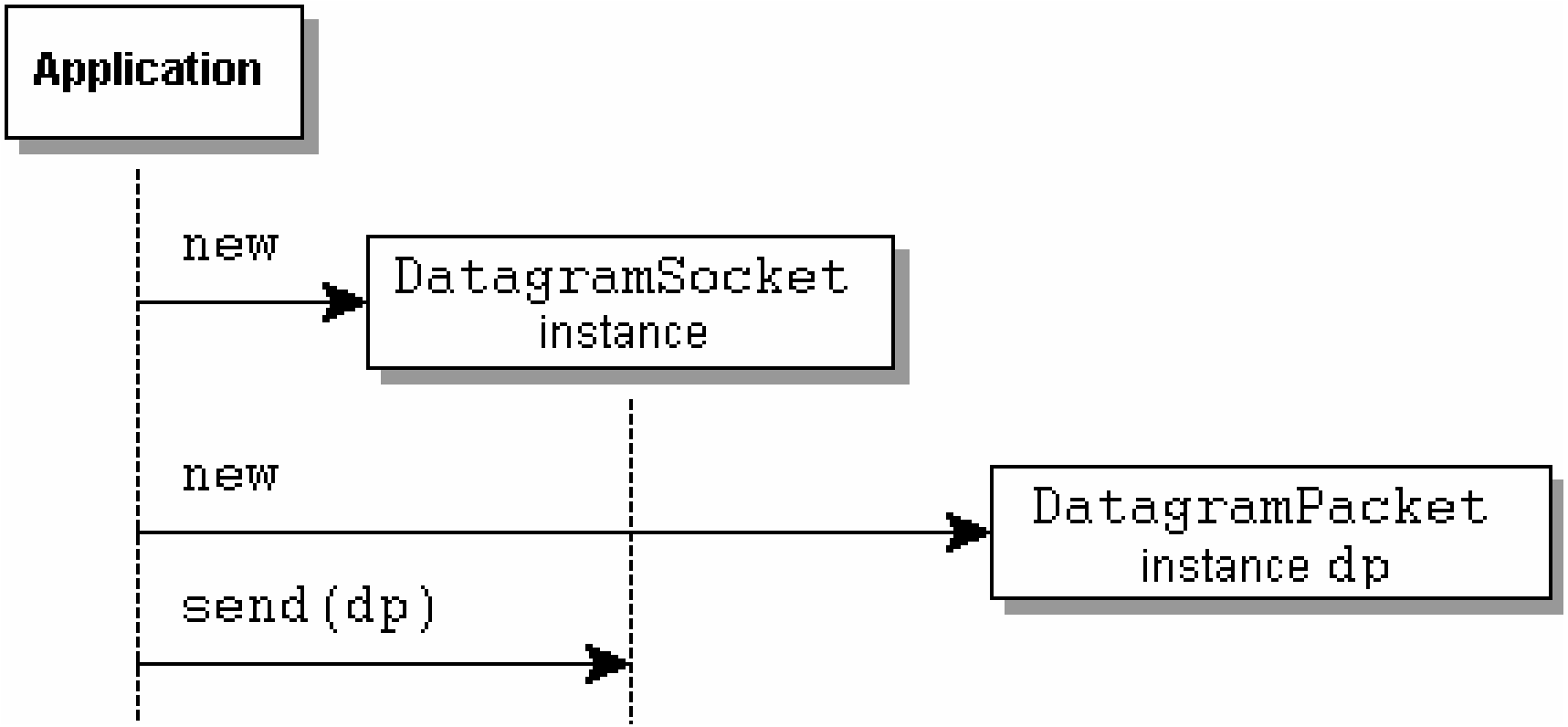
Класс DatagramPacket

- Экземпляры класса являются прототипами дейтаграмм-сообщений
- Конструкторы
  - `DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)`
  - И прочие…
- Методы
  - `byte[] getData()`
  - `int getLength()`
  - `int getOffset()`
  - `SocketAddress getSocketAddress()`
  - `void setSocketAddress(SocketAddress address)`
  - `void setData(byte[] buf, int offset, int length)`
  - И прочие…

Класс DatagramSocket

- Экземпляры являются не ориентированными на соединение сокетами
- Конструкторы
  - `DatagramSocket()`
  - `DatagramSocket(int port, InetAddress laddr)`
  - И другие…
- Методы
  - `void bind(SocketAddress addr)`
  - `void close()`
  - `void connect(InetAddress address, int port)`
  - `void send(DatagramPacket p)`
  - `void receive(DatagramPacket p)`
  - И другие…

Передача дейтаграмм



Uniform Resource Locator

- URL – адрес ресурса в Интернет
- Имя протокола
  Протокол, используемый для связи
- Имя хоста
  Имя компьютера, на котором расположен ресурс
- Имя файла
  Путь к файлу на компьютере
- Номер порта
  Номер порта для соединения (необязателен)
- Ссылка
  Ссылка на именованный якорь (необязательна)
- Может быть абсолютным и относительным

```
URL gamelan = new URL("http", "www.gamelan.com", 80,
                      "pages/Gamelan.network.html");
```

Прямое чтение из URL

```
import java.net.*;
import java.io.*;

public class URLReader {
  public static void main(String[] args) throws Exception {
    URL yahoo = new URL("http://www.yahoo.com/");
    BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                              yahoo.openStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null) {
      System.out.println(inputLine);
    }
    in.close();
  }
}
```

Чтение из URL-соединения

```
import java.net.*;
import java.io.*;
public class URLConnectionReader {
  public static void main(String[] args) throws Exception {
    URL yahoo = new URL("http://www.yahoo.com/");
    URLConnection yc = yahoo.openConnection();
    BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                              yc.getInputStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null) {
      System.out.println(inputLine);
    }
    in.close();
  }
}
```

Запись в URL-соединение

```
import java.io.*;
import java.net.*;
public class Reverse {
  public static void main(String[] args) throws Exception {
    if (args.length != 1) {
      System.err.println("Usage:  java Reverse" +
                         "string_to_reverse");
      System.exit(1);
    }
    String stringToReverse = URLEncoder.encode(args[0],
                                 "US-ASCII");
    URL url = new URL(
                "http://java.sun.com/cgi-bin/backwards");
```

Запись в URL-соединение

```
    URLConnection connection = url.openConnection();
    connection.setDoOutput(true);
    PrintWriter out = new PrintWriter(
                          connection.getOutputStream());
    out.println("string=" + stringToReverse);
    out.close();
    BufferedReader in = new BufferedReader(
                          new InputStreamReader(
                            connection.getInputStream()));
    String inputLine;
    while ((inputLine = in.readLine()) != null)
      System.out.println(inputLine);
    in.close();
  }
}
```

Спасибо за внимание!

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 992 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 14.10.2013.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Remote Method Invocation

Занятие 4

Гаврилов А.В.

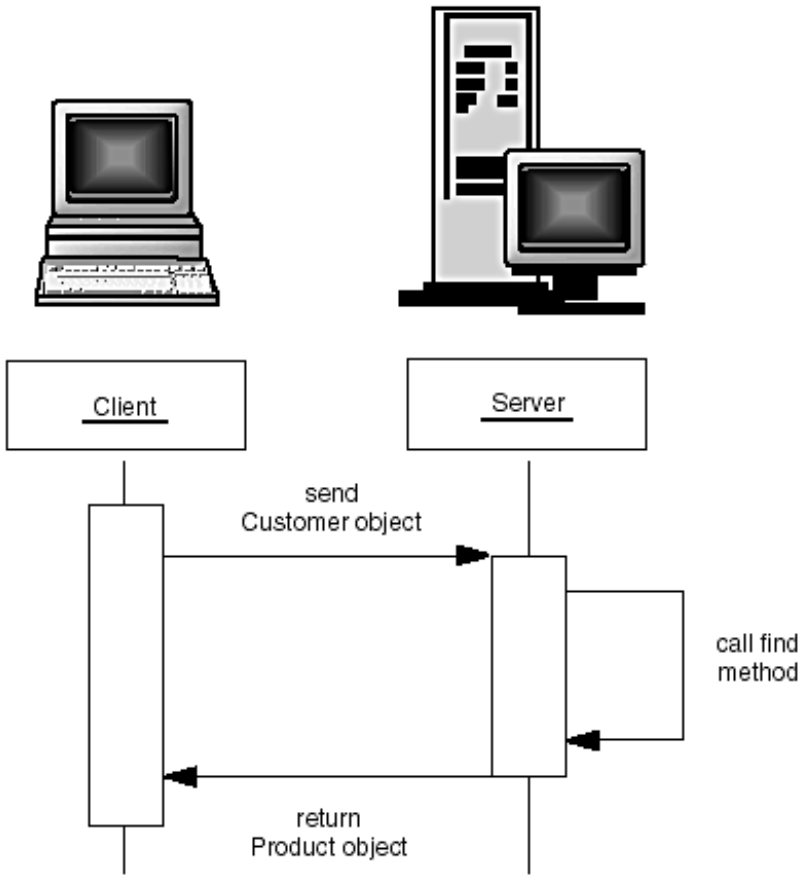
Самара
2013

План занятия

- Общие принципы RMI
- Элементы распределенной системы RMI
- Порядок разработки и запуска RMI-приложений
- Нововведения в Java5

Remote Method Invocation

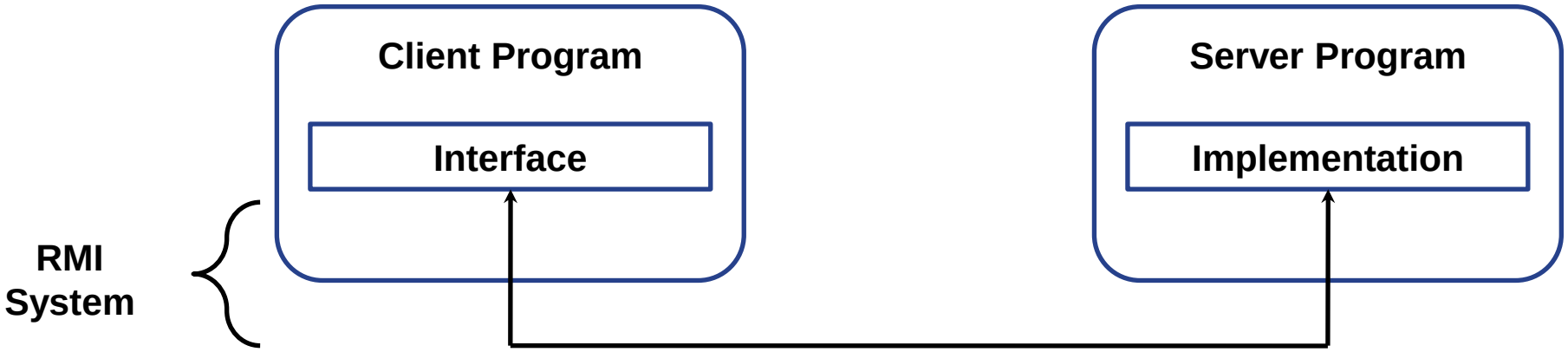
- Основной принцип: **определение** поведения и **реализация** этого поведения считаются разными понятиями
- RMI дает возможность разделить и выполнить на разных JVM код, **определяющий** поведение, и код, **реализующий** поведение



Remote Method Invocation

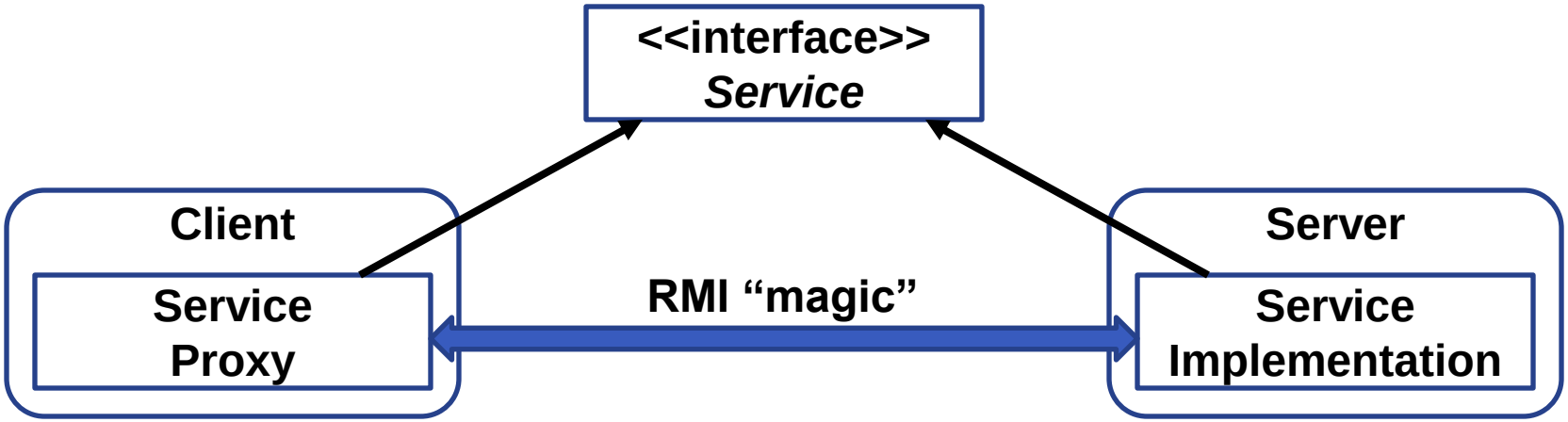
- В RMI удаленная служба определяется при помощи интерфейса Java
- Реализация удаленной службы кодируется в классе, реализующем интерфейс
- Ключ к пониманию RMI:
  - интерфейсы определяют поведение
  - классы определяют реализацию

RMI: принцип действия



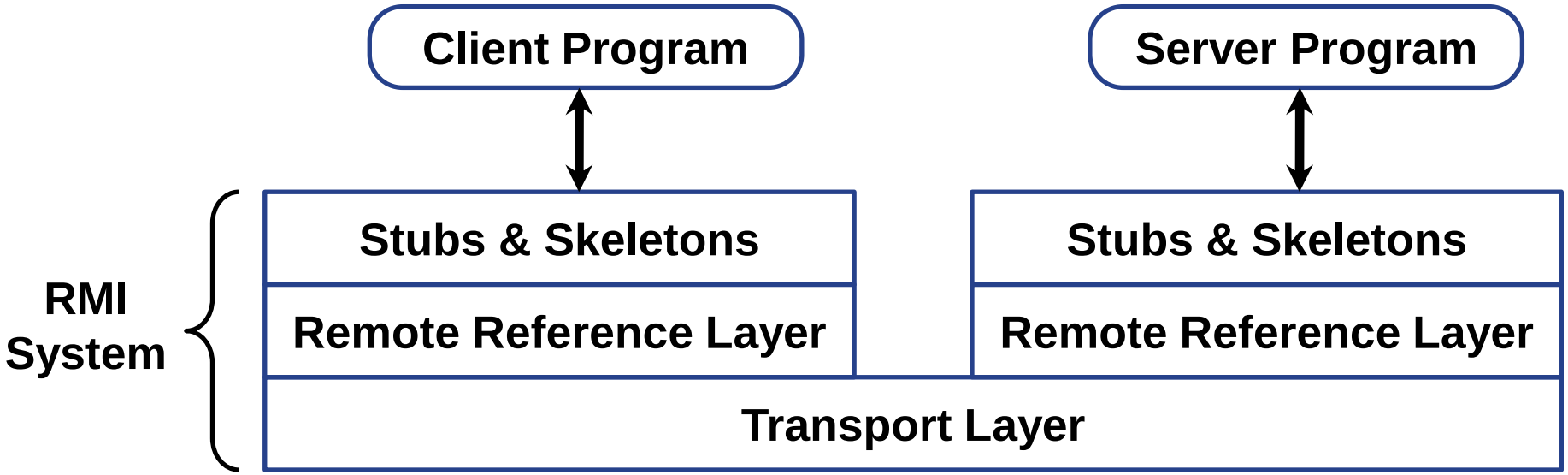
- Интерфейсы Java не содержат исполняемого кода
- RMI поддерживает два класса, реализующих один и тот же интерфейс:
  - первый класс является реализацией поведения и исполняется на сервере
  - второй класс работает как промежуточный интерфейс для удаленной службы и исполняется на клиентской машине

RMI: принцип действия



- Клиентская программа вызывает методы прокси-объекта, RMI передает запрос на удаленную JVM и направляет его в реализацию объекта
- Любые возвращаемые из реализации значения передаются назад в прокси-объект и затем в клиентскую программу

Уровни архитектуры RMI



- Уровень заглушки и скелета
- Уровень удаленной ссылки
- Транспортный уровень

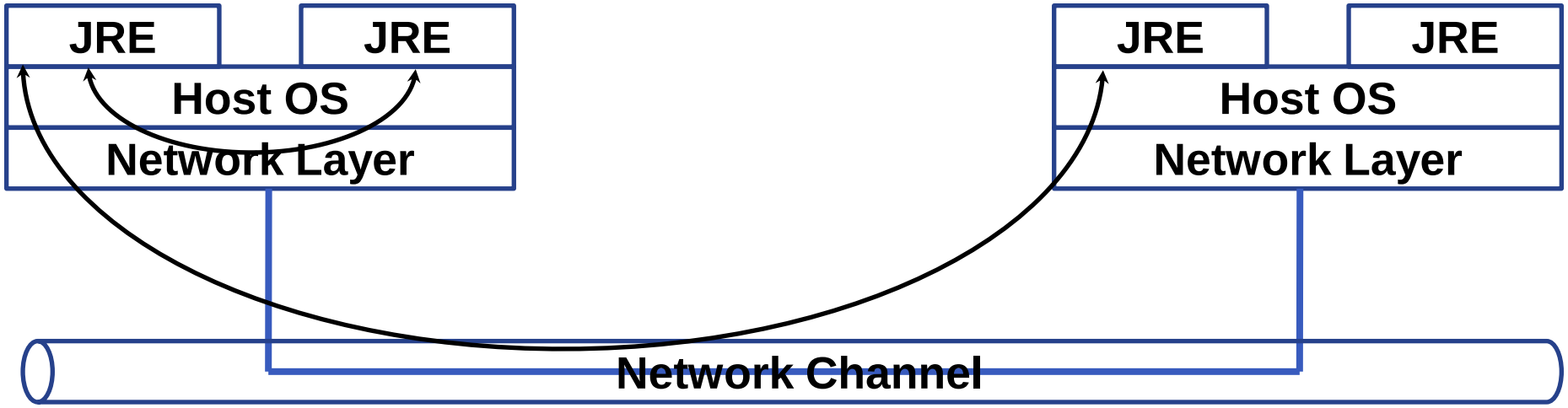
Уровень заглушки и скелета

- Непосредственно с ним взаимодействует разработчик
- Перехватывает вызовы методов, произведенные клиентом при помощи ссылки типа интерфейса, и переадресует их в удаленную службу RMI
- Основан на паттерне проектирования Proxy (Заместитель)

Уровень удаленной ссылки

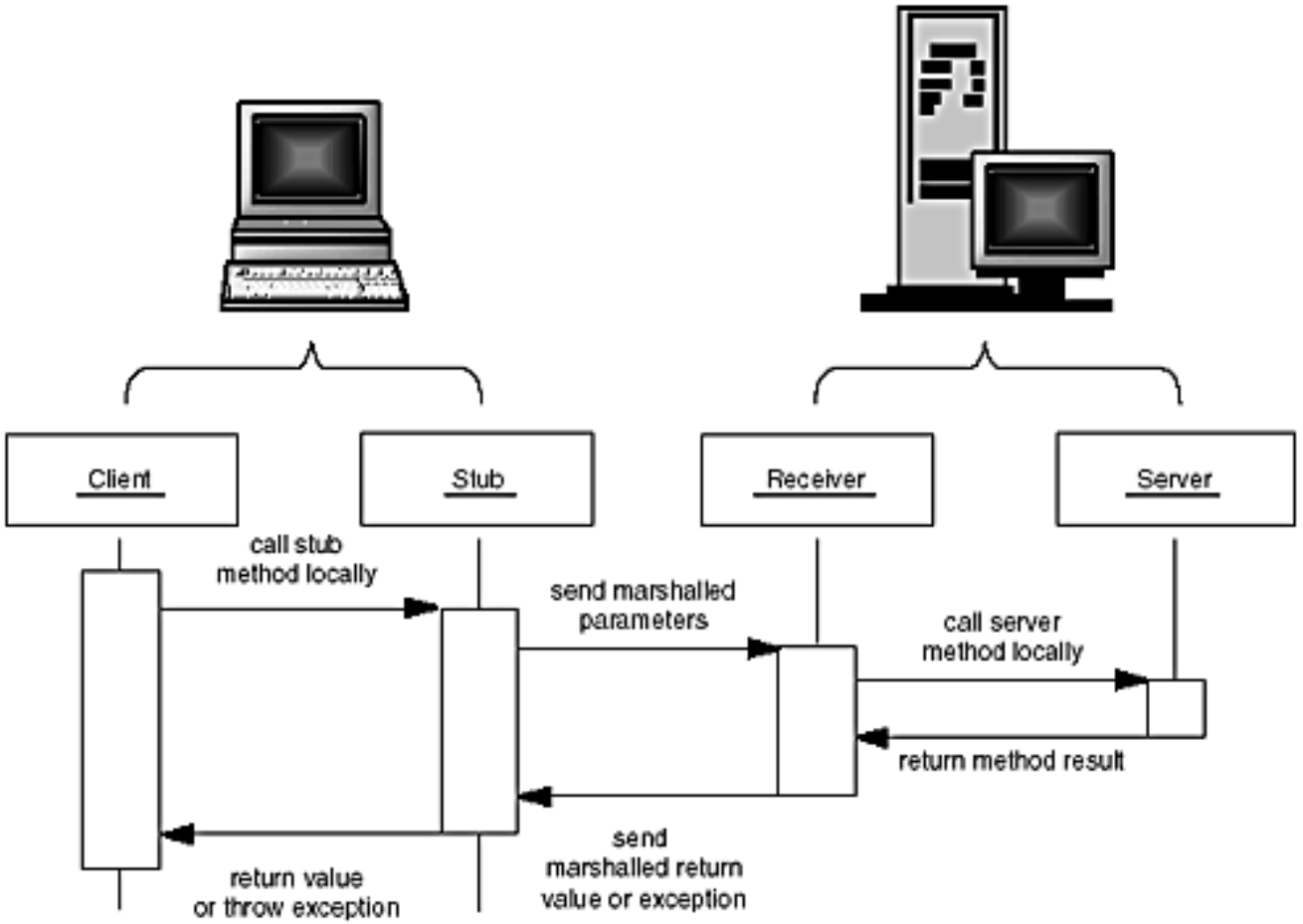
- Удаленная ссылка (remote reference)
  - Может включать в себя адрес компьютера, адрес приложения и адрес собственно объекта
  - Ссылка на удаленный объект должна быть получена в начале работы с этим объектом с помощью **службы именования**
- Этот уровень понимает, как интерпретировать и управлять ссылками на удаленные объекты

Транспортный уровень



- Основан на соединениях TCP/IP между сетевыми машинами
- Обеспечивает основные возможности соединения и некоторые стратегии защиты от несанкционированного доступа
- Поддерживаются протоколы RMI-JRMP и RMI-IIOP

Вызов удаленного метода



Действия при вызове удаленного метода

Заглушка

- Высылает серверу пакет с идентификатором удаленного объекта, описанием вызываемого метода и упакованными параметрами
- Получает пакет от сервера, распаковывает результат

Получатель

- Разбирает параметры (unmarshaling)
- Находит объект
- Вызывает нужный метод
- Получает и упаковывает результат (marshaling)
- Отсылает пакет заглушке

Передача параметров

Аргументы методов и возвращаемое значение могут быть следующих типов:

- **Простые типы**
- **Объектные типы**
- **Удаленные объектные типы**

Параметры простых типов

- Когда в качестве параметра в удаленный метод передается простой тип данных, RMI передает их по значению
- RMI делает копию значения простого типа и передает ее в удаленный метод
- Если метод возвращает значение простого типа, также используется передача по значению
- Значения передаются между JVM в стандартном, машинно-независимом формате; это позволяет JVM, работающим на разных платформах, надежно взаимодействовать друг с другом

Параметры объектных типов

- RMI передает между JVM сам объект, а не ссылку на него, т.е. объект передается по значению
- Когда удаленный метод возвращает объект, в вызывающую программу передается копия объекта
- Для передачи состояния объекта RMI использует сериализацию: состояние объекта преобразуется в набор байтов, пересылаемых по сети

Параметры удаленных объектных типов

- При передаче в качестве параметра или возвращаемого значения ссылки на заглушку удаленного объекта сериализация не используется
- Вместо этого передается удаленная ссылка
- Получатель получает для работы локальную ссылку на заглушку удаленного объекта
- Это еще один способ получить ссылку на удаленный объект

Синтаксис вызова

- Синтаксис вызова такой же, как и при локальном вызове

```
centralWarehouse.getQuantity("SuperSucker 100 Vacuum Cleaner");
```

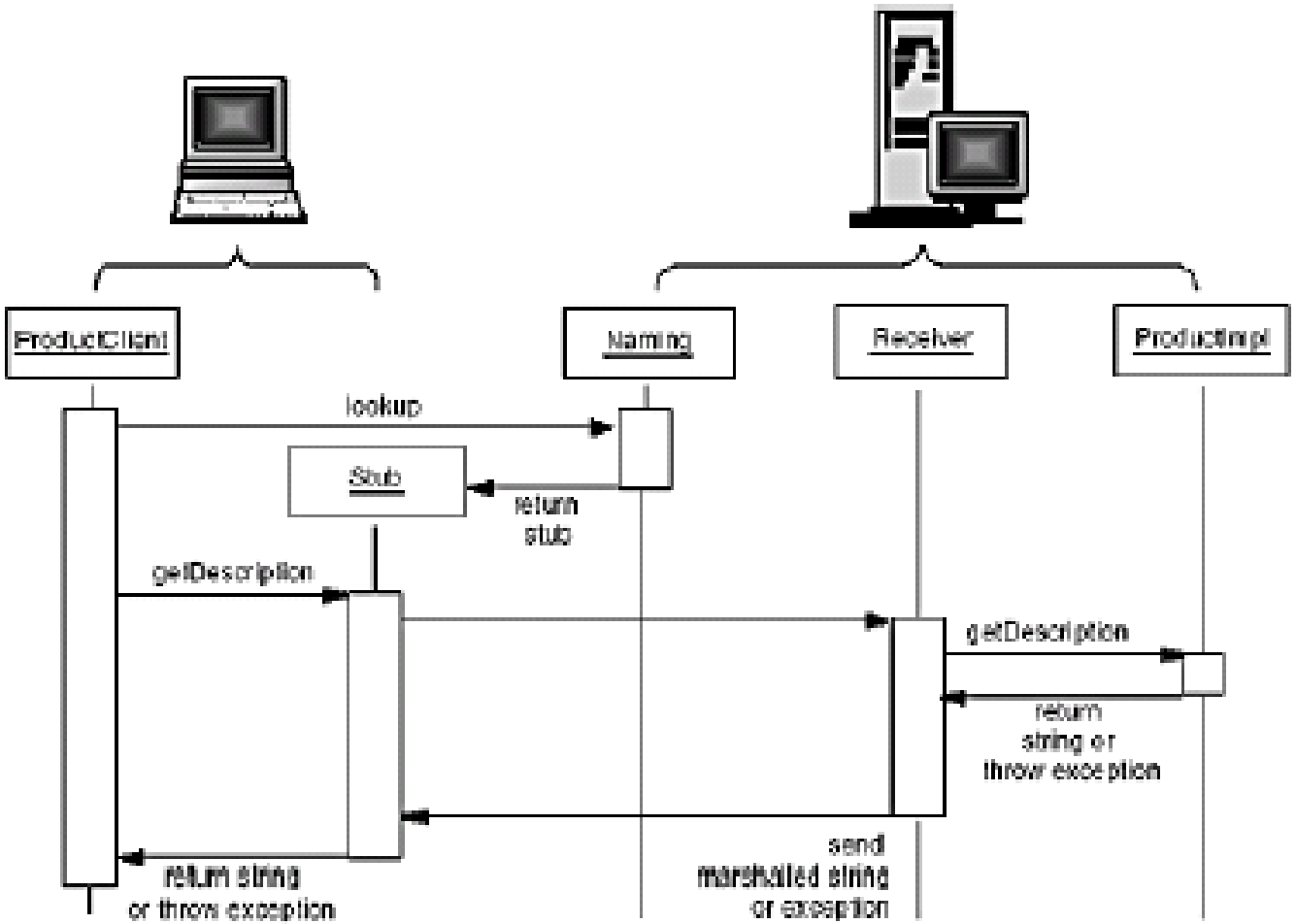
- Используются ссылки интерфейсных типов

```
interface Warehouse extends Remote {
  int getQuantity(String description) throws RemoteException;
  ...
}
```

Динамическая загрузка классов

- Класс заглушки должен быть доступен клиенту
- RMI-клиенты могут сами динамически загружать классы заглушек
- Также могут быть загружены дополнительные классы, необходимые для передачи параметров
- Для обеспечения корректности применяется менеджер безопасности (security manager)

Пример работы



Именование удаленных объектов

- Как клиент находит удаленный объект RMI?
- **Клиенты находят удаленные объекты, используя службу имен или каталогов**
- RMI может использовать различные службы, включая Java Naming and Directory Interface (JNDI)
- RMI включает в себя простую службу – реестр RMI (rmiregistry)
- Реестр RMI работает на каждой машине, содержащей объекты удаленных служб и принимающей запросы на обслуживание (по умолчанию используется порт 1099)

На стороне сервера

- Программа сервера создает удаленный объект, создавая локальный объект, реализующий нужную функциональность
- Затем программа экспортирует этот объект в RMI
- Как только объект экспортирован, RMI создает службу прослушивания, ожидающую соединения с клиентом и запроса к объекту
- После экспорта сервер регистрирует объект в реестре RMI, используя публичное имя

На стороне клиента

- Доступ к реестру RMI обеспечивается через статический класс **`Naming`**
- Он предоставляет метод **`lookup()`**, который клиент использует для запросов к реестру
- Метод принимает URL, указывающий на имя хоста и имя требуемой службы
- URL принимает следующий вид: rmi://<host_name> [:<name_service_port>] /<service_name>
- Метод возвращает удаленную ссылку на объект

Основные элементы распределенной RMI-системы

- Интерфейс удаленного объекта
- Класс, реализующий удаленный объект
- Файлы классов stub'a и skeleton'a.
- Программа серверной части
- Служба именования RMI
- Провайдер файлов классов (HTTP- или FTP-сервер)
- Программа-клиент

Соглашения именования классов

Без суффикса `Product`	Удаленный интерфейс
Суффикс Impl `ProductImpl`	Серверный класс, реализующий интерфейс
Суффикс Server `ProductServer`	Программа на сервере, создающая и регистрирующая объекты
Суффикс Client `ProductClient`	Клиентская программа, вызывающая удаленные методы
Суффикс _Stub `ProductImpl_Stub`	Класс заглушки, автоматически генерируется программой rmic
Суффикс _Skel `ProductImpl_Skel`	Класс скелета, автоматически генерируется программой rmic; нужен для RMI 1.1

Порядок разработки серверной части

- Определение интерфейса удаленного объекта
- Написание класса, реализующего этот интерфейс
- Создание программы серверной части, которая реально создает объект и регистрирует его
- Запуск специального компилятора (`rmic`), автоматически создающего код для заглушки

Описание интерфейса Product

```
import java.rmi.*;

/**
 * The interface for remote product objects.
 */
 public interface Product extends Remote
 {
  /**
   * Gets the description of this product.
   * @return the product description
   */
  String getDescription() throws RemoteException;
}
```

Реализация интерфейса ProductImpl

```
import java.rmi.*;
import java.rmi.server.*;

public class ProductImpl extends UnicastRemoteObject
                         implements Product {
  public ProductImpl(String n) throws RemoteException {
    name = n;
  }

  public String getDescription() throws RemoteException{
    return "I am a " + name + ". Buy me!";
  }

  private String name;
}
```

Сервер ProductServer

```
import java.rmi.*;
import java.rmi.server.*;

public class ProductServer {
  public static void main(String args[]) {
    try {
      System.out.println("Constructing server implementations...");
      ProductImpl p1 = new ProductImpl("Blackwell Toaster");
      ProductImpl p2 = new ProductImpl("ZapXpress Microwave Oven");
      System.out.println("Binding server implementations to registry...");
      Naming.rebind("toaster", p1);
      Naming.rebind("microwave", p2);
      System.out.println ("Waiting for invocations from clients...");
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Порядок работы клиентской части

- Запуск менеджера безопасности (Security Manager)
- Поиск удаленного объекта
- Вызов какого-либо метода удаленного объекта

Клиент ProductClient

```
import java.rmi.*;
import java.rmi.server.*;

public class ProductClient {
  public static void main(String[] args) {
    System.setProperty("java.security.policy", "client.policy");
    System.setSecurityManager(new RMISecurityManager());
    String url = "rmi://localhost/";
    // change to "rmi://yourserver.com/"
    try {
      Product c1 = (Product)Naming.lookup(url + "toaster");
      Product c2 = (Product)Naming.lookup(url + "microwave");
      System.out.println(c1.getDescription());
      System.out.println(c2.getDescription());
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
}
```

Файл политики безопасности

- Определяет права на доступ к различным ресурсам
- Используется менеджером безопасности
- Необходимо любому загружаемому коду с любого места разрешить:
  - Соединяться или принимать соединения по непривилегированным портам (> 1024) с любого хоста
  - Подключаться к порту 80 (HTTP-порт)

```
grant
{
  permission java.net.SocketPermission
  "*:1024-65535", "connect";
  permission java.net.SocketPermission
  "*:80", "connect";
};
```

Разделение кода для распределенного приложения

Server

- Папка, где располагается сервер
- Не должна быть доступна клиенту
- Должна содержать, как минимум, следующие файлы:
  - `ProductServer.class`
  - `ProductImpl.class`
  - `Product.class`
  - `ProductImpl_Stub.class`

Разделение кода для распределенного приложения

Client

- Папка, где располагается клиент
- Должна содержать, как минимум, следующие файлы:
  - `ProductClient.class`
  - `client.policy`
- Если интерфейс удаленного объекта известен заранее, также должна содержать файл:
  - `Product.class`

Разделение кода для распределенного приложения

Download

- Содержит классы, используемые с данного сервера
- Классы из нее могут быть загружены клиентом динамически
- Указывается как значение переменной `java.rmi.server.codebase`
- Должна содержать, как минимум, следующие файлы:
  - `Product.class`
  - `ProductImpl_Stub.class`

Запуск серверной части

- Запуск программы RMIRegistry

```
UNIX: rmiregistry &

Windows: start rmiregistry
```

- Запуск программы сервера удаленного объекта

```
UNIX: java -Djava.rmi.server.codebase=file:classDir/
        ProductServer &

Windows: start java -Djava.rmi.server.codebase=file:classDir/
           ProductServer
```

Запуск клиентской части

- Файл политики безопасности должен быть доступен менеджеру безопасности

- Запуск производится как запуск обычного приложения Java

```
java ProductClient
```

Нововведения Java5

- Стала необязательной компиляция заглушек с помощью rmic
- Расширились возможности службы именования
- Немного изменился подход к регистрации объекта на сервере
- Общие принципы и порядки разработки и работы приложений сохранились

Интерфейс и реализация в стиле Java5

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
  String sayHello() throws RemoteException;
}
```

```
public class HelloImpl implements Hello {
  public HelloImpl() {}

  public String sayHello() {
    return "Hello, world!";
  }
}
```

Сервер в стиле Java5

```
import java.rmi.registry.*;
import java.rmi.server.*;

public class HelloServer {
  public static void main(String args[]) {
    try {
      HelloImpl obj = new HelloImpl();
      Hello stub = (Hello) UnicastRemoteObject.exportObject(obj, 0);
      Registry registry = LocateRegistry.getRegistry();
      registry.bind("Hello", stub);
      System.err.println("Server ready");
    }
    catch (Exception e) {
      System.err.println("Server exception: " + e.toString());
    }
  }
}
```

Спасибо за внимание!

Дополнительные источники

- Хорстманн, К.С. Java2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010. – 816 с.
- Grosso, W. Java RMI [Текст] / William Grosso. – O'Reilly, 2001. – 572 с.
- Harold, E.R. Java Network Programming [Текст] / Elliotte Rusty. – O'Reilly, 2004. – 504 с.
- Remote Method Invocation Home [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, дата доступа: 14.10.2013.
- Trial: RMI [Электронный ресурс]. – Режим доступа: http://download.oracle.com/javase/tutorial/rmi/index.html, дата доступа: 14.10.2013.
- jGuru: Remote Method Invocation (RMI) [Электронный ресурс]. – Режим доступа: http://www.dse.disco.unimib.it/ds/extra/rmiTutorial.pdf, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

HyperText Markup Language Cascading Style Sheets

Занятие 5

Гаврилов А.В.

Самара
2013

План занятия

- Общие сведения об HTML
- Структура документа, основные теги
- Взаимодействие с сервером
- Основные принципы CSS

Давным-давно в одной далёкой галактике…

- Generalized Markup Language (GML)
  - Charles Goldfarb, Edward Mosher ,Raymond Lorie (IBM, 1969)
  - В тексте размещаются т.н. теги, которые определяют структуру текста в терминах абзацев, заголовков, списков и т.д.
  - Форматирование документа может производиться автоматически для конкретного устройства с применением конкретного профиля

```
:h1.Chapter 1: Introduction
:p.GML supported hierarchical containers, such as
:ol
:li.Ordered lists (like this one),
:li.Unordered lists, and
:li.Definition lists
:eol.
as well as simple structures.
:p.Markup minimization (later generalized and formalized in SGML), allowed the
end-tags to be omitted for the "h1" and "p" elements.
```

SGML

- Standard Generalized Markup Language, 1970
  - Разметка носит декларативный характер
  - Разметка должна регламентироваться строгими правилами
- Document Type Definition

```
<anthology>
  <poem><title>The SICK ROSE</title>
    <stanza>
      <line>O Rose thou art sick.</line>
      <line>The invisible worm,</line>
      <line>That flies in the night</line>
      <line>In the howling storm:</line>
    </stanza>
    <stanza>
      <line>Has found out thy bed</line>
      <line>Of crimson joy:</line>
      <line>And his dark secret love</line>
      <line>Does thy life destroy.</line>
    </stanza>
  </poem>

  <!-- more poems go here -->

</anthology>
```

HTML

- HyperText Markup Language –
  язык разметки для web-документов
- История
  - HTML – Tim Berners-Lee, 1991
  - HTML 2.0 – IETF,1995 г.
  - HTML 3.0 – W3C, 1995 г.
  - HTML 3.2 – W3C, 1997 г.
  - HTML 4.0 – W3C, 1997 г.
  - HTML 4.1 – W3C, 1999 г.
  - HTML 5 – WHATWG, W3C
  - XHTML 1.0 – W3C, 1998 г.
  - XHTML 1.1 – W3C, 2000 г.
  - XHTML 2.0 – W3C,
    закрыт в 2009 г.
  - XHTML5 – ???

HTML-файлы

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>
      This is a paragraph
      with a line break, isn't it?
    </p>
    <p>No, it isn't. This is a paragraph <br/> with a line break.</p>
    <a href="http://www.w3.org/standards/webdesign/htmlcss">Read more</a>
    <br/>
    <img src="http://www.w3.org/html/logo/img/html5-display.png"/>
  </body>
</html>
```

Объявление типа документа

Заголовок

Тело

Открывающий тег

Текст

Закрывающий тег

Ссылка

Тег без наполнения

Атрибут

Результат

Hello World!

This is a paragraph with a line break, isn't it?

No, it isn't. This is a paragraph
with a line break.

Read more

Теги

- Открывающий тег (<tagname>)
- Закрывающий тег (</tagname>)
  - Есть теги, традиционно не требующие закрытия (<p>, <br>, <img>, …)
  - Лучше их всё равно явно закрывать
  - Для тегов без наполнения лучше использовать сокращённую форму (<hr/>)
- Теги вкладываются друг в друга
  - Желательно образуя древовидную структуру
- HTML не чувствителен к регистру тегов
  - Рекомендуется писать в нижнем регистре

Атрибуты тегов

- Атрибут добавляет тегу информацию, не включая её во внутренности тега
  `<tagname attrname="attrvalue">`
- Атрибуты бывают обязательными и необязательными
- Значения атрибутов можно не заключать в кавычки
  - Лучше всё-таки заключать
- В ряде случаев у значений атрибутов можно не указывать единицы измерения
  - Лучше всё-таки указывать
- Существуют атрибуты-маркеры, не требующие значения
- Существуют атрибуты событий, обычно используются со скриптами

Текст

- Выводимый текст располагается в теле документа внутри тегов
- Форматирование текста в документе игнорируется
  - Если он только не заключён в специальный тег, например <pre>
- Реальное форматирование определяется тегами
- Текст формируется в блоки
  - <p> – абзац
  - – текстовый блок
  - <pre> – текстовый блок с предварительным форматированием
  - <h1>, …, <h6> – заголовки
  - , <ul>, <dl> – списки
  - <table> – таблицы
  - <form> – форма для обмена данными
  - …

Символьные мнемоники

Числовая	Символьная	Символ	Описание
&#034;	&quot;	"	Кавычка
&#038;	&amp;	&	Амперсанд
&#060;	&lt;	<	Знак меньше
&#062;	&gt;	>	Знак больше
			Неразрывный пробел
&#162;	&cent;	¢	Знак цента
&#169;	&copy;	©	Знак копирайта
&#171;	&laquo;	«	Левая угловая кавычка
&#174;	&reg;	®	Зарегистрированная товарная марка
&#177;	&plusmn;	±	Плюс-минус
&#187;	&raquo;	»	Правая угловая кавычка

Объявления типа документа

- **Строгий (Strict)**
  Не содержит устаревших и нерекомендованных тегов
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

- **Переходный (Transitional)**
  Поддерживает старые теги в целях совместимости
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

- **С фреймами (Frameset)**
  Переходный с добавлением тегов для фреймов
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
```

Заголовок документа
<title>, <base>

- Заголовок определяется тегом **<head>**, остальные теги располагаются внутри
- **<title>**
  - Позволяет указать заголовок документа, отображающийся в браузере
  - `<title>Заголовок</title>`
- **<base>**
  - Указывает базовый адрес для относительных адресов и объект по умолчанию для открытия страниц
  - `<base href="http://www.site.ru/folder/"/>`
  - `<base target="_blank"/>`

Заголовок документа
<script>, <link>

- `<script>`
  - Описывает скрипт в документе или ссылку на внешний скрипт
  - Может быть не только в заголовке
  - В заголовке обычно размещают скрипты, которые должны выполняться в первую очередь
  - `<script type="text/javascript">…</script>`
  - `<script src="scriptfile.js"></script>`

- `<link>`
  - Устанавливает связь со внешним документом
  - `<link rel="stylesheet" type="text/css" href="st.css"/>`

Заголовок документа
<meta>

- `<meta>`
  - Задаёт различного рода дополнительную информацию о документе
  - Атрибуты
    - `name` – имя метатега
    - `http-equiv` – конвертирует данные в заголовок HTTP
    - `content` – значение атрибута
    - `charset` – кодировка документа
  - `<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>`
  - `<meta http-equiv="Cache-Control" content="no-cache"/>`
  - `<meta http-equiv="Refresh" content="1; url=new.htm"/>`
  - `<meta name="description" http-equiv="description" content="Курс технологий сетевого программирования, лекция по HTML и CSS."/>`
  - `<meta name="keywords" http-equiv="keywords" content="HTML; CSS; JavaEE"/>`

Тело документа

- Тело документа располагается внутри тега `<body>`
- Абзацы располагаются внутри тега `<p>`
- Заголовки располагаются внутри тегов `<h1>`, …, `<h6>`
- Если необходимо использовать исходное форматирование, применяется тег `<pre>`
- Принудительный перевод каретки делается тегом `<br/>`
- Горизонтальное отчёркивание делается тегом `<hr/>`
- Комментарии располагаются внутри тега `<!-- … -->`

Управление отображением символов

Теги, управляющие формой отображения

Тег	Значение
<i>	Курсив
<b>	Полужирный
<tt>	Телетайп
<u>	Подчёркивание
<s>	Перечёркивание
<big>	Увеличенный шрифт
<small>	Уменьшенный шрифт
<sub>	Подстрочные символы
<sup>	Надстрочные символы

Теги, характеризующие тип информации

Тег	Значение
	Смысловое выделение
<cite>	Цитирование
	Смысловое усиление
<code>	Код программы
<samp>	Вывод программы
<kdb>	Ввод с клавиатуры
<var>	Переменная
<dfn>	Определение
<q>	Короткая цитата

Использование текстовых тегов

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Hello HTML</title>
 </head>
 <body>
   <!-- Демонстрация тегов -->
   <h1>Управление отображением</h1>
   <hr/>
   <h2>Форма отображения</h2>
   <p>
     <i>italic</i> <b>bold</b> <tt>teletype</tt> <u>underline</u> <s>strike</s> <br/>
     <big>big</big> <small>small</small> <sup>sup</sup> <sub>sub</sub>
   </p>
   <hr/>
   <h2>Тип информации</h2>
   <p>
     <em>emphasis</em> <cite>cite</cite> <strong>strong</strong> <code>code</code> <br/>
     <samp>samp</samp> <kbd>kbd</kbd> <var>var</var> <dfn>dfn</dfn> <q>quote</q>
   </p>
 </body>
</html>
```

Результат

Управление отображением

Форма отображения

italic **bold** `teletype` underline ~~strike~~
big small ^{sup} $_{sub}$

Тип информации

emphasis *cite* **strong** `code`
`samp` `kbd` *var* dfn "quote"

Оформление списков

- Неупорядоченные списки `<ul>`
- Упорядоченные списки `<ol>`
- Заголовок списка `<lh>`
- Элемент списка `<li>`
- Список определений `<dl>`
  - Термин `<dt>`
  - Определение `<dd>`
- Списки могут быть вложенными

```
<ol type="I">
  <lh>List examples</lh>
  <li>Ordered list
    <ol>
      <li>One</li>
      <li>Two</li>
    </ol>
  </li>
  <li>Unordered list
    <ul type="square">
      <li>Something</li>
      <li>More</li>
    </ul>
  </li>
  <li>Definition list
    <dl>
      <dt>HTML</dt>
      <dd>HyperText Markup Language</dd>
      <dt>CSS</dt>
      <dd>Cascading Style Sheets</dd>
    </dl>
  </li>
<ol>
```

Результат

List examples

I. Ordered list
    1. One
    2. Two

II. Unordered list
    - Something
    - More

III. Definition list

HTML
: HyperText Markup Language

CSS
: Cascading Style Sheets

Ссылки

- `<a>`
  - Ссылки на внешние ресурсы
    - `<a href="URL">reference object</a>`
  - Определение внутренних якорей
    - `<a name="AncorName">object</a>`
  - Ссылки на внутренние якоря
    - `<a href="#AncorName">reference object</a>`
  - Атрибут target определяет место, в котором откроется ссылка:
    - `_blank` – новое окно (вкладка)
    - `_self` – текущее окно (фрейм)
    - `_parent` – родительский фрейм
    - `_top` – окно браузера (поверх всех фреймов)
    - имя фрейма – указанный по имени фрейм
- Ссылки могут быть абсолютными, по полному URL
- Ссылки могут быть относительными
  - Отсчитываются от текущего адреса
  - Отсчитываются от адреса, заданного тегом `<base>`

Использование ссылок

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>References</title>
 </head>
 <body>
   <a name="top"><h1>Демонстрация ссылок</h1></a>
   <hr/>
   <p><a href="local.html">Ссылка</a> на страницу, расположенную в этом же
   каталоге.</p>
   <p><a href="http://ru.wikipedia.org/" target=_blank>Ссылка</a> на внешний
   ресурс.</p>
   <p><a href="http://www.docbook.org/tdg51/en/html/ch00.html#pref-
   whyread">Ссылка</a> на внешний ресурс с указанием якоря.</p>
   <p><a href="#top">Ссылка</a> на внутренний якорь.</p>
 </body>
</html>
```

Результат

Демонстрация ссылок

Ссылка на страницу, расположенную в этом же каталоге.

Ссылка на внешний ресурс.

Ссылка на внешний ресурс с указанием якоря.

Ссылка на внутренний якорь.

Вывод изображений

- `<img>`
  - Обязательный атрибут `src` задаёт URL изображения
  - Атрибут `alt` задаёт текст, выводимый вместо изображения, если его нельзя отобразить
  - Атрибуты `width` и `height` задают ширину и высоту области, куда будет вставлено изображение
  - Атрибут `usemap` вместе с дополнительными тегами `<map>` и `<area>` позволяет задать на изображении области разной формы, являющиеся ссылками
  - Атрибут `ismap` делает изображение ссылкой, при нажатии на которую на сервер будут отправлены координаты нажатия
- Изображения могут быть ссылками
- Изображения могут быть вставлены в текст
- Параметрами вывода изображений можно управлять
- Атрибуты `width` и `height` лучше указывать

Вывод изображений

```
<p> Это слон: <img src="elephant.jpg" width="600px" height="450px" alt="Слон,
   стоящий на хоботе"/>.
</p>
<p>
  <a href="http://www.sagradafamilia.cat/index.php">
    <img src="http://students.cis.uab.edu/kmvaughn/sagradafamiliawhole.jpg"
   width="100px" height="100px" alt="Sagrada Familia"/>
  </a>   
  <a href="http://www.casabatllo.es/">
    <img src="http://www.reedconsulting.com/ctdc/199912es/casa-batllo.jpg"
   width="75px" height="100px" alt="Casa Batllo"/>
  </a>   
  <a href="http://www.parkguell.es/">
    <img src="http://www.barcelona4all.com/Pics/ParkGuell4.jpg" width="134px"
   height="100px" alt="Guell Park"/>
  </a>
</p>
```

Результат

Это слон:

Таблицы

- `<table>`
  - Определяет таблицу
- `<tr>`
  - Определяет строку
- `<td>`
  - Определяет ячейку в строке
  - С помощью атрибутов colspan и rowspan ячейка может «захватывать» соседние ячейки
- `<caption>`
  - Заголовок таблицы
- `<th>`
  - Ячейка с заголовком
- …

- В ячейках могут содержаться другие объекты: текст, таблицы, рисунки
- Для элементов таблицы можно указывать параметры:
  - Границы
  - Ширина
  - Отступы от краёв ячеек
  - Расстояния между ячейками
  - Выравнивания
- В пустых ячейках лучше писать ` `
- Таблицы можно использовать не только как таблицы данных, но и как средство вёрстки

Таблицы

```
<table width="90%" border="1px" cellspacing="0px" cellpadding="4px" align="center">
  <caption>Таблица 1. Демонстрация возможностей таблиц</caption>
  <tr>
    <th colspan="4">Заголовок таблицы</th>
  </tr>
  <tr>
    <th width="15%">Строка 1</th>
    <td align="left" width="15%">left</td>
    <td align="center" width="20%">center</td>
    <td align="right" width="*">right</td>
  </tr>
  <tr>
    <th>Строка 2</th>
    <td rowspan="2" valign="bottom" align="right">bottom</td>
    <td rowspan="2" colspan="2" align="center" valign="middle">
      <a href="http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BA%D1%82-
    %D0%B4%D0%B8%D1%81%D0%BA">
        <img src="http://sparccd.spb.ru/images_index/cd_small.png"/></a></td>
  </tr>
  <tr height="400px">
    <th>Строка 3</th>
  </tr>
</table>
```

Результат

Таблица 1. Демонстрация возможностей таблиц

Заголовок таблицы			
Строка 1	left	center	right
Строка 2			
Строка 3	bottom		

HTTP

- **HTTP** (Словарь: Компьютерный англ.) сокр. от Hypertext Transfer Protocol
- Протокол передачи гипертекстовых файлов протокол уровня приложений для распределенных информационных систем гипермедиа, позволяющий общаться системам с различной архитектурой; используется при передаче HTML-файлов по сети WWW

HTTP

- Протокол для получения информации
- Протокол, работающий без установления сессии (в отличии от, например, FTP)
- Документы, которые получает пользователь, могут содержать ссылки на другие документы
- Документы, которые получает пользователь, могут быть отформатированы с использованием шрифтов, таблиц, графики, видео и музыки

HTTP – Get

- Основная команда для получения документов

GET *адрес*

GET http://www.yandex.ru/

Возвращает HTML документ

GET http://img.yandex.ru/i/logo-big-txt.gif

Возвращает бинарный GIF-файл

HTTP – Get

- Сервер может вернуть не только уже сохранённый на его диске документ, но и сгенерировать специально для Вас личный уникальный неповторимый HTML-документ
- Но для этого ему надо знать, какие именно данные мы хотим от него получить
- Для этого можно в URL (в адресе документа) указать дополнительные параметры:
  http://www.yandex.ru/yandsearch?text=something

HTTP – Post

- А если размер данных слишком большой?
- Тогда используется вторая команда: POST
- Данные включаются в тело запроса

POST http://www.yandex.ru/yandsearch

text=somebigdocumenttext

otherparam=otherbigcontent

Взаимодействие с сервером

- `<form>`
  - Определяет форму с экранными компонентами
  - Атрибут `method` определяет команду HTTP (get, post) и способ передачи параметров
  - Атрибут `action` определяет адрес документа для обработки формы
- Введенные в форме значения становятся значениями параметров запроса, имена которых определяются именами компонентов формы
- `<fieldset>`
  - Группирует компоненты в форме
  - `<legend>` определяет название группы
- `<input>`
  - Определяет элемент формы
  - Тип элемента формы указывается с помощью атрибута `type`

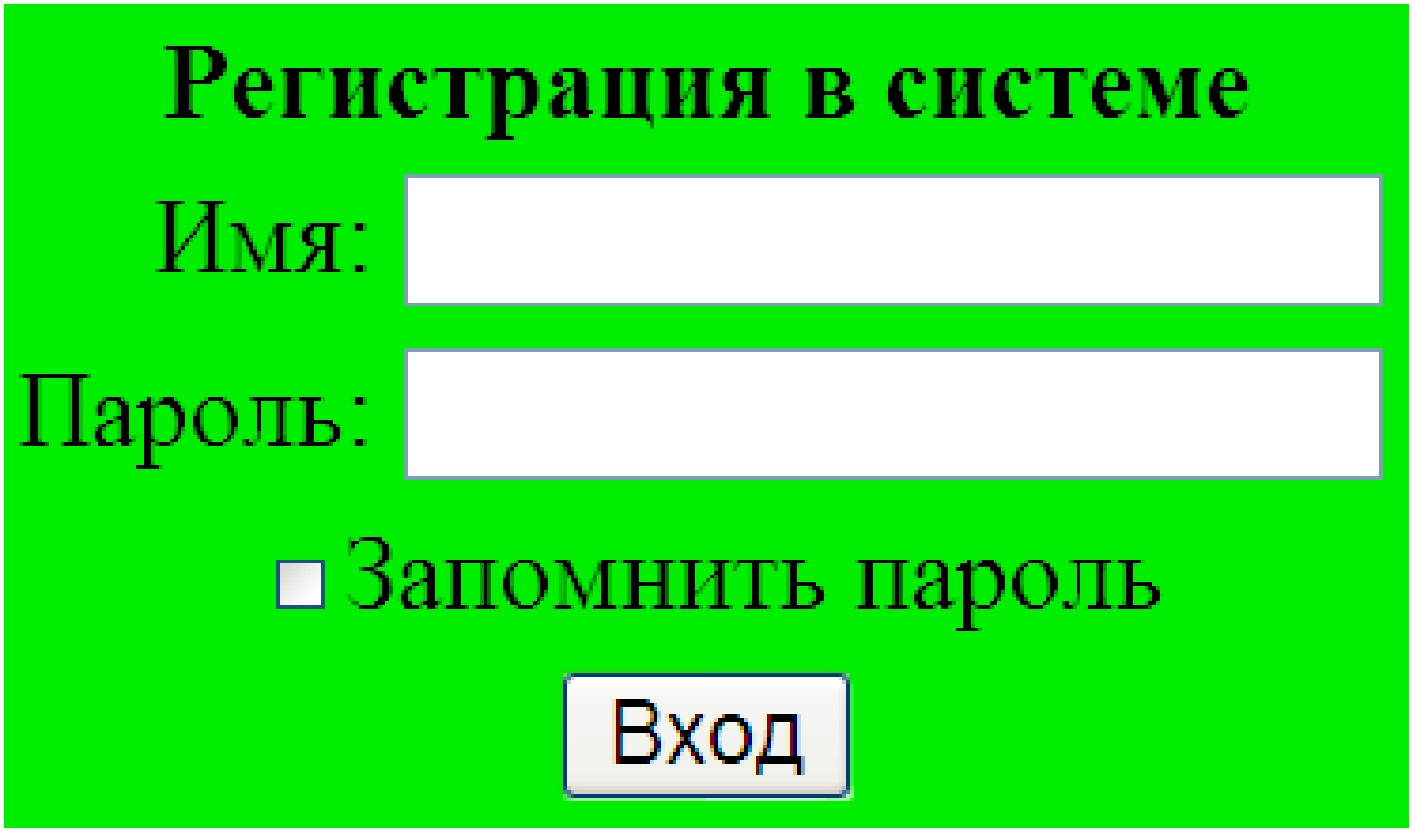
Элементы форм

Тип	Вид элемента
text	Текстовый редактор, можно указать размеры и выравнивание
checkbox	Флажок, можно указать, выбран ли он
radio	Переключатель, можно указать, выбран ли он; переключатели в одной группе должны иметь одинаковые имена
image	Кнопка с изображением
password	Текстовый редактор со скрытием символов
file	Средство загрузки файлов на сервер
reset	Кнопка для очистки формы
submit	Кнопка подтверждения и отправки формы
…	

Пример формы

```
<form name="input" action="HtmlFormAction.jsp" method="post">
  <table bgcolor="ltgreen">
    <tr>
      <th colspan="2" align="center">Регистрация в системе</th>
    </tr>
    <tr>
      <td align="right">Имя:</td>
      <td align="left"><input type="text" name="name"/></td>
    </tr>
    <tr>
      <td align="right">Пароль:</td>
      <td align="left"><input type="password" name="password"/></td>
    </tr>
    <tr>
      <td colspan="2" align="center"><input type="checkbox"/>Запомнить пароль</td>
    </tr>
    <tr>
      <td colspan="2" align="center"><input type="submit" value="Вход"/></td>
    </tr>
  </table>
</form>
```

Результат



Форматирование элементов

- Оригинально форматирование определялось браузером
- Часть тегов имеет атрибуты, определяющие форматирование
- Даже придумывались новые теги, обеспечивающие только форматирование
- Всё это – нарушение первоначальной идеи: **структура и формат должны быть разделены!!!**

Cascading Style Sheets

- Отображение определяется стилями
- Стили могут задаваться различными образами для различных элементов
- При наложении на одном элементе нескольких стилей приоритеты однозначно определяют отображение
- Возможности форматирования значительно шире, чем в HTML

Возможности CSS

- CSS1, 1996 г./1999 г.
  - Параметры шрифтов
  - Цвета
  - Границы
  - Атрибуты текста
  - Выравнивание элементов
  - Свойства блоков и отступы
- CSS2, 1998 г./2011 г.
  - Блочная вёрстка
  - Типы носителей
  - Расширенный механизм селекторов
  - …

Селекторы CSS

- Селектор определяет:
  - условие срабатывания
  - устанавливаемые свойства
  - значения свойств
- Различные свойства принято писать в различных строках
- Свойства:
  - простые, принимающие единственное значение
  - составные, принимающие наборы значений

```
селектор, селектор {
  свойство: значение;
  свойство: значение;
  ...
  свойство: значение;
}
```

Базовые селекторы

- Элементов
```
p {font-family: Garamond, serif;}
```
- Классов
```
.note {color: red; background: yellow;
font-weight: bold;}
```
- Идентификаторов
```
#paragraph1 {margin: 0;}
```
- Псевдоклассов
```
a:hover {color:yellow;}
```
- Псевдоэлементов
```
p:first-letter {font-size: 32pc;}
```

Способы указания стиля

- Внешняя таблица стилей
  - `<link rel="stylesheet" type="text/css" href="st.css"/>`
- Таблица стилей документа
  - Определяется в заголовке документа
  - `<style> … </style>`
- Атрибут стиля элемента
  - `<p style="color:red; font-family: courier">`

Пример использования стилей

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <title>Styles</title>
  <style type="text/css">
   .myclass {
    width: 200px; background: #E1E1E1; padding: 5px; padding-right: 20px; border: solid 1px
    black; float: left;
   }
   #myid {
    width: 200px; background: #00A0A0; padding: 5px; border: solid 1px black; float: left;
    position: relative; top: 40px; left: -70px;
   }
   p { font-family: arial; font-size: 200%; }
  </style>
 </head>
 <body>
  <div class="myclass">Я пришёл к тебе с приветом<br/> Рассказать, что солнце встало,</div>
  <div id="myid">Что оно горячим светом <p>По листам затрепетало</p></div>
 </body>
</html>
```

Результат

Я пришёл к тебе с приветом
Рассказать, что солнце
встало,

Что оно горячим светом

По листам
затрепетало

Приоритеты стилей

- Стили браузера
- Стили в настройках браузера
- Стили документа:
  - Подключенные к документу стили
  - Стили в заголовке документа
  - Атрибуты `style` тега
  - Стили, помеченные словом `!important`

Спасибо за внимание!

Дополнительные источники

- Дакетт, Дж. Основы веб-программирования с использованием HTML, XHTML и CSS [Текст] / Джон Дакетт. – М. : Эксмо, 2011. – 768 с.
- Шафер, С. HTML, XHTML и CSS. Библия пользователя [Текст] / Стивен Шафер. – М. : Издательский дом «Вильямс», 2011. – 656 с.
- Хеник, Б. HTML и CSS. Путь к совершенству [Текст] / Бен Хеник. – СПб. : Питер, 2011. – 336 с.
- HTML & CSS [Электронный ресурс]. – Режим доступа: http://www.w3.org/standards/webdesign/htmlcss, дата доступа: 14.10.2013.
- Htmlbook.ru [Электронный ресурс]. – Режим доступа: http://htmlbook.ru/, дата доступа: 14.10.2013.
- HTML справочник [Электронный ресурс]. – Режим доступа: http://html.manual.ru/, дата доступа: 14.10.2013.
- HTML справочник [Электронный ресурс]. – Режим доступа: http://www.html-info.net/, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

eXtensible Markup Language

Занятие 6

Гаврилов А.В.

Самара
2013

План занятия

- Общие принципы
- Document type definition
- XML Schema
- SAX и DOM
- Работа с SAX и DOM в Java
- Запись XML в Java
- XML-сериализация в Java

Отличия XML от HTML

- XML чувствителен к регистру
- В XML нельзя «опускать» закрывающие тэги
- В XML часто встречаются тэги, одновременно открывающие и закрывающие
  `<img src="coffeecup.png"/>`
- В XML значения атрибутов должны быть заключены в кавычки
- В XML все атрибуты должны иметь значения

Пример XML

```
<configuration>
  <title>
    <font> <name>Helvetica</name> <size>36</size> </font>
  </title>
  <body> <name>Times Roman</name> <size>12</size> </body>
  <window> <width>400</width> <height>200</height> </window>
  <color>
    <red>0</red>
    <green>50</green>
    <blue>100</blue>
  </color>
  <menu>
    <item>Times Roman</item>
    <item>Helvetica</item>
  </menu>
</configuration>
```

Структура XML-документа

- Заголовок

```
<?xml version="1.0"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ex01-1.xsl"?>
```

- Объявления типа документа

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.
//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

- Корневой элемент

```
<configuration>
</configuration>
```

Структура XML-документа

- Смешанное наполнение не рекомендуется

```
<font>
        Helvetica
        <size>36</size>
</font>
```

- Существуют атрибуты

```
<font>
        <name>Helvetica</name>
        <size unit="pt">36</size>
</font>
```

Некоторые инструкции

- Символы

```
&#233, &#x2122
```

- Стандартные символы

```
&lt; &gt; &amp; &quot; &apos;
```

- Инструкции обработки

```
<?xml version="1.0"?>
```

- Комментарии

```
<!-- This is a comment. -->
```

Правильный документ

- Начинается с объявления
- Содержит один уникальный корневой элемент
- Все открытые теги закрываются
- Учтена чувствительность к регистру
- Теги корректно вложены друг в друга
- Значения всех атрибутов заключены в кавычки
- Специальные символы задаются с помощью инструкций

Document Type Definition (DTD)

- Содержит правила, описывающие структуру документа
- Транслятор может автоматически проверять документ на соответствие этим правилам
- Описывает дочерние элементы и атрибуты для каждого элемента
- Включение в XML-документ

```
<!DOCTYPE имя [правила]>

<!DOCTYPE configuration SYSTEM "config.dtd">

<!DOCTYPE configuration SYSTEM
"http://myserver.com/config.dtd">
```

Регулярные выражения

Правило	Смысл
E*	0 или больше вхождений E
E+	1 или больше вхождений E
E?	0 или 1 вхождение E
E1|E2|…|En	Одно из E1, E2, …, En
E1, E2, …, En	Последовательность E1, E2, …, En
#PCDATA	Текст
(#PCDATA|E1|…|En)*	Смешанное наполнение
ANY	Произвольный дочерний тэг
EMPTY	Нет дочерних тэгов

Примеры выражений

```
Описание меню
<!ELEMENT menu (item)*>

Описание шрифта
<!ELEMENT font (name,size)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT size (#PCDATA)>

Описание главы в книге
<!ELEMENT chapter
(intro,(heading,(para|image|table|note)+)+)>
```

Описание атрибутов: типы

Тип	Смысл
CDATA	Произвольная строка
(A1|A2|…|An)	Один из строковых атрибутов A1, A2, …, An
NMTOKEN, NMTOKENS	Одна или более строк, записанных по правилам имен
ID	Уникальный ID
IDREF, IDREFS	Одна или более ссылка на уникальный ID
ENTITY, ENTITIES	Ссылки на внешние сущности

Описание атрибутов: значения

Значение	Смысл
#REQUIRED	Атрибут обязателен
#IMPLIED	Атрибут опционален
A	Атрибут опционален, если значение не указано, то принимается равным A
#FIXED A	Атрибут не указывается или равен A

Примеры выражений

```
<!ATTLIST font style (plain|bold|italic|bold-italic)
plain>
<!ATTLIST size unit CDATA #IMPLIED>

<!ELEMENT gridbag (row)*>
<!ELEMENT row (cell)*>
<!ATTLIST cell gridwidth CDATA "1">
<!ATTLIST cell gridheight CDATA "1">
<!ATTLIST cell fill (NONE|BOTH|HORIZONTAL|VERTICAL)
"NONE">
<!ATTLIST cell anchor (CENTER|NORTH|NORTHEAST|EAST
|SOUTHEAST|SOUTH|SOUTHWEST|WEST|NORTHWEST)
"CENTER">
<!ATTLIST cell ipadx CDATA "0">
<!ATTLIST cell ipady CDATA "0">
```

XML Schema

- Предназначена для того же, что и DTD
- Для описания правил используется непосредственно XML
- Имеет более гибкие возможности, чем DTD
  - Расширяема
  - Более гибкие возможности
  - Есть понятие типа данных
  - Есть понятие пространства имен
- Сложнее в восприятии и программировании средств, ее обрабатывающих
- www.w3.org/XML/Schema
  http://www.w3schools.com/Schema/default.asp

Поддержка типов данных

- Проще описывать допустимое содержимое документа
- Проще проверять корректность данных
- Проще накладывать ограничения на данные
- Проще определять формат данных

XML Schema описывается на XML

- Не требуется изучение еще одного языка
- Вы можете использовать свой любимый XML-редактор для работы со схемой
- Вы можете работать со схемой программно
- Вы можете изменять свою схему с помощью XSLT

Документ и тип DTD

```
<?xml version="1.0"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

XML Schema для документа

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">

<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```

Указание типа документа

```
<?xml version="1.0"?>

<!DOCTYPE note SYSTEM "http://www.w3schools.com/dtd/note.dtd">

<note>
  <!-- Some content -->
</note>
```

```
<?xml version="1.0"?>

<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com note.xsd">
 <!-- Some content -->
</note>
```

Простые элементы

- Не могут содержать других элементов
- Не могут содержать атрибутов
- Содержат только текст
- «Только текст» должен иметь тип
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:date
  - И т.д.
- Могут накладывать ограничения на содержание
- Могут иметь значения по умолчанию

Примеры описания простых элементов

```
<xs:element name="lastname" type="xs:string"/>
<xs:element name="color" type="xs:string" default="red"/>
<xs:element name="color" type="xs:string" fixed="red"/>

<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="([a-z])*"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Комплексные элементы

- Четыре вида комплексных элементов
  - Пустые элементы
  - Внутри только другие элементы
  - Внутри только текст
  - Внутри и текст, и элементы
- Все четыре вида могут иметь атрибуты

Атрибуты

- Описывают атрибуты тега
- По форме описания очень напоминают простые типы
- Кроме прочего, можно указать обязательность атрибута

```
<xs:attribute name="lang" type="xs:string"/>
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Пустые комплексные элементы

```
<xs:element name="product">
  <xs:complexType>
    <xs:attribute name="prodid"
                  type="xs:positiveInteger"/>
  </xs:complexType>
</xs:element>


<xs:element name="product" type="prodtype"/>

<xs:complexType name="prodtype">
  <xs:attribute name="prodid"
                type="xs:positiveInteger"/>
</xs:complexType>
```

Комплексные элементы с элементами

- Можно управлять порядком появления элементов
  - <xs:all>
  - <xs:choice>
  - <xs:sequence>
- Можно управлять количеством элементов
  - Атрибут minOccurs
  - Атрибут maxOccurs

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
      maxOccurs="10" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Комплексные элементы с текстом

```
<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="basetype">
        ....
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="somename">
  <xs:complexType>
    <xs:simpleContent>
      <xs:restriction base="basetype">
        ....
      </xs:restriction>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Комплексные элементы со смешанным содержимым

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name"
                  type="xs:string"/>
      <xs:element name="orderid"
                  type="xs:positiveInteger"/>
      <xs:element name="shipdate"
                  type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Extensible Stylesheet Language (XSL)

- Комплекс технологий, связанных с преобразованием и представлением XML-документов
- Обычно используется для преобразования документов в XML, HTML, текст и PDF (XSL-FO)
- XSL Transformations (XSLT) – язык, на котором описываются правила преобразования
- XPath – язык, позволяющий формулировать используемые в процессе преобразования выражения, использующие различные фрагменты документа
- http://www.w3.org/Style/XSL/
  http://www.w3schools.com/xsl/

XPath

- Вспомогательный язык, позволяющий обращаться к элементам документа
- Имя элемента представляется в виде пути `/bookstore/book/title`
- Обращение может происходить и к атрибутам
- http://www.w3.org/TR/xpath
  http://www.w3schools.com/Xpath/default.asp

Примеры выражений XPath

Выражение	Результат
`bookstore`	Все дочерние элементы для элемента bookstore
`/bookstore`	Корневой элемент bookstore
`bookstore/book`	Все элементы book, дочерние для bookstore
`//book`	Все элементы book в документе
`bookstore//book`	Все элементы book в рамках элемента bookstore
`@lang`	Атрибуты lang
`.`	Текущий элемент
`..`	Родительский элемент

Принципы XSL

- Контекстно-зависимый язык
- Основные элементы – выводимый текст и шаблоны
  - Текст просто выводится
  - Шаблоны описывают некоторые действия
    - Могут быть вызваны явно
    - Могут быть вызваны неявно, по условию совпадения шаблона
- Имеются средства управления ходом выполнения
- Позволяет создавать и вызывать бибилиотеки с помощью тега `<xsl:include href="..."/>`

Пример XSL

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <html>
 <body>
  <h2>My CD Collection</h2>
  <table border="1">
   <tr bgcolor="#9acd32">
    <th>Title</th>
    <th>Artist</th>
   </tr>
   <xsl:for-each select="catalog/cd">
    <tr>
     <td><xsl:value-of select="title"/></td>
     <td><xsl:value-of select="artist"/></td>
    </tr>
   </xsl:for-each>
  </table>
 </body>
 </html>
</xsl:template>
</xsl:stylesheet>
```

Пример XSL

```
<xsl:template
   match="note|important|warning|caution|tip">
  <xsl:choose>
    <xsl:when test="$admon.graphics != 0">
      <xsl:call-template
        name="graphical.admonition"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:call-template
        name="nongraphical.admonition"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Пример XSL

```
<xsl:template name="graphical.admonition">
  <xsl:variable name="id">
    <xsl:call-template name="object.id"/>
  </xsl:variable>
  <xsl:variable name="graphic.width">
     <xsl:apply-templates select="." mode="admon.graphic.width"/>
  </xsl:variable>
...
  <xsl:if test="$admon.textlabel != 0 or title">
    <fo:block xsl:use-attribute-sets="admonition.title.properties">
      <xsl:apply-templates select="." mode="object.title.markup"/>
    </fo:block>
  </xsl:if>
  <fo:block xsl:use-attribute-sets="admonition.properties">
     <xsl:apply-templates/>
  </fo:block>
...
</xsl:template>
```

Использование XSLT

- Включение в XML-файл строкой заголовка

```
<?xml version="1.0"?>
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="ex01-1.xsl"?>
```

- Обработку выполняет XSLT-Processor
  - Большинство современных браузеров имеют встроенные процессоры
  - Существуют программы процессоров
    - Самостоятельные приложения
    - Программные модули, в т.ч. на Java

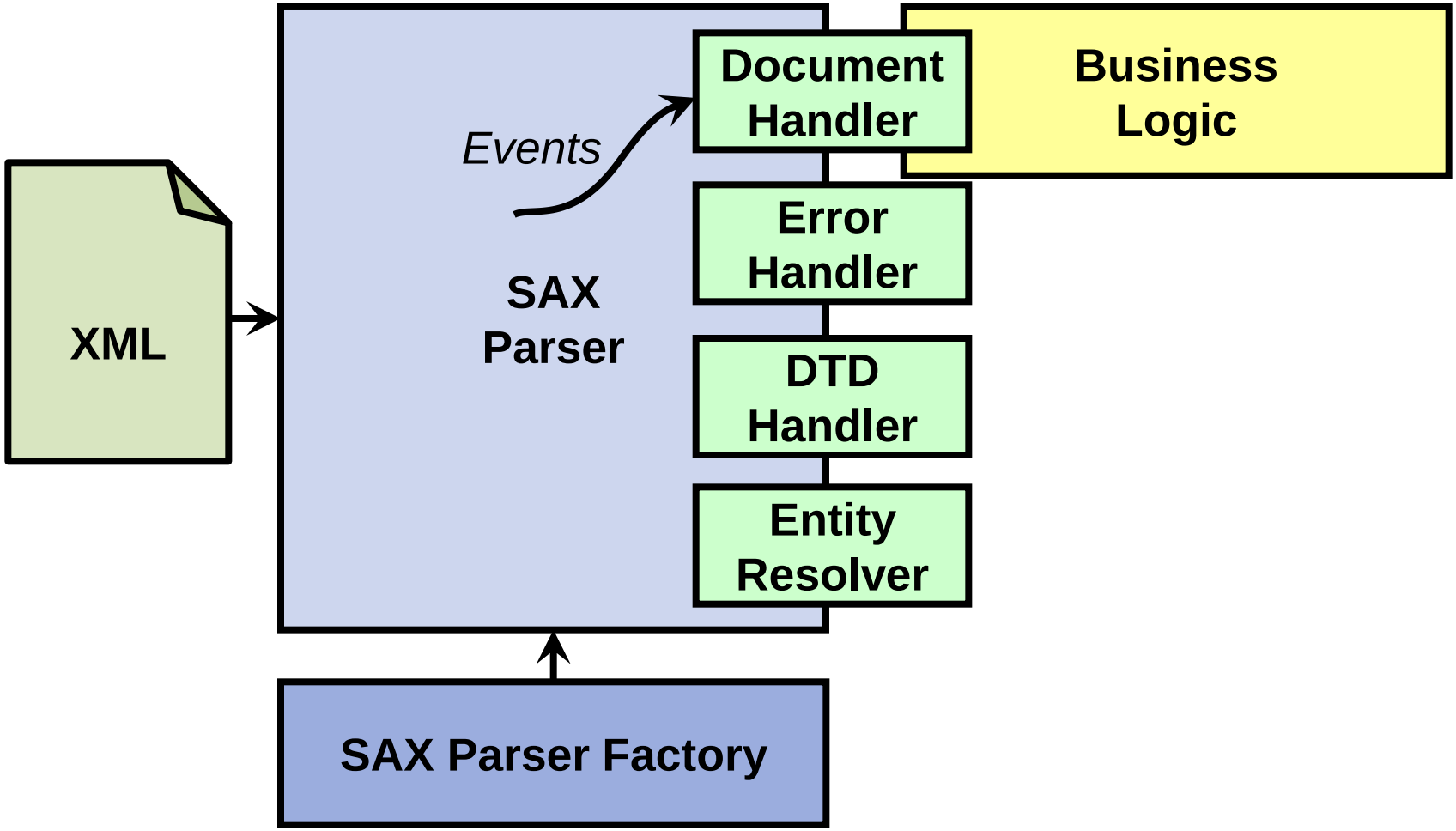
Обработка XML

Два подхода

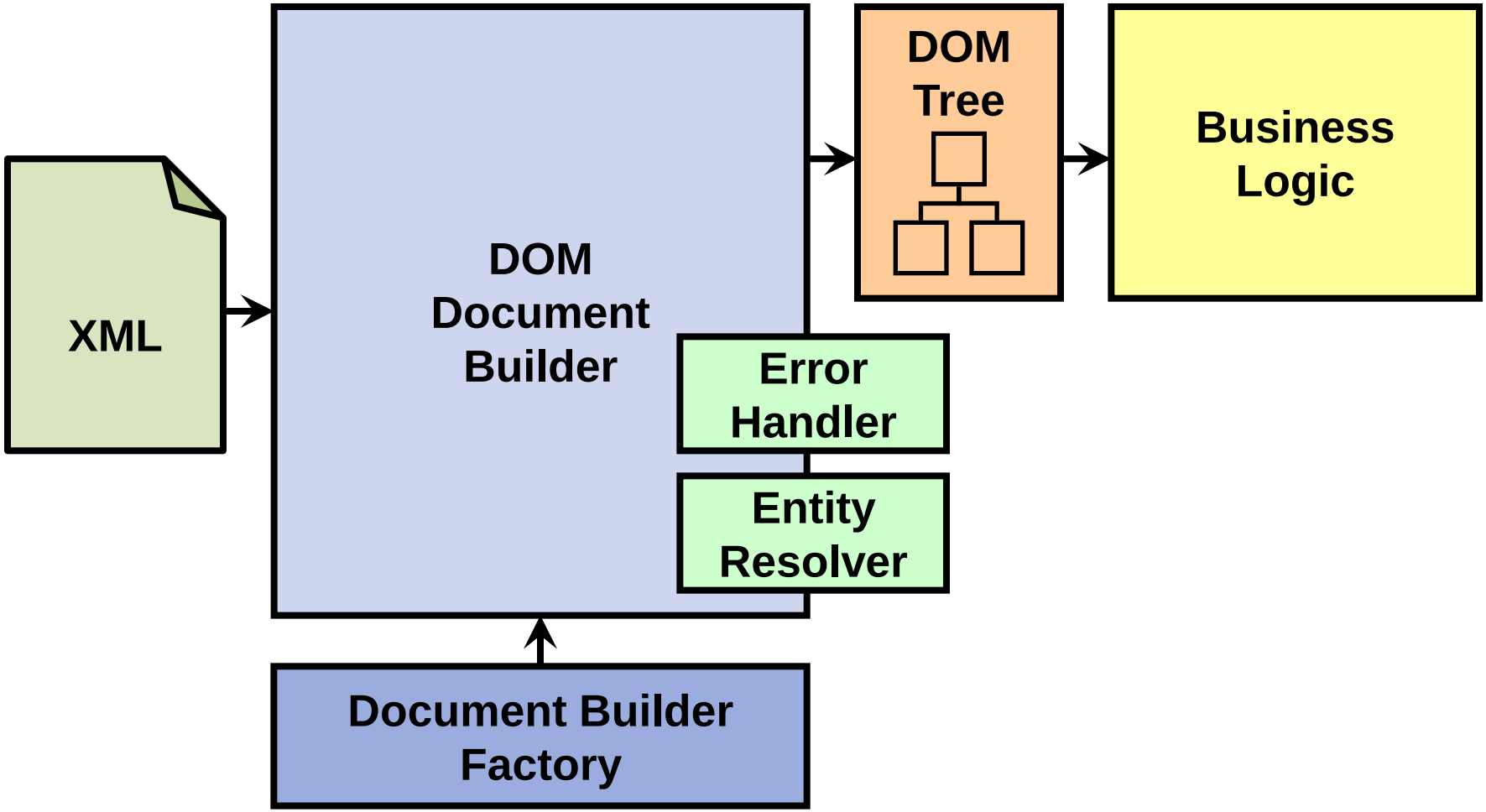
- **Simple API for XML (SAX)**
  Порождает события в процессе чтения XML документа

- **Document Object Model (DOM)**
  Представляет XML документ в форме древовидной структуры элементов

Логика SAX


Document Handler
Business Logic
Events
Error Handler
XML
SAX Parser
DTD Handler
Entity Resolver
SAX Parser Factory

Логика DOM


XML
DOM
Document
Builder
DOM
Tree
Business
Logic
Error
Handler
Entity
Resolver
Document Builder
Factory

Особенности SAX и DOM

SAX	DOM
Модель обработки событий	Древовидная структура данных
Последовательный доступ (поток событий)	Произвольный доступ (структура данных в памяти)
Используется мало памяти (порождаются только события)	Используется много памяти (документ загружен полностью)
Для обработки частей документа (обработка релевантных событий)	Для редактирования документа (обработка данных в памяти)
Для однократной обработки документа	Для многократной обработки документа

Работа с XML в Java

- Стандартные интерфейсы
  - В оригинале описаны на Interface Definition Language (OMG IDL)
  - Пакет **`org.w3c.dom`**
  - Пакет **`org.xml.sax`**
- Реализующие классы
  - Предоставляются отдельно…
  - JAVA API for XML Processing (JAXP)
    Пакет **`javax.xml`**

Работа с SAX

- Обработку документа производит транслятор, передающий информацию зарегистрировавшимся обработчикам событий

- Обработчики должны реализовывать интерфейсы
  - `org.xml.sax.ContentHandler`
  - `org.xml.sax.DTDHandler`
  - `org.xml.sax.EntityResolver`
  - `org.xml.sax.ErrorHandler`
  - `org.xml.sax.Locator`
  - `org.xml.sax.ext.DeclHandler`
  - `org.xml.sax.ext.EntityResolver2`
  - `org.xml.sax.ext.LexicalHandler`
  - `org.xml.sax.ext.Locator2`

SAX

SAX2

Пакет javax.xml.parsers

- Класс **`SAXParserFactory`**
  Образец проектирования Singleton, позволяет настроить и получить экземпляр фабрики для производства **`SAXParser`**

- Класс **`SAXParser`**
  Непосредственно транслятор, экземпляры получаются от фабрики **`SAXParserFactory`**

Семантика документа

- Возникающие события
  - `startElement` / `endElement`
    Открывающий и закрывающий тэг
  - `characters`
    Символы
  - `startDocument` / `endDocument`
    Начало и конец документа

- Интерфейс `ContentHandler`
  - `startElement()`
  - `endElement()`
  - `characters()`
  - `startDocument()`
  - `endDocument()`

Создание обработчика событий

- Реализация нужного интерфейса, настройка на него используемого транслятора

- Использование класса `org.xml.sax.helpers.DefaultHandler` или `org.xml.sax.helpers.DefaultHandler2` реализующих все интерфейсы обработки событий (все методы имеют пустые тела)

Пример. Часть 1

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import javax.xml.parsers.*;
import java.io.IOException;

public class ReaderSAX extends DefaultHandler {
  private int indent = 0;
  final static int INDENT = 4;

  public void startDocument() throws SAXException {
    printString ("Начало документа");
  }

  public void endDocument() throws SAXException {
    printString ("Конец документа");
  }
```

Пример. Часть 2

```
public void startElement (String uri, String localName,
                          String qName, Attributes attributes)
  throws SAXException {
  indent += INDENT;
  printString ("Элемент " + qName + ":");
}

public void endElement (String uri, String localName,
                        String qName) throws SAXException {
  printString ("Конец элемента " + qName + ".");
  indent -= INDENT;
}

public void warning (SAXParseException e) throws SAXException {
  System.out.println ("Предупреждение :" + e.getPublicId());
}
```

Пример. Часть 3

```
public void error (SAXParseException e) throws SAXException {
  System.out.println ("Ошибка :" + e.getPublicId());
}

public void fatalError (SAXParseException e) throws SAXException {
  System.out.println ("Фатальная ошибка :" + e.getPublicId());
}

public void characters (char[] ch, int start, int length)
  throws SAXException {
  indent += INDENT;
  String str = new String (ch, start, length);
  printString (str);
  indent -= INDENT;
}
```

Пример. Часть 4

```
public void printString (String str) {
  String ind_s;
  if (indent > 0) {
    char[] ind = new char[indent];
    java.util.Arrays.fill(ind, ' ');
    ind_s = new String (ind, 0, indent);
  }
  else {
    ind_s = "";
  }
  System.out.println (ind_s + str);
}
```

Пример. Часть 5

```
  public static void main (String[] args) {
    DefaultHandler handler = new ReaderSAX();
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser parser = null;
    try {
      parser = factory.newSAXParser();
      parser.parse ("test.xml", handler);
    } catch (ParserConfigurationException e) {
       e.printStackTrace();
    } catch (SAXException e) {
       e.printStackTrace();
    } catch (IOException e) {
       e.printStackTrace();
    }
  }
}
```

Работа с DOM

- Считывание документа, опять же, реализует транслятор
- Результат считывания возвращается в виде дерева объектов, реализующих интерфейс `org.w3c.dom.Node`
- Дальнейшая обработка ведется уже на уровне бизнес-логики

Пакет org.w3c.dom

- Базовый интерфейс **`Node`**, содержит основные методы работы с узлом
- От него наследуют специфические интерфейсы для конкретных видов узлов:
  - **`Document`**
  - **`Element`**
  - **`Text`**
  - **`Comment`**
  - **`Attr`**
  - и др.
- Каждый интерфейс добавляет новую функциональность (например **`Document`**, является фабрикой для создания остальных узлов)

Пакет javax.xml.parsers

- Класс `DocumentBuilderFactory`
  Образец проектирования Singleton, позволяет настроить и получить экземпляр фабрики для производства `DocumentBuilder`

- Класс `DocumentBuilder`
  Непосредственно транслятор, экземпляры получаются от фабрики `DocumentBuilderFactory`

Пример. Часть 1

```
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import javax.xml.parsers.*;
import java.io.IOException;

public class ReaderDOM {
  private int indent = 0;
  final static int INDENT = 4;
  private static final String[] TYPE_NAMES = {
    "ELEMENT_NODE", "ATTRIBUTE_NODE", "TEXT_NODE",
    "CDATA_SECTION_NODE", "ENTITY_REFERENCE_NODE",
    "ENTITY_NODE", "PROCESSING_INSTRUCTION_NODE",
    "COMMENT_NODE", "DOCUMENT_NODE", "DOCUMENT_TYPE_NODE",
    "DOCUMENT_FRAGMENT_NODE", "NOTATION_NODE"
  };
  public String getTypeName (short nodeType) {
    return TYPE_NAMES[nodeType - 1];
  }
```

Пример. Часть 2

```
public static void main(String[] args) {
  try {
    DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse("test.xml");
    ReaderDOM obj = new ReaderDOM();
    obj.myPrint (document);
  } catch (ParserConfigurationException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  } catch (SAXException e) {
    e.printStackTrace();
  }
}
```

Пример. Часть 3

```
public void myPrint (Document document) {
  printDocInfo (document);
  printNodeInfo (document);
}
public void printDocInfo (Document document) {
  printString ("Имя узла документа : " + document.getNodeName());
}
public void printElementInfo (Element element) {
  NamedNodeMap nnm = element.getAttributes();
  for (int i = 0; i < nnm.getLength(); i++) {
    Attr attr = (Attr)nnm.item (i);
    printString ("Attribute name = " + attr.getName() +
                 ", value = " + attr.getValue());
  }
}
public void printTextInfo (Text text) {
  printString ("Значение поля : " + text.getData());
}
```

Пример. Часть 4

```
public void printNodeInfo (Node node) {
  indent += INDENT;
  printString ("Node name = " + node.getNodeName() +
               ", node type = " + getTypeName (node.getNodeType()));
  if (node instanceof Element) {
    printElementInfo ((Element)node);
  } else if (node instanceof Text) {
    printTextInfo ((Text)node);
  }
  NodeList nl = node.getChildNodes();
  for (int i = 0; i < nl.getLength(); i++) {
          printNodeInfo (nl.item (i));
  }
  indent -= INDENT;
}
```

Пример. Часть 5

```
  public void printString (String str) {
    String ind_s;
    if (indent > 0) {
      char[] ind = new char[indent];
      java.util.Arrays.fill(ind, ' ');
      ind_s = new String (ind, 0, indent);
    }
    else {
      ind_s = "";
    }
    System.out.println (ind_s + str);
  }
}
```

Запись XML

- Средствами пакета `javax.xml.transform`
- Средствами API третьих фирм JDOM (www.jdom.org)

Пример. Часть 1

```
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.IOException;

public class WriterDOM {
  public static void main(String[] args) {
    try {
      DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = null;
      builder = factory.newDocumentBuilder();
      Document document = builder.parse("test.xml");
      DOMSource dom_source = new DOMSource (document);
      StreamResult out_stream = new StreamResult("test2.xml");
```

Пример. Часть 2

```
TransformerFactory tFactory = TransformerFactory.newInstance();
Transformer transformer = tFactory.newTransformer(/* !!!! */);

// Вспомогательные действия, связанные с тем, что такая
// элементарная трансформация не "копирует" директиву
// !DOCTYPE. В зависимости от PUBLIC- или SYSTEM-описания DTD,
// можно использовать разные свойства transformer'а

DocumentType docType = document.getDoctype();
String systemID = docType.getSystemId();
String publicID = docType.getPublicId();
String res = publicID + "\" \"" + systemID;
//  transformer.setOutputProperty
//  (OutputKeys.DOCTYPE_SYSTEM, systemID);
transformer.setOutputProperty (OutputKeys.DOCTYPE_PUBLIC, res);
```

Пример. Часть 3

```
      // Прочие настройки преобразователя

      transformer.transform (dom_source, out_stream);
    } catch (ParserConfigurationException e) {
      e.printStackTrace();
    } catch (TransformerConfigurationException e){
      e.printStackTrace();
    } catch (TransformerException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    } catch (SAXException e) {
      e.printStackTrace();
    }
  }
}
```

Настройка преобразователя

- Метод создания объекта преобразователя `TransformerFactory.newTransformer()` имеет 2 формы:
  - без аргументов – будет создаваться «копия» исходного документа
  - с аргументом типа `Source` – ссылка на загруженный объект xml-документа, в котором описано XSL-преобразование
- Метод `Transformer.setOutputProperty()` позволяет настроить некоторые параметры вывода (см. класс `OutputKeys`)

Размышления на тему

- Итак, что мы научились делать:
  - Считывать информацию из XML-документов
    - SAX
    - DOM
  - Записывать информацию в XML-документы
- Какой еще инструмент был бы удобен?..
- А если бы мы умели записывать и считывать из XML непосредственно объекты Java?..

Шаг 1. Сохранение JavaBeans

- В версии JavaSE 1.4 для объектов JavaBeans появились механизмы, сходные с сериализацией
- Реализовывали их классы `java.beans.XMLEncoder` и `java.beans.XMLDecoder`
- Недостаток: механизм основан на интроспекции, требует соблюдения правил именования и т.д.

Пример. Часть 1

```
XMLEncoder e = new XMLEncoder(
                new BufferedOutputStream(
                 new FileOutputStream("Test.xml"))); e.writeObject(new
JButton("Hello, world"));
e.close();
```

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.0" class="java.beans.XMLDecoder">
  <object class="javax.swing.JFrame">
    <void property="name">
      <string>frame1</string>
    </void>
    <void property="bounds">
      <object class="java.awt.Rectangle">
        <int>0</int> <int>0</int>
        <int>200</int> <int>200</int>
      </object>
    </void>
```

Пример. Часть 2

```
 <void property="contentPane">
      <void method="add">
        <object class="javax.swing.JButton">
          <void property="label">
            <string>Hello, world</string>
          </void>
        </object>
      </void>
    </void>
    <void property="visible">
      <boolean>true</boolean>
    </void>
  </object>
</java>
```

Шаг 2. Java Architecture for XML Binding (JAXB)

- В версии JavaSE 1.6 появились новые механизмы JAXB
- Связанные с ними классы находятся в пакете `javax.xml.bind`
- Позволяют производить «сериализацию» объектов и их структур в XML
- Классы объектов должны быть специальным образом подготовлены
- Активно использует механизм аннотаций…

Пример. RootClass

```
import javax.xml.bind.annotation.*;

@XmlRootElement
public class RootClass {
  private int value;

  @XmlElement
  private NodeClass name = new NodeClass();

  public RootClass() {
    value = 0;
    name.setInnerValue("");
  }

  public NodeClass getName() { return name; }
  public int getValue() { return value; }
  public void setValue(int newValue) { value = newValue; }
}
```

Пример. NodeClass (1)

```
public class NodeClass {
  private String innerValue = "";
  private double rval = Math.random();

  public String getInnerValue() {
    return innerValue;
  }

  public void setInnerValue(String newInnerValue) {
    innerValue = newInnerValue;
  }

  public void print() {
    System.out.println(rval);
  }
}
```

Пример. WriterJAXB

```
import javax.xml.bind.*;
import java.io.*;

public class WriterJAXB {
  public static void main(String[] args){
    try {
      RootClass object1 = new RootClass();
      object1.setValue(5);
      object1.getName().setInnerValue("ABC");
      JAXBContext jc = JAXBContext.newInstance(RootClass.class);
      Marshaller m = jc.createMarshaller();
      OutputStream os = new FileOutputStream("test.xml");
      m.marshal(object1, os);
      os.close();
    }
    catch (JAXBException e) {e.printStackTrace();}
    catch (FileNotFoundException e) {e.printStackTrace();}
    catch (IOException e) {e.printStackTrace();}
  }
}
```

Содержимое файла после выполнения (1)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rootClass>
  <name>
    <innerValue>ABC</innerValue>
  </name>
  <value>5</value>
</rootClass>
```

- Что сохранено:
  - значение **name**, помеченное аннотацией
    - значение **innerValue**, не помеченное аннотацией
    - значение **rval** не сохранено
  - значение **value**, не помеченное аннотацией
- Сохранились элементы, являющиеся свойствами JavaBeans

Пример. NodeClass (2)

```
import javax.xml.bind.annotation.*;
public class NodeClass {
  private String innerValue = "";
  @XmlElement
  private double rval = Math.random();

  public String getInnerValue() {
    return innerValue;
  }

  public void setInnerValue(String newInnerValue) {
    innerValue = newInnerValue;
  }

  public void print() {
    System.out.println(rval);
  }
}
```

Содержимое файла после выполнения (2)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rootClass>
  <name>
    <rval>0.9878295088863659</rval>
    <innerValue>ABC</innerValue>
  </name>
  <value>5</value>
</rootClass>
```

- Сохранились элементы, являющиеся свойствами JavaBeans
- Сохранились элементы, помеченные аннотациями

Пакет javax.xml.bind.annotation

- Содержит разнообразнейшие аннотации, описывающие параметры маршалинга и анмаршалинга
- `@XmlRootElement`
  Обозначает корневой элемент сохраняемой структуры
- `@XmlElement`
  Обозначает поля и свойства (для JavaBeans)
- `@XmlTransient`
  Обозначает то, что поле не будет сохраняться

Пример. ReaderJAXB

```
import javax.xml.bind.*;
import java.io.*;

public class ReaderJAXB {
  public static void main(String[] args) {
    try {
      JAXBContext jc = JAXBContext.newInstance(RootClass.class);
      InputStream is = new FileInputStream("test.xml");
      Unmarshaller um = jc.createUnmarshaller();
      RootClass object2 = (RootClass) um.unmarshal(is);
      System.out.println(object2.getValue());
      System.out.println(object2.getName().getInnerValue());
      is.close();
    }
    catch (JAXBException e) {e.printStackTrace();}
    catch (FileNotFoundException e) {e.printStackTrace();}
    catch (IOException e) {e.printStackTrace();}
  }
}
```

Спасибо за внимание!

Дополнительные источники

- Хорстманн, К.С. Java2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010. – 816 с.
- Extensible Markup Language (XML) [Электронный ресурс]. – Режим доступа: http://www.w3.org/XML/, дата доступа: 14.10.2013.
- XML Tutorial [Электронный ресурс]. – Режим доступа: http://www.w3schools.com/xml/, дата доступа: 14.10.2013.
- Язык XML – практическое введение [Электронный ресурс]. – Режим доступа: http://www.codenet.ru/webmast/xml/, доступ: 14.10.2013.
- Java API for XML Processing [Электронный ресурс]. – Режим доступа: http://download.oracle.com/javaee/1.4/tutorial/doc/JAXPIntro.html, дата доступа: 14.10.2013.
- Java Architecture for XML Binding (JAXB) [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/articles/javase/index-140168.html, дата доступа: 14.10.2013.
- JAXB Reference Implementation [Электронный ресурс]. – Режим доступа: http://jaxb.java.net/, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Введение в JavaScript

Занятие 7

Гаврилов А.В.

Самара
2013

План занятия

- Общие сведения о JavaScript
- Основы синтаксиса
- Основы работы с объектами
- Основы работы с документом и браузером

JavaScript

- JavaScript – это язык программирования
  - Интерпретируемый
  - Объектно-ориентированный
  - Со слабой типизацией
  - С динамической типизацией
  - С автоматическим управлением памятью
  - Вместо наследования классов используются прототипы объектов
  - Функции являются объектами

Области применения

- Скриптовые фрагменты серверных приложений
- Виджеты
- Прикладное программное обеспечение
- Букмарклеты (небольшие приложения, размещаемые в закладках браузера)
- Пользовательские скрипты в браузерах

Области применения

- Программы на стороне клиента в web-приложениях
  - Код встраивается в HTML-страницу
  - Код выполняется браузером
  - Возможно изменение структуры страницы, её элементов и их параметров
  - Изменения могут происходить без перезагрузки страницы
  - Могут выполняться дополнительные действия
    - Взаимодействие с браузером
    - Взаимодействие с сервером
    - Бизнес-логика в целом

Здравствуй, мир!!!

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Проверка JS</title>
  </head>
  <body>
    <h1>
      <script>
        document.write("Hello, world!!!");
      </script>
    </h1>
  </body>
</html>
```

Размещение в HTML-документе

- Тег <script>

```
<html>
  <body>
    <p id="demo">This is a paragraph.</p>
    <script type="text/javascript">
      document.getElementById("demo").innerHTML=Date();
    </script>
  </body>
</html>
```

- Атрибуты тегов (обработка событий)

```
<a href="delete.jsp" onclick="return confirm('Вы уверены?');">Удалить</a>
```

- Гипертекстовые ссылки

```
<a href="javascript:document.mainform.submit();">Отправить</a>
```

Тег <script>

- Указание языка
  - По умолчанию – именно JavaScript
  - В явном виде – с помощью атрибута
    - Устаревший вариант `language="JavaScript"`
    - Современный вариант `type="text/javascript"`
- Две формы кода
  - Код размещается внутри тега
  - Код размещается в отдельном файле (обычно с расширением `js`), файл указывается с помощью атрибута `src`
  - Второй способ «перекрывает» первый
- Типичное использование
  - В заголовке документа – для описания функций и предварительных действий
  - В теле документа – для условной генерации документа

Обработка браузером

- Обычно код интерпретируется
  - Возможны Just-In-Time решения
- Код выполняется по мере обнаружения
  - Обращаться к элементам документа, описанным в документе позднее – не лучшая мысль
  - Однако обращаться к позднее описанным сущностям из функций можно, если сущности уже будут существовать на момент выполнения функции
- Возможно возникновение ошибок в ходе выполнения
  - Блок скрипта не завершит работу
  - В консоли браузера появится сообщение об ошибке

Собственно, код

- JavaScript чувствителен к регистру!
  - HTML – нечувствителен к регистру (хотя есть рекомендации)
  - Соседство атрибутов HTML и событий/методов JavaScript может фрустрировать (например, `onClick` и `onclick`)

- Выполнение кода
  - Функции выполняются при явном вызове
  - Просто код в теге `<script>` выполняется по мере обнаружения
  - Код, указанный в атрибутах тегов как обработка событий, выполняется при возникновении событий
  - Код, указанный в ссылках, выполняется при срабатывании ссылки

Комментарии

```
<html>
  <body>
    <script>
      document.write("Это будет выведено.");
      // document.write("А это не будет выведено...");
      /*
      document.write("И это не будет выведено!");
      document.write("И это тоже не будет, надо же...");
      */
      <!-- document.write("И это не будет?!!");
    </script>
  </body>
</html>
```

Комментарии

- Однострочные
  - От символов `//` до конца строки
  - От символов `<!--` до конца строки

```
<script type="text/javascript">
<!--
document.write("Вы не увидите никаких следов кода,");
document.write("если ваш браузер не поддерживает JavaScript!!!");
//-->
</script>
```

- Многострочные
  - Начало `/*`
  - Конец `*/`
  - Не могут быть вложенными

Литералы и переменные

- Литералы
  - Числовые
    `10  010 0x10 1.1 .1 1. 1e1`
  - Строковые
    `"Строка" 'Строка' "I'm a string" '\'' "\""`
- Переменные
  - Могут объявляться с помощью ключевого слова **`var`**
    - `var a = 5, b = " лет";`
    - Если объявление вне функции – переменная глобальная, иначе – локальная
  - Могут объявляться и без ключевого слова **`var`**
    - `c = "И так тоже можно"`
    - Переменная всегда глобальная…
  - Объявление переменных «на лету» + их глобальность + чувствительность к регистру = отличный способ выстрелить себе в ногу
  - Тип переменной определяется её содержимым
  - Приведение типов автоматическое

Операторы

Приоритет	Оператор	Запись
1	Доступ к элементу	`. []`
	Создание объекта	`new`
2	Вызов функции	`()`
3	Инкремент (две формы)	`++`
	Декремент (две формы)	`--`
4	Логическое отрицание	`!`
	Побитовое отрицание	`~`
	Унарный плюс	`+`
	Унарный минус	`-`
	Тип объекта	`typeof`
	Вычисление выражения без возврата значения	`void`
	Удаление свойства объекта	`delete`

Операторы

Приоритет	Оператор	Запись
5	Умножение	`*`
	Деление	`/`
	Остаток от деления	`%`
6	Сложение, конкатенация	`+`
	Вычитание	`-`
7	Побитовые сдвиги	`<< >> >>>`
8	Сравнение	`< <= > >=`
	Проверка наличия свойства в объекте	`in`
	Проверка «типа» объекта	`instanceof`
9	Равенство и строгое (с типом) равенство	`== != === !===`

Операторы

Приоритет	Оператор	Запись
10	Побитовое И	`&`
11	Побитовое исключающее ИЛИ	`^`
12	Побитовое ИЛИ	`|`
13	Логическое И	`&&`
14	Логическое ИЛИ	`||`
15	Тернарный условный оператор	`?:`
16	Присваивания	`= += -= *= /= %= <<= >>= >>>= &= ^= |=`
17	Разделитель последовательности выражений	`,`

Управление порядком выполнения

- Операторные скобки
  - `{...}`
- Ветвление
  - `if (условие) инструкция;`
  - `if (условие) инструкция; else инструкция;`
  - Нелогические значения `0`, `""`, `null`, `undefined` и `NaN` считаются ложью, всё остальное – истиной

Управление порядком выполнения

- Цикл с предусловием
  - `while(условие) инструкция;`
- Цикл с постусловием
  - `do { ... } while(условие);`
- Итерационный цикл
  - `for(инициализация; условие; действие) инструкция;`
- Цикл по элементам
  - `for(переменная in объект) инструкция;`

Управление порядком выполнения

- Прерывание блока
  - `break;`
  - `break имяМетки;`
- Переход на следующую итерацию
  - `continue;`
  - `continue имяМетки;`
- Возврат из метода или обработчика
  - `return;`
  - `return значение;`

Управление порядком выполнения

- Блок переключателей
  - ```
    switch (выражение) {
      case значение: инструкции;
      ...
      default: инструкции}
    ```
  - Для сравнения используется строгое равенство
  - Можно работать со строками
  - Предложений `case` может быть много
  - Для прерывания используется `break`
  - Есть предложение `default`


    ```

Функции

- Объявление функции (function declaration)

```
function имя(парамр1, ... параметрN) {
  тело функции
}
```

- Формальные и фактические параметры различаются
- Внутри функции объявление переменных с `var` даёт локальные переменные
- Функции создаются предварительно, до выполнения кода

```
writeLine("Итого:");
writeLine(745);
function writeLine(line) {
  document.write(line);
  document.write("<br/>");
}
```

Функции – это объекты

- Функции – это объекты, у которых есть имя и значение, и которые можно передавать в функции

```
writeLine(writeLine);
/* Будет выведено следующее:
function writeLine(line) { document.write(line); document.write("<br/>"); }
*/
```

- Функция-выражение (function expression)

```
function(параметр1, ..., параметрN) {
  тело функции
}
```

  - Полученную функцию можно присвоить в переменную
  - Функции создаются не заранее, а когда до них доходит выполнение
  - Условное объявление функций (переменная объявляется заранее, реализация – потом)
  - Лучше такой формой не злоупотреблять

Функции – это объекты

- Можно вызывать функции, полученные в качестве объектов, и передавать им параметры

```
function showResult(a, b, calculator) {
  document.write("Операнд 1: " + a + "<br/>");
  document.write("Операнд 2: " + b + "<br/>");
  document.write("Результат: " + calculator(a, b) + "<br/>");
}
function sum(a, b) {
  return a + b;
}
showResult(5, 7, sum); //
showResult(8, 3, function (a, b) { return a - b; });
```

Объекты

- Являются по сути ассоциированными массивами (картами, map)
  - ключи – имена свойств (только строки)
  - значения – значения свойств (любые типы, **включая функции**)
- Доступ к свойствам
  - `ссылка.имяСвойства`
  - `ссылка["имяСвойства"]`
- Свойства добавляются и исключаются динамически
- Переменные хранят ссылки на объекты
- Свойства объектов являются переменными

Операции со свойствами

```
var employee = {}; // Создание пустого объекта

employee.name = "King"; // Добавление свойств
employee.salary = 5000;

document.write(employee.name + ": " + employee.salary); // Чтение свойств

delete employee.salary // Удаление свойства

for (p in employee) { // Ещё одно чтение свойств
  document.write(typeof(p) +  " " + p + " " + employee[p]);
}
/* Будет выведено
King: 5000
string name King
/*
```

Операции со свойствами

```
// Литеральное создание объекта
var rectangle = {
  width: 200,
  height: 300,
  getArea: function() { return this.width * this.height; }
}; // this в методах при обращении к свойствам обязателен!!!

// Изменение свойств
rectangle.width = rectangle.height = 5;
rectangle.getPerimeter = function() {
  return 2 * (this.width + this.height);
};

// Проверка работы
document.write(rectangle.getArea() + "<br/>");
document.write(rectangle.getPerimeter() + "<br/>");
```

Функции-конструкторы

- Конструкторами объектов являются функции
  - Их принято называть с большой буквы
  - Вызываются с помощью **new**
  - Могут иметь параметры
  - Формируют объект, используя слово **this**
  - Возвращают ссылку на созданный объект

```
function Employee(name, salary) {
  this.name = name;
  this.salary = salary;
  this.fired = false;
}
var king = new Employee("King", "5000");
```

Встроенные объекты

- Конструкторы
  - Math
  - Date
  - RegExp
  - Function
  - Array
  - ...
- «Обёртки»
  - String
  - Number
  - Boolean

Наследование

- Базовым является не наследование классов, а наследование объектов
- Родительский объект называется прототипом
- Механизмы наследования заметно отличаются от классического ООП с классами
- Можно эмулировать почти что классические классы со всеми их возможностями и проблемами

Обработка исключений

- Обработка исключений

```
try {
  // код, потенциально выбрасывающий исключения
}
catch (e) {
  // код обработки ошибки
}
finally {
  // код, всегда выполняющийся в конце
}
```

- Блока `catch` или `finally` может не быть
- Для непользовательских методов ошибка обычно «имеет тип» `Error`, свойства которого могут отличаться в зависимости от браузера

Выбрасывание исключений

- Метод вправе выбросить своё исключение
```
throw ссылкаНаОбъектИсключения;
```
- Можно выбросить вообще любой объект
```
throw 12345;
```
- Но лучше выбрасывать что-нибудь вразумительное
```
throw {name: "OhShit!", message: "Ohhhh"}
throw new Error("Oooops!");
```

Смена контекста

- Смена текущего объекта

```
var employee = {name: "King", salary: 5000};
with (employee) {
  document.write(name + ": " + salary);
}
```

- Смена контекста this у функции/метода

```
function raiseSalary(addition, factor) {
  this.salary = addition + this.salary * factor;
}
var petrov = {position: "Salesman", salary: 1000};
var president = {name: "King", salary: 5000};
raiseSalary.call(petrov, 100, 1.1);
raiseSalary.apply(president, [0, 1.2]);
```

Пользовательские массивы

- Одномерные
  - Но можно создать массив массивов, в т.ч. непрямоугольный
  - Обращение по индексу с помощью оператора `[]`
  - Нумерация элементов с 0
  - Есть поле `length`, хранящее количество элементов

- Способы создания
  - Пустой массив
    ```
    var a1 = new Array();
    ```
  - Массив с заданным количеством элементов
    ```
    var a2 = new Array(10);
    ```
  - Массив с заданными элементами
    ```
    var a3 = new Array(10, "и это не длина", 5.5, '!');
    ```
  - Литеральная форма с заданными элементами
    ```
    var a4 = [10, "и это не длина", 5.5, '!'];
    ```

Пользовательские массивы

- Динамические
  - Изменение значения `length`
  - Добавление новых элементов
  - Явно не указанные элементы получают значение `undefined`

```
<script>
  var a = [1, 2];
  a[5] = 5;
  document.write(a[4] + "<br/>");
  document.write(a[5] + "<br/>");
  a.length = 2;
  a.length = 5;
  document.write(a[4] + "<br/>");
  document.write(a[5] + "<br/>");
</script>
```

Сортировка пользовательских массивов

- Выполняется с помощью метода `sort()` массива
- По умолчанию – это сортировка в лексикографическом порядке
- Если требуется другой порядок, то следует задать свой критерий сравнения в виде функции

```
<script>
  var a = [1, 2, 15, 23];
  a.sort();
  document.write(a + "<br/>"); // 1,15,2,23
  function compareNumeric(a, b) {
    return a - b;
  }
  a.sort(compareNumeric);
  document.write(a + "<br/>"); // 1,2,15,23
</script>
```

Дополнительные методы пользовательских массивов

Инвертирование порядка элементов	`reverse()`
Добавление элемента в конец	`push(element)`
Удаление элемента из конца	`pop()`
Добавление элемента в начало	`unshift(element)`
Удаление элемента из начала	`shift()`
Объединение в строку с указанием разделителя	`join(separator)`
Удаление и вставка элементов	`splice(start, [deleteCount, add1, ..., addN])`
Копирование части массива	`slice(begin, end)`

Дополнительные методы пользовательских массивов

```
<html>
  <body>
    <script>
      var a = [1, 2, 15, 23];
      a.reverse();
      document.write(a + "<br/>"); // 23,15,2,1
      a.push("добавка 1");
      document.write(a + "<br/>"); // 23,15,2,1,добавка 1
      a.pop();
      document.write(a + "<br/>"); // 23,15,2,1
      a.unshift("добавка 2");
      document.write(a + "<br/>"); // добавка 2,23,15,2,1
      a.shift();
      document.write(a + "<br/>"); // 23,15,2,1
      //...
```

Дополнительные методы пользовательских массивов

```
      // ...
      var str = a.join("; ");
      document.write(str + "<br/>"); // 23; 15; 2; 1
      var b = str.split("; ");
      document.write(b + "<br/>"); // 23,15,2,1
      a.splice(2, 1, "Замена");
      document.write(a + "<br/>"); // 23,15,Замена,1
      var c = a.slice(1, a.length);
      c[0] = "И что будет?";
      document.write(a + "<br/>"); // 23,15,Замена,1
      document.write(c + "<br/>"); // И что будет?,Замена,1
    </script>
  </body>
</html>
```

Виды объектов

- Встроенные
  - По сути – библиотеки и базовые объекты
- Пользовательские
  - Всё, что создаёт пользователь-программист
- Серверные
  - Определяют и предоставляют взаимодействие с сервером
- Клиентские
  - Определяют и предоставляют взаимодействие с браузером и документом

Клоуны BOM и DOM

- Browser Object Model
  - Объектная модель для взаимодействия с браузером
  - В настоящий момент не стандартизована
  - Базовый объект – `window`
- Document Object Model
  - Объектная модель для взаимодействия с документом
  - В целом стандартизирована
  - Базовый объект – `document`

Объект window

- Глобальный объект
- Все объявляемые переменные и объекты становятся его свойствами
- Содержит либо напрямую информацию о документе, либо ассоциированный массив (карт, map) фреймов (frames)
- Имеет свои свойства, методы и события

Взаимодействие с пользователем

```
// Окно с сообщением
alert("Случилось страшное!!!");

// Окно с запросом на подтверждение
// Возвращает true или false
confirm("Хотите поговорить об этом?..");

// Окно со вводом строки
// Возвращает введённую строку
prompt("Вам слово!", "говорите сюда");
```

Создание и закрытие новых окон

- Создание
  - Параметры задаются строкой
  - Поведение зависит от браузера
  - Возвращается ссылка на объект окна

```
var ncwin = window.open('http://www.netcracker.com',
 'Сайт компании', 'directories=no,height=300,location=no,' +
 'menubar=no,resizable=yes,scrollbars=yes,' +
 'status=no,toolbar=no,width=300');
```

- Закрытие

```
ncwin.close();
```

Запуск новых потоков

- Запуск нового потока
  - `setTimeout("код", времяВМиллисекундах)`
  - Возвращает ссылку на поток
- Остановка запущенного разового потока
  - `clearTimeOut(ссылкаНаПоток)`
- Периодический запуск нового потока
  - `setInterval("код", времяВМиллисекундах)`
  - Возвращает ссылку на поток
- Остановка запущенного периодического потока
  - `clearInterval(ссылкаНаПоток)`

Свойства window

- `location` – объект «типа» URL, текущий адрес
  - `href` – собственно URL
  - Набор свойств, соответствующих фрагментам URL
  - Методы
  - `replace()`
  - `reload()`
- `history` – объект, хранящий историю переходов по страницам
  - `forward()`
  - `back()`
  - `go(количествоСтраниц)`
- `navigator` – объект, хранящий данные о браузере
- `opener` – ссылка на родительское окно

Объект document

- В нём и живёт DOM
- Это тоже ассоциированный массив специфического вида
- Все теги получают в соответствие объект
  - Имя объекта определяется значением атрибута `name` тега
  - Атрибуты тегов становятся свойствами соответствующих объектов
  - Не все свойства объектов соответствуют атрибутам тегов
- Объекты формируются по ходу чтения документа браузером
- Некоторые виды объектов объединяются в дополнительные массивы (формы, ссылки и т.д.)
- У объектов могут быть события
- У объектов могут быть дополнительные методы

Прямая запись в документ

- Методы
  - `document.write(значение)`
  - `document.writeln(значение)`
- Используются для генерирования (в т.ч. условного) содержимого документа
- Можно использовать только если документ ещё «открыт»
  - Для загружаемых документов – документ ещё не до конца прочитан браузером
  - Для документов в динамически открытых окнах – после вызова `document.open()` и до вызова `document.close()`
- Если документ уже закрыт, то его можно изменять только через объекты и управление ими

События объектов

- Обычно именуются `onчто-нибудь`
- Обработчик можно указать прямо в HTML-коде
- Обработчик можно задать программно
- Возврат из некоторых обработчиков null означает прекращение обработки

```
<html>
 <body>
  <p
onMouseOver="document.getElementById('addition').innerHTML = 'А он есть...'"
onMouseOut="document.getElementById('addition').innerHTML = ''">
    Видишь суслика?..</p>
  <p id="addition"></p>
  <img id="picture" src="http://turbo-suslik.ru/images/suslik.jpg"
       width="150px" height="150px"/>
  <script>
    document.getElementById("picture").onclick = function() {
      alert("Вот же он, суслик!")};
  </script>
 </body>
</html>
```

Динамическое изменение DOM

```
<html>
  <body>
    <table border="1px" id="table">
      <head id="head"> <td>Фамилия</td> <td>Имя</td> </head>
      <tr> <td>Иванов</td> <td>Иван</td> </tr>
    </table>
    <form name="mainform">
      Фамилия:
      <input type="text" name="sname"/><br/>
      Имя:
      <input type="text" name="fname"/><br/>
      <input type="submit" name="addition" value="Добавить"
         onClick="appendToTable(document.mainform.fname.value,
document.mainform.sname.value); return false;"/>
    </form>
```

Динамическое изменение DOM

```
    <script>
      function appendToTable(fname, sname) {
        var table = document.getElementById("table");
        var row = document.createElement("tr");
        var surname = document.createElement("td");
        var firstname = document.createElement("td");
        surname.innerHTML = sname;
        firstname.innerHTML = fname;
        row.appendChild(surname);
        row.appendChild(firstname);
        table.appendChild(row);
      }
    </script>
  </body>
</html>
```

Работа с формами

- Доступ к форме
  - По имени как к свойству документа
  - В массиве `document.forms`
- Доступ к элементам
  - По имени как к свойству формы
  - В массиве `elements` формы
- Атрибуты тегов элементов формы и самой формы – свойства соответствующих объектов
- События формы
  - `onsubmit`
  - `onreset`
- Методы формы
  - `submit()`
  - `reset()`

Программирование гиперссылок

- Доступ к гиперссылкам
  - По имени/идентификатору
  - В массиве `document.links[]`
  - Объект имеет «тип» URL
- Объект ссылки имеет свойства и события
- Ссылка может иметь программный вид
  - `javascript:код`
  - результат работы кода может быть показан в браузере
  - Результатом считается результат последнего выражения
  - Приём `void(0);`
  - Так можно не только в ссылке, но и в `action` у форм

Использование ссылок

```
<html>
  <body>
    <table border="1px" id="table">
      <head id="head"> <td>Фамилия</td> <td>Имя</td> </head>
      <tr> <td>Иванов</td> <td>Иван</td> </tr>
    </table>
    <form name="mainform">
      Фамилия: <input type="text" name="sname"/><br/>
      Имя: <input type="text" name="fname"/><br/>
    </form>
    <a href="javascript:appendToTable(document.mainform.fname.value,
                                      document.mainform.sname.value);
            void(0);">Добавить</a>
    <a href="javascript:document.mainform.submit();">
      Отправить данные</a>
```

Некоторые замечания

- Часто JavaScript воспринимается как набор приёмов по программированию клиентской части
- Это не так
- JavaScript – интересный и богатый язык с широкими возможностями
- Он просто недооценён сообществом
- И детальное его изучение требует значительного времени…

Что осталось за бортом

- Объектная модель, наследование, паттерны и т.д.
- Детали работы с DOM и BOM
- Свойства, методы и события объектов, соответствующих HTML-тегам
- Встроенные объекты и их возможности
- Регулярные выражения
- Работа с графикой
- Работа с анимацией
- Возможности в сочетании с CSS
- Работа с Cookie
- …

Спасибо за внимание!

Дополнительные источники

- Флэнаган, Д. JavaScript. Подробное руководство [Текст] / Дэвид Флэнаган. – СПб. : Символ-плюс, 2008. – 992 с.
- Крокфорд, Д. JavaScript. Сильные стороны [Текст] / Д. Крокфорд. – СПб. : Питер, 2013. – 176 с.
- JavaScript [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/JavaScript, дата доступа: 14.10.2013.
- JavaScript Tutorial [Электронный ресурс]. – Режим доступа: http://www.w3schools.com/js/, дата доступа: 14.10.2013.
- Современный учебник JavaScript [Электронный ресурс]. – Режим доступа: http://learn.javascript.ru/, дата доступа: 14.10.2013.
- JavaScript [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Servlets

Занятие 8

Гаврилов А.В.

Самара
2013

План занятия

- Понятие сервлета
- Элементы сервлетов
- HTTP-сервлеты

Добавление информации на сервер

- Как добавить статичные файлы на ваш HTTP-сервер?
- Просто поместить их в некоторый каталог
- Как добавить динамическое наполнение на ваш HTTP-сервер?
- Написать программу, обрабатывающую запросы GET и POST

Сервлеты

- Сервлет – это класс на Java, расширяющий возможности сервера, к которому происходят обращения в рамках модели «запрос-отклик»
- Гипотетически могут поддерживать любой тип запросов
- Чаще всего используются HTTP-сервлеты

Жизненный цикл сервлета

- Если экземпляр сервлета на существует, контейнер:
  - Загружает класс сервлета
  - Создает его экземпляр
  - Вызывает метод `init()`
- При получении запроса вызывается метод `service()`, которому передаются объекты запроса и отклика
- Если контейнеру требуется удалить объект сервлета, то вызывается метод `destroy()`

Особенности сервлетов

- Существуют механизмы реагирования на события, связанные с деятельностью сервлетов в приложении
  - `javax.servlet.`*`Событие`*`Listener`
  - `javax.servlet.`*`Событие`*`Event`
- Для обработки события требуется
  - написать класс, реализующий интерфейс
  - снабдить этот класс аннотацией `@WebListener` (пакет `javax.servlet.annotation`)

Особенности сервлетов

- Интерфейс `SingleThreadModel` в спецификации Servlet 2.4 объявлен как deprecated
- Требуется учитывать то, что сервлет является разделяемым ресурсом и работает с разделяемыми ресурсами
- Не следует использовать поля класса для хранения состояния, а особенно для передачи состояния между вызовами

Объект запроса

- Реализует интерфейс `javax.servlet.ServletRequest`
- Позволяет узнать характеристики запроса, в том числе значения параметров
  - `String getParameter(String name)`
  - `Enumeration getParameterNames()`
  - и ряд других методов
- Позволяет получить доступ к потоку (stream) запроса
  - `ServletInputStream getInputStream()`
  - `BufferedReader getReader()`
  - Можно вызвать только один из этих двух методов

Объект запроса

- Позволяет получить параметры запроса
  - `int getContentLength()`
  - `String getContentType()`
  - `String getLocalAddr()`
  - `String getProtocol()`
  - `String getRemoteAddr()`
  - `boolean isSecure()`

Объект отклика

- Реализует интерфейс `javax.servlet.ServletResponse`
- Основные методы:
  - `ServletOutputStream getOutputStream()`

    Далее для вывода используются обычные средства бинарного вывода
  - `PrintWriter getWriter()`

    Далее для вывода используются обычные средства символьного вывода
  - Можно вызвать только один из этих двух методов

Объект отклика

- Позволяет установить параметры отклика
  - Для установки типа отклика используется метод `setContentType()`
    - Список типов: http://www.iana.org/assignments/media-types/index.html
  - Для установки кодировки используется метод `setCharacterEncoding()`
  - Сбрасывает текущий буфер клиенту `flushBuffer()`
  - Очищает буфер и сбрасывает статус `reset()`

HTTP Servlet

- `javax.servlet.http.HttpServlet`
  Абстрактный класс, позволяющий создавать сервлеты, удобные для Web
- Наследные классы должны переопределять хотя бы один из методов обработки запроса
- Обычно это один из методов
  - ```
    void doGet(HttpServletRequest req,
                    HttpServletResponse resp)
    ```
  - ```
    void doPost(HttpServletRequest req,
                    HttpServletResponse resp)
    ```
- Классы снабжаются аннотацией `@WebServlet`

Пример класса сервлета

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet("/MyServlet.html")
public class MyServlet extends HttpServlet {
  protected void doGet(HttpServletRequest request,
                       HttpServletResponse response) throws IOException {
    PrintWriter out = response.getWriter();
    try {
      out.print("<html><body>This is text document, which has "
                + "<a href=\"other.html\">link</a>"
                + " to another document</body></html>");
    }
    finally {
      out.close();
    }
  }
}
```

Нетекстовый сервлет

```
@WebServlet("/image.png")
public class ImageServlet extends HTTPServlet {

...
    response.setContentType("image/png");
    OutputStream out = response.getOutputStream();
    try {
      BufferedImage im = new BufferedImage(640, 480,
                                BufferedImage.TYPE_INT_RGB);
      ...
      ImageIO.write(im, "png", out);
    } finally {
      out.close();
    }
...
```

Использование других web-ресурсов

- Включение ресурса

```
RequestDispatcher dispatcher =
    getServletContext().getRequestDispatcher("/banner");
if (dispatcher != null)
  dispatcher.include(request, response);
```

- Передача управления

```
RequestDispatcher dispatcher = request.
    getRequestDispatcher("/template.jsp");
if (dispatcher != null)
  dispatcher.forward(request, response);
```

Поддержка сессий

- В HTTP-сервлетах в объектах запроса и отклика добавляются дополнительные методы
- Среди них появляются методы работы с сессиями, включая работу с Cookies
- Для передачи значений между запросами можно использовать атрибуты сессии
  - `HttpSession.getAttribute(String name)`
  - `HttpSession.setAttribute(String name, Object value)`
  - И ряд других методов

Поддержка сессий

```
public class CashierServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                     HttpServletResponse response)
                     throws ServletException, IOException {
    // Get the user's session and shopping cart
    HttpSession session = request.getSession();
    ShoppingCart cart =
      (ShoppingCart)session.getAttribute("cart");
    //...
  }
}
```

```
out.println("<a href=\"" +
response.encodeURL(request.getContextPath() +
"/bookcatalog") + "\">");
```

Фильтрация

- Получаемые запросы и формируемые отклики могут быть подвергнуты фильтрации
- API для классов фильтрации определен интерфейсами **`Filter`**, **`FilterConfig`**, **`FilterChain`** пакета **`javax.servlet`**
- Включение фильтров осуществляется с помощью аннотации **`@WebFilter`**

Общий дескриптор развёртывания web.xml

- Определяет параметры развёртывания web-приложения в контейнере
- Некоторые из этих параметров даже не могут быть указаны с помощью аннотаций классов, т.к. относятся к web-приложению в целом
- Указанные в дескрипторе параметры перекрывают параметры, указанные с помощью аннотаций классов

Пример дескриптора развертывания web.xml

```
<?xml version="1.0" encoding="UTF-8"?> <web-app version="3.0"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <listener>
        <listener-class>org.tempuri.MyListner</listener-class>
    </listener>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>org.tempuri.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyServlet.html</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>/MyServlet.html</welcome-file>
    </welcome-file-list>
</web-app>
```

Спасибо за внимание!

Дополнительные источники

- Дейтел, Х.М. Технологии программирования на Java 2. Книга 3. Корпоративные системы, сервлеты, JSP, Web-сервисы [Текст] / Х.М. Дейтел, П.Дж. Дейтел, С.И. Сантари. – М. : Бином-пресс, 2003. – 672 с.
- Перри, Б.У. Java сервлеты и JSP. Сборник рецептов [Текст] / Брюс У. Перри. – М. : Кудиц-пресс, 2009. – 768 с.
- Соломон, М.К. Oracle. Программирование на языке Java [Текст] / Мартин К. Соломон, Нирва Морисco-Леруа, Джули Басу. – М. : Лори, 2010. – 512 с.
- Курванян, Б. Программирование web-приложений на языке Java [Текст] / Буди Курванян. – М. : Лори, 2009. – 880 с.
- JavaEE APIs & Docs [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/apis-139520.html, дата доступа: 21.10.2011.
- JavaEE Tutorials [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html, дата доступа: 21.10.2011.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Java Server Pages

Занятие 9

Гаврилов А.В.

Самара
2013

План занятия

- Задачи и принципы JSP
- Структура и элементы JSP-страниц
- Объекты и директивы
- Взаимодействие с HTML-формами
- Недостатки раннего JSP
- Expression Language
- Custom tags
- JSTL

Java Server Pages

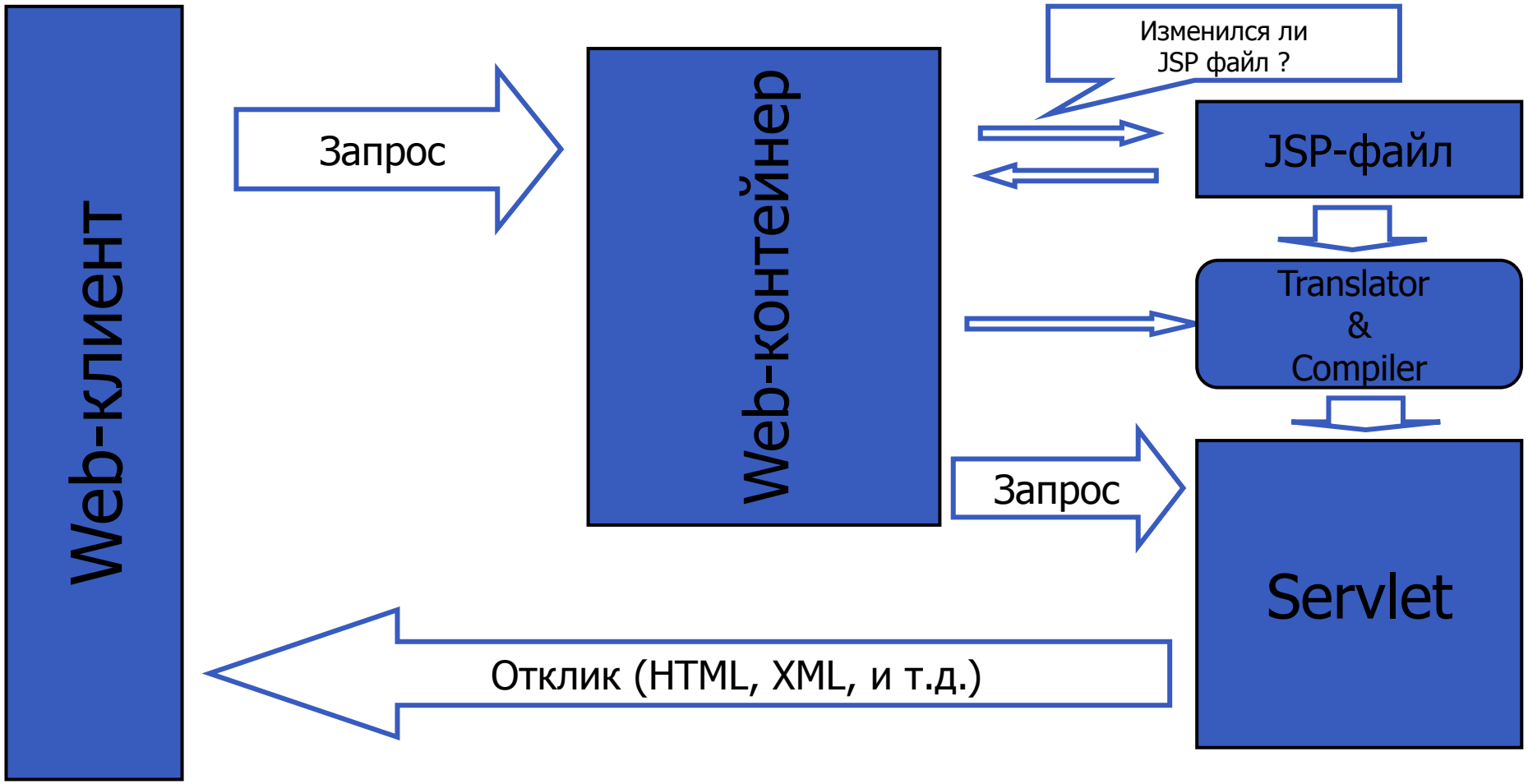
- JSP – это технология, которая упрощает создание web страниц (и не только) с динамически изменяющимся (во время генерации страницы) содержимым

```
<html>
<body>
<p>HI<%=
    (request.getParameter("name")!=null)?
    ", " + request.getParameter("name"):""
%>!!!</p>
</body>
</html>
```

JSP-файл

- JSP файл – это текстовый документ, содержимое которого можно разбить на две части
  - статический текст (каркас) – создается при помощи специальных редакторов
  - динамический (генерируемый во время выполнения страницы) – обычно пишется в среде разработки программирования

Принцип работы



Трансляция JSP-файла

- JSP-файл транслируется в файл java
- Процесс трансляции управляется директивами (`<%@ page [...] %>`)
- Полученный файл компилируется в класс сервлета
- Далее с ним идет работа как с сервлетом
- Конфигурация сервера может включать ряд настроек по работе с jsp-страницами

Обработка ошибок

- Во время трансляции и компиляции
  - поведение зависит от используемого сервера
  - класс сервлета создан не будет
- Во время выполнения:
  - если для данной JSP страницы (или приложения) определена «страница на случай ошибки» (errorPage), то будет выведена указанная страница
  - иначе – в зависимости от сервера и его настроек

Пример сообщения об ошибке в процессе развертывания

Виды JSP

- **JSP-страница**
  - Это страница из статических и динамических элементов
  - Статическая часть имеет произвольный формат

- **JSP-документ**
  - XML-файл, содержащий тэги особого вида
  - Имеет ту же функциональность, что и JSP-страница, но требует представления в виде тэгов ряда элементов

Создание страницы

- Создание статического содержимого. Для этого удобно использовать редакторы HTML, XML и пр.
- Вставка динамических объектов и элементов сценариев JSP. В созданный файл вставляются элементы <%? [...] %>

Элементы JSP-страницы

- Статический текст
- Элементы сценариев
  - Комментарии `<%-- […] --%>`
  - Объявления `<%! […] %>`
  - Скриптлеты `<% […] %>`
  - Выражения `<%= […] %>`
  - Директивы `<%@ ? […]%>`
- Стандартные объекты

Комментарии

- `<%-- […] --%>`
- Содержат текст, который не будет включен в отклик страницы
- Содержимое комментария игнорируется транслятором
- Не путайте с комментариями HTML `<!-- […] -->`

JSP-страница с комментариями

- JSP-страница

```
<html><body>
  Hello World!
  <%-- This is a JSP comment --%>
  <!-- This is a HTML comment -->
</body></html>
```

- Фрагмент примерного кода сервлета

```
out.write("<html><body>\n  Hello World!\n  ");
out.write("\n  <!-- This is a HTML comment -->"
          + "\n</body></html>\n");
```

Объявления

- <%! […] %>
- Обычно содержат объявления переменных, методов, классов и интерфейсов
- При трансляции преобразуются в элементы класса сервлета (поля, методы и вложенные типы)
- Нельзя использовать поля для хранения значений, использующихся в рамках одного запроса
- При работе с внутренними классами есть особенности

JSP-страница с объявлениями

- JSP-страница

```
<%!
    private String str = "Message: ";
    public String createMessage(String arg) {
        return str + arg;
    }
%>
```

- Фрагмент примерного кода сервлета

```
public final class index_jsp
  extends org.apache.jasper.runtime.HttpJspBase
  implements org.apache.jasper.runtime.JspSourceDependent {
    private String str = "Message: ";
    public String createMessage(String arg) {
        return str + arg;
    }
```

Скриптлеты

- `<% [...] %>`
- Содержат динамически выполняемый код
- Преобразуются во фрагменты методов сервлета, формирующих отклик
- Синтаксически являются фрагментами обычных Java-методов
- Каждая завершенная инструкция заканчивается знаком точки с запятой

JSP-страница со скриптлетами

- JSP-страница

```
<html><body>
<%  String server_name = System.getProperty("os.name");
    if (server_name == null)
      server_name = "Undefined.";  %>
</body></html>
```

- Фрагмент примерного кода сервлета

```
out.write("<html><body>\n");
String server_name = System.getProperty("os.name");
if (server_name == null)
  server_name = "Undefined.";
out.write("\n");
out.write("</body></html>\n");
```

Выражения

- `<%= [...] %>`
- Содержит выражения, значения которых будут выведены в отклик
- Синтаксически являются вычислимыми выражениями Java, для которых может быть получено текстовое представление
- В конце выражений точки с запятой нет!

JSP-страница с выражениями

- JSP-страница

```
<html><body>
OS Name: <%=server_name%>
<br><%=createMessage("It works!")%>
</body></html>
```

- Фрагмент примерного кода сервлета

```
out.write("<html><body>\n");
out.write("OS Name: ");
out.print(server_name);
out.write("\n");
out.write("<br>");
out.print(createMessage("It works!"));
out.write("\n");
out.write("</body></html>\n");
```

Директива page

- `<%@ page […] %>`
- Позволяет управлять процессом трансляции JSP-страницы в сервлет
- Часто используемые параметры
  - Тип отклика и кодировка
  - Автоматическая инициализация сессии
  - Импортируемые типы
  - Страница для обработки ошибок
    - Является ли данная страница обработчиком
    - Какая страница используется как обработчик

Тип отклика, кодировка и импорт типов

- JSP-страница

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@page session="true"%>
<%@page import="java.util.*, java.lang.reflect.*" %>
```

- Фрагмент примерного кода сервлета

```
import java.util.*;
import java.lang.reflect.*;

...
  response.setContentType("text/html; charset=UTF-8");
  session = pageContext.getSession();
...
```

Возникновение и обработка ошибок (JSP-страницы)

- Основная страница

```
<%@page contentType="text/html; charset=UTF-8"%>
<%@page errorPage="errorMessage.jsp"%>
<html><body>
<% if(true) throw new Exception("Something happened!");%>
</body></html>
```

- Страница-обработчик

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page isErrorPage="true" %>
<html><body>
An error has occurred! Please, contact administrator.
<%someLogMethod(exception);%>
</body></html>
```

Возникновение и обработка ошибок (примерный код сервлетов)

- Основная страница

```
try {
  pageContext = _jspxFactory.getPageContext(this, request,
    response, "errorMessage.jsp", true, 8192, true);
  if(true) throw new Exception("Something happened!");
} catch (Throwable t) {
 _jspx_page_context.handlePageException(t);
}
```

- Страница-обработчик

```
Throwable exception =
  org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);
...
out.write("<html><body>\n");
out.write("An error has occurred! Please, contact administrator.\n");
someLogMethod(exception);
out.write("</body></html>\n");
```

Директива include

- Предназначена для включения в формируемый отклик содержимого другого файла
- Включение статическое: происходит на этапе трансляции
- Включаться могут как статические, так и динамические ресурсы
- Существует специальный вид JSP-страниц – JSP Fragments (JSPF)

Директива include (JSP-страницы)

- Основная страница

```
<%@page contentType="text/html; charset=UTF-8"%>
<html><body>
<%@include file="WEB-INF/jspf/fragment.jspf" %>
</body></html>
```

- Включаемая страница

```
<%@ page pageEncoding="UTF-8" %>
Some fragment text
```

Директива include (примерный код сервлетов)

- Основная страница

```
public final class index_jsp ... {

  private static java.util.Vector _jspx_dependants;
  static {
    _jspx_dependants = new java.util.Vector(1);
    _jspx_dependants.add("/WEB-INF/jspf/fragment.jspf");
  }

  public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
    throws java.io.IOException, ServletException {
    ...
    out.write("<html><body>\n");
    out.write("Some fragment text\n");
    out.write("</body></html>\n");
    ...
}
```

Стандартные объекты

- **`page`**
  Ссылка на текущий объект (по сути – объект сервлета)
- **`config`**
  Имеет тип `javax.servlet.ServletConfig`, позволяет получить контекст сервлета и параметры сервлета
- **`application`**
  Имеет тип `javax.servlet.ServletContext`, определяет контекст сервлета
- **`pageContext`**
  Имеет тип `javax.servlet.jsp.PageContext`, определяет контекст jsp-страницы

Стандартные объекты

- **`session`**
  Имеет тип **`javax.servlet.http.HttpSession`**, представляет текущую сессию
- **`request`**
  Имеет тип **`javax.servlet.http.HttpServletRequest`**, представляет текущий запрос
- **`response`**
  Имеет тип **`javax.servlet.http.HttpServletResponse`**, представляет текущий отклик
- **`out`**
  Имеет тип **`javax.servlet.jsp.JspWriter`**, представляет поток для вывода, по функциональности схож с классом **`PrintWriter`**

Стандартные объекты
(примерный код сервлета)

```
PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
JspWriter out = null;
Object page = this;

...

pageContext = _jspxFactory.getPageContext(this, request,
                response, null, true, 8192, true);
application = pageContext.getServletContext();
config = pageContext.getServletConfig();
session = pageContext.getSession();
out = pageContext.getOut();
```

Тэги <jsp:…>

- Являются еще одним средством управления ходом трансляции страницы
- В основном предназначены для формата JSP-документов
- Позволяют:
  - Описывать теги и их составляющие
  - Перенаправлять обработку запросов
  - Работать с компонентами JavaBeans
  - Включать в код страницы обращения к апплетам

Тэги <jsp:include>

- `<jsp:include page="inclPage">`
- Включает в страницу статический или динамический ресурс
- Включение происходит динамически, во время выполнения

```
<jsp:include page="{relativeURL | <%= expression %>}"
      flush="true | false" >
   <jsp:param name="parameterName"
       value="{parameterValue|<%=expr%>}|${…}” />
</jsp:include>
```

Тэги <jsp:include>

- JSP-страница

```
<html><body>
Before
<jsp:include page="simple.html"></jsp:include>
After
</body></html>
```

- Примерный код сервлета

```
out.write("<html><body>\n");
out.write("Before\n");
org.apache.jasper.runtime.JspRuntimeLibrary.include(request,
  response, "simple.html", out, false);
out.write("\n");
out.write("After\n");
out.write("</body></html>\n");
```

Тэги <jsp:forward>

- `<jsp:forward page="inclPage">`
- Передает обработку запроса другому ресурсу
- Могут быть указаны дополнительные параметры

```
<jsp:forward page="{relativeURL | <%= expression %>}">
   <jsp:param name="parameterName"
       value="{parameterValue|<%=expr%>}|%{…}" />
</jsp:forward>
```

Тэги <jsp:forward>

- JSP-страница

```
<html><body>
Before
<jsp:forward page="simple.html"></jsp:forward>>
After
</body></html>
```

- Примерный код сервлета

```
out.write("<html><body>\n");
out.write("Before\n");
if (true) {
  _jspx_page_context.forward("simple.html");
  return;
}
out.write("\n");
out.write("After\n");
out.write("</body></html>\n");
```

Взаимодействие с HTML-формами

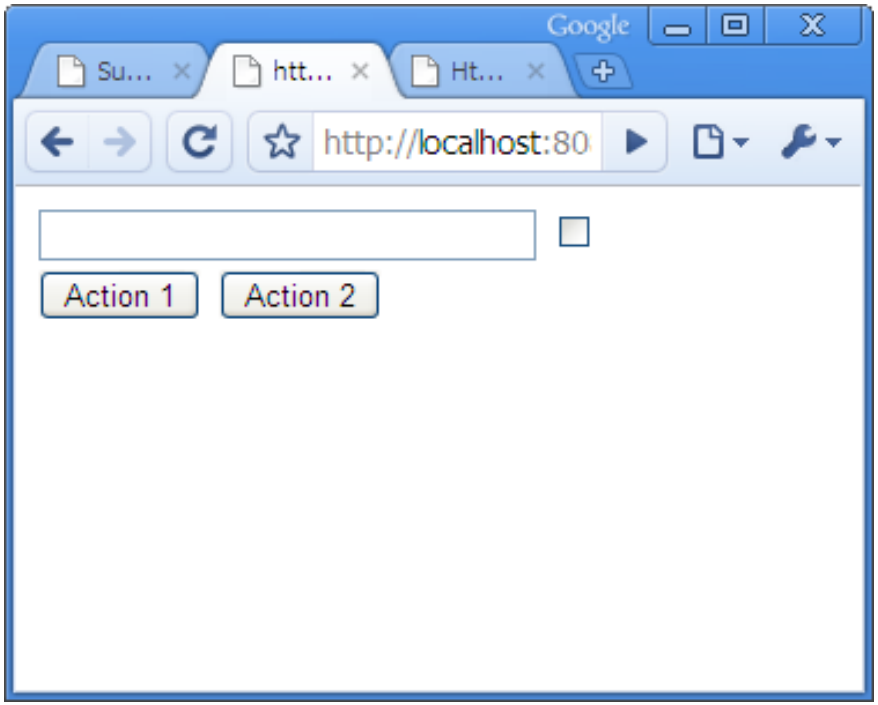
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
    <form name="SimpleForm" action="index.jsp">
        <input type="text" name="NameEditor" value="" size="30" />
        <input type="checkbox" name="SomeCheckBox" value="ON" />
        <br>
        <input type="submit" value="Action 1" name="Action1Button" />
        <input type="submit" value="Action 2" name="Action2Button" />
    </form>
    <%  Enumeration names = request.getParameterNames();
        if (names.hasMoreElements()) {
            String name = null;
            String value = null;%>
```

Взаимодействие с HTML-формами

```
<table border="1">
        <thead>
            <tr>
              <th>Parameter name</th>
              <th>Parameter value</th>
            </tr>
        </thead>
        <tbody>
            <% while (names.hasMoreElements()) {
            name = (String) names.nextElement();
            value = request.getParameter(name);%>
              <tr>
                <td><%=name%></td>
                <td><%=value%></td>
              </tr>
            <% }%>
        </tbody>
    </table>
    <% } %>
</html>
```

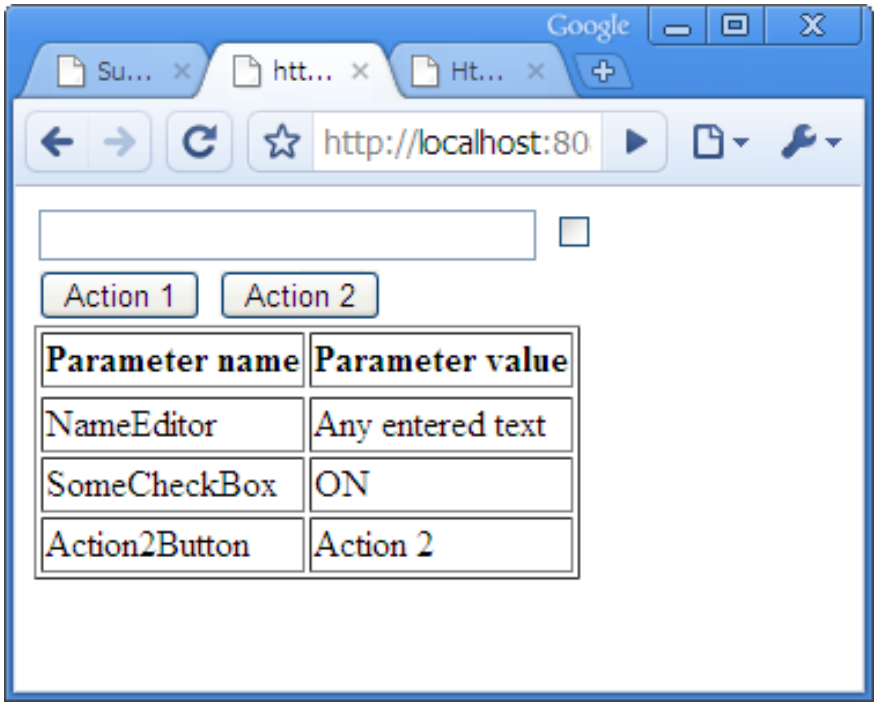
Форма при первом запуске



- Адрес:

http://host:8080/WebApplication1/index.jsp

Форма после нажатия кнопки



- Адрес:
  http://localhost:8080/WebApplication1/index.jsp?NameEditor=Any+entered+text&SomeCheckBox=ON&Action2Button=Action+2

Взаимодействие с HTML-формами

- Элементы HTML-управления должны находиться в форме
- Для формы указывается адрес для перехода
- Значения элементов ввода записываются в параметры запроса
- В новой странице значения элементов ввода автоматически не заполняются
- Элемент управления, активация которого привела к смене страницы, вносится в параметры запроса

Что было не так?..

- Смешение Java-кода и тегов страницы
- Много «типовых» действий
- «Некомфортность» работы на Java в ходе обработки запросов
  - Громоздкость кода
  - Необходимость знания особенностей преобразования в сервлет

Смена парадигмы

- **Было**

JSP нужно для того, чтобы Java-программистам было удобнее писать под Web

- **Стало**

JSP-программист может и не знать Java, и даже лучше, если он ее знать не будет

Основные идеи

- Альтернативный язык
  - Expression Language
- Использование библиотек тегов
  - особенно JSTL – JSP Standard Tag Library
- Ориентация на компоненты в смысле JavaBeans и использование понятия свойств
- Общие принципы остаются прежними

Как мы писали раньше

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title></head> <body>
 <h1>Hello page</h1><br/>
 <% if (request.getParameter("name") != null) {
      out.print("<b>Hello, ");
      out.print(request.getParameter("name"));
      out.println("!</b><br/>");
    } %>
<form name="oldform" action="old.jsp">
  What's your name?<br>
  <input type="text" name="name" value="" size="50" />
  <input type="submit" value="Tell name" name="tell" />
</form> </body> </html>
```

Как мы пишем теперь

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html> <head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title></head> <body>
  <h1>Hello page</h1><br/>
  <c:if test="${!empty param.name}" var="val" scope="request">
    <b>Hello, ${param.name}!</b><br/>
  </c:if>
  <form name="oldform" action="old.jsp">
    What's your name?<br>
    <input type="text" name="name" value="" size="50" />
    <input type="submit" value="Tell name" name="tell" />
  </form>
</body></html>
```

Expression Language

- Является альтернативой выражениям и скриптлетам
- Динамическое чтение данных из…
  - JavaBeans-компонентов
  - Неявных объектов
  - Различных структур данных
- Динамическая запись данных в…
  - Формы
  - JavaBeans-компоненты
- Вызов статических и публичных методов
- Динамическое выполнение простых операций

Виды выражений EL

- По моменту вычисления
  - Немедленное вычисление
  - Отложенное вычисление
- По цели выражения
  - Обращение к данным
  - Вызов метода
- По режиму доступа
  - Только чтение
  - Чтение и запись данных

Немедленное и отложенное вычисление

- Немедленное вычисление (JSP)
  - `${ выражение }`
  - Позволяет лишь получить значение выражения

- Отложенное вычисление (JSF)
  - `#{ выражение }`
  - Выражение может использоваться различным образом в разные моменты жизненного цикла
  - Позволяет не только получать значения, но и записывать их (в т.ч. для свойств бинов)

Применение выражений с немедленным вычислением

- Выражения EL могут применяться:
  - в статическом тексте
  - в качестве значений атрибутов тегов

```
<some:tag>
     some text ${expr} some text
</some:tag>

<some:tag value="${expr}"/>

<some:tag value="some${expr}${expr}text${expr}"/>
```

Обращение к данным

- Компоненты JavaBeans
  - ${beanName.propertyName}
  - ${beanName["propertyName"]}
- Коллекции
  - ${beanName.listName[i]}
  - ${beanName.mapName["string key"]}
- Перечислимые типы JavaSE
  - ${reference == "enumConstant"}
- Неявные объекты
  - ${param["paramName"]}
- Вычисление выражений
  - ${customer.age + 20}

Вызов методов

- Без параметров
  - `${object.method}`
  - `${object["method"]}`
- С параметрами
  - `${object.method(param1, ..., paramN}`
  - `${object["method"](param1, ..., paramN)}`

Литералы в выражениях

- Логические
  - `true`
  - `false`
- Целочисленные
  - `136`
- С плавающей точкой
  - `3.14`
  - `1.0e10`
- Строковые
  - `"string"`
  - `'literal'`
- Ссылочные
  - `null`

Операторы в выражениях

- Доступ к элементам
- Операторные скобки
- Унарные
- Арифметические
- Сравнение
- Логические

- `[]  .`
- `()`
- `-  not  !  Empty`
- `*  /  div  %  mod`
- `+  -`
- `<  >  <=  >=`
  `lt  gt  le  ge`
- `==  !=  eq  ne`
- `&& and`
- `||  or`
- `? :`

Неявные объекты

- Объекты доступа к другим объектам
  - `pageScope`
  - `requestScope`
  - `sessionScope`
  - `applicationScope`
- Часто используемые объекты
  - `pageContext`
  - `session`
  - `request`
  - `response`
  - `param`
  - `cookie`
  - `...`

Примеры выражений EL

Выражение	Результат
`${(10*10) ne 100}`	false
`${'a' < 'b'}`	true
`${1.2E4 + 1.4}`	12001.4
`${!empty param.Add}`	true, если параметр Add не null и не пустая строка
`${pageContext.request.contextPath}`	Путь контекста
`${param['mycom.productId']}`	Значение параметра mycom.productId
`${departments[deptName]}`	Значение элемента deptName в departments
`${header["host"]}`	Имя хоста

JSP-страница с применением EL

- JSP-страница

```
${20 + 335}<br/>
${!empty param.Add}<br/>
```

- Фрагмент примерного кода сервлета

```
out.write((java.lang.String)
  org.apache.jasper.runtime.PageContextImpl.evaluateExpression(
    "${20 + 335}", java.lang.String.class,
    (PageContext)_jspx_page_context, null));
out.write("<br/>\n");
out.write((java.lang.String)
  org.apache.jasper.runtime.PageContextImpl.evaluateExpression(
    "${!empty param.Add}", java.lang.String.class,
    (PageContext)_jspx_page_context, null));
out.write("<br/>\n");
```

Custom tags

- Определяются пользователем и описывают часто используемые операции
- Описываются и распространяются в виде библиотек тегов, определяющих семейства тегов и содержащих объекты, реализующие теги
- Для использования тегов необходимо:
  - Объявить библиотеку тегов
  - Сделать ее доступной для web-приложения

```
<prefix:tag attr1="value" ... attrN="value" />
<prefix:tag attr1="value" ... attrN="value" >
  body
</prefix:tag>
```

Библиотеки тегов

- Библиотеки тегов описываются в tld-файлах
- Библиотеки могут быть
  - публичными и храниться в сети
  - локальными и храниться в каталоге WEB-INF
- Для подключения библиотеки необходимо использовать директиву **taglib**

```
<%@ taglib prefix="tlt" uri="/WEB-INF/iterator.tld"%>

<%@ taglib prefix="c"
uri="http://java.sun.com/jsp/jstl/core"%>
```

Обработка custom tags

- В процессе компиляции jsp-страницы теги будут преобразовываться в java-код в соответствии с описанием в библиотеке тегов

- Если соответствующая библиотека не была подключена, относящиеся к ней теги будут трактоваться как статическая часть jsp-страницы и просто выводиться в отклик

JSP-страница с применением custom tags

- JSP-страница

```
<c:if test="${!empty param.name}" var="nameExists" scope="request">
    <b>Hello, ${param.name}!</b><br/>
</c:if>
```

- Фрагмент примерного кода сервлета

```
org.apache.taglibs.standard.tag.rt.core.IfTag _jspx_th_c_if_0 =
  (org.apache.taglibs.standard.tag.rt.core.IfTag)
  _jspx_tagPool_c_if_var_test_scope.get(
    org.apache.taglibs.standard.tag.rt.core.IfTag.class);
_jspx_th_c_if_0.setPageContext(_jspx_page_context);
_jspx_th_c_if_0.setParent(null);
_jspx_th_c_if_0.setTest(((java.lang.Boolean)
  org.apache.jasper.runtime.PageContextImpl.evaluateExpression(
    "${!empty param.name}", java.lang.Boolean.class,
    (PageContext)_jspx_page_context, null)).booleanValue());
```

JSP-страница с применением custom tags

- Фрагмент примерного кода сервлета

```
_jspx_th_c_if_0.setVar("nameExists");
_jspx_th_c_if_0.setScope("request");
int _jspx_eval_c_if_0 = _jspx_th_c_if_0.doStartTag();
if (_jspx_eval_c_if_0 != javax.servlet.jsp.tagext.Tag.SKIP_BODY) {
  do {
    out.write("<b>Hello, ");
    out.write((java.lang.String)
      org.apache.jasper.runtime.PageContextImpl.evaluateExpression(
      "${param.name}", java.lang.String.class,
      (PageContext)_jspx_page_context, null));
    out.write("!</b><br/>\n");
    int evalDoAfterBody = _jspx_th_c_if_0.doAfterBody();
    if (evalDoAfterBody !=
      javax.servlet.jsp.tagext.BodyTag.EVAL_BODY_AGAIN)
      break;
  } while (true);
}
```

JSP Standard Tag Library

- Библиотека включает в себя набор тегов, функциональность которых часто используется в web-приложениях
- Эти теги должны обрабатываться единообразно в рамках всех контейнеров
- Это позволяет:
  - избежать использования различных библиотек
  - единообразно организовывать JSP-страницы

Библиотеки JSTL

- JSTL поставляется в виде набора библиотек:
  - Различные направления функциональности
  - Различные пространства имен
- Библиотеки и соответствующие пути
  - Core *http://java.sun.com/jsp/jstl/core*
  - XML *http://java.sun.com/jsp/jstl/xml*
  - Internationalization *http://java.sun.com/jsp/jstl/fmt*
  - Database *http://java.sun.com/jsp/jstl/sql*
  - Functions *http://java.sun.com/jsp/jstl/functions*

Библиотеки JSTL

Библиотека	Функциональность	Префикс
Core	Работа с переменными	c
	Управление ходом выполнения	
	Работа с URL	
	Разное	
XML	Базовые	x
	Управление выполнением	
	Преобразования	
I18n	Локализация	fmt
	Форматирование сообщений	
	Форматирование чисел и дат	
Database	SQL	sql
Functions	Возможности коллекций	fn
	Работа со строками	

Библиотека Core

Функциональность	Тэги
Работа с переменными	`remove` `set`

Управление ходом выполнения	`choose (when, otherwise)`
	`forEach`
	`forTokens`
	`if`

Работа с URL	`import (param)`
	`redirect (param)`
	`url (param)`

Разное	`catch` `out`

Библиотека XML

Функциональность	Тэги

Базовая	`out`
	`parse`
	`set`

Управление ходом выполнения	`choose (when, otherwise)`
	`forEach`
	`if`

Преобразования	`transform (param)`

Пример использования

```
<c:if test="${!empty param.Add}">
  <c:set var="bid" value="${param.Add}"/>
  <jsp:useBean id="bid" type="java.lang.String" />
  <sql:query var="books"
       dataSource="${applicationScope.bookDS}">
    select * from PUBLIC.books where id = ?
    <sql:param value="${bid}" />
  </sql:query>
  <c:forEach var="bookRow" begin="0" items="${books.rows}">
    <jsp:useBean id="bookRow" type="java.util.Map" />
    <jsp:useBean id="addedBook" class="database.Book"
         scope="page" />
    ...
    <% cart.add(bid, addedBook); %>
  ...
</c:if>
```

Спасибо за внимание!

Дополнительные источники

- Дейтел, Х.М. Технологии программирования на Java 2. Книга 3. Корпоративные системы, сервлеты, JSP, Web-сервисы [Текст] / Х.М. Дейтел, П.Дж. Дейтел, С.И. Сантари. – М. : Бином-пресс, 2003. – 672 с.
- Перри, Б.У. Java сервлеты и JSP. Сборник рецептов [Текст] / Брюс У. Перри. – М. : Кудиц-пресс, 2009. – 768 с.
- Соломон, М.К. Oracle. Программирование на языке Java [Текст] / Мартин К. Соломон, Нирва Морисco-Леруа, Джули Басу. – М. : Лори, 2010. – 512 с.
- Курванян, Б. Программирование web-приложений на языке Java [Текст] / Буди Курванян. – М. : Лори, 2009. – 880 с.
- JavaEE APIs & Docs [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/apis-139520.html, дата доступа: 14.10.2013.
- JavaEE Tutorials [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Java Server Faces

Занятие 10

Гаврилов А.В.

Самара
2013

План занятия

- Принципы JSF
- Модель данных
- Порядок обработки запросов
- Элементы JSF

Недостатки JSP

- Возможности «чистого» HTML недостаточны
  - Необходимо использование других языков и технологий
- Необходимость обработки параметров запросов
- Необходимость перехода к модели данных на стороне сервера

Технология Java Server Faces

- Фреймворк для Web-приложений на Java, предназначенный для описания пользовательского интерфейса с помощью компонентов, находящихся на сервере
- Основные элементы технологии:
  - API для представления UI-компонентов и управления их состоянием; управления событиями, преобразования данных; определения навигации по страницам; и т.д.
  - Библиотеки тегов для описания UI-компонентов на jsf-страницах

Возможности Java Server Faces

- Создание пользовательского интерфейса на основе компонентов
- Добавление компонентов на страницу простым указанием тегов
- Привязка генерируемых компонентами событий к коду на стороне сервера
- Привязка состояния компонентов к данным на сервере
- Сохранение и восстановление состояния компонентов пользовательского интерфейса вне процесса обработки запроса

Как мы писали раньше
Класс JavaBean

```
package calc;
public class Calculator {
  private double a = 0;
  private double b = 0;
  private double result = 0;
  private boolean resultExists = false;
  public Calculator() {}
  public double getA() { return a; }
  public void setA(double a) { this.a = a; }
  public double getB() { return b; }
  public void setB(double b) { this.b = b; }
  public double getResult() {resultExists = false; return result; }
  public void doSum() { result = a + b; resultExists = true; }
  public void doSubtr() { result = a - b; resultExists = true; }
  public void doMult() { result = a * b; resultExists = true; }
  public void doDev() { result = a / b; resultExists = true; }
  public boolean isResultExists() { return resultExists; }
}
```

Как мы писали раньше
JSP-страница, часть 1

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="calc.Calculator" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <body>
    <%
      Calculator calc = (Calculator) session.getAttribute("calc");
      if (calc == null) {
        calc = new Calculator();
        session.setAttribute("calc", calc);
      }
      if (request.getParameter("aEd") != null) {
        calc.setA(Double.parseDouble((String) request.getParameter("aEd")));
      }
      if (request.getParameter("bEd") != null) {
        calc.setB(Double.parseDouble((String) request.getParameter("bEd")));
      }
...
```

Как мы писали раньше
JSP-страница, часть 2

```
...

      if (request.getParameter("sum") != null) {
        calc.doSum();
      }
      else if (request.getParameter("subtr") != null) {
        calc.doSubtr();
      }
      else if (request.getParameter("mult") != null) {
        calc.doMult();
      } else if (request.getParameter("dev") != null) {
        calc.doDev();
      }
    %>

...
```

Как мы писали раньше
JSP-страница, часть 3

```
...

    <form name="calcForm" action="index.jsp">
      First:<input type="text" name="aEd"
                value="<%=calc.getA()%>" size="15"/><br/>
      Second:<input type="text" name="bEd"
                value="<%=calc.getB()%>" size="15" /><br/>
      <input type="submit" value="+" name="sum" />
      <input type="submit" value="-" name="subtr" />
      <input type="submit" value="*" name="mult" />
      <input type="submit" value="/" name="dev" />
      <%
        if (calc.isResultExists()) {
      %>
      <br/>Result: <%=calc.getResult()%>
      <%}%>
    </form>
  </body>
</html>
```

Как мы пишем теперь
web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
  </context-param>
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name><url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <session-config><session-timeout>30</session-timeout></session-config>
  <welcome-file-list>
    <welcome-file>faces/index.xhtml</welcome-file>
  </welcome-file-list>
</web-app>
```

Как мы пишем теперь
Класс JavaBean

```
package calc;

import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;

@ManagedBean(name = "Calculator")
@SessionScoped
public class Calculator
    implements Serializable {
    ...
}
```

Как мы пишем теперь
Facelet

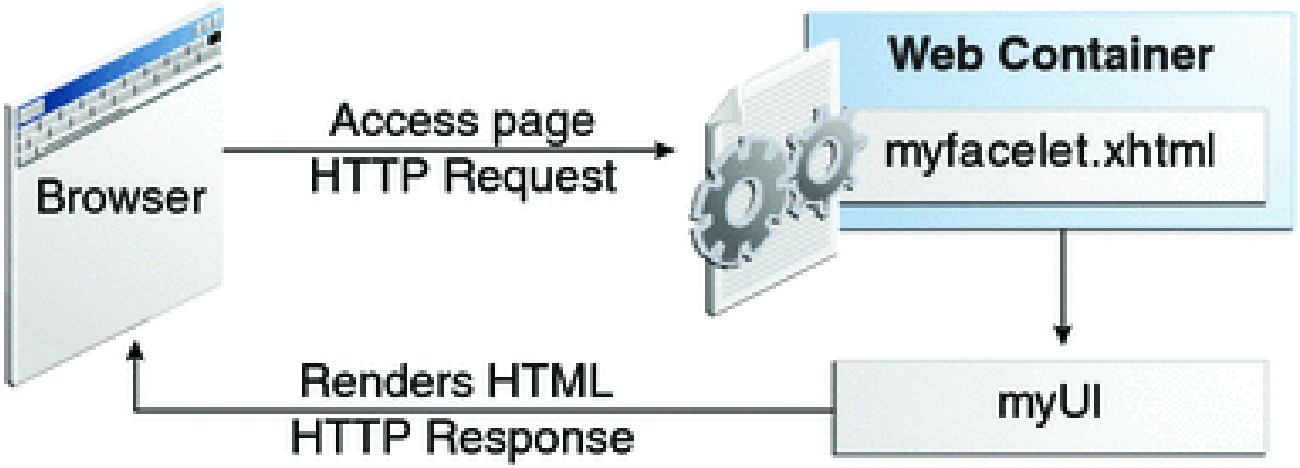
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:c="http://java.sun.com/jsp/jstl/core">
    <h:head>
        <title>Calculator</title>
    </h:head>
    <h:body>
        <h:form>
            First: <h:inputText label="a" value="#{Calculator.a}"/><br/>
            Second: <h:inputText label="b" value="#{Calculator.b}"/><br/>
            <h:commandButton action="#{Calculator.doSum}" value="+" />
            <h:commandButton action="#{Calculator.doSubtr}" value="-" />
            <h:commandButton action="#{Calculator.doMult}" value="*" />
            <h:commandButton action="#{Calculator.doDev}" value="/" />
            <c:if test="${Calculator.resultExists}">
                <br/>Result: <h:outputText value="${Calculator.result}" />
            </c:if>
        </h:form>
    </h:body>
</html>
```

Элементы JSF-приложения

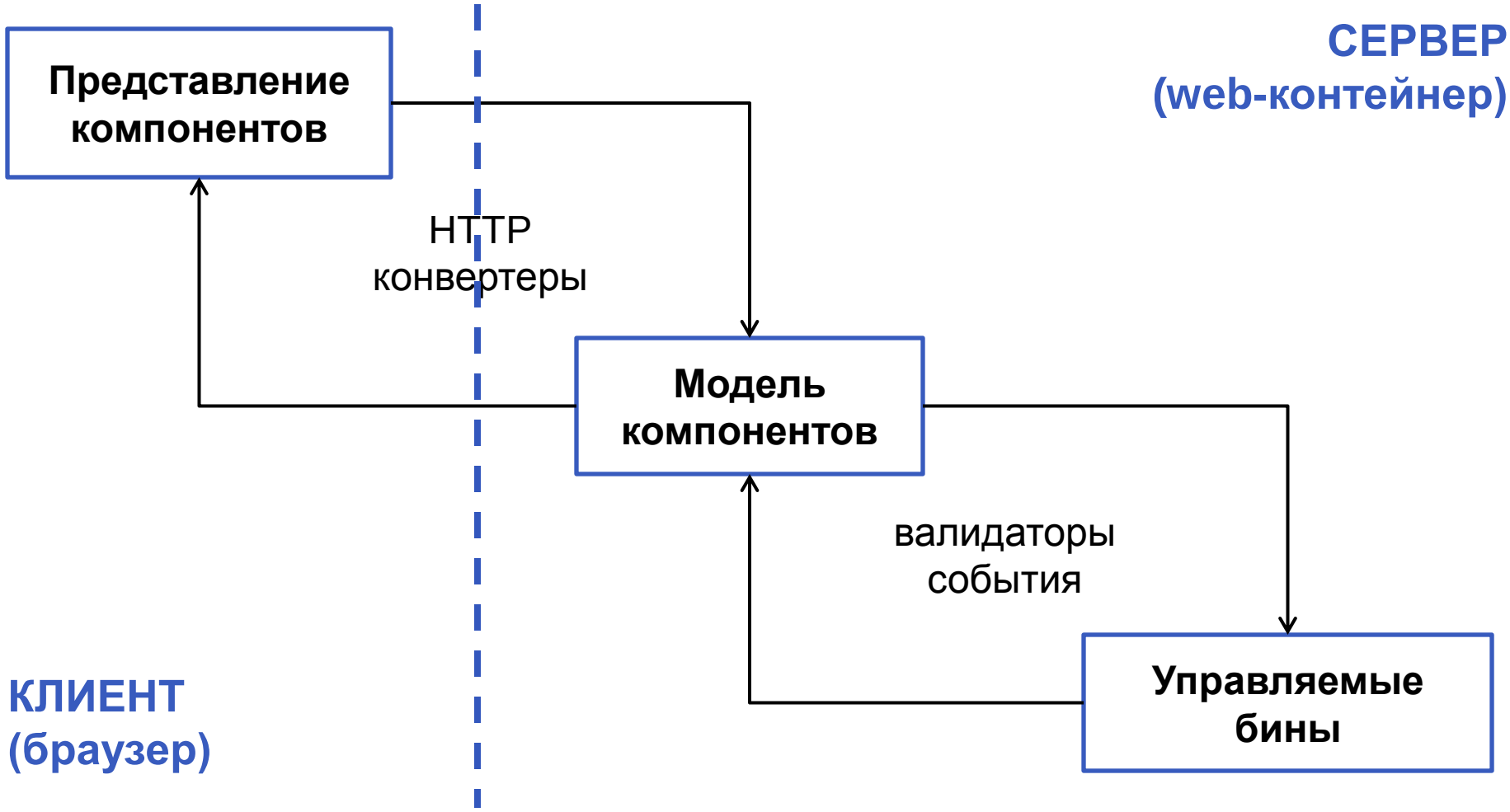
- Набор страниц
  - JSP (классический html + дополнительные теги)
  - Facelets (xhtml + дополнительные теги)
- Набор используемых бинов
- Опционально, конфигурационный файл
- Опционально, дескриптор развертывания
- Дополнительные классы
- Библиотеки дополнительных тегов

Обработка запроса

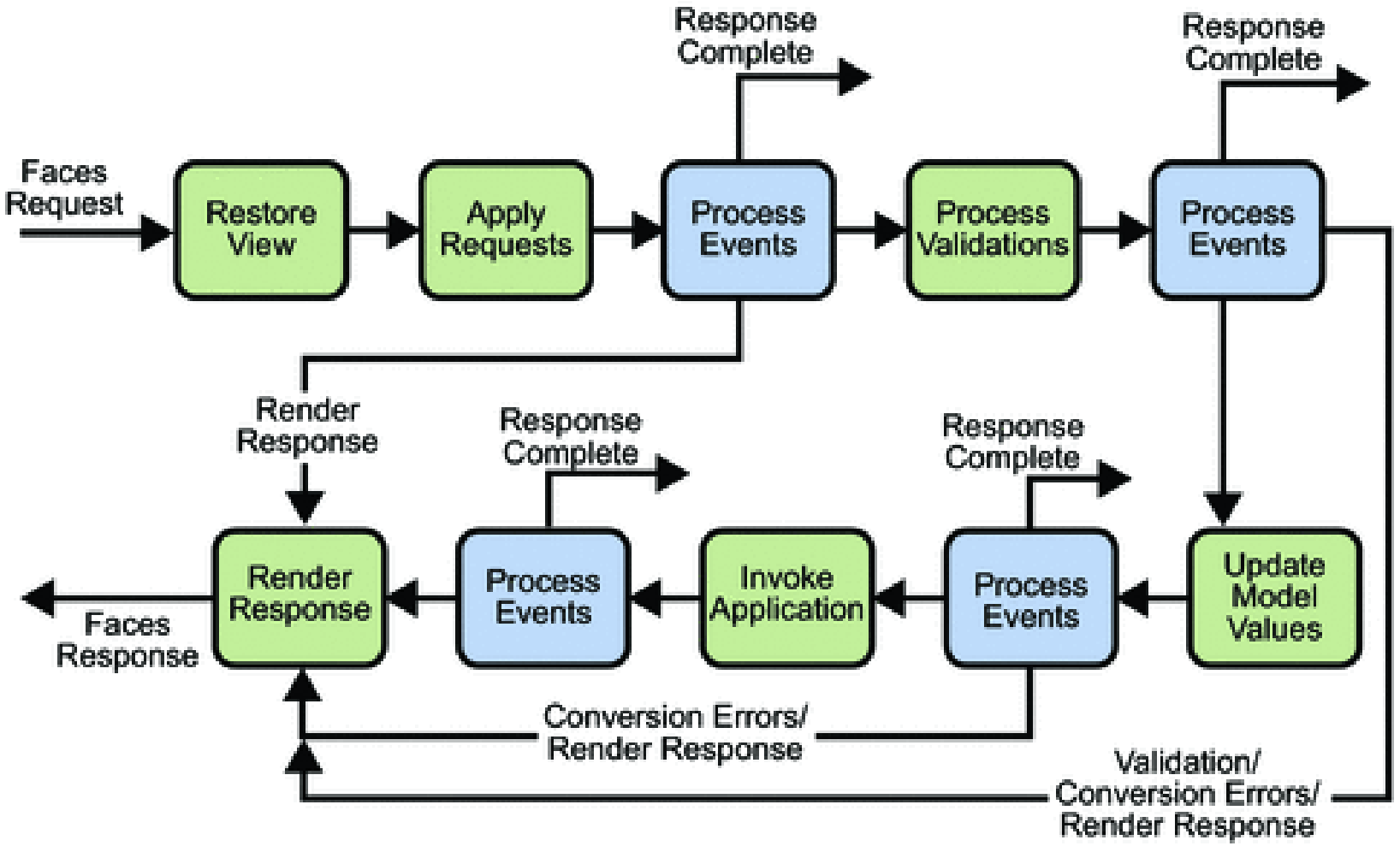
- Facelet включает JSF-теги, описывающие UI-компоненты
- Программа UI управляет объектами, использующимися в xhtml-странице:
  - Компоненты UI, описанные тегами на странице
  - Зарегистрированными для компонентов слушателями событий, валидаторами и конвертерами
  - POJO-объектами, инкапсулирующими данные и бизнес-логику



Взаимодействие элементов



Жизненный цикл обработки запросов



Жизненный цикл обработки запросов

- Значительно отличается от жизненного цикла в случае JSP
- Выражения EL вида `#{ выражение }` (отложенное вычисление) имеют различный смысл на различных стадиях
  - На этапе формирования отклика позволяют вычислять значения, отправляемые в отклик
  - На этапе обработки запроса позволяют изменять состояние компонентов на сервере
- Операции жизненного цикла отличаются в случае первого и последующих запросов

Facelets

- Особенности
  - Для создания страниц используется XHTML (transitional)
  - Использование библиотек тегов (через пространства имен)
  - Поддержка Expression Language

- Преимущества
  - Повторное использование кода (шаблоны и компоненты)
  - Возможность настройки и корректировки работы компонентов
  - Быстрая компиляция
  - Проверка выражений EL на этапе компиляции
  - Быстрый рендеринг компонентов

Библиотеки тегов

Библиотека	URI	Префикс	Содержимое
Java Server Faces Facelets Tag Library	http://java.sun.com/jsf/facelets	ui:	Теги для работы с шаблонами
Java Server Faces HTML Tag Library	http://java.sun.com/jsf/html	h:	Теги для компонентов документа
JavaServer Faces Core Tag Library	http://java.sun.com/jsf/core	f:	Теги дополнительных действий, не зависящих от технологии рендеринга
JSTL Core Tag Library	http://java.sun.com/jsp/jstl/core	c:	JSTL Core
JSTL Functions Tag Library	http://java.sun.com/jsp/jstl/functions	fn:	JSTL Functions

Управляемые бины Managed beans

- С точки зрения языка представляют собой обычный класс JavaBean-компонента
- Класс снабжается аннотацией @ManagedBean
- Жизненный цикл экземпляра компонента управляется контейнером
- В основном свойства бина «подкладываются» под свойства UI-компонента, а его методы «выполняют действия» компонента
- Характеристики указываются с помощью дополнительных аннотаций

Пределы сохранения компонентов

- Всё web-приложение
  - `@ApplicationScoped`
- Клиентская сессия
  - `@SessionScoped`
- Одна страница (представление)
  - `@ViewScoped`
- Один запрос
  - `@RequestScoped`
- Отсутствие сохранения
  - `@NoneScoped`

Конвертеры

- Когда бин привязан к UI-компоненту, существует 2 формы данных компонента
  - В представлении: нечто, что может интерпретировать и вводить пользователь
  - В модели: данные, хранящиеся в бине в виде значений типов данных
- Конвертеры применяются для перевода из одной формы в другую
- Существуют стандартные конвертеры для стандартных типов
  - `DoubleConverter`
  - `LongConverter`
  - ...

Конвертеры

- Существует возможность разрабатывать и использовать новые типы конвертеров
- Для конвертеров можно указывать сообщение, которое следует выводить в случае ошибки

```
<h:inputText id="ccno"
    size="19"
    converterMessage="#{ErrMsg.userNoConvert}"
    converter="CreditCardConverter"
    required="true">
        ...
</h:inputText>
```

Валидаторы

- Функционирование компонента может подразумевать ограничения стандартных типов
- Для проверки соответствия значений требованиям бизнес-логики приложения применяются валидаторы
- Существует ряд стандартных валидаторов
  - `DoubleRangeValidator`
  - `LongRangeValidator`
  - `LengthValidator`

Валидаторы

- Валидаторы указываются с помощью специальных тегов
- Существует возможность разрабатывать и использовать новые виды валидаторов

```
<h:inputText id="userNo" label="User Number"
  value="#{UserNumberBean.userNumber}">
    <f:validateLongRange
      minimum="#{UserNumberBean.minimum}"
      maximum="#{UserNumberBean.maximum}" />
</h:inputText>
```

Выбор перехода

- «Исход» страницы определяется атрибутом action компонента, который был использован пользователем
- Данный атрибут может быть:
  - Статичным

```
<h:commandButton id="submit" action="success" value="Submit" />
```

  - Вычислимым

```
<h:commandButton id="submit" action="#{userNumberBean.getStatus}"
value="Submit" />
```

- Если в случае вычислимого атрибута метод не возвращает ничего или возвращает `null`, то переход совершается на эту же страницу

Значения исходов

- URL

```
<h:commandLink action="page1.xhtml">URL</h:commandLink>
```

- Имя фейслета (без расширения)

```
<h:commandLink action="page2">facelet name</h:commandLink>
```

- Имя случая для правил навигации

```
<h:commandLink action="exit">navigation rule</h:commandLink>
```

Правила навигации

- Определяет правила перехода между страницами приложения
- По сути образует граф состояний с случаями переходов
- Сами правила указываются в конфигурационном файле faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config version="2.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd">
    <navigation-rule>
        <from-view-id>/index.xhtml</from-view-id>
        <navigation-case>
            <from-outcome>exit</from-outcome>
            <to-view-id>/page3.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>
</faces-config>
```

Сообщения

- Сообщения, выводимые, например, в случае ошибок конвертеров, можно хранить в отдельных файлах с расширением properties

```
ConversionError=Input value is not a String
FormatInvalid=Input value does not match legal pattern(s) {0}
```

- Для использования сообщений их необходимо зарегистрировать в конфигурационном файле faces-config.xml

```
<application>
  <resource-bundle>
    <base-name>guessNumber.ApplicationMessages</base-name>
    <var>ErrMsg</var>
  </resource-bundle>
</application>
```

- Существуют механизмы, облегчающие локализацию программ

За бортом…

- Библиотеки тегов и их возможности
- Шаблоны документов и встраиваемые в шаблоны фейслеты
- Встроенный AJAX
- Написание собственных конвертеров
- Написание собственных валидаторов
- Написание собственных компонентов
- Дополнительные библиотеки компонентов
- …

Общая ситуация

Спасибо за внимание!

Дополнительные источники

- Гери, Д.М. Java Server Faces. Библиотека профессионала [Текст] / Дэвид Гери, Кей Хорстманн. – М. : Издательский дом «Вильямс», 2011. – 544 с.
- JavaEE APIs & Docs [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/apis-139520.html, дата доступа: 14.10.2013.
- JavaEE Tutorials [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html, дата доступа: 14.10.2013.
- Expression Language [Электронный ресурс]. – Режим доступа: http://uel.java.net/, дата доступа: 14.10.2013.
- Введение в Java Server Faces [Электронный ресурс]. – Режим доступа: http://www.ibm.com/developerworks/ru/edu/j-jsf1/, дата доступа: 14.10.2013.
- Java Server Faces Technology [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Java Naming & Directory Interface

Занятие 11

Гаврилов А.В.

Самара
2013

План занятия

- Общие принципы JNDI
- Основные термины JNDI
- Классы и интерфейсы JNDI
- Настройка JNDI

Служба имен

- **Задача**

  Обеспечение доступа к тем или иным ресурсам по произвольным именам, которые сопоставляются с этими ресурсами в момент их опубликования в службе имен

- **Примеры**

  DNS
  LDAP
  Файловая система ОС

Составные части JNDI

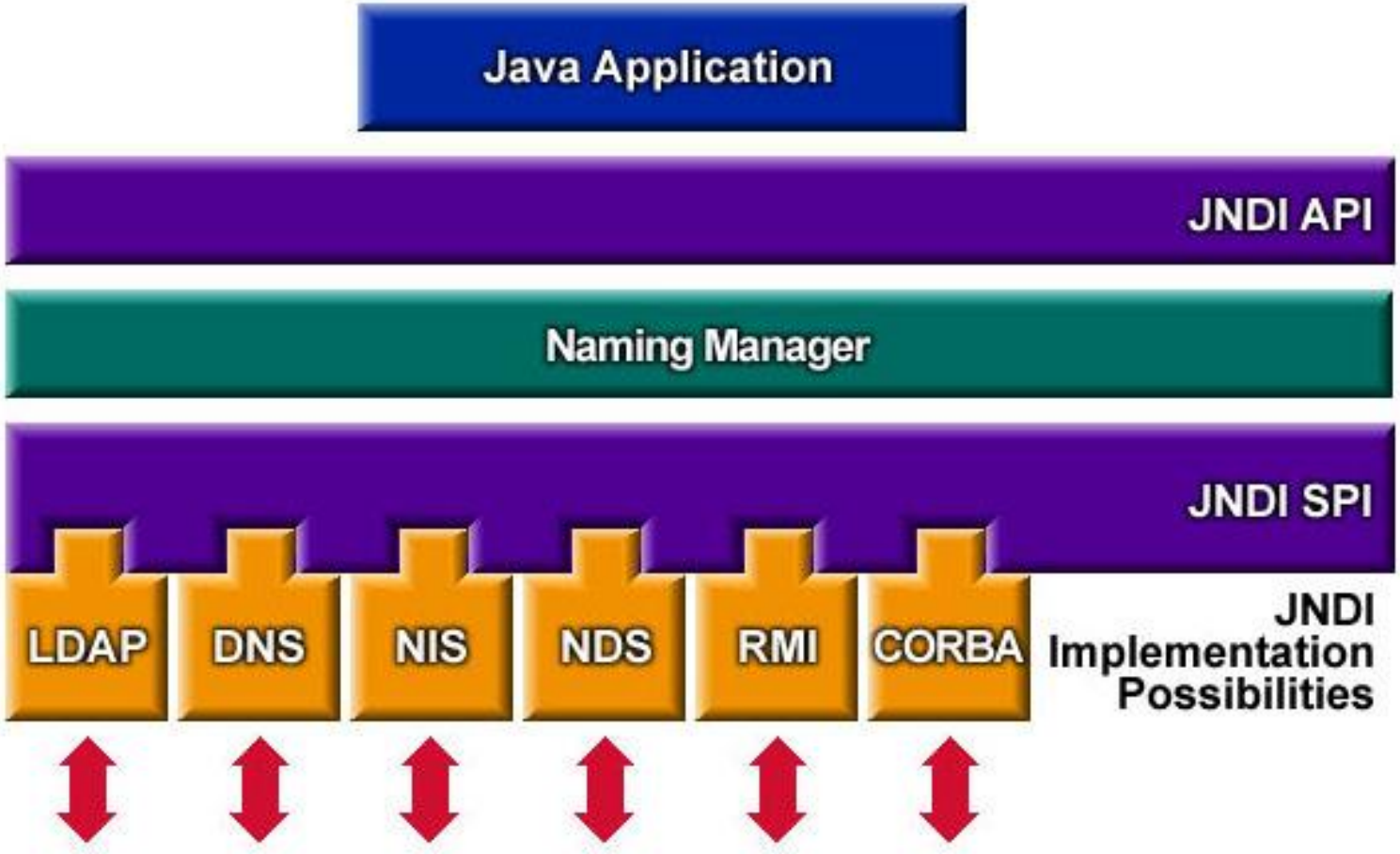
- **Java Naming & Directory Interface API**

  Универсальный набор интерфейсов и методов Java, который позволяет обратиться к некоторой «обобщенной» службе имен

- **Java Naming & Directory Interface SPI**

  Service Provider Interface – позволяет подключить для использования практически любую реальную реализацию службы имен

Архитектура JNDI



Структура JNDI

- **Naming Service**

  Служба имен – обеспечивает сопоставление произвольного (возможно, с некоторыми ограничениями) имени с некоторым ресурсом и позволяет строить иерархию таких ассоциаций

- **Directory Service**

  Служба каталогов – обеспечивает сопоставление с ресурсом произвольного списка атрибутов и их значений

Атомарное имя
Atomic Name

- Такое имя, которое уже нельзя разделить на отдельные части (с точки зрения службы имен, разумеется)
- Пример: в файловой системе это относительное имя файла, напр. Myfile.txt
- JNDI сама по себе не накладывает ограничений на атомарное имя

Составное имя
Compound Name

- Совокупность одного или нескольких атомарных имен
- Позволяет строить иерархию имен и сопоставленных с ними объектов, и определять место объекта в иерархии
- Пример: dir1/dir2/myfile.txt
- Относительно, как и атомарное имя

Сопоставление имени и объекта

- **Binding** (привязка, связывание) – процесс сопоставления атомарного имени и некоторого ресурса (объекта)
- Получается пара «имя-объект», часто называемая связанным именем
- Пример: «имя файла – его содержимое»

Разрешение имени

- **Resolving** (разрешение имени) – операция, в результате выполнения которой пользователь получает доступ к ресурсу, задав его имя
- Если составное имя содержит несколько атомарных имен, то операция разрешения применяется последовательно к каждому из них
- Пример: поиск файла по полному имени
- Результат зависит от конкретной реализации JNDI

Контекст
Context

- Объект, задачей которого является управление совокупностью связанных имен, относящихся к данному контексту
- Пример: в файловой системе – каталог
- Может содержать «обычные» связанные имена и другие контексты
- Позволяет выполнение ряда действий

Начальный контекст
Initial Context

- Т.к. все имена являются относительными, необходим некоторый контекст, относительно которого строятся (интерпретируются) все остальные имена
- Пример: сетевой диск ОС
- Существует стандартный интерфейс получения доступа к начальному контексту
- В общем случае за создание начального контекста отвечает некоторое приложение (или класс) – фабрика контекстов

Система и пространство имен

- **Naming System** (система имен) – совокупность контекстов одного типа, которые используют одно и то же соглашение об именах
  - Пример – система каталогов ОС
- **NameSpace** (пространство имен) – совокупность всех имен в системе

Композитное имя
Composite Name

- Имя в «федерации», являющейся объединением нескольких смешанных систем имен
- Состоят из фрагментов, интерпретируемых в различных системах имен
- Пример: http://www.mycompany.com/public/myfile.htm

Пакеты JNDI

- **`javax.naming`**
  Взаимодействие со службой имен
- **`javax.naming.directory`**
  Взаимодействие со службой каталогов
- **`javax.naming.spi`**
  «Подключение» к JNDI конкретной реализации служб имен и каталогов
- **`javax.naming.event`**
  Работа с событиями, возникающими при работе с ресурсами

Интерфейсы и классы службы имен

- `Name`
- `NameClassPair`
- `Binding`
- `NamingEnumeration`
- `NamingException`
- `Context`
- `InitialContext`

Интерфейс Name

- Представляет имя – упорядоченный набор элементов
- Имя может быть композитным, составным или атомарным
- Предоставляет методы работы с именем и его элементами:
  - Добавление и удаление элементов
  - Получение частей имени в виде строк и других имен
  - Определение характеристик имени

Класс NameClassPair

- Объект класса представляет пару «имя объекта» – «имя класса объекта»
- Оба значения хранятся в виде строк
- Предоставляет методы для получения и замены имен

Класс Bindind

- Объект класса представляет собой связанное имя
- Содержит пару «имя объекта» – «объект»
- Расширяет класс `NameClassPair`
- Добавляются методы работы с объектом

Интерфейс NamingEnumeration

- Используется при работе со списками, возвращаемыми методами служб именования и каталогов
- Является наследником `Enumeration`
- По сути является итератором по некоторому набору объектов
- Концептуальное отличие в возможности выбрасывания исключений `NamingException`

Класс NamingException

- Родительский класс всех исключений служб JNDI
- Объявлено почти во всех методах служб именования и каталогов
- Имеет некоторое (достаточно большое) количество наследников для конкретных исключительных ситуаций
- Имеет специфические методы описания состояния исключения

Интерфейс Context

- Представляет контекст службы имен, состоящий из набора связей «имя-объект»
- Содержит методы работы с элементами контекста и изменения его состояния
- В основном и используется программистами
- Содержит методы работы с элементами контекста:
  - Связывание и удаление элементов `bind()`, `rebind()`, `unbind()`
  - Разрешение имени `lookup()`
  - Создание и удаление вложенных контекстов `createSubcontext()`, `destroySubcontext()`
  - Получение списков элементов `list()`, `listBindings()`
  - Ряд других методов

Класс InitialContext

- Реализует интерфейс `Context`
- Объект класса представляет собой начальный контекст
- В процессе выполнения конструкторов происходит обращение к фабрикам контекстов
- Параметры создания задаются так или иначе: явно или неявно

Настройка JNDI

- Настройка производится либо явно при работе с контекстом, либо неявно указанием значений переменных среды
- Переменных много, могут встречаться специфические
- Имена переменных указаны как константы интерфейса `Context`

java.naming.factory.initial

- Указывает имя класса фабрики начального контекста
- Класс предоставляется тем, кто создает конкретную реализацию службы имен
- Не имеет значения по умолчанию
- Если значение не указано, то используется значение среды переменной Java
  java –Djava.naming.factory.initial=
  com.sun.jndi.cosnaming.CNCtxFactory myapp

java.naming.provider.url

- Задает информацию о конкретной используемой службе
- Служба может находится и на другом хосте
- Если не задана, то используется значение переменной среды Java
- Пример:
  java.naming.provider.url=iiop://имя_хоста:900

Параметры среды контекста

- Доступны при работе с контекстом через его методы

```
Hashtable getEnvironment()

Object addToEnvironment(
   String propName, Object propVal)

Object removeFromEnvironment(
   String propName)

...
```

Параметры среды начального контекста

```
java.util.Property prop = new java.util.Property();
prop.put(Context.INITIAL_CONTEXT_FACTORY,
  "com.sun.jndi.cosnaming.CNCtxFactory");
prop.put(Context.PROVIDER_URL, "iiop://my_host:900");
Context initContext = new InitialContext(prop);
```

```
java -Djava.naming.factory.initial=
com.sun.jndi.cosnaming.CNCtxFactory
-Djava.naming.provider.url=iiop://my_host:900
myApplication

...

Context initContext = new InitialContext();
```

Файл ресурсов JNDI

- Является еще одним способом задания параметров JNDI
- Имеет имя `jndi.properties`
- Хранит пары вида `имя_свойства=значение_свойства`
- Может находиться в jar-файле
- Поиск файла выполняется в каталогах, сопоставленных с приложением, указанных в CLASSPATH, а также по пути `$JAVA_HOME/lib/jndi.properties`

Спасибо за внимание!

Дополнительные источники

- Хорстманн, К.С. Java2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010. – 816 с.
- Цимбал, А. А. Технологии создания распределенных систем. Для профессионалов [Текст] / А. Цимбал, М. Аншина. – СПб. : Питер, 2003. – 576 с.
- Java Naming and Directory Interface (JNDI) [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/jndi/index.html, дата доступа: 14.10.2013.
- The JNDI Tutorial [Электронный ресурс]. – Режим доступа: http://download.oracle.com/javase/jndi/tutorial/, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Java Database Connectivity

Занятие 12

Гаврилов А.В.

Самара
2013

План занятия

- Структура JDBC
- Виды драйверов
- Основные интерфейсы
- Виды подключений
- Выполнение SQL-операторов
- Взаимодействие с СУБД

JDBC

- **Java Database Connectivity** – стандарт взаимодействия между базами данных и Java-приложениями
- API для доступа к SQL-совместимым базам данных
- Интерфейсная модель, обеспечивающая взаимодействие с базой данных

http://www.oracle.com/technetwork/java/javase/jdbc/index.html

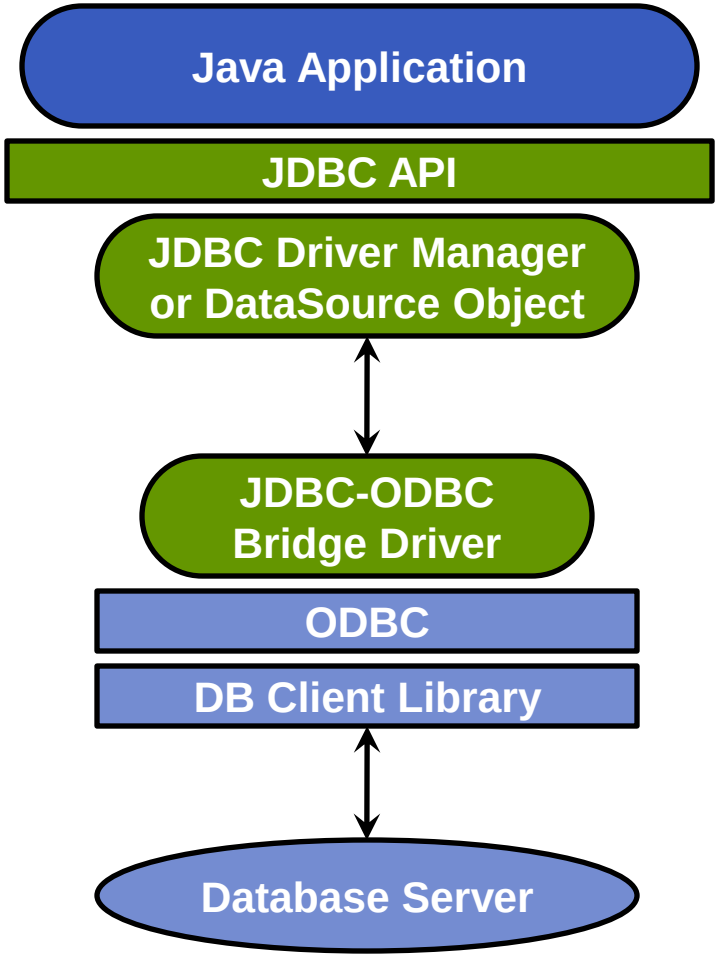
Состав JDBC

- **JDBC API**
  Содержит набор классов и интерфейсов, определяющий Java-ориентированный доступ к базам данных. Объявлены в пакетах `java.sql` и `javax.sql`

- **JDBC-driver**
  Специфический для каждой базы данных драйвер, с помощью которого JDBC превращает вызовы уровня API в «родные» команды сервера баз данных. Бывают 4-х типов

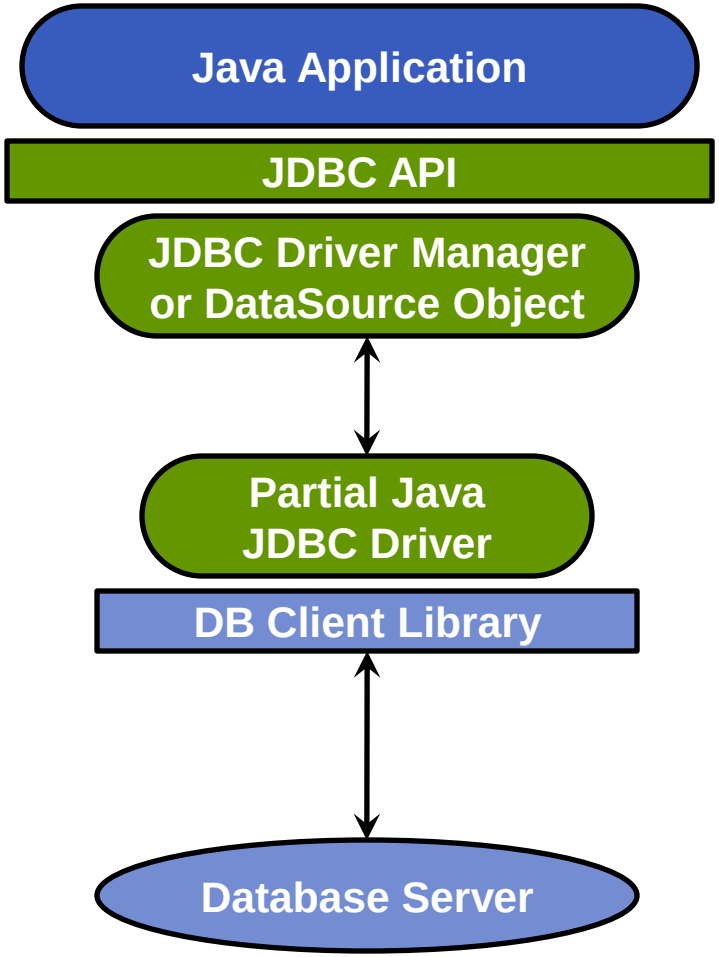
Тип драйвера 1
JDBC-ODBC Bridge

- Отображают вызовы JDBC-API в вызовы другого API (не источника данных)
- Классический пример – отображение в вызовы ODBC-драйвера
- Проблемы переносимости и эффективности



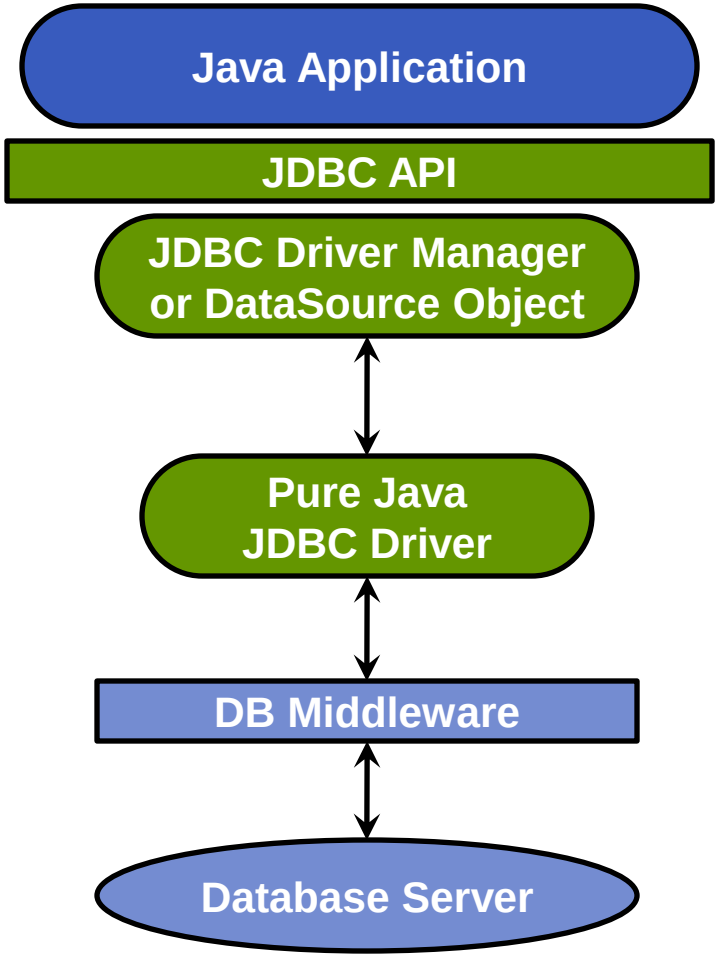
Тип драйвера 2
Native API Partly Java Technology-enabled

- Используется специфический для конкретного источника данных специализированный API
- В большинстве случаев создается в виде native-библиотек
- Проблемы те же



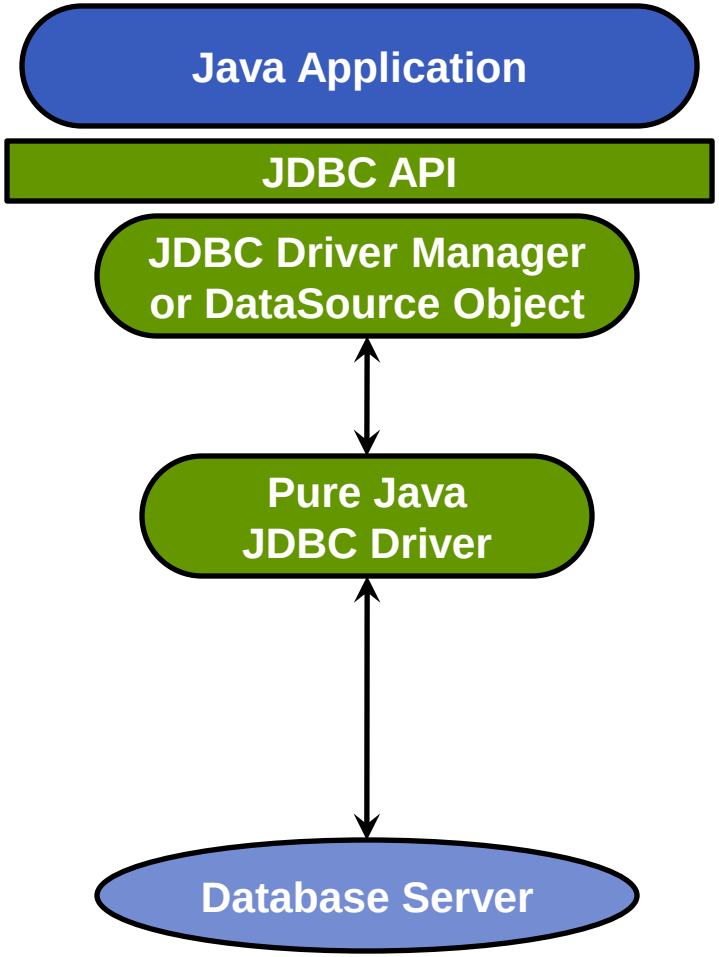
Тип драйвера 3
Pure Java Driver for Database Middleware

- Полностью написан на Java
- Обеспечивает обращение клиента к клиентской части сервера БД
- На клиентской машине должно быть запущено промежуточное ПО, которое получает запросы клиента и передает их серверу БД



Тип драйвера 4
Direct-to-Database Pure Java

- Полностью написан на Java
- Драйвер сам, без дополнительного промежуточного сервера, способен общаться по сети с сервером БД
- Иногда такие драйвера называют «тонкими» (thin)



Основные интерфейсы

- **`javax.sql.DataSource`**
  Загрузка и запуск JDBC-драйвера, получение соединения с БД
- **`java.sql.Connection`**
  Формирование запросов к источнику данных и управление транзакциями
- **`java.sql.Statement`**
  `java.sql.PreparedStatement`
  `java.sql.CollableStatement`
  Позволяют отправить запрос к источнику данных

Основные интерфейсы

- **`java.sql.ResultSet`**
  Позволяет перемещаться по набору данных, возвращаемых оператором SELECT, и считывать значения отдельных полей в текущей записи
- **`java.sql.ResultSetMetaData`**
  Позволяет получить информацию о структуре набора данных: количество полей, их названия, тип и т.д.
- **`java.sql.DatabaseMetaData`**
  Позволяет получить информацию о структуре самого источника данных

Загрузка драйвера (DriverManager)

- Указание в свойстве

```
java -Djdbc.drivers=interbase.interclient.Driver
```

- Явная регистрация

```
DriverManager.registerDriver(Driver driver)
```

- Явная загрузка

```
Class.forName("oracle.jdbc.driver.OracleDriver")
```

Подключение (DriverManager)

Используется методы **DriverManager**

```
public static Connection getConnection(
  String url)throws SQLException

public static Connection getConnection(
  String url, String user, String password)
  throws SQLException

public static Connection getConnection(
  String url, Properties info)
  throws SQLException
```

URL соединения

- Задается в формате:
  `jdbc:субпротокол:дополнительное_имя`
- **JDBC**
  Ключевое слово, представляющее тип драйвера базы
- **Субпротокол**
  Определяет тип базы, с которой должно устанавливаться соединение
- **Дополнительное_имя**
  Дополнительная информация, необходимая для установки соединения
  с конкретным типом базы данных

```
jdbc:oracle:thin:@SERVER:PORT:DBNAME
jdbc:oracle:thin:@ServerName:1521:TSP
```

Пример (DriverManager)

```
import java.sql.*;

class TestThinApp {
  public static void main (String args[])
    throws ClassNotFoundException, SQLException {
    Class.forName("oracle.jdbc.driver.OracleDriver");

    Connection conn = DriverManager.getConnection(
      "jdbc:oracle:thin:@dssnt01:1521:dssora01",
      "scott","tiger");
    // ...
    conn.close( );
  }
}
```

Подключение
(DataSource. Версия 1)

```
import java.sql.*;
import oracle.jdbc.pool.*;

class TestThinDSApp {
  public static void main (String args[])
    throws ClassNotFoundException, SQLException {
    OracleDataSource ds = new OracleDataSource ( );
    ds.setDriverType("thin");
    ds.setServerName("dssw2k01");
    ds.setPortNumber(1521);
    ds.setDatabaseName("orcl");
    ds.setUser("scott");
    ds.setPassword("tiger");
    Connection conn = ds.getConnection( );
    //...
    conn.close( );
  }
}
```

Регистрация подключения

```
import java.sql.*;
import java.util.*;
import javax.naming.*;
import oracle.jdbc.pool.*;

public class TestDSBind {
  public static void main (String args [])
    throws SQLException, NamingException {
    Context ctx = null;
    Properties prop = new Properties( );
    prop.setProperty(Context.INITIAL_CONTEXT_FACTORY,
      "com.sun.jndi.fscontext.RefFSContextFactory");
    prop.setProperty(Context.PROVIDER_URL, "file:/JNDI/JDBC");
    ctx = new InitialContext(prop);
    OracleDataSource ds = new OracleDataSource();
    //... Настройка источника данных
    ctx.bind("joe", ds);
  }
}
```

Использование подключения

```
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
import java.util.*;

public class TestDSLookUp {
  public static void main (String[] a)
    throws SQLException, NamingException {
    Context ctx = null;
    Properties prop = new Properties( );
    prop.setProperty(Context.INITIAL_CONTEXT_FACTORY,
      "com.sun.jndi.fscontext.RefFSContextFactory");
    prop.setProperty(Context.PROVIDER_URL, "file:/JNDI/JDBC");
    ctx = new InitialContext(prop);
    DataSource ds = (DataSource)ctx.lookup("joe");
    Connection conn = ds.getConnection( );
    // ...
    conn.close( );
  }
}
```

Возникает вопрос…

- И зачем все это надо?..

- А вот за этим:
  - Настройка и использование могут происходить в различных местах
  - При использовании необходимо знать только имя объекта
  - Чаще всего настройка и связывание с именем производятся средствами сервера приложений

Подключение
(DataSource. Версия 2)

```
InitialContext ic =
  new InitialContext();

DataSource ds = ic.lookup(
  "java:comp/env/jdbc/myDB");

Connection conn = ds.getConnection( );
//...

conn.close( );
```

java.sql.Connection

- Это интерфейс
- Реализующие его объекты представляют собой установленное соединение
- Содержит методы:
  - Порождения объектов запросов и команд
  - Порождения объектов специфических типов
  - Управления транзакциями
  - Закрытия соединения

java.sql.Statement

- Интерфейс предназначен для формирования SQL-команд, которые не содержат параметров, и для получения результата их выполнения

- Экземпляр может быть получен с помощью методов интерфейса `Connection`:
  `Statement createStatement()`

Виды SQL-операторов

- **«SELECT-операторы»**

  Возвращают набор данных как множество записей (возможно, пустое)

- **«UPDATE-операторы»**

  Возвращают либо признак завершения, либо один или несколько вспомогательных параметров

Примеры выполнения операторов

```
Statement statS =
          connection.createStatement();
ResultSet rs = statS.executeQuery(
               "SELECT * FROM my_table");


Statement statU =
          connection.createStatement();
int numberOfDeletedRecords =
          statU.executeUpdate(
               "DELETE FROM my_table");
```

Примеры выполнения операторов

```
String sql = "...";
Statement stat = connection.createStatement();
boolean b = stat.execute(sql);
if (b == true) {          //Выполнен SELECT-оператор
  ResultSet rs = stat.getResultSet();
  int id;
  while(rs.next())
  {
    id = rs.getInt(1);
    ...
  }
}
else {                    //Выполнен UPDATE-оператор
  int n_records = stat.getUpdateCount();
  ...
}
```

Порядок выполнения запроса

- Анализ и разбор полученного выражения
- Компиляция в некоторый промежуточный код, специфический для каждого конкретного сервера
- Выполнение кода в режиме интерпретации

java.sql.PreparedStatement

- Позволяет использовать запросы с параметрами
- Позволяет сократить время выполнения последующих запросов
- Параметры указываются знаком «?»

```
PreparedStatement stat =
connection.prepareStatement(
  "SELECT * FROM my_table WHERE F1 = ?");

stat.setString(1, "MyString");
ResultSet rs = stat.executeQuery();
```

java.sql.ResultSet

- Методы навигации
  `first()`, `last()`, `next()`, `previous()`, ...
- Методы получения значений
  `getInt()`, `getFloat()`, `getString()`, ...
- Методы модификации значений
  `updateString()`, `updateBoolean()`, ...
- Методы управления записями
  `insertRow()`, `deleteRow()`, ...

Информация о метаданных

- Метаданные набора данных

```
ResultSet rs = stat.executeQuery(sqlString);
ResultSetMetaData rsmd = rs.getMetaData();
```

- Метаданные источника данных

```
DatabaseMetaData dbmd =
connection.getMetadata()
```

Транзакции

- В транзакции участвует только один источник данных
- Транзакции обеспечивает JDBC-совместимый источник данных
- Методы управления транзакциями определены в интерфейсе `java.sql.Connection`

Управление транзакциями

- `public void setAutoCommit(boolean autoCommit)`
- `public void commit()`
- `public void rollback()`
- `public void rollback(Savepoint savepoint)`
- `public Savepoint setSavepoint(String name)`
- `public void releaseSavepoint(Savepoint savepoint)`

Типы данных

JDBC Type	Java Type
CHAR	`String`
VARCHAR	`String`
NUMERIC	`java.math.BigDecimal`
BOOLEAN	`boolean`
INTEGER	`int`
FLOAT	`double`
BINARY	`byte[]`
DATE	`java.sql.Date`
TIME	`java.sql.Time`
BLOB	`java.sql.Blob`

Прочие особенности

- Интерфейс `CallableStatement` используется при работе с хранимыми процедурами
- Соединения и результаты запросов следует закрывать
- Существуют исключения `java.sql.SQLException`

Спасибо за внимание!

Дополнительные источники

- Хорстманн, К.С. Java2. Библиотека профессионала. Том 2. Тонкости программирования [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010. – 816 с.
- Цимбал, А. А. Технологии создания распределенных систем. Для профессионалов [Текст] / А. Цимбал, М. Аншина. – СПб. : Питер, 2003. – 576 с.
- Trial: JDBC Database Access [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html, дата доступа: 14.10.2013.
- JDBC Tutorial [Электронный ресурс]. – Режим доступа: http://www.tutorialspoint.com/jdbc/index.htm, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Enterprise Java Beans

Занятие 13

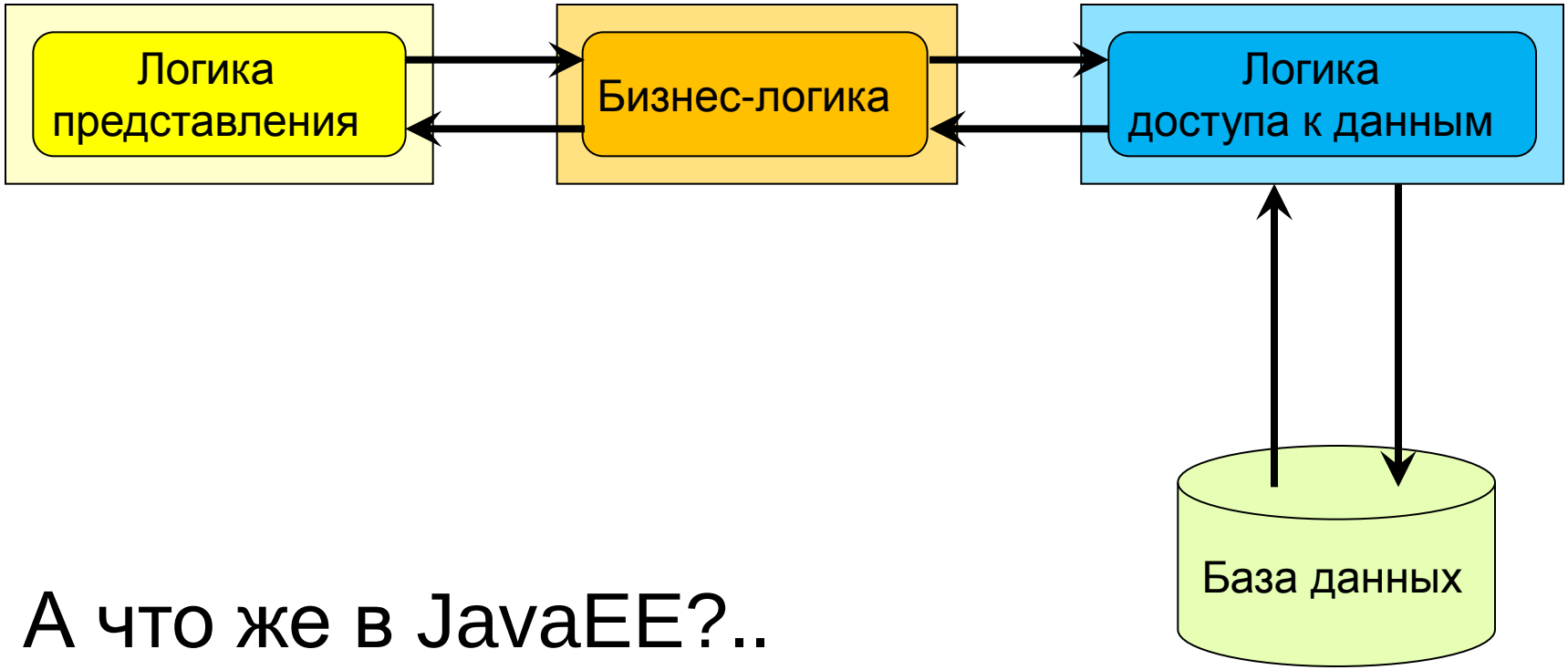
Гаврилов А.В.

Самара
2013

План занятия

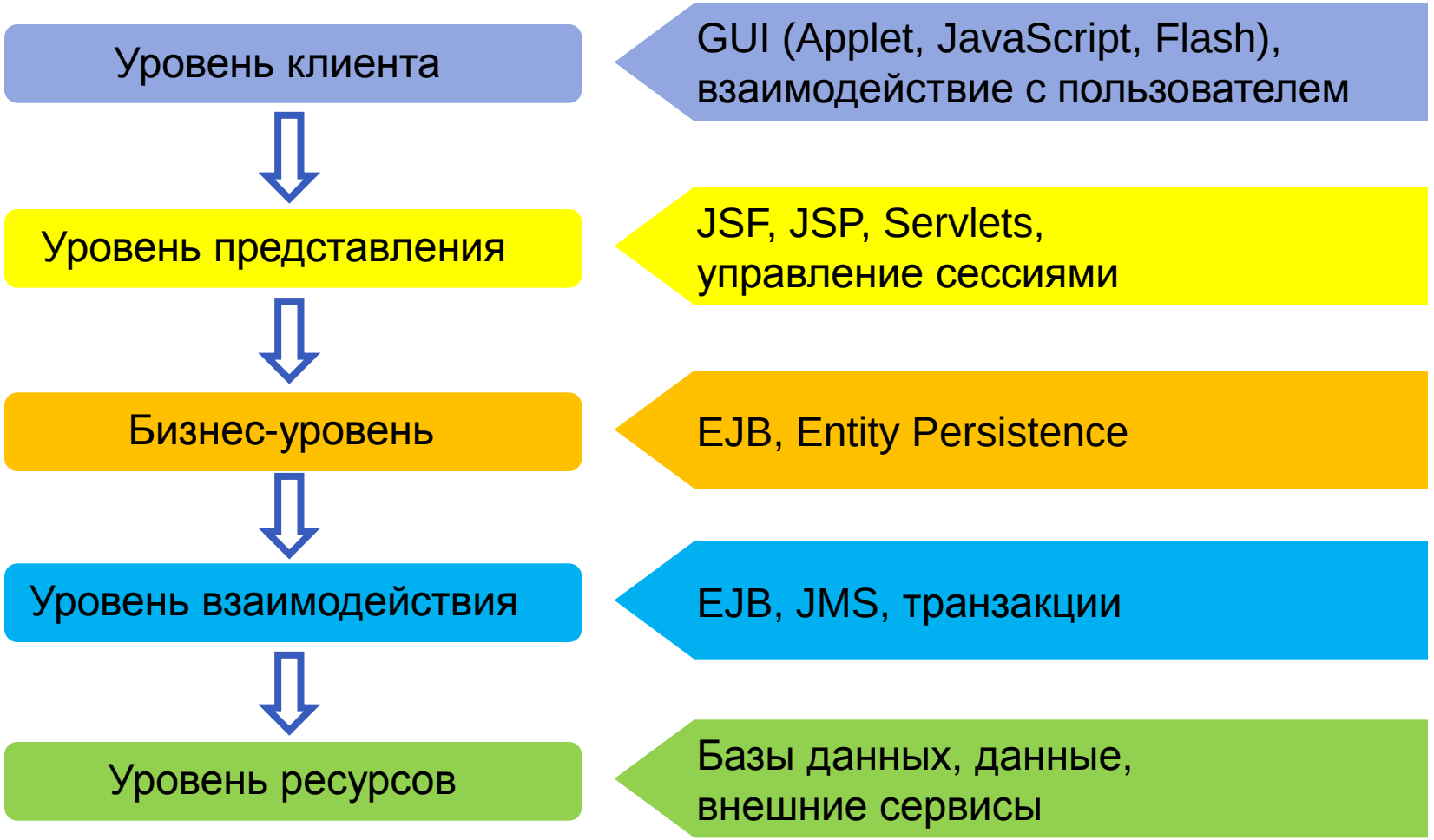
- Задачи EJB
- Элементы EJB
- Порядок разработки EJB-компонента
- Особенности EJB-компонентов

Трехслойная система



А что же в JavaEE?..

Общая схема


Уровень клиента
GUI (Applet, JavaScript, Flash), взаимодействие с пользователем
Уровень представления
JSF, JSP, Servlets, управление сессиями
Бизнес-уровень
EJB, Entity Persistence
Уровень взаимодействия
EJB, JMS, транзакции
Уровень ресурсов
Базы данных, данные, внешние сервисы

Производственные кофейные бобы

Enterprise Java Bean – это…

- компонент,
- написанный на Java,
- находящийся на сервере,
- работающий в контейнере сервера приложений,
- реализующий бизнес-логику приложения
  - некоторый код, обеспечивающий работу приложения
  - клиент получает доступ к сервисам приложения путем вызова методов компонента

Преимущества EJB

- Упрощение разработки кода бизнес-логики
- Упрощение разработки уровня представления
- Упрощение разработки сетевых приложений за счет повторного использования кода
- Упрощение разработки в целом за счет использования функциональности контейнера

Когда использовать EJB?

- Приложение должно быть легко масштабируемым
- Необходимо обеспечивать целостность данных в транзакциях
- Приложение может иметь клиентов различных видов

Виды EJB

- **Session Beans**
  - Stateful
  - Stateless
  - Singleton
- **Message Driven Beans**

Составные части компонента

- Класс компонента
- Вспомогательные классы
- Интерфейсы
- Дескриптор развертывания

Компонент и контейнер

- Компонент как совокупность всех своих частей выкладывается в сервер приложений
- Процесс развертывания управляется информацией из аннотаций и дескриптора развертывания
- Контейнер дописывает необходимые для работы компонента классы
- Контейнер регистрирует компонент в службе имен
- Контейнер управляет жизнью компонента, клиент может лишь инициировать работу

Интерфейсы компонента

- Описывают непосредственно действия, выполняемые с конкретным экземпляром компонента
- Интерфейс описывает видение компонента клиентом
- У одного компонента может быть много интерфейсов
- У компонента может не быть явного интерфейса
- Объект, удовлетворяющий типу интерфейса возвращается службой имён или в результате внедрения зависимости

Удаленные клиенты

- Могут быть запущены в JVM, отличной от JVM компонента
- Могут быть Web-компонентами, Java-приложениями или другими EJB
- «Прозрачность» компонента для клиента
- В основе взаимодействия лежит RMI, со всеми вытекающими последствиями

Локальные клиенты

- Запущенные только в той же JVM, что и компонент
- Могут быть Web-компонентами и другими EJB
- Отсутствие «прозрачности»
- Возможны тесные взаимосвязи между компонентом и клиентом-EJB

Факторы при выборе вида интерфейса

- Тесная взаимосвязанность компонентов
- Вид клиента
- Распределенность приложения
- Производительность

Соглашения именования

Элемент	Синтаксис	Пример
Имя компонента	<name>Bean	ProductBean
Класс компонента	<name>Bean	ProductBean
Бизнес-интерфейс	<name>	Product

Порядок разработки компонента

- Выбор вида компонента
- Выбор вида интерфейсов
- Написание интерфейсов класса компонента
- Написание класса компонента

Особенности EJB-компонентов

- С точки зрения модели первичны интерфейсы компонента, а не класс
- Класс компонента может даже не реализовывать интерфейсы
- Контейнер дописывает классы, реализующие интерфейсы, опираясь на предоставленный класс
- Контейнер генерирует вспомогательные классы (например, RMI-заглушки)

Особенности EJB-компонентов

- Контейнер реализует все промежуточные элементы взаимодействия клиента с компонентом
- Компонент имеет методы жизненного цикла, вызываемые контейнером
- Параметры функционирования компонента определяются аннотациями и в дескрипторах развертывания

Особенности EJB-компонентов

- При разработке EJB-компонентов приходится соблюдать множество правил, описываемых в модели JavaEE
- Многие из этих правил лежат за пределами синтаксиса и не могут быть проверены средой разработки
- При нарушении правил возникают ошибки в процессе развертывания компонента

Немного приятного

- Современные средства разработки значительно облегчают создание EJB-компонентов
  - Различного рода wizard-приложения
  - Управление характеристиками компонента через графический интерфейс
  - Визуальное редактирование дескрипторов развертывания
  - Средства отладки

Спасибо за внимание!

Дополнительные источники

- Соломон, М.К. Oracle. Программирование на языке Java [Текст] / Мартин К. Соломон, Нирва Морисco-Леруа, Джули Басу. – М. : Лори, 2010. – 512 с.
- Курванян, Б. Программирование web-приложений на языке Java [Текст] / Буди Курванян. – М. : Лори, 2009. – 880 с.
- JavaEE APIs & Docs [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/apis-139520.html, дата доступа: 14.10.2013.
- JavaEE Tutorials [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html, дата доступа: 14.10.2013.
- Enterprise JavaBeans Technology [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/ejb/index.html, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Session Beans

Занятие 14

Гаврилов А.В.

Самара
2013

План занятия

- Задачи и применение сессионных компонентов
- Жизненный цикл сессионных компонентов
- Элементы сессионных компонентов
- Обращение клиента к компоненту

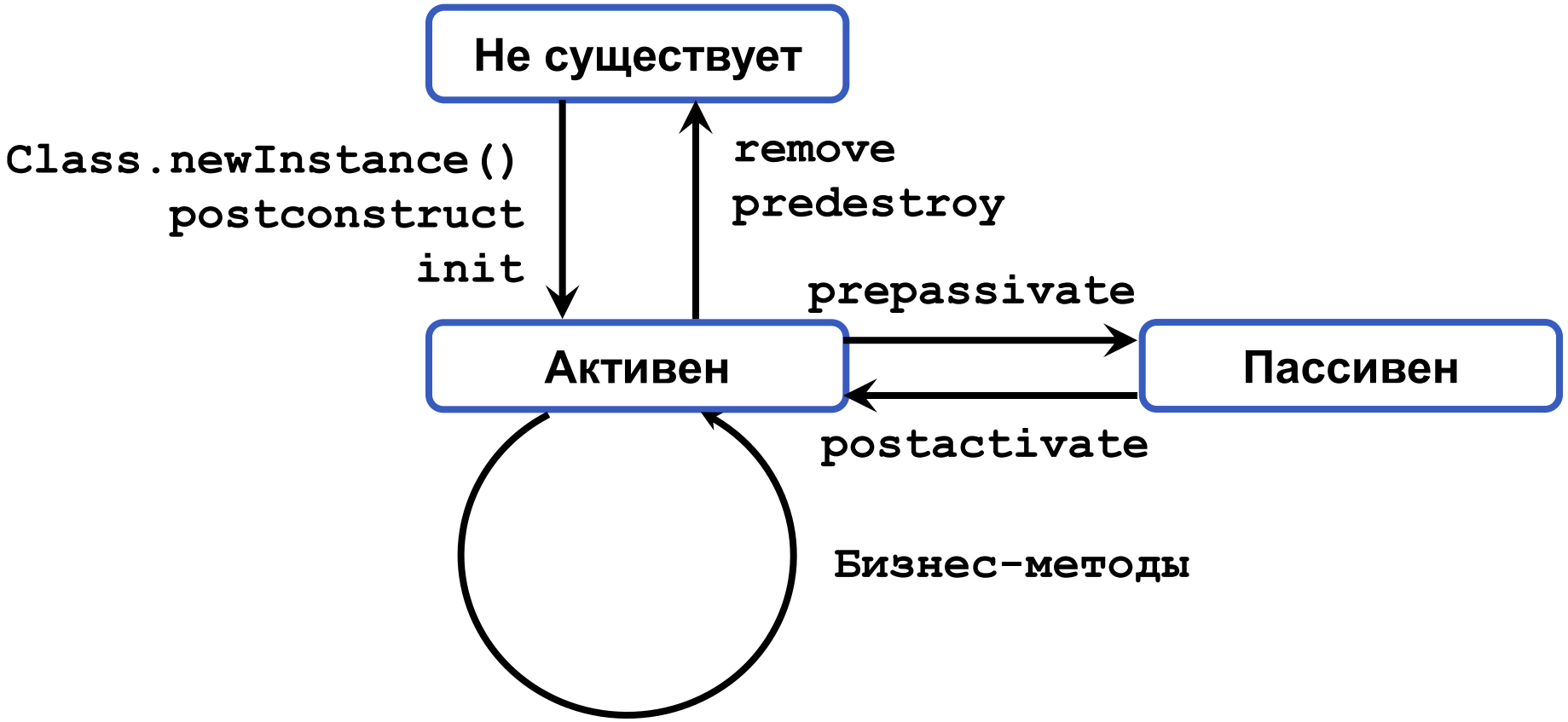
Сессионный компонент

- **Основная задача – представление клиента на стороне сервера и реализация бизнес-логики**
- Клиент вызывает методы компонента в процессе работы с приложением
- Методы компонента скрывают от клиента реальную сложность приложения

Stateful Session Beans

- Класс снабжается аннотацией `@Stateful`
- Сохраняют состояние между вызовами клиентом методов
- Сохраняют состояние на протяжении всей сессии клиента
- Для каждого клиента создается свой экземпляр компонента
- Набор объектов (pool) не создается
- В некоторых случаях могут быть сохранены контейнером в промежуточное хранилище

Жизненный цикл Stateful-компонентов



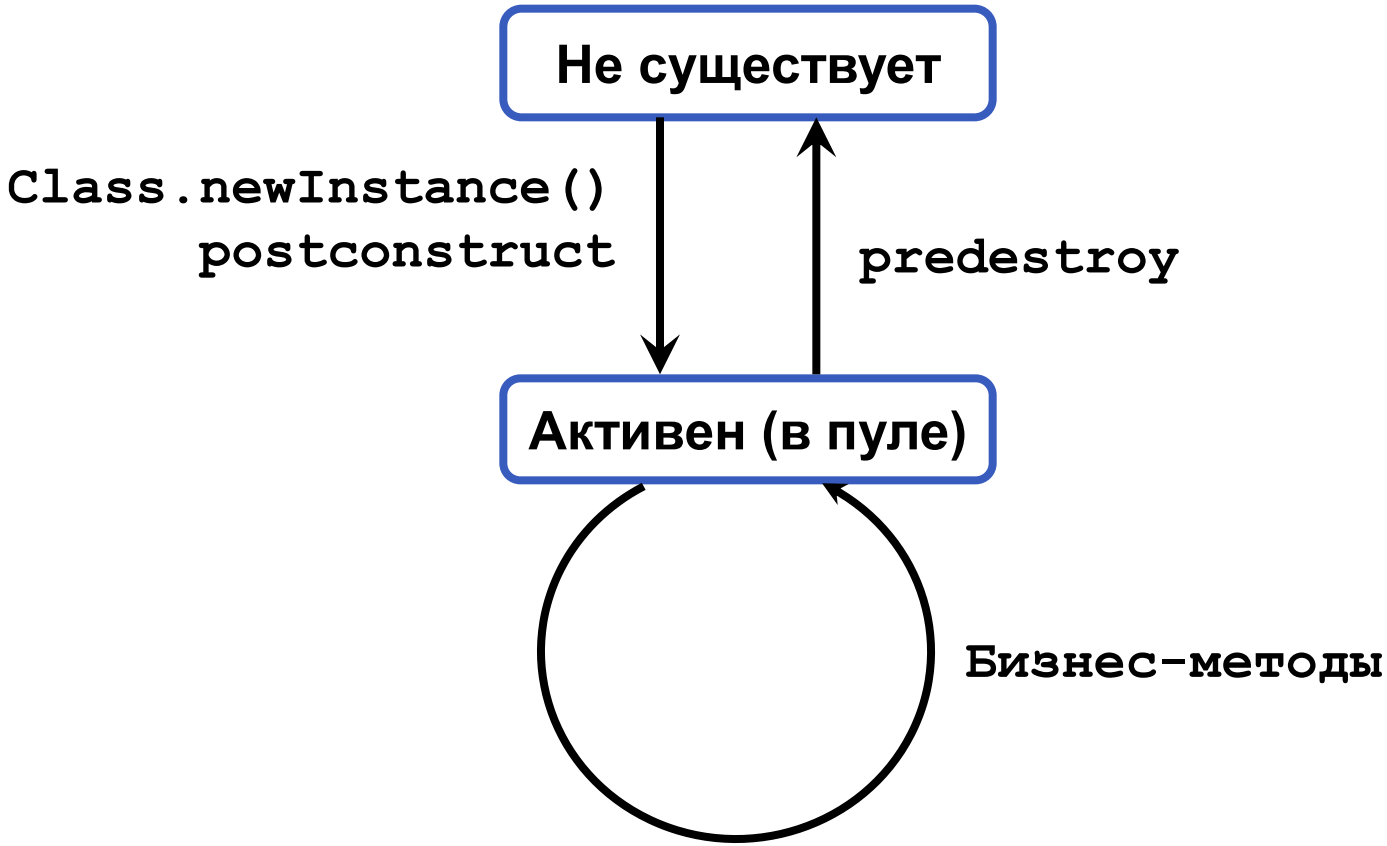
Когда используют Stateful-компоненты

- Состояние компонента описывает взаимодействие с клиентом
- Необходимо сохранять информацию о клиенте на время вызовы методов
- Необходим посредник между клиентом и другими компонентами, скрывающий от клиента сложность приложения
- Необходимо управление совместной работой нескольких компонентов

Stateless Session Beans

- Класс снабжается аннотацией `@Stateless`
- Не сохраняют состояния между вызовами клиентом методов
  - Вернее, вы не можете на это рассчитывать
- Обычно сервер хранит набор экземпляров компонента (pool)
- Все экземпляры эквивалентны
- Даже записанные подряд вызовы клиента могут адресоваться различным объектам
- Обеспечивают лучшую масштабируемость и скорость работы

Жизненный цикл Stateless-компонентов



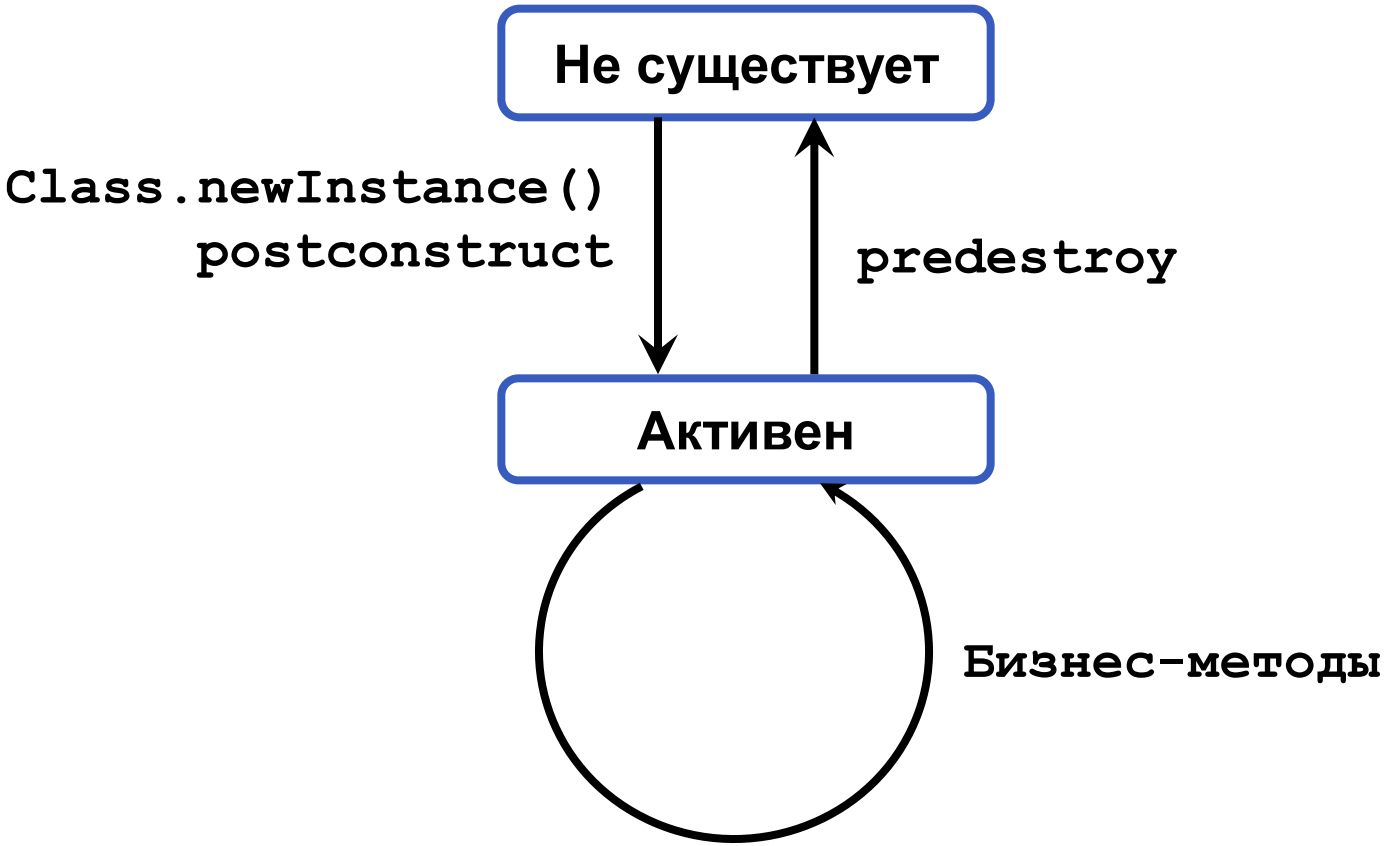
Когда используют Stateless-компоненты

- Состояние компонента не хранит специфических для клиента данных
- Методы компонента реализуют однотипные для всех клиентов действия
- Необходимо реализовать web-сервис

Singleton Session Beans

- Класс снабжается аннотацией `@Singleton`
- Один экземпляр создаётся единожды и существует всё время жизни приложения
- Сохраняет своё состояние между вызовами клиентов
- Является объектом конкурентного доступа

Жизненный цикл Singleton-компонентов



Когда используют Singleton-компоненты

- Состояние компонента должно быть видимо в пределах всего приложения
- Компонент должен быть доступен в конкурентном режиме нескольким нитям приложения
- Необходимо выполнять действия при запуске и завершении приложения
- Необходимо реализовать web-сервис

Бизнес-интерфейс Session Bean

- Обычный интерфейс Java
- Не наследует ни от каких специальных интерфейсов
  - Ни от базового интерфейса
  - Ни даже от `javax.rmi.Remote`
- В случае, если интерфейс планируется для использования в качестве удаленного, типы параметров методов и возвращаемых значений должны удовлетворять требованиям RMI

Класс Session Bean

- Должен быть помечен аннотацией `@Stateful`, `@Stateless` или `@Singleton`
- Не наследует от специальных типов
- Должен реализовывать методы, заявленные в бизнес-интерфейсах
- Может содержать методы жизненного цикла, помеченные специальными аннотациями
- Может содержать методы удаления экземпляра компонента, помеченные аннотацией `@Remove`

Пример сессионного компонента

- Бизнес-интерфейс

```
import javax.ejb.Remote;

@Remote
public interface Sum {
  double sum(double a, double b);
}
```

- Класс

```
import javax.ejb.Stateless;

@Stateless
public class SumBean implements Sum {
  public SumBean() { }
  public double sum(double a, double b) { return a + b; }
}
```

Класс и интерфейсы Session Bean

- Класс может реализовывать свои интерфейсы

```
@Remote
public interface BeanRemote {...}

@Stateless
public class Bean imlements BeanRemote {...}
```

- Класс может не реализовывать свои интерфейсы

```
public interface BeanLocal {...}

@Statefull
@Local(BeanRemote.class)
public class Bean {...}
```

- Считается, что лучше все-таки реализовывать

Бизнес-интерфейсы Session Bean

- Если аннотации `@Local` и `@Remote` применяются к…
  - классу, то:
    - сами интерфейсы можно не аннотировать
    - класс может не реализовывать интерфейсы
  - интерфейсам, то:
    - интерфейсы указываются через реализацию классом
    - отсутствие аннотации означает `@Local`
- Способы указания интерфейсов можно смешивать
- Нельзя использовать один и тот же интерфейс и как локальный, и как удаленный

Бизнес-интерфейсы Session Bean

- Компонент может иметь
  - Локальный интерфейс
  - Удаленный интерфейс
  - Локальный и удаленный интерфейсы
- **Интерфейсов каждого вида может быть более одного…**
- В качестве бизнес-интерфейсов расцениваются все реализуемые классом интерфейсы, кроме:
  - `java.io.Serializable`
  - `java.io.Externalizable`
  - Интерфейсы пакета `javax.ejb`

Пример бизнес-интерфейсов Session Bean

```
import javax.ejb.Remote;

@Remote
public interface FirstRemote {
    String operation1();
}
```

```
import javax.ejb.Remote;

@Remote
public interface SecondRemote {
    String operation2();
}
```

Пример класса Session Bean с двумя интерфейсами

```
import javax.ejb.Stateless;

@Stateless
public class FunnyBean implements
                          FirstRemote, SecondRemote {
  public String operation1() {
    return "Result of the first operation";
  }

  public String operation2() {
    return "Result of the second operation";
  }
}
```

Сессионный компонент без интерфейса

- Разрешается отдельно не описывать бизнес-интерфейс компонента
- Бизнес-интерфейсом считаются все публичные методы, объявленные в классе компонента
- Для доступа в таком режиме везде, где это требуется, вместо имени интерфейса используется имя класса
- Снабжается дополнительной аннотацией `@LocalBean`

Методы жизненного цикла в классе Session Bean

- Имена методов жизненного цикла в стандарте не закреплены
- Нужные методы определяются по наличию аннотаций
  - `javax.annotation.PostConstruct`
  - `javax.annotation.PreDestroy`
  - `javax.ejb.PrePassivate`
  - `javax.ejb.PostActivate`
- К методам предъявляются особые требования

Методы удаления компонента

- Помечаются аннотацией `javax.ejb.Remove`
- После вызова метода работа контейнера с экземпляром компонента завершается
- Метод удаления может быть указан в бизнес-интерфейсах, т.е. процесс удаления может инициировать клиент

Бизнес-методы в классе Session Bean

- Должны иметь модификатор `public`
- Не могут иметь модификаторов `static` и `final`
- Могут выбрасывать исключения
- Если к методу открыт доступ через удаленный интерфейс, то метод должен удовлетворять требованиями RMI
- Имя метода не должно начинаться с `ejb`

Параметры аннотаций @Stateful, @Stateless, @Singleton

- **`description`**
  Текстовое описание компонента
- **`name`**
  Простое имя (ejb-name) компонента
- **`mappedName`**
  Специфическое для платформы имя (JNDI-name)
  - Формат имени не регламентируется
  - Сервер даже не обязан учитывать это имя

Дескрипторы развертывания

- Дескрипторы развертывания существуют
  - Общий дескриптор ejb-jar.xml описывает ejb-модуль
  - Специфический дескриптор (например, sun-ejb-jar.xml) описывает специальные настройки конкретного сервера
- Дескрипторы развертывания могут отсутствовать

Дескрипторы развертывания

- Если присутствует общий дескриптор развертывания, то указанные в нем параметры перекрывают параметры, указанные с помощью аннотаций
  - Нельзя изменить тип компонента
  - Если значение параметра не указано в дескрипторе, оно извлекается из аннотаций
- Дескрипторы являются инструментом сборщиков модулей и приложений, позволяющим изменять параметры компонентов без изменения кода

Пример общего дескриптора развертывания

```
<?xml version="1.0" encoding="UTF-8"?>

<ejb-jar xmlns = "http://java.sun.com/xml/ns/javaee"
  version = "3.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd">
  <enterprise-beans>
    <session>
      <ejb-name>MyBean</ejb-name>
      <business-remote>session.FirstRemote</business-remote>
      <ejb-class>session.FunnyBean</ejb-class>
      <session-type>Stateless</session-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Доступ к компоненту через JNDI

- Объекты автоматически регистрируются в JNDI
- Имя компонента может быть указано специально и быть специфическим для конкретного сервера приложений
- Также присваиваются универсальные JNDI-имена, причем как для доступа по удаленным интерфейсам, так и по локальным

Определение специальных имен

- JNDI-имя может быть явно указано в специальном дескрипторе развертывания

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Application Server
9.0 EJB 3.0//EN" "http://www.sun.com/software/appserver/dtds/sun-ejb-jar_3_0-
0.dtd">
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>BeanClassName</ejb-name>
      <jndi-name>ManualJNDIName</jndi-name>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

- Сервер может самостоятельно присваивать дополнительные имена
  - Например, в GlassFish они имеют вид
    - `FullBeanClassName#FullBeanInterfaceName`
    - `ManualJNDIName#FullBeanInterfaceName`

Portable JNDI names

- Имя в пределах модуля
  - `java:module/BeanName[!FullInterfaceName]`
  - `java:module/NoInterface!beans.NoInterface`
- Имя в пределах приложения
  - `java:app/ModuleName/BeanName[!FullInterfaceName]`
  - `java:app/Test-ejb/RemoteTwice!beans.RemoteTwiceSecondRemote`
- Глобальное имя
  - `Java:global/ApplicationName/ModuleName/BeanName[!FullInterfaceName]`
  - `java:global/Test/Test-ejb/LocalOnly!beans.LocalOnlyLocal`

Получение доступа через JNDI

- Получение ссылки

```
Context context = new InitialContext();
LocalOnlyLocal localOnlyLocal = (LocalOnlyLocal)context.lookup(
    "java:app/Test-ejb/LocalOnly!beans.LocalOnlyLocal");
NoInterface noInterface = (NoInterface)context.lookup(
    "java:global/Test/Test-ejb/NoInterface!beans.NoInterface");
RemoteOnlyRemote remoteOnlyRemote =
    (RemoteOnlyRemote)context.lookup("MyManuaJNDIName");
```

- Использование ссылки

```
localOnlyLocal.someMethod();
noInterface.someOtherMethod(someParameter);
remoteOnlyRemote.someRemoteMethod(RMICompatibleParameter);
```

Доступ к компоненту через внедрение зависимостей

- Позволяет избежать постоянных обращений к JNDI
- Необходимое поле или метод просто получают аннотацию, помогающую реализовать связь с ресурсом
- Аннотация для доступа к EJB-компонентам:
  - `javax.ejb.EJB`

Внедрение зависимостей

- Работает в объектах, жизненным циклом которых управляет контейнер
- В начале жизни объекта перед выполнением бизнес-методов контейнер внедряет зависимости:
  - Присваивает значение поля
  - Вызывает метод установки значения

@EJB

- Позволяет внедрять ссылки на экземпляры EJB-компонентов
- Основные параметры:
  - name – JNDI-имя компонента
  - beanName – простое имя компонента
  - beanInterface – тип получаемой ссылки
- Если значение параметров не указано, оно восстанавливается по типу ссылки, куда производится внедрение

Особенности @EJB

- При внедрении с локальными интерфейсами в рамках одного приложения никаких дополнительных действий не требуется
- При использовании с компонентами EJB 3 обычно достаточно указания интерфейса
- При внедрении компонентов Session Stateful каждое новое внедрение дает новый объект

Примеры использования @EJB

```
//EJB 3, локальный доступ
@EJB
SomeBeanLocal sbl;

//EJB 3, удаленный доступ
@EJB
AnotherBeanRemote abr;
```

Асинхронные вызовы методов сессионных компонентов

- Обычный вызов метода компонента является блокирующим
- Если запрашиваемое действие является длительным, это может быть неудобно клиенту
- Возможное решение:
  - Использование Java Message System
  - Использование асинхронных методов

Асинхронные методы

- Аннотация `@Asynchronous`
  - Для метода – делает метод асинхронным
  - Для класса – делает все бизнес-методы асинхронными
- Возвращаемое значение метода
  - `void`
  - `java.util.concurrent.Future<t>` (интерфейс)
- В качестве конкретной реализации возвращается объект класса `javax.ejb.AsyncResult<T>`

Пример асинхронного метода

```
@Asynchronous
public Future<String> processPayment(Order order)
                      throws PaymentException {
    ...
    if (SessionContext.wasCancelled()) {
      // clean up
    } else {
      String status = ...;
      return new AsyncResult<String>(status);
    }
    ...
}
```

Интерфейс Future<T>

- `boolean cancel(boolean mayInterruptIfRunning)`
  - Прерывает выполнение асинхронного
  - Или отменяет прерывание
- `boolean isCancelled()`
  - Проверяет, был ли прерван асинхронный вызов
- `boolean isDone()`
  - Проверяет, завершено ли выполнение вызова
  - Не блокирующий метод

Интерфейс Future<T>

- `T get()`
  - Блокирующий метод
  - Возвращает результат выполнения вызова
  - Выбрасывает исключение `ExecutionException`, если вызов выбросил исключение

- `T get(long timeout, TimeUnit unit)`
  - Блокирующий метод
  - Возвращает результат выполнения вызова, но ждёт не более указанного времени, иначе выбрасывает `TimeOutException`
  - Выбрасывает исключение `ExecutionException`, если вызов выбросил исключение

Спасибо за внимание!

Дополнительные источники

- Соломон, М.К. Oracle. Программирование на языке Java [Текст] / Мартин К. Соломон, Нирва Морисco-Леруа, Джули Басу. – М. : Лори, 2010. – 512 с.
- Курванян, Б. Программирование web-приложений на языке Java [Текст] / Буди Курванян. – М. : Лори, 2009. – 880 с.
- JavaEE APIs & Docs [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/apis-139520.html, дата доступа: 14.10.2013.
- JavaEE Tutorials [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html, дата доступа: 14.10.2013.
- Enterprise JavaBeans Technology [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/ejb/index.html, дата доступа: 14.10.2013.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Asynchronous JavaScript and XML

Занятие 15

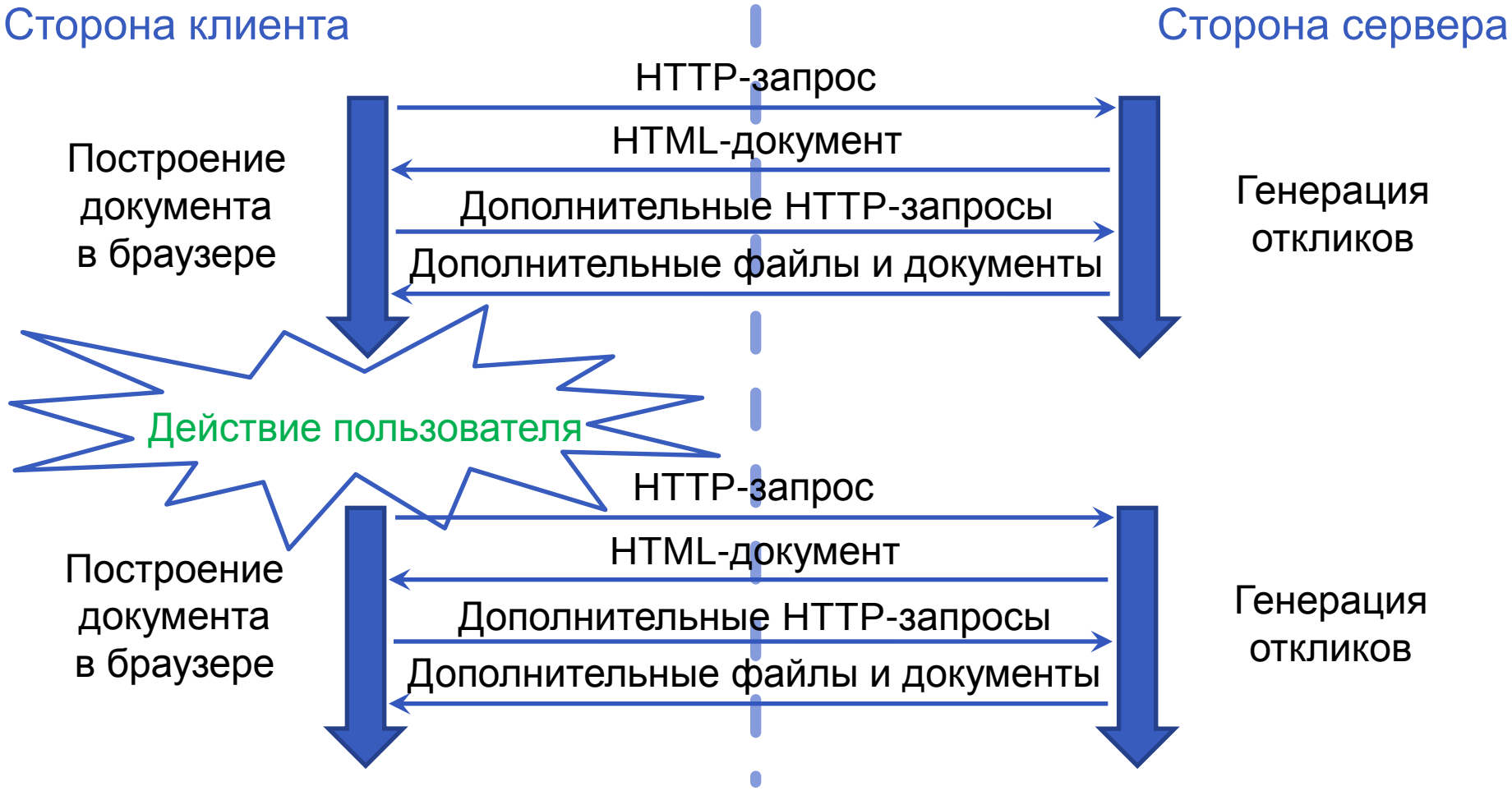
Гаврилов А.В.

Самара
2013

План занятия

- AJAX
- Виды передаваемых данных
- Класс XMLHttpRequest
- Отладка web-приложений
- jQuery

Общий порядок работы с документами



Пример
Класс для работы с единицами измерения (1)

```
package weights;

import java.util.*;

public class MassUnit {
    private String name;
    private double weightInGrams;
    private static final Map units = new HashMap();
    private static final Collection allUnits;

    static {
        MassUnit mu;
        mu = new MassUnit("kilogram", 1000); units.put(mu.getName(), mu);
        mu = new MassUnit("pound", 453.59237); units.put(mu.getName(), mu);
        mu = new MassUnit("ounce", 28.349523125); units.put(mu.getName(), mu);
        mu = new MassUnit("carat", 0.2); units.put(mu.getName(), mu);
        allUnits = units.values();
    }

    ...
```

Пример
Класс для работы с единицами измерения (2)

```
...

public MassUnit(String name, double weightInGrams) {
    if (weightInGrams <= 0 || name == null) {
        throw new IllegalArgumentException();
    }
    this.name = name;
    this.weightInGrams = weightInGrams;
}

public String getName() {
    return name;
}

public double getWeightInGrams() {
    return weightInGrams;
}

...
```

Пример
Класс для работы с единицами измерения (3)

```
    ...

    public static String[][] getConversion(double weight, String massUnitName) {
        MassUnit mu = (MassUnit) units.get(massUnitName.toLowerCase());
        if (mu == null)
            throw new IllegalArgumentException();
        weight *= mu.getWeightInGrams();
        String[][] result = new String[allUnits.size()][2];
        Iterator iterator = allUnits.iterator();
        MassUnit currentUnit;
        for (int current = 0; iterator.hasNext(); current++) {
            currentUnit = (MassUnit) iterator.next();
            result[current][0] = currentUnit.getName();
            result[current][1] = Double.toString(weight /
                                    currentUnit.getWeightInGrams());
        }
        return result;
    }
}
```

Пример
HTML-страница

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Weights converter</title>
    </head>
    <body>
        <form action="converter" method="get">
            <p>
                <input type="text" size="40" name="weight">
                <select size="1" name="unit">
                    <option>kilogram</option>
                    <option>pound</option>
                    <option>ounce</option>
                    <option>carat</option>
                </select>
                <input type="submit" value="convert"/>
            </p>
        </form>
    </body>
</html>
```

Пример
Фрагмент кода сервлета (1)

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    try {
        boolean correct = true;
        double weight;
        String unit;
        String[][] conversions = null;
        try {
            weight =
                 Double.parseDouble(request.getParameter("weight"));
            unit = request.getParameter("unit");
            conversions = MassUnit.getConversion(weight, unit);
        } catch (Throwable ex) {
            correct = false;
        }
```

Пример
Фрагмент кода сервлета (2)

```
    if (correct) {
            out.print("<html><body><table>");
            for (int i = 0; i < conversions.length; i++) {
                out.print("<tr><td>");
                out.print(conversions[i][1]);
                out.print("</td><td>");
                out.print(conversions[i][0]);
                out.print("</td></tr>");
            }
            out.print("</table></body></html>");
        } else {
            out.print("<html><body>Error!</body></html>");
        }
    } finally {
        out.close();
    }
}
```

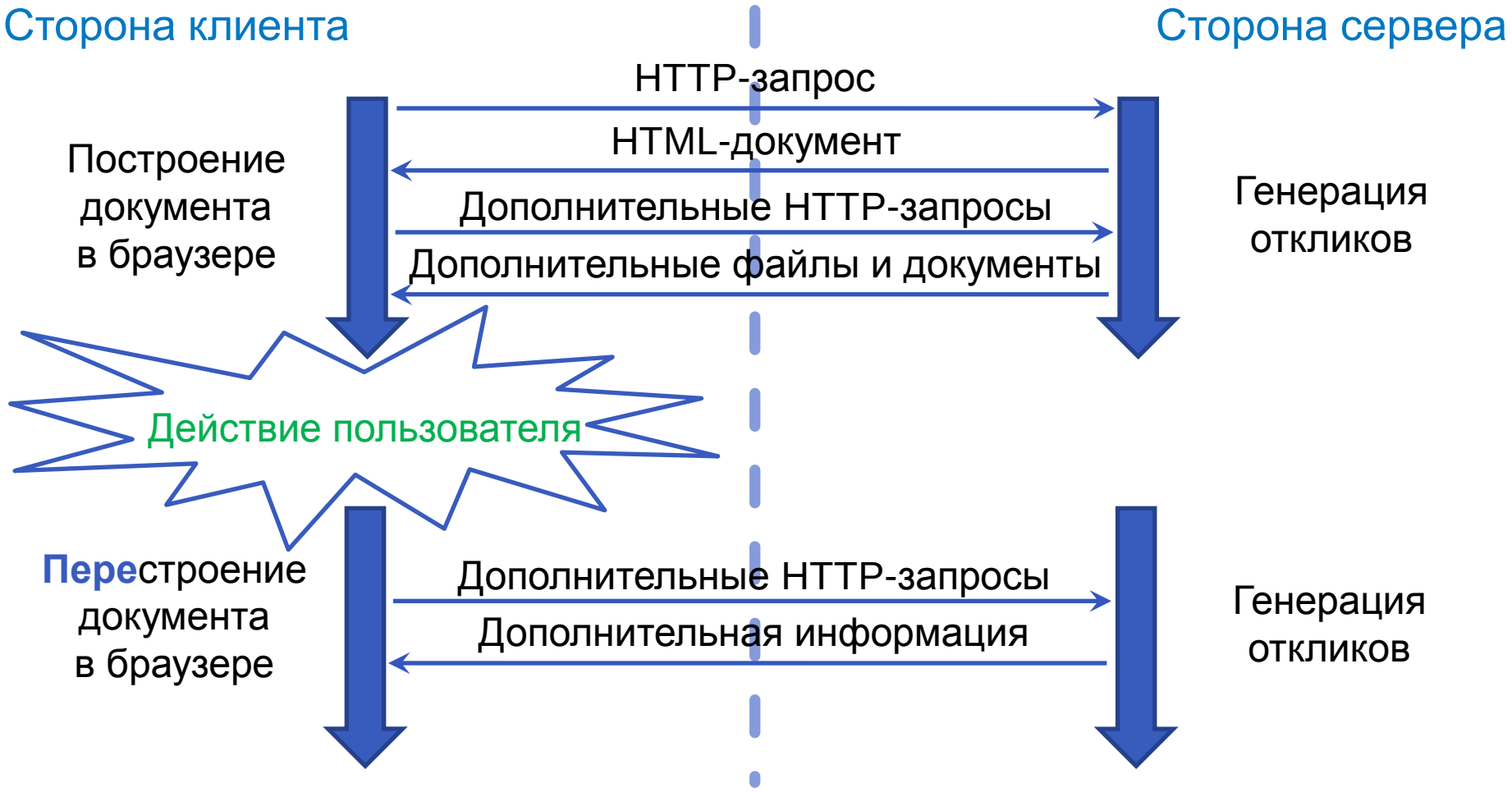
Результат

- Исходная страница

- Результат нажатия на кнопку

5.0	kilogram
176.36980974790205	ounce
25000.0	carat
11.023113109243878	pound

А что если бы?..


Сторона клиента
Сторона сервера
HTTP-запрос
HTML-документ
Построение документа в браузере
Дополнительные HTTP-запросы
Дополнительные файлы и документы
Генерация откликов
Действие пользователя
Перестроение документа в браузере
Дополнительные HTTP-запросы
Дополнительная информация
Генерация откликов

Пример
HTML-страница

```
<html>
    <head>
        <!-- ... -->
        <script type="text/javascript" src="converter.js"></script>
    </head>
    <body onload="init()">
        <form action="">
            <p>
                <input type="text" size="40" id="weight-field">
                <select size="1" id="unit-selector">
                    <option>kilogram</option> <option>pound</option>
                    <option>ounce</option> <option>carat</option>
                </select>
                <input type="submit" value="convert"
                       onclick="doConversion(); return false;">
            </p>
            <p id="result-place"></p>
        </form>
    </body>
</html>
```

Пример

Фрагмент кода сервлета

```
...
response.setContentType("text/plain;charset=UTF-8");
...
if (correct) {
    for (int i = 0; i < conversions.length; i++) {
        out.print(conversions[i][0]);
        out.print(" ");
        out.print(conversions[i][1]);
        out.print(", ");
    }
} else {
    out.println("");
}
...
```

Пример
converter.js

```
var req;
var weightField;
var unitSelector;

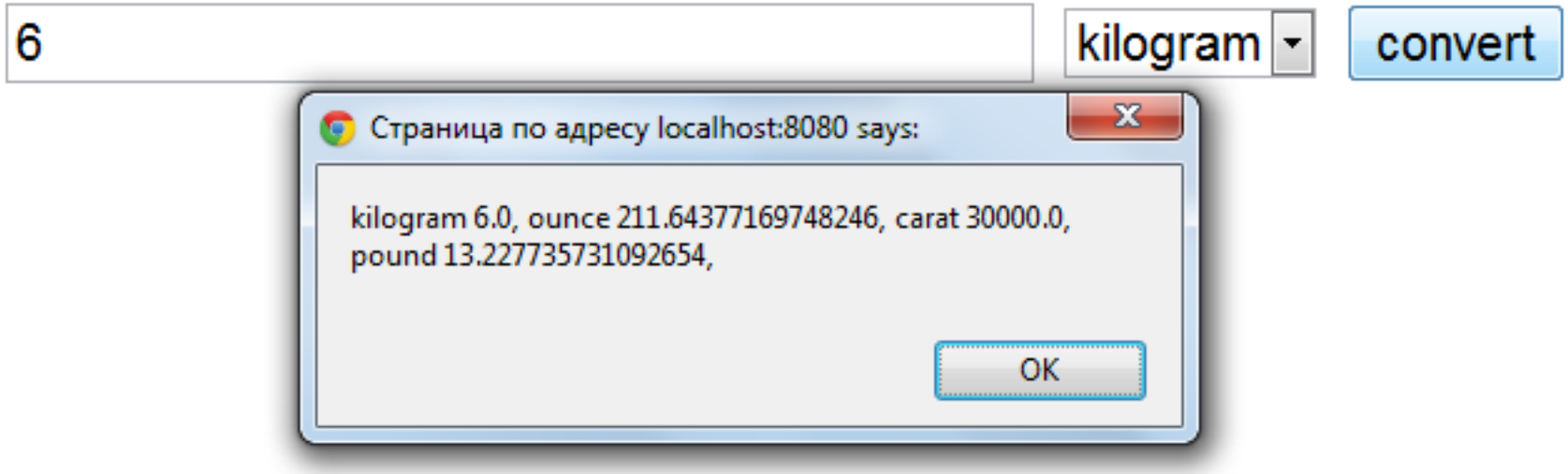
function init() {
    weightField = document.getElementById("weight-field");
    unitSelector = document.getElementById("unit-selector");
}

function doConversion() {
    var url = "converter?unit=" + escape(unitSelector.value) +
              "&weight=" + escape(weightField.value);
    req = new XMLHttpRequest();
    req.open("GET", url, false);
    req.send(null);
    alert(req.responseText);
}
```

Результат

- Исходная страница

- Результат нажатия на кнопку



История развития подхода

- 1991 – Страницы загружаются полностью
- 1995 – Java Applets, могут асинхронно взаимодействовать с сервером
- 1996 – Internet Explorer, тег iframe, допускающий асинхронную загрузку
- 1999 – Internet Explorer 5, ActiveX-объект XMLHTTP
  - Mozilla, Safari и Opera реализуют аналогичные возможности в виде JavaScript-объекта XMLHttpRequest
  - InternetExplorer 7 также вводит поддержку XMLHttpRequest
- 2000-2005 – создание приложений (Microsoft, Google, …)
- 2005 – появление термина AJAX (Джесси Джеймс Гаррет)
- 2006 – Объект XMLHttpRequest становится частью стандарта (W3C)

AJAX

- Asynchronous JavaScript and XML
  - Допускает синхронные вызовы
  - Язык – необязательно JavaScript
  - Формат передачи данных – необязательно XML

- Основные идеи
  - Динамическое обращение к серверу
  - Динамическое изменение содержимого страницы
  - Центральный объект – XMLHttpRequest
  - Для асинхронного вызова указывается функция обратного вызова, в которой обычно проверяются параметры выполнения запроса

Порядок работы приложения

- Сервер
  - На сервере реализовано что-нибудь, обрабатывающее HTTP-запросы
  - Оно как-то генерирует какой-то отклик в зависимости от параметров
- Клиент
  - Для элемента(ов) документа пишется обработчик события(й)
  - В обработчике производится запрос на сервер
  - Получается и интерпретируется отклик сервера
  - Средствами JavaScript изменяется документ в браузере

Популярные форматы передачи данных

- Простой текст
  - Небольшой объем данных
  - Неиерархические данные
- HTML
  - Пересылается фрагмент HTML-документа
  - Отклик можно встроить прямо в документ
- XML
  - Данные как таковые
  - Можно обрабатывать в стиле DOM
- JSON
  - Данные в виде описания JavaScript-объекта
  - Обращение к свойствам JavaScript-объекта
- JavaScript
  - Выполняемый на стороне клиента код

Пример (HTML)
Фрагмент кода сервлета

```
...
response.setContentType("text/html;charset=UTF-8");
...
if (correct) {
    out.print("<table>");
    for (int i = 0; i < conversions.length; i++) {
        out.print("<tr><td>");
        out.print(conversions[i][1]);
        out.print("</td><td>");
        out.print(conversions[i][0]);
        out.print("</td></tr>");
    }
    out.println("</table>");
} else {
    out.println("");
}
...
```

Пример (HTML)
Фрагмент HTML-страницы

```
<body onload="init()">
    <form action="">
        <p>
            <input type="text" size="40" id="weight-field"
                   onkeyup="doConversion();">
            <select size="1" id="unit-selector"
                   onchange="doConversion()">
                <option>kilogram</option>
                <option>pound</option>
                <option>ounce</option>
                <option>carat</option>
            </select>
        </p>
        <p id="result-place"/>
    </form>
</body>
```

Пример (HTML)
converter.js (1)

```
var req;
var resultPlace;
var weightField;
var unitSelector;

function init() {
    weightField = document.getElementById("weight-field");
    unitSelector = document.getElementById("unit-selector");
    resultPlace = document.getElementById("result-place");
}

function callback() {
    if (req.readyState == 4) {
        if (req.status == 200) {
            resultPlace.innerHTML = req.responseText;
        }
    }
}
```

Пример (HTML)
converter.js (2)

```
function doConversion() {
    // Формируем адрес с параметрами
    var url = "converter?unit=" +
              escape(unitSelector.value) +
              "&weight=" + escape(weightField.value);

    // Создаем объект запроса
    req = new XMLHttpRequest();

    // Указываем метод, адрес и асинхронность
    req.open("GET", url, true);

    // Указываем функцию для обратного вызова
    req.onreadystatechange = callback;

    // Отправляем запрос
    req.send(null);
}
```

Пример (XML)
Фрагмент кода сервлета

```
...
response.setContentType("text/xml;charset=UTF-8");
...
if (correct) {
    out.print("<converter>");
    for (int i = 0; i < conversions.length; i++) {
        out.print("<weight><value>");
        out.print(conversions[i][1]);
        out.print("</value><unit>");
        out.print(conversions[i][0]);
        out.print("</unit></weight>");
    }
    out.println("</converter>");
} else {
    out.println("<converter/>");
}
...
```

Пример (XML)
Фрагмент HTML-страницы

```
<body onload="init()">
    <form action="">
        <p>
            <input type="text" size="40" id="weight-field"
                   onkeyup="doConversion();">
            <select size="1" id="unit-selector"
                   onchange="doConversion()">
                <option>kilogram</option>
                <option>pound</option>
                <option>ounce</option>
                <option>carat</option>
            </select>
        </p>
        <table id="conversion-table"><tr><td></td></tr></table>
    </form>
</body>
```

Пример (XML)
converter.js (1)

```
var req;
var conversionTable;
var weightField;
var unitSelector;

function init() {
    weightField = document.getElementById("weight-field");
    conversionTable = document.getElementById("conversion-table");
    unitSelector = document.getElementById("unit-selector");
}

function doConversion() {
    var url = "converter?unit=" + escape(unitSelector.value) +"&weight="
+ escape(weightField.value);
    req = new XMLHttpRequest();
    req.open("GET", url, true);
    req.onreadystatechange = callback;
    req.send(null);
}
```

Пример (XML)
converter.js (2)

```
function callback() {
    clearTable();
    if (req.readyState == 4) {
        if (req.status == 200) {
            parseMessages(req.responseXML);
        }
    }
}

function appendWeight(weight, unit) {
    var row; var cell;
    conversionTable.style.display = 'table';
    row = document.createElement("tr");
    cell = document.createElement("td");
    cell.textContent = weight; row.appendChild(cell);
    cell = document.createElement("td");
    cell.textContent = unit; row.appendChild(cell);
    conversionTable.appendChild(row);
}
```

Пример (XML)
converter.js (3)

```
function parseMessages(responseXML) {
    if (responseXML == null) {
        return false;
    } else {
        var converter = responseXML.getElementsByTagName("converter")[0];
        if (converter.childNodes.length > 0) {
            for (loop = 0; loop < converter.childNodes.length; loop++) {
                var weight = converter.childNodes[loop];
                var unit = weight.getElementsByTagName("unit")[0];
                var value = weight.getElementsByTagName("value")[0];
                appendWeight(value.childNodes[0].nodeValue,
                             unit.childNodes[0].nodeValue);
            }
        }
        return true;
    }
}
```

Пример (XML)
converter.js (4)

```
function clearTable() {
    if (conversionTable.getElementsByTagName("tr").length > 0) {
        conversionTable.style.display = 'none';
        for (loop = conversionTable.childNodes.length -1;
             loop >= 0 ;
             loop--) {
            conversionTable.removeChild(
              conversionTable.childNodes[loop]);
        }
    }
}
```

JSON

- JavaScript Object Notation
- Формат представления данных, основанный на подмножестве языка JavaScript (литералы и массивы объектов)
- Предложен Дугласом Крокфордом в 2002 г.
- Несмотря на основанность на JavaScript, является форматом, не зависящим от языка

Две формы создания объекта в JavaScript

```
var oAuto = new Object();
oAuto.firm = "Audi";
oAuto.model = "A6";
oAuto.year = 2008;
oAuto.price = 78000;
```

```
var oAuto = {
   "firm":"Audi",
   "model":"A6",
   "year":2008,
   "price":78000
};
```

Пример описания на JSON

- { описание объекта}
- [элементы массива]
- "свойство" : значение

```
{ "general":
  {
    "cars":[
      {
        "firm":"BMW",
        "model":"X5",
        "year":2007,
        "price":99000
      },
      {
        "firm":"Audi",
        "model":"A6",
        "year":2008,
        "price":78000
      }
    ]
  }
}
```

Преобразование в JSON и обратно

- Серверная сторона
  - Существуют библиотеки преобразования данных для различных языков
  - http://www.json.org/json-ru.html
- Клиентская сторона
  - Прямое вычисление JavaScript-кода

```
var oAuto = eval("(" + sData + ")");
```

  - Использование библиотек

```
var oAuto= JSON.parse(sData);
```

Пример (JSON)
Фрагмент кода сервлета

```
...
response.setContentType("application/json;charset=UTF-8");
...
org.json.JSONWriter jw = new org.json.JSONWriter(out);
jw.array();
if (correct) {
    for (int i = 0; i < conversions.length; i++) {
        jw.object();
        jw.key("unit");
        jw.value(conversions[i][0]);
        jw.key("weight");
        jw.value(conversions[i][1]);
        jw.endObject();
    }
}
jw.endArray();
...
```

Пример (JSON)

Фрагмент converter.js

```
function callback() {
    clearTable();
    if (req.readyState == 4) {
        if (req.status == 200) {
            parseMessages(req.responseText);
        }
    }
}
function parseMessages(responseText) {
    if (responseText == null) {
        return false;
    } else {
        var converter = JSON.parse(responseText);
        if (converter.length > 0) {
            for (loop = 0; loop < converter.length; loop++) {
                appendWeight(converter[loop].weight, converter[loop].unit);
            }
        }
        return true;
    }
}
```

Методы класса XMLHttpRequest

Метод	Краткое описание
abort()	Отменяет текущий запрос, удаляет все заголовки, ставит текст ответа сервера в null.
getAllResponseHeaders()	Возвращает полный список HTTP-заголовков в виде строки. Заголовки разделяются знаками переноса (CR+LF). Если флаг ошибки равен true, возвращает пустую строку. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
getResponseHeader(headerName)	Возвращает значение указанного заголовка. Если флаг ошибки равен true, возвращает null. Если заголовок не найден, возвращает null. Если статус 0 или 1, вызывает ошибку INVALID_STATE_ERR.
open(method, URL, async)	Определяет метод, URL и синхронность/асинхронность выполнения запроса.
send(content)	Отправляет запрос на сервер.
setRequestHeader(label, value)	Добавляет HTTP-заголовок к запросу.

Свойства класса XMLHttpRequest

Свойство	Тип	Краткое описание
onreadystatechange	EventListener	Обработчик события, которое происходит при каждой смене состояния объекта.
readyState	unsigned short	Текущее состояние объекта (0 — не инициализирован, 1 — открыт, 2 — отправка данных, 3 — получение данных и 4 — данные загружены).
responseText	DOMString	Текст ответа на запрос. Если состояние не 3 или 4, возвращает пустую строку.
responseXML	Document	Текст ответа на запрос в виде XML, который затем может быть обработан посредством DOM. Если состояние не 4, возвращает null.
status	unsigned short	HTTP-статус в виде числа (404 — «Not Found», 200 — «OK» и т. д.)
statusText	DOMString	Статус в виде строки («Not Found», «OK» и т. д.). Если статус не распознан, браузер пользователя должен вызвать ошибку INVALID_STATE_ERR.

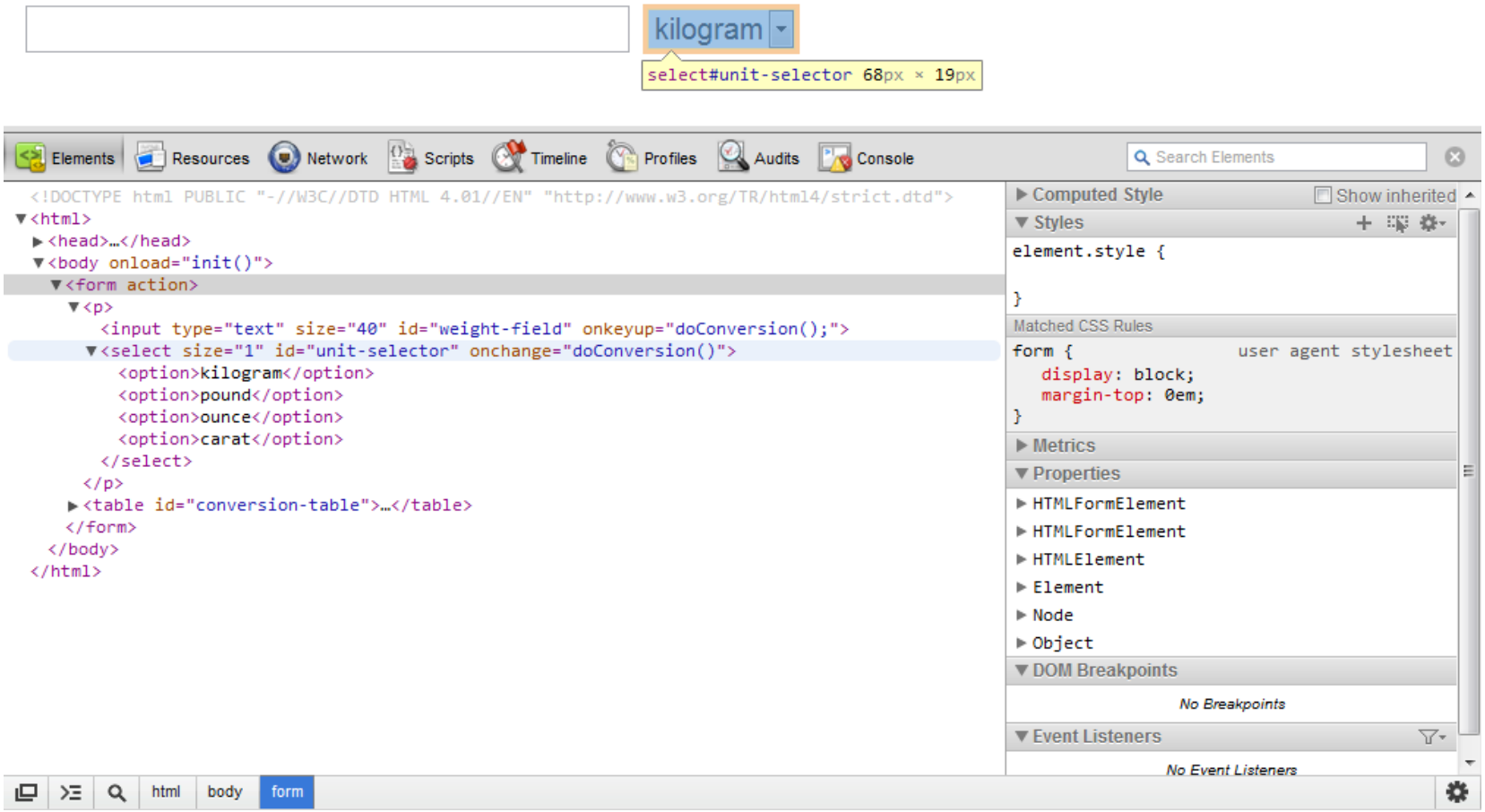
Как отлаживать?..

- На стороне сервера
  - Административная консоль сервера
  - Вывод сообщений в журналы сервера
  - Использование сервера в режиме отладки
- На стороне клиента
  - Тривиальные средства JavaScript
  - Специализированные утилиты, входящие в состав браузеров или поставляющиеся в виде расширений браузеров

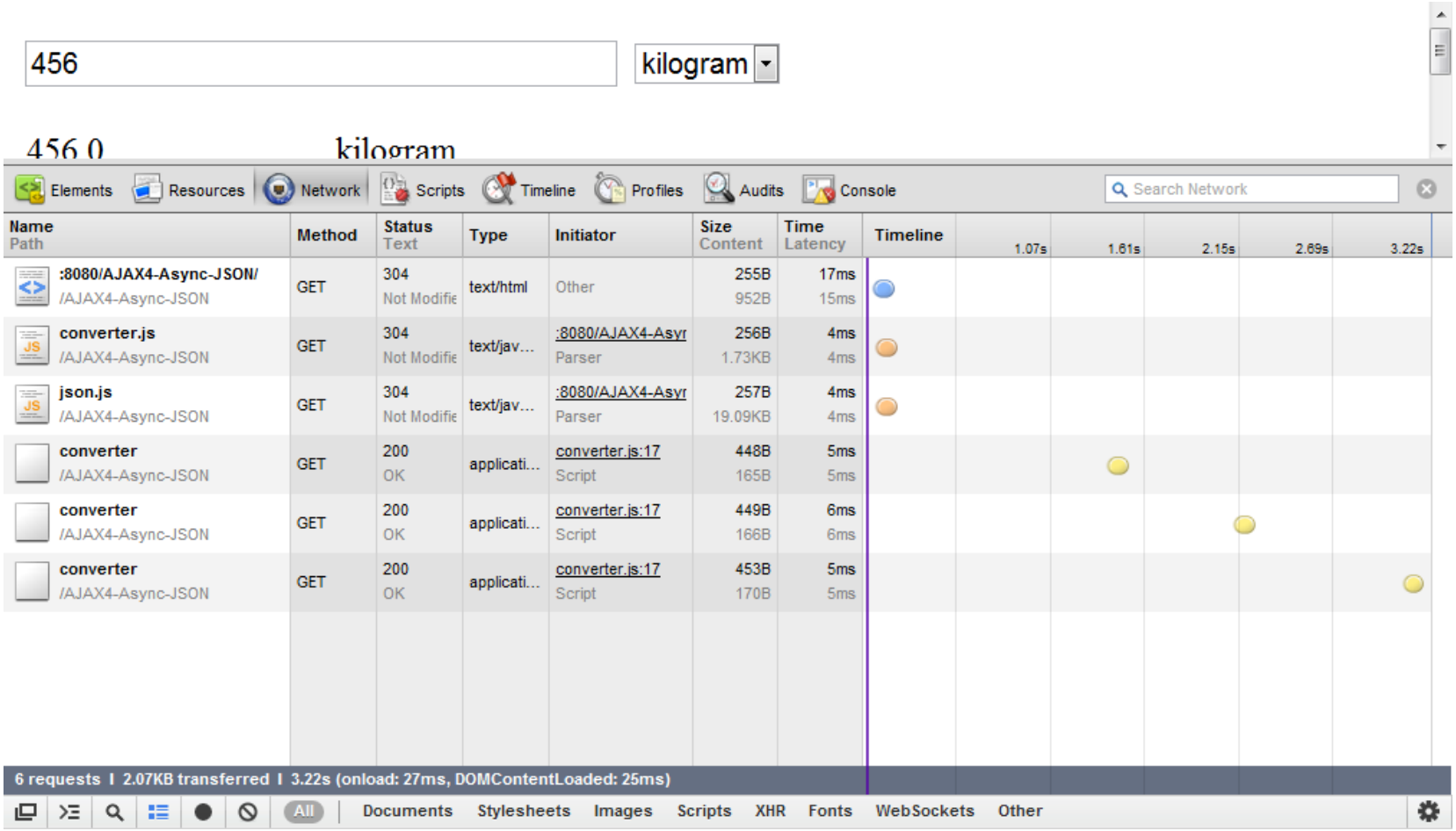
Инструменты разработчика в браузерах

- Типичные возможности
  - Просмотр структуры HTML/CSS
  - Отслеживание и управление кэшем браузера
  - Мониторинг сетевых запросов
  - Редактирование и отладка JavaScript-кода
  - Анализ, профилирование и оптимизация кода страниц
- Примеры
  - Internet Explorer – Средства разработчика, HTTPWatch
  - FireFox – FireBug, HTTPWatch
  - Chrome – Инструменты разработчика

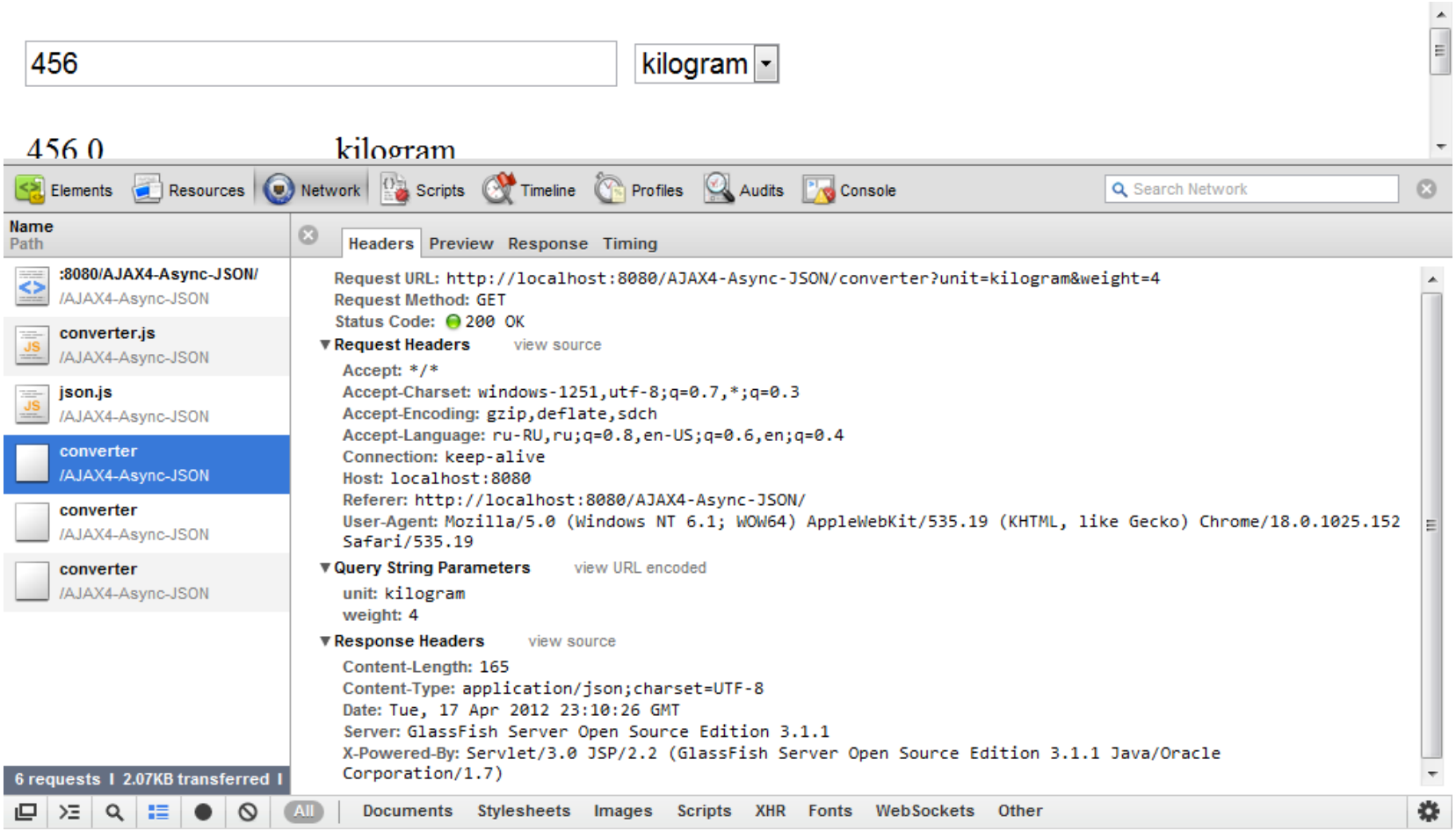
Пример работы с HTML/CSS



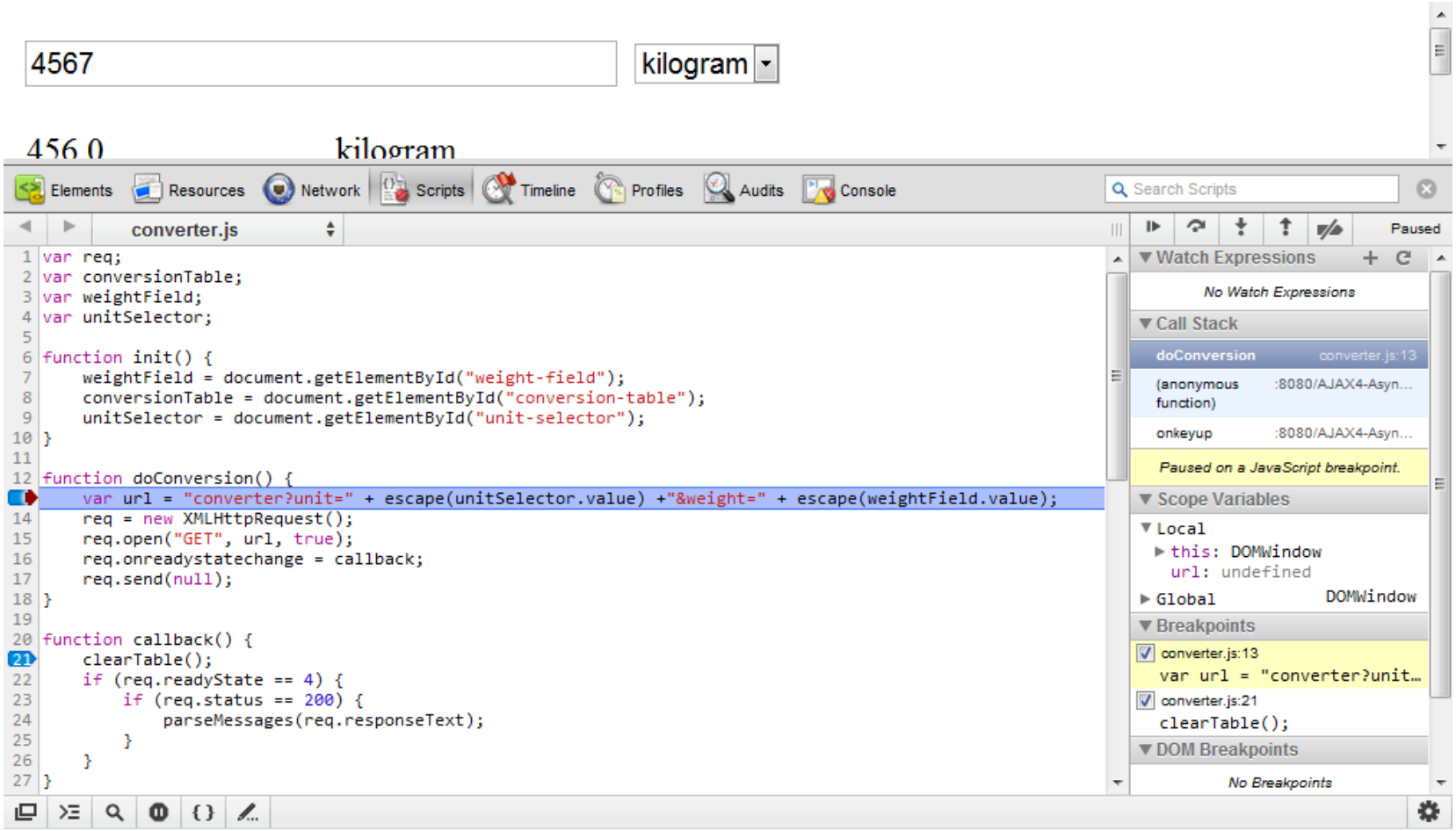
Пример анализа сетевых запросов



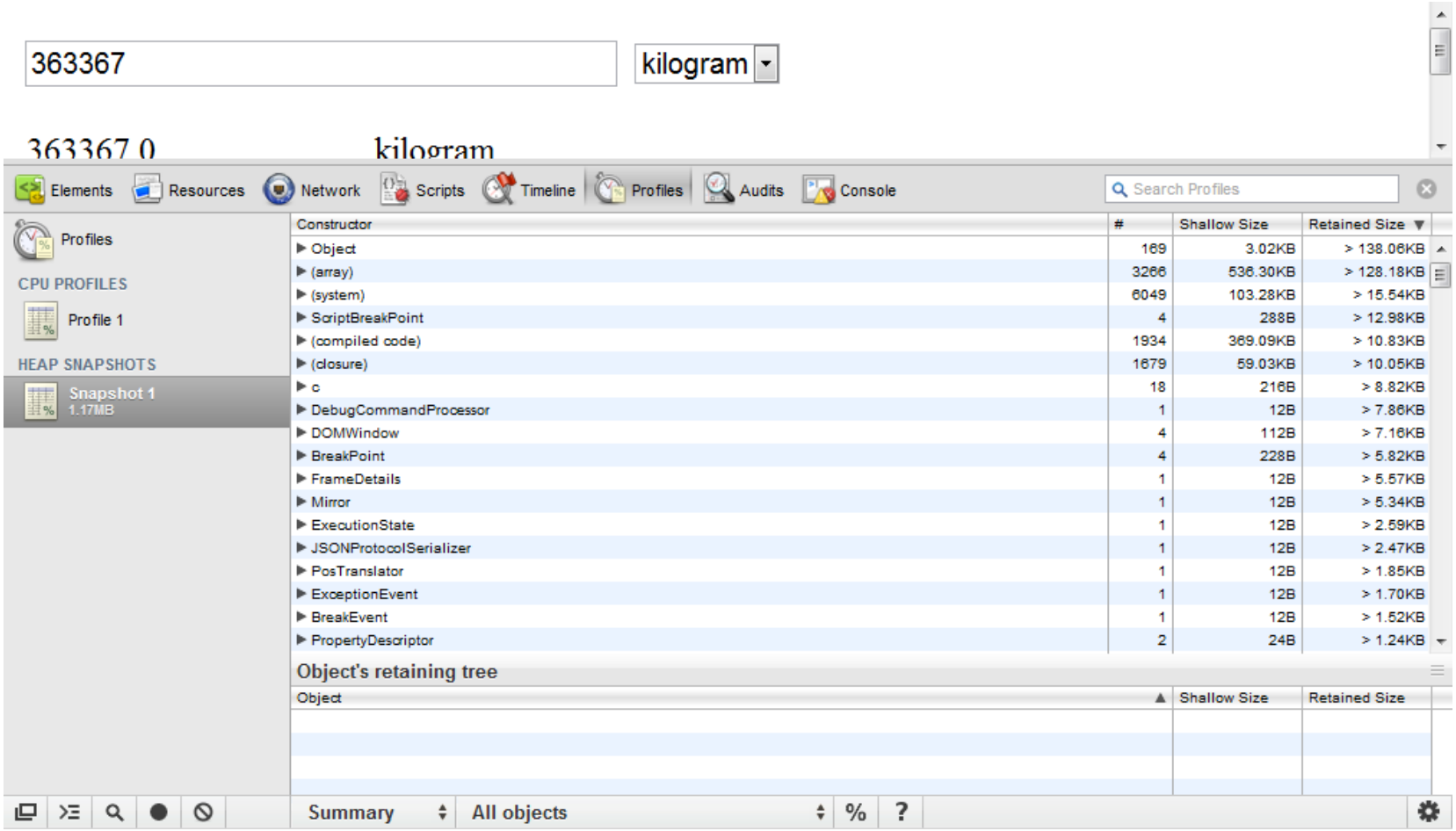
Пример анализа конкретного сетевого запроса



Пример отладки JavaScript-кода



Пример профилирования JavaScript-кода



Промежуточные выводы

- Преимущества
  - Уменьшение объемов передаваемых данных?
  - Уменьшение нагрузки на сервер?
  - Ускорение реакции интерфейса
  - Асинхронная реакция интерфейса
- Недостатки
  - Проблемы с интеграцией со стандартными средствами браузеров
  - Недоступность динамического содержимого поисковым системам
  - Требуются иные методики учета статистики сайтов
  - Требование выполнения JavaScript на стороне клиента
  - **Значительное усложнение архитектуры проекта**

Промежуточные выводы

- Чистый JavaScript в работе с документом не очень удобен
- Объект XMLHttpRequest тоже не блещет удобством в работе
- Однако AJAX позволяет создавать принципиально новый вид web-приложений
- Вывод: нужны фреймворки и библиотеки поверх JavaScript и AJAX

jQuery

- JavaScript-библиотека для решения типовых задач на стороне клиента
- Кроссбраузерная библиотека
- Ориентирована на «ненавязчивый» JavaScript, при котором структура документа (HTML) отделена от его поведения (JavaScript)
- Поставляется в двух версиях
  - Standard (около 250Кб) – применяется на этапе разработки приложений
  - Mini (около 94Кб, 32Кб в архиве) – применяется в готовых приложениях
- Начало работ – 2006 г. (Джон Резиг)
- Текущая версия – 1.8.3
- http://jquery.com/

jQuery

- Возможности
  - Упрощение работы с HTML-документом
  - Упрощение работы с CSS
  - Работа с событиями
  - Выполнение анимации
  - Создание AJAX-приложений
- Основная идея
  - Find things
  - Do stuff

Как действовать

- Центральная функция – $
  - Это псевдоним функции jQuery
  - Параметры – условия выбора объектов (find things)
  - Методы – действия над объектами (do stuff)
- Целесообразно начинать работу тогда, когда документ полностью загружен в память

```
$(document).ready(function(){
   // Какой-то код
   }
)
```

Примеры селекторов

```
$('#sidebar');     // выбор элемента с id = sidebar
$('.post');        // выбор элементов с class = post
$('div#sidebar'); // выбор элемента div с id = sidebar
$('div.post');     // выбор элементов div с class = post
$('div span');     // выбор всех span элементов в элементах div
$('div').find('span'); // выбор всех span элементов в элементах div
$('div > span');       // выбор всех span элементов в элементах
                       // div, где span является прямым потомком div'a
$('div, span');        // выбор всех div и span элементов
$('span + img');       // выбор всех img элементов перед которыми идут
                       // span элементы
$('span ~ img');       // выбор всех img элементов после первого
                       // элемента span
$('#banner').prev();   // выбор предыдущего элемента от найденого
$('#banner').next();   // выбор следующего элемента от найденого
```

Примеры селекторов

```
$('*');                    // выбор всех элементов
$('p > *');                // выбор всех потомков элементов p
$('p').children();         // --
$('p').parent();           // выбор всех прямых предков элементов p
$('* > p');                // выбор всех предков элементов
$('p').parents();          // --
$('p').parents('div'); // выбор всех предков элемента p, которые есть
                           // div
$('div:first');      // выбираем первый div в доме
$('div:last');       // выбираем последний div в доме
$('div:not(.red)'); // выбираем div'ы, у которых нет класса red
$('div:even');       // выбираем четные div'ы
$('div:odd');        // выбираем нечетные div'ы
...
```

Примеры событий

- change – изменение значения элемента (значение, при потери фокуса, элемента отличается от изначального, при получении фокуса)
- click – клик по элементу (порядок событий – mousedown, mouseup, click)
- dblclick – двойной клик по элементу
- resize – изменение размеров элементов
- scroll – скроллинг элемента
- select – выбор текста (актуален только для input[type=text] и textarea)
- submit — отправка формы
- focus – фокус на элементе – актуально для input[type=text], но в современных браузерах работает и с другими элементами
- blur – фокус ушел с элемента – актуально для input[type=text] – срабатывает при клике по другому элементу на странице или по событию клавиатуры (к примеру переключение по tab)
- focusin – фокус на элементе, данное событие срабатывает на предке элемента, для которого произошло событие focus
- focusout – фокус ушел с элемента, данное событие срабатывает на предке элемента, для которого произошло событие blur

Примеры событий

- keydown – нажатие клавиши на клавиатуре
- keypress – нажатие клавиши на клавиатуре (порядок событий – keydown, keypress, keyup)
- keyup – отжатие клавиши на клавиатуре
- load – загрузка элемента (img)
- unload – выгрузка элемента (window)
- mousedown – нажатие клавиши мыши
- mouseup – отжатие клавиши мыши
- mousemove – движение курсора
- mouseenter – наведение курсора на элемент, не срабатывает при переходе фокуса на дочерние элементы
- mouseleave – вывод курсора из элемента, не срабатывает при переходе фокуса на дочерние элементы
- mouseover – наведение курсора на элемент
- mouseout – вывод курсора из элемента

Функция $.ajax

- Главная функция для выполнения AJAX
- Основные параметры
  - async – асинхронность запроса, по умолчанию true
  - cache – вкл/выкл кэширование данных браузером, по умолчанию true
  - contentType – по умолчанию “application/x-www-form-urlencoded”
  - data – передаваемые данные
  - dataType – тип данных возвращаемых в callback-функцию (xml, html, script, json, text, _default)
  - processData – по умолчанию отправляемые данные заворачиваются в объект, и отправляются как “application/x-www-form-urlencoded”
  - scriptCharset – кодировка
  - timeout – время таймаут в миллисекундах
  - type – GET либо POST
  - url – url запрашиваемой страницы

Функция $.ajax

- Обработка возвращаемых из запроса данных производится в обработчиках событий объекта:
  - beforeSend – срабатывает перед отправкой запроса
  - error – если произошла ошибка
  - success – если ошибок не возникло
  - complete – срабатывает по окончанию запроса
- Упрощенные методы
  - $.load – загрузка HTML-кода в выбранный элемент на странице
  - $.get – AJAX-запрос через HTTP-метод get
  - $.post – AJAX-запрос через HTTP-метод post
  - $.getScript – AJAX-запрос, получающий от сервера JavaScript и выполняющий его
  - …

Пример (plain text, HTML)

HTML-страница

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Weights converter</title>
        <script type="text/JavaScript" src="jquery.js"></script>
        <script type="text/javascript" src="converter.js"></script>
    </head>
    <body>
        <form action="">
            <p>
                <input type="text" size="40" id="weight-field">
                <select size="1" id="unit-selector">
                    <option>kilogram</option> <option>pound</option>
                    <option>ounce</option> <option>carat</option>
                </select>
                <input type="submit" value="convert" id="convert-button"/>
            </p>
            <p id="result-place"></p>
        </form>
    </body>
</html>
```

Пример (plain text)
converter.js

```
$(document).ready(function(){
    $("#convert-button").click(function(){
      $("#result-place").load("converter",
        "weight="
        + $("#weight-field").val()
        + "&unit="
        + $("#unit-selector").val());
      return false;
    });
  }
)
```

Пример (HTML)
converter.js

```
function refresh(){
    $("#result-place").load("converter", "weight=" +
                $("#weight-field").val() + "&unit=" +
                $("#unit-selector").val());
    return false;
}

$(document).ready(function(){
    $("#unit-selector").change(function() {
        return refresh()});
    $("#weight-field").keyup(function() {
        return refresh()});
  }
)
```

Пример (XML, JSON)

HTML-страница

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Weights converter</title>
        <script type="text/JavaScript" src="jquery.js"></script>
        <script type="text/javascript" src="converter.js"></script>
    </head>
    <body>
        <form action="">
            <p>
                <input type="text" size="40" id="weight-field">
                <select size="1" id="unit-selector">
                    <option>kilogram</option> <option>pound</option>
                    <option>ounce</option> <option>carat</option>
                </select>
            </p>
            <table id="conversion-table"><tr><td></td></tr></table>
        </form>
    </body>
</html>
```

Пример (XML)
converter.js

```
function refresh(){
    $.get("converter", {weight: $("#weight-field").val(),
                        unit: $("#unit-selector").val()},
        function (xml) {
            $("#conversion-table tr").remove();
            $(xml).find("weight").each(function() {
                $("#conversion-table").append("<tr><td>" +
                $(this).find("value").text() + "</td><td>"  +
                $(this).find("unit").text() + "</td></tr>");
        });
    }, "xml");
    return false;
}

$(document).ready(function(){
    $("#unit-selector").change(function() {return refresh()});
    $("#weight-field").keyup(function() {return refresh()});
})
```

Пример (JSON)
converter.js

```
function refresh(){
    $.get("converter", {weight: $("#weight-field").val(),
                        unit: $("#unit-selector").val()},
        function (json) {
            $("#conversion-table tr").remove();
            $(json).each(function() {
                $("#conversion-table").append("<tr><td>" +
                this.weight + "</td><td>"  + this.unit +
                "</td></tr>");
        });
    }, "json");
    return false;
}

$(document).ready(function(){
    $("#unit-selector").change(function() {return refresh()});
    $("#weight-field").keyup(function() {return refresh()});
})
```

Выводы

- AJAX – это суровая действительность и неотъемлемая часть Web 2.0
- Клиентская часть становится значительно сложнее
- Серверная часть не сложнее с технологической точки зрения
- Но архитектура серверного приложения тоже становится сложнее
- Фреймворки и библиотеки облегчают жизнь, но всё-таки не до конца
- Требуется чёткое понимание того, когда AJAX нужен, а когда нет

Спасибо за внимание!

Дополнительные источники

- Алмаер, Д. AJAX [Текст] / Дион Алмаер, Джастин Гетланд, Бен Гэлбрайт. – М. : Лори, 2013. – 342 с.
- Овчаренко, А. AJAX на примерах [Текст] / Андрей Овчаренко. – СПб. : БХВ-Петербург, 2009. – 432 с.
- Закас, Н. AJAX для профессионалов [Текст] / Николас Закас, Джереми Мак-Пик, Джо Фоссет. – СПб. : Символ-Плюс, 2008. – 488 с.
- Ajax (programming) [Электронный ресурс]. – Режим доступа: http://en.wikipedia.org/wiki/Ajax_(programming), дата доступа: 14.10.2013.
- AJAX Tutorial [Электронный ресурс]. – Режим доступа: http://www.w3schools.com/ajax/, дата доступа: 14.10.2013.
- AJAX Tutorial [Электронный ресурс]. – Режим доступа: http://www.tutorialspoint.com/ajax/, дата доступа: 14.10.2014.
- jQuery.ajax() [Электронный ресурс]. – Режим доступа: http://api.jquery.com/jQuery.ajax/, дата доступа: 14.10.2014.

Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Технологии сетевого программирования

Модель Java Enterprise Edition

Занятие 16

Гаврилов А.В.

Самара
2013

План занятия

- Системы уровня предприятия
- Принципы и архитектура JavaEE
- Элементы систем уровня предприятия
- Структура приложений
- Серверы приложений и их конфигурирование

Программное обеспечение уровня предприятия

- ПО уровня предприятия (Enterprise Software) – ПО, предназначенное для решения задач уровня предприятия

- Обеспечивает работу и сопровождает бизнес-процессы предприятия с целью увеличения качества и эффективности работы предприятия в целом

Программные системы уровня предприятия

- Основные характеристики
  - Производительность
  - Масштабируемость
  - Надежность
- Современные приложения уровня предприятия
  - являются распределенными
  - подразумевают работу с базой данных

До «нашей эры»

- Основное средство разработки – JDK и дополнительные пакеты
- Продукты независимых производителей (например, CORBA)
- API различных производителей, ориентированные на создание систем уровня предприятия

Java Enterprise Edition

- Платформа для создания систем предприятия
- Представлена Sun Microsystems в июне 1999 г. под названием Java 2 Platform Enterprise Edition (J2EE)
- Приложения, входящие в систему, в общем случае должны работать в разнородной среде, обеспечиваемой продуктами разных производителей

Состав JavaEE

- **Спецификация**
  Определяет набор требований к реализации JavaEE
- **Программная модель**
  В виде руководства разработчика, объясняет порядок использования различных средств
- **Платформа JavaEE**
  Набор интегрированных API и инструментов разработки
- **Реализация сервисных средств**
- **Тест на совместимость**

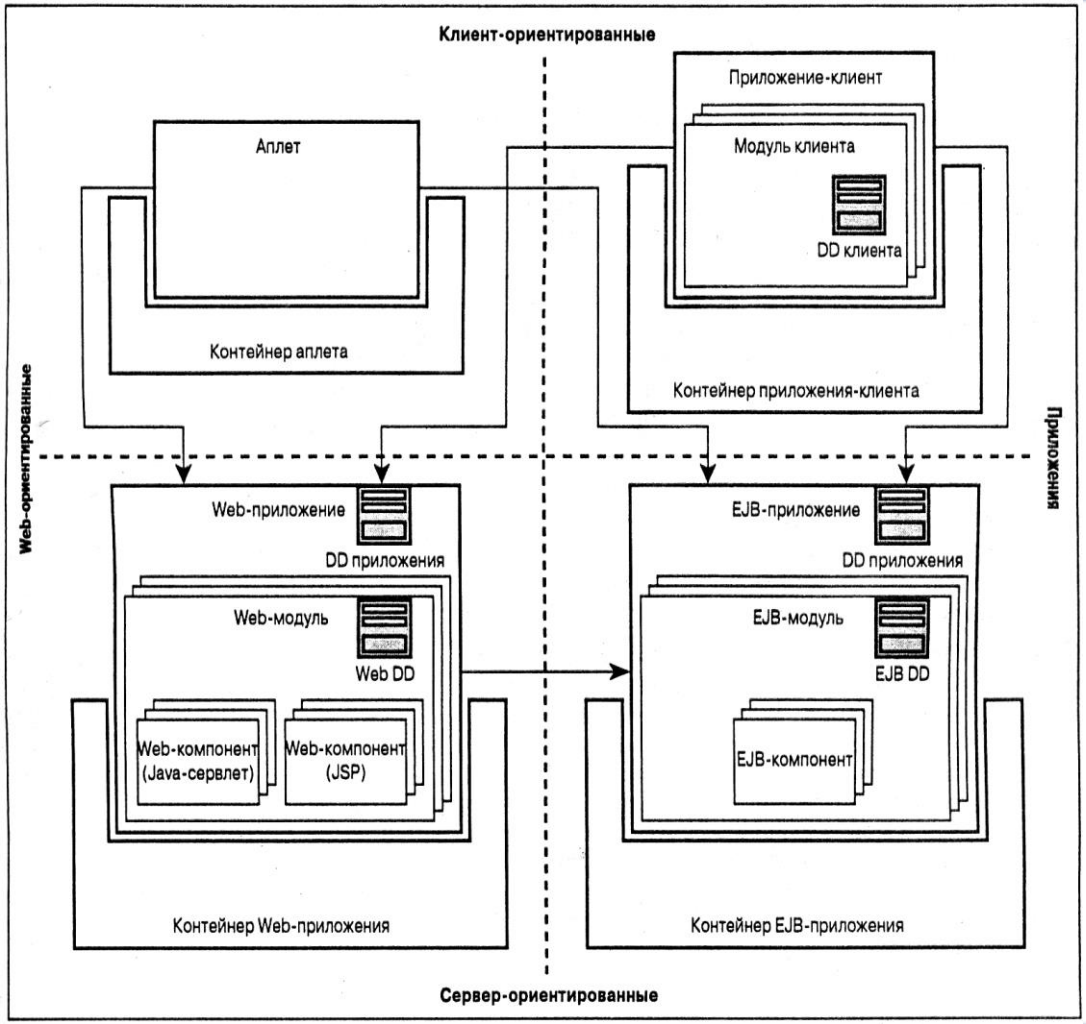
Версии JavaEE

- **J2EE 1.2**
  J2SE 1.2, EJB 1.1, Servlet 2.2, JSP 1.1, …
- **J2EE 1.3**
  J2SE 1.3, EJB 2.0, Servlet 2.3, JSP 1.2, …
- **J2EE 1.4**
  J2SE 1.4, EJB 2.1, Servlet 2.4, JSP 2.0, …
- **JavaEE 5**
  JavaSE 1.5, EJB 3.0, Servlet 2.5, JSP 2.1, JSF 1.2, JAXB 2.0, Java Persistence 1.0, …
- **JavaEE 6**
  JavaSE 1.6, EJB 3.1, Servlet 3.0, JSP 2.2, JSF 2.0, JAXB 2.2, Java Persistence 2.0, …

Реализации JavaEE

- Платформа и сервисные средства могут быть реализованы сторонними производителями
- Реализация должна гарантировать работу компонентов, если они созданы в соответствии со спецификацией и программной моделью
- Реализация может обладать специфическими особенностями
- Реализация играет роль промежуточного ПО (Middleware), в котором выполняются программные модули

Компоненты и контейнеры JavaEE

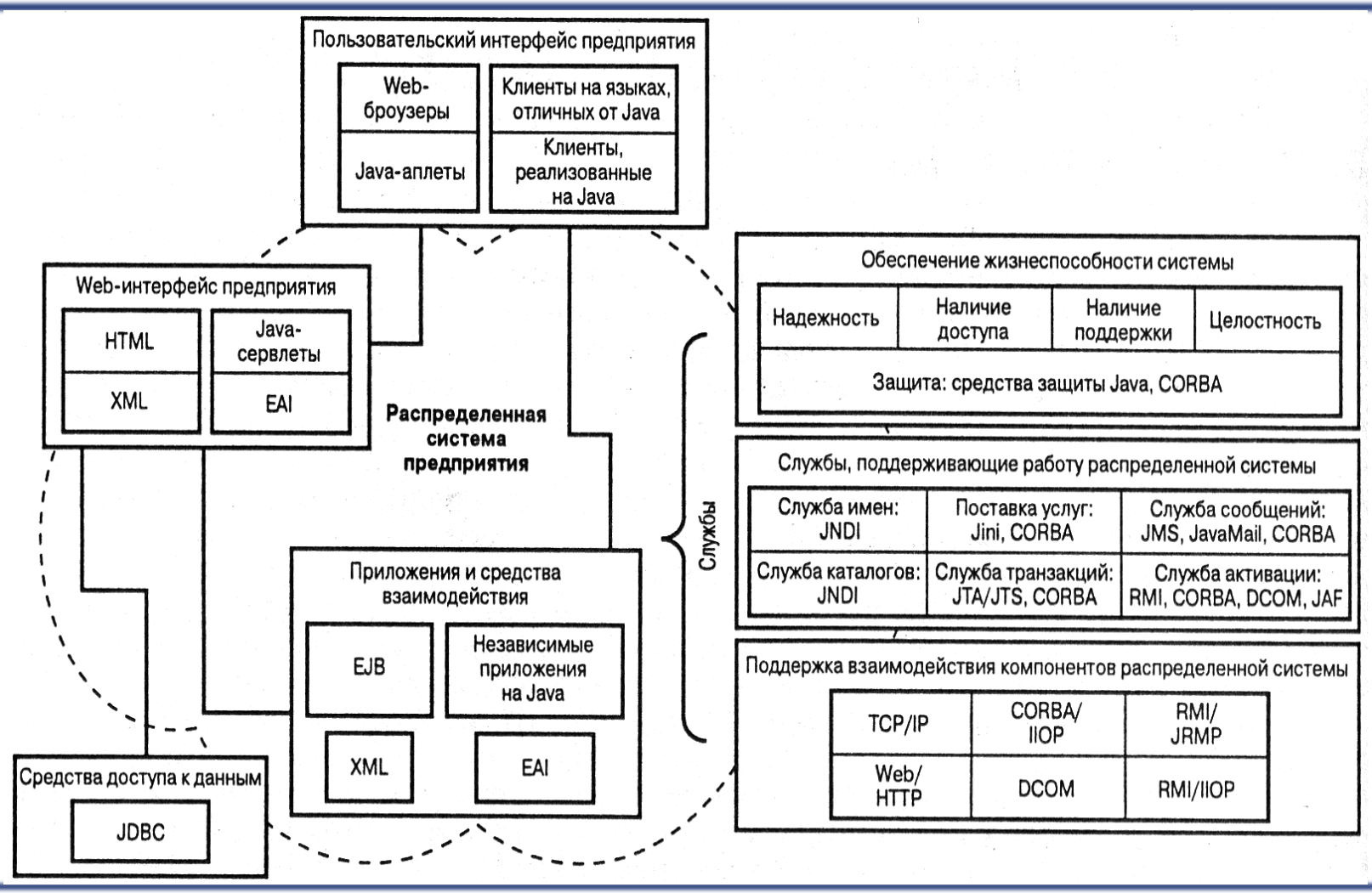


Средства, доступные из контейнера

- Платформа JavaSE
- Java Enterprise API
- Реализации Java Enterprise API

- Служба развертывания (доставки)
- Средства управления

Архитектура системы уровня предприятия



Пользовательские интерфейсы

- Java-апплеты и Java-приложения, выполняющиеся на персональных компьютерах и рабочих станциях
- Традиционные приложения, предназначенные для просмотра содержимого WEB
- Клиенты, взаимодействующие с компонентами на сервере с помощью технологий распределенных коммуникаций

Средства доступа к данным предприятия

- Основой является JDBC API
  - Применение на Web-слое JavaEE
  - Применение в Entity Beans, Entity Persistence
  - Применение в независимых приложениях, выполняемых вне JavaEE

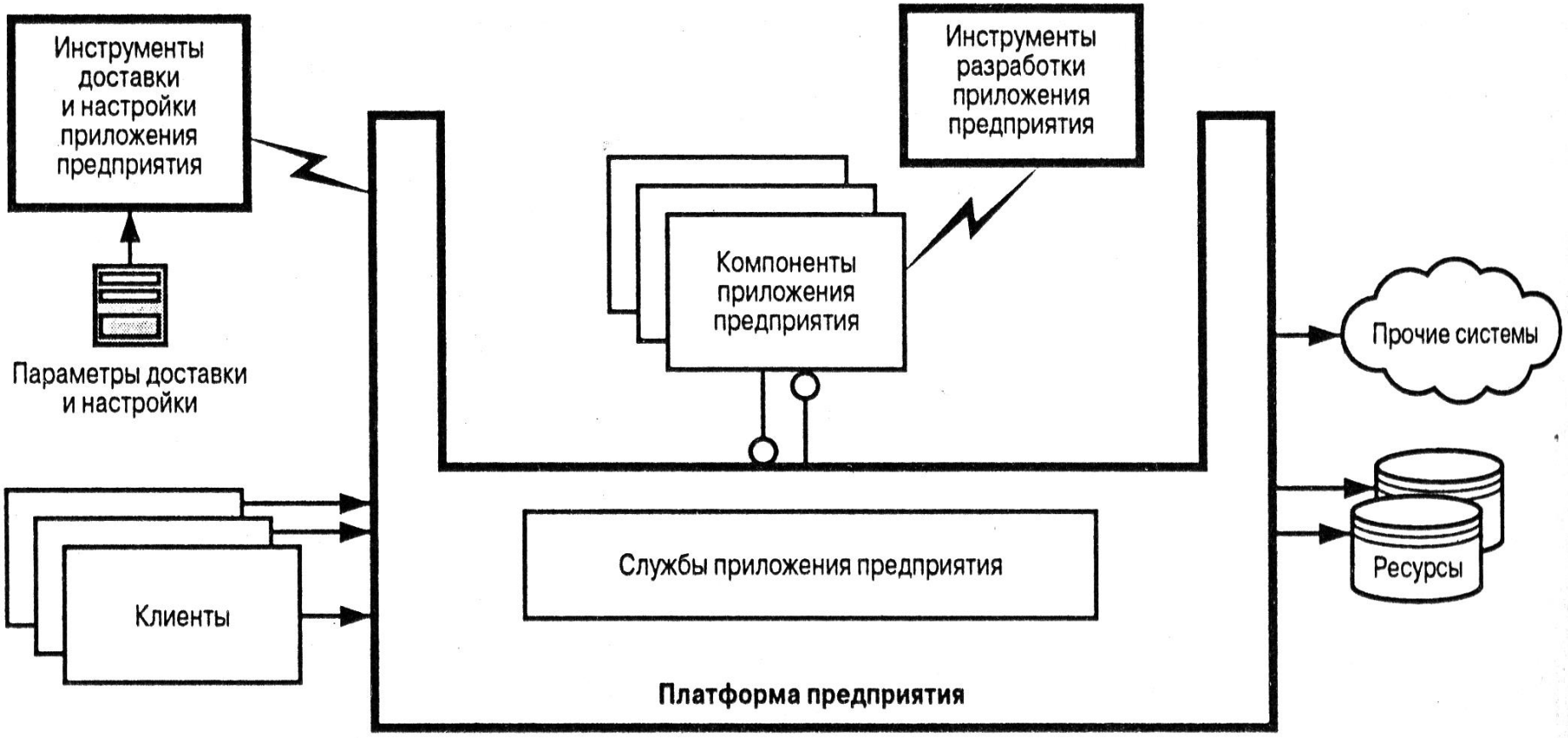
Средства распределенных вычислений

- **Технологии**
  RMI, CORBA, TCP/IP
- **На серверной стороне**
  Web-сервер, сервер приложений, независимые приложения
- **На клиентской стороне**
  Средства предоставления интерфейсов пользователям

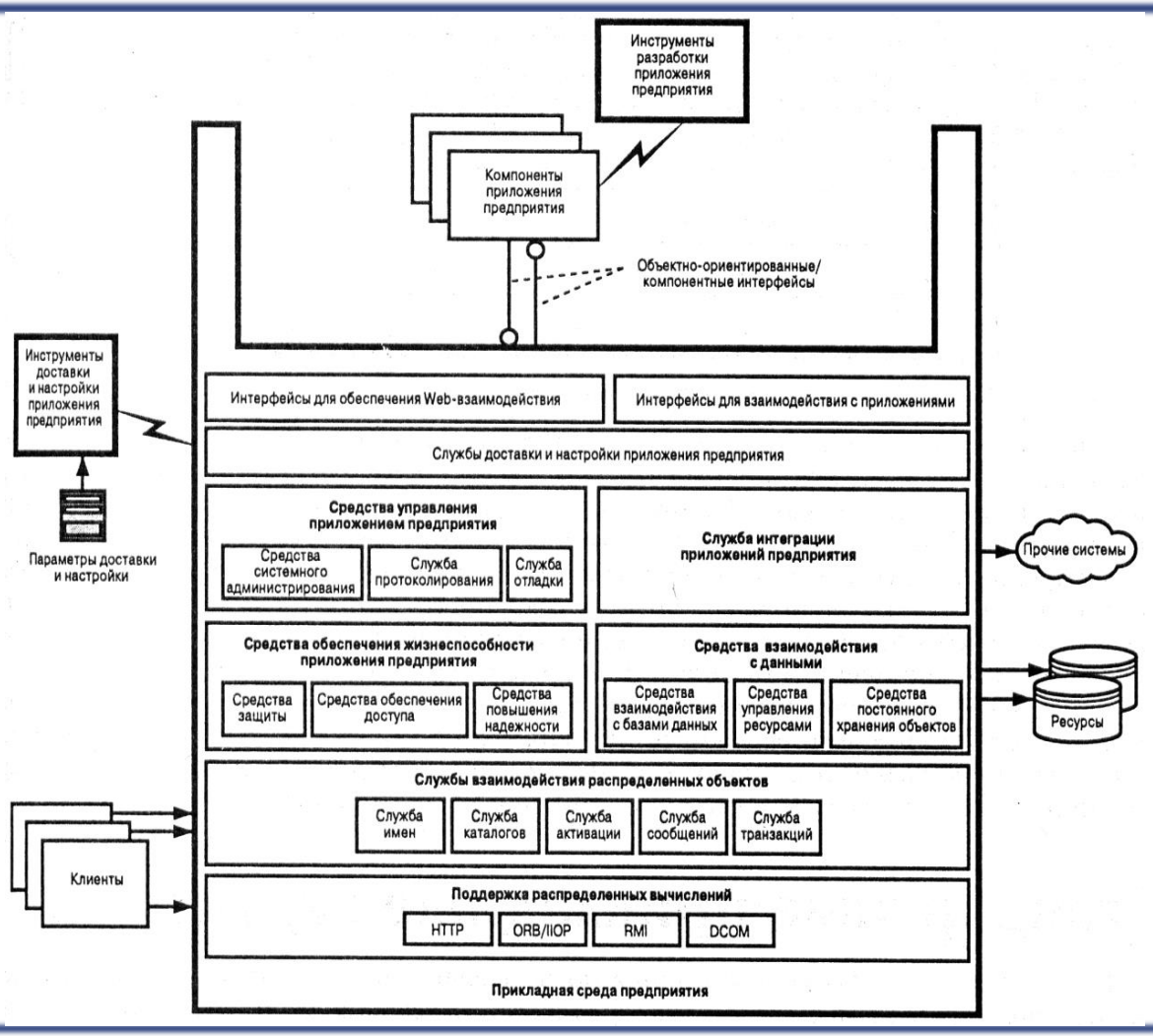
Службы распределенных систем

- **Служба имен**
  Поиск именованных объектов в сети
- **Служба каталогов**
  Дополнительные возможности поиска
- **Служба сообщений**
  Асинхронное взаимодействие в сети
- **Служба активации**
  Активация объектов по запросу клиента
- **Служба транзакций**
  Выполнение взаимозависимых операций над распределенными ресурсами

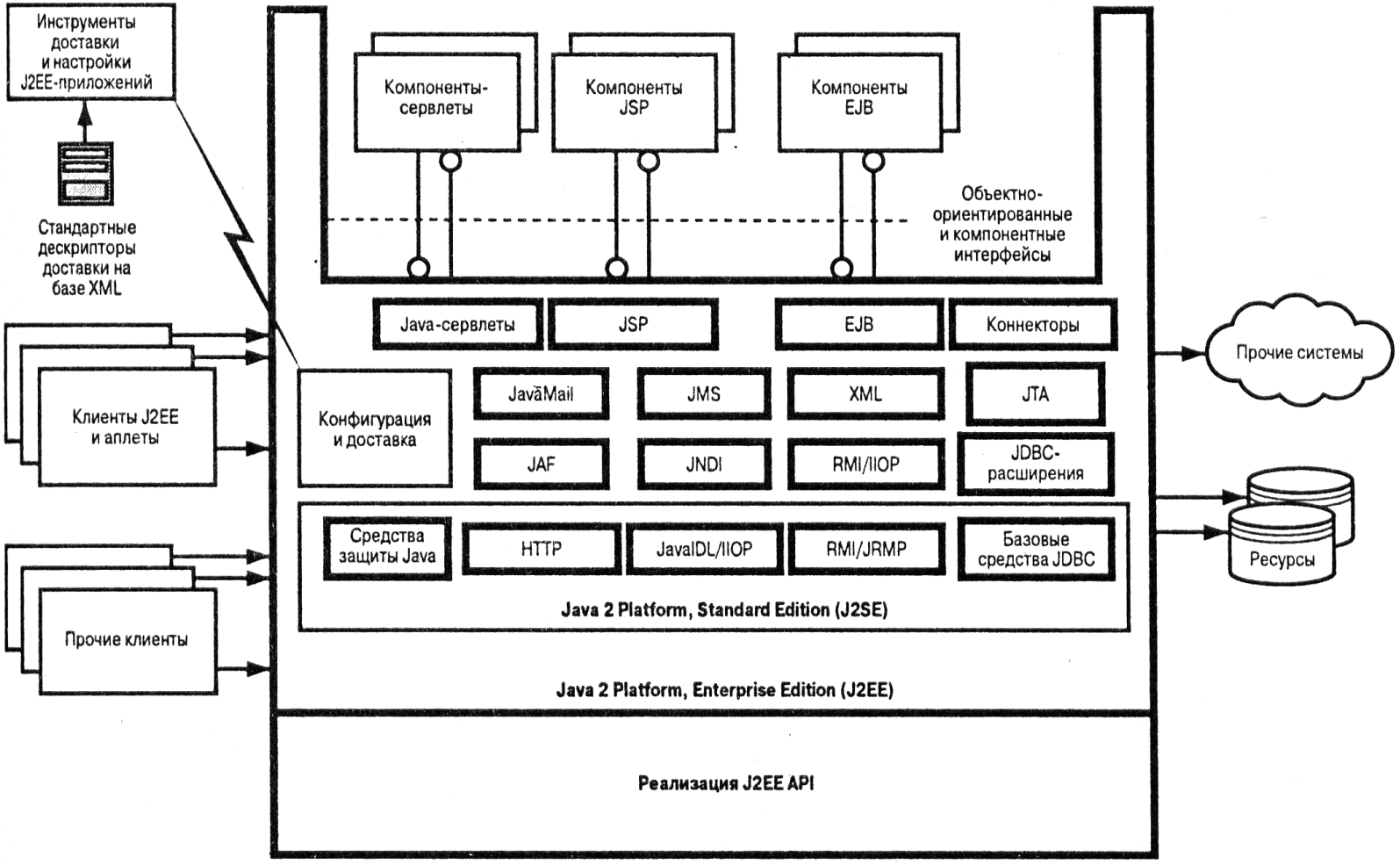
Архитектура прикладной среды предприятия



Базовая архитектура сервера приложений



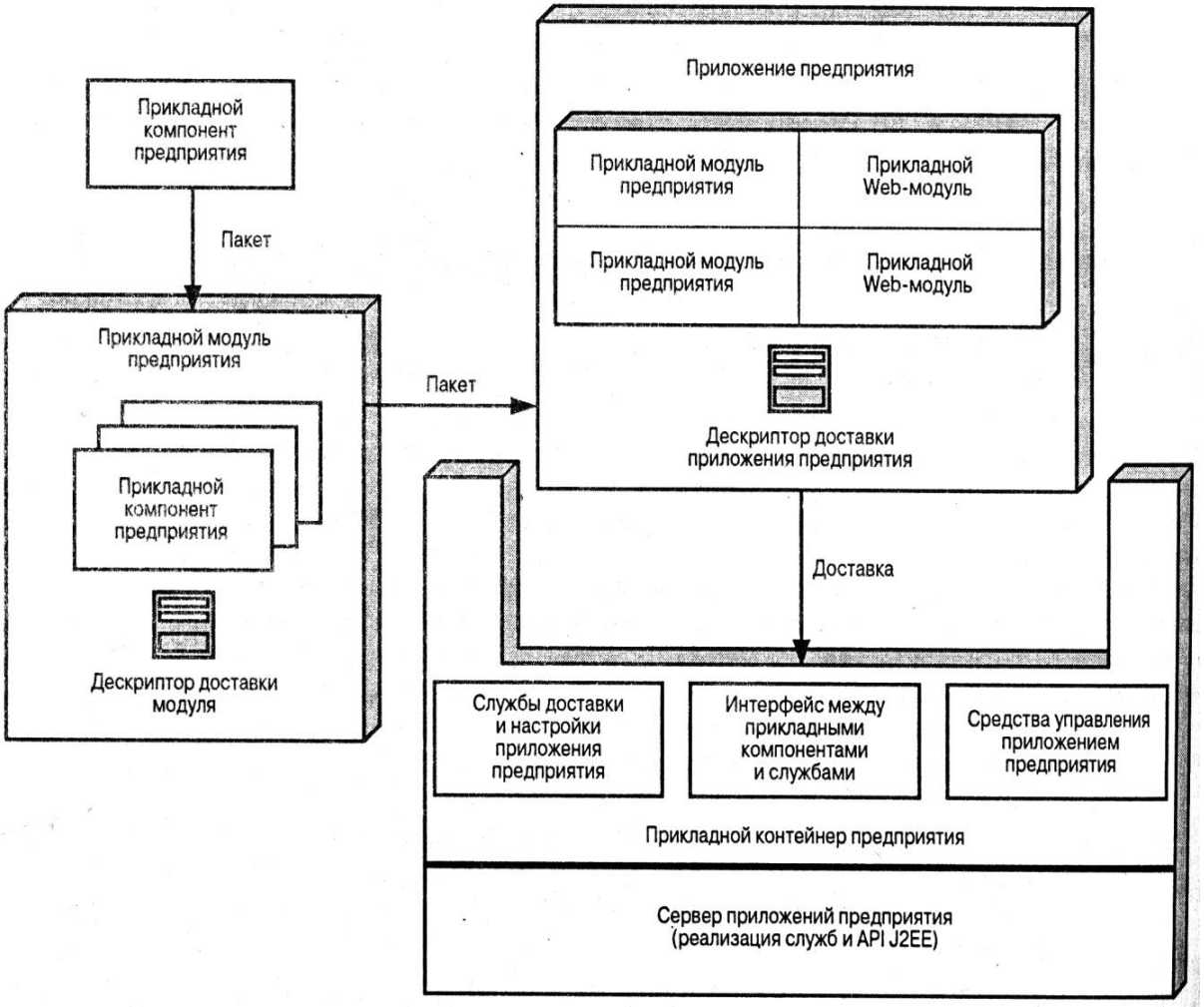
Архитектура серверного приложения JavaEE



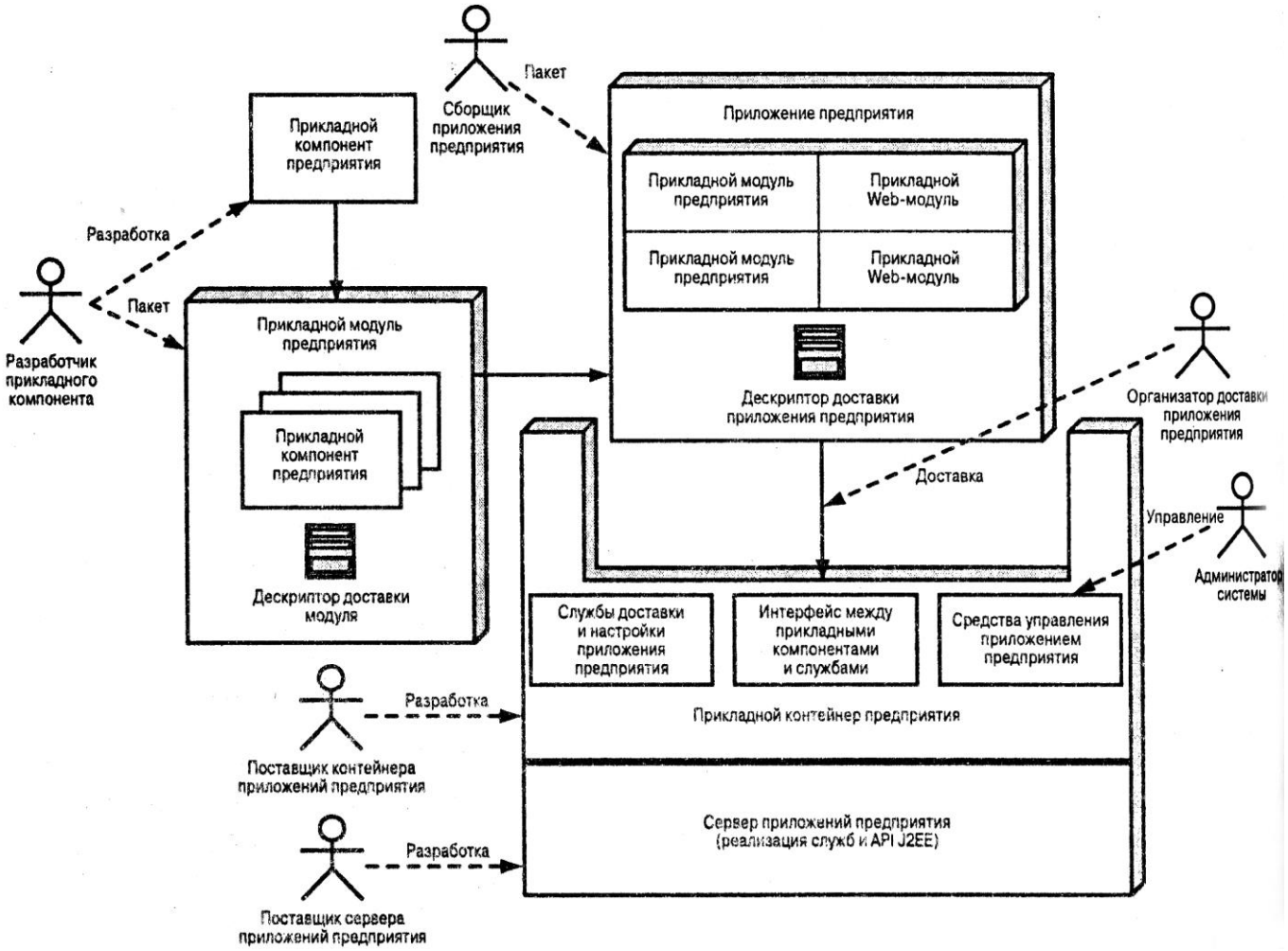
Элементы серверного приложения

- **Прикладные компоненты**
  Бизнес-логика и работа с данными
- **Прикладные модули**
  Один или несколько компонентов, обеспечивают работу службы
- **Приложение уровня предприятия**
  Набор связанных между собой модулей и Web-приложений
- **Прикладной контейнер**
  Среда для выполнения приложений предприятия
- **Сервер приложений (Application Server)**
  Коммуникационные услуги, управление транзакциями и т.д.

Архитектура приложения в рамках сервера приложений



Роли поставщиков элементов



Поставщик сервера приложений

- Обеспечивает функционирование инфраструктуры, реализуемой сервером приложений
- В среде JavaEE обеспечивает работу JavaEE-служб и API
- Поскольку стандартные интерфейсы для взаимодействия контейнера с сервером приложений не определены, роль совпадает с ролью поставщика контейнера

Поставщик контейнера

- Реализует среду выполнения, интерфейсы служб, логику взаимодействия, а также средства развертывания, настройки и управления
- Совпадает с ролью поставщика сервера приложений

Серверы приложений

- J2EE 1.4
  - Sun Application Server 8.x (Sun)
  - Oracle Application Server 10g (Oracle)
  - JBoss 4.x (JBoss)
  - IBM WebSphere Application Server 6.x (IBM)
  - WebLogic 8.x (BEA Systems)
  - Apache Geronimo 1.x
  - …
- JavaEE5
  - Sun Application Server 9.x (Sun), based on GlassFish
  - Oracle Application Server 11 (Oracle)
  - Jboss 5.x (JBoss)
  - IBM WebSphere Application Server 7.x (IBM)
  - WebLogic 8.x (BEA Systems)
  - Apache Geronimo 2.x
  - GlassFish 2.x
  - WebSphere Community Edition (based on Apache Geronimo)

Разработчик прикладного компонента

- Специалист в конкретной области и имеет четкое представление о бизнес-логике и данных, необходимых приложению

- Основная задача – создание прикладных компонентов, объединение их в единый модуль и написание дескриптора развертывания, описывающего структуру и взаимозависимость компонентов

Сборщик приложения уровня предприятия

- Объединяет один или несколько прикладных модулей и представляет их в виде приложения уровня предприятия

- Определяет свойства дескриптора доставки приложения и дополняет либо сообщает о необходимости дополнить дескрипторы развертывания модулей информацией, необходимой для сборки

Организатор развертывания приложения

- Развертывает собранное приложение предприятия в среде «контейнер/сервер».
- Должен знать особенности той среды «контейнер/сервер», в которой должно выполняться приложение
- Может добавлять в дескриптор развертывания модуля или приложения необходимые свойства

Администратор прикладной системы

- Управляет взаимодействием программных компонентов, обменом с базами данных
- Следит за безопасностью и использованием ресурсов
- Выполняет прочие обязанности администратора

Структура серверных приложений

- **EJB-модули**
  Содержат классы и интерфейсы EJB-компонентов, дескрипторы развертывания, вспомогательные классы и дополнительные ресурсы различного вида. Физически – один файл с расширением `jar`

- **Модули Web-компонентов**
  Содержат статические web-ресурсы, классы сервлетов, JSP-страницы, дескрипторы развертывания, вспомогательные классы и дополнительные ресурсы различного вида. Физически – один файл с расширением `war`

JAR-модуль

- Представляет собой архив
- Обладает своей собственной «файловой системой», «каталогами» и служебной информацией
- Обычно содержит служебный каталог META-INF, в котором обычно находится файлы:
  - MANIFEST.MF
  - Дескриптор развертывания
  - Специфический дескриптор развертывания

WAR-модуль

- **Document root**
  - Статические web-ресурсы
  - Страницы JSP
  - Классы, используемые на стороне клиента

- **WEB-INF**
  - Дескриптор развертывания web-приложения (web.xml)
  - Каталог classes, в котором содержатся классы, используемые на стороне сервера, и подкаталоги
  - Каталог lib (вместе с подкаталогами), в которых находятся jar-файлы с используемыми библиотеками
  - TLD-файлы описания tag-библиотек
  - Произвольные каталоги, например, для хранения графической информации

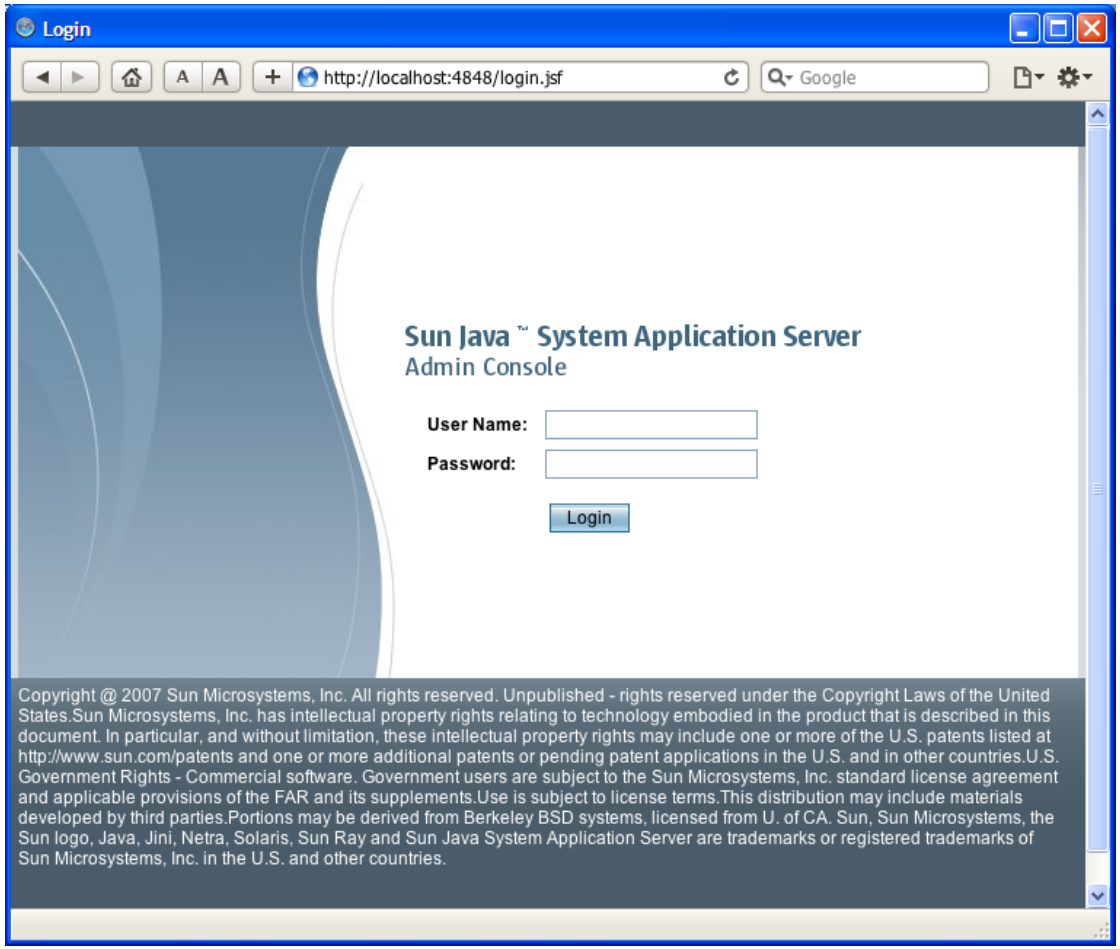
EAR-модуль

- Предназначен для хранения JavaEE-приложения в целом
- Обычно содержит JAR- и WAR-архивы, из которых состоит приложение
- По сути – тот же JAR-архив
- В каталоге META-INF должен находиться дескриптор развертывания (application.xml)

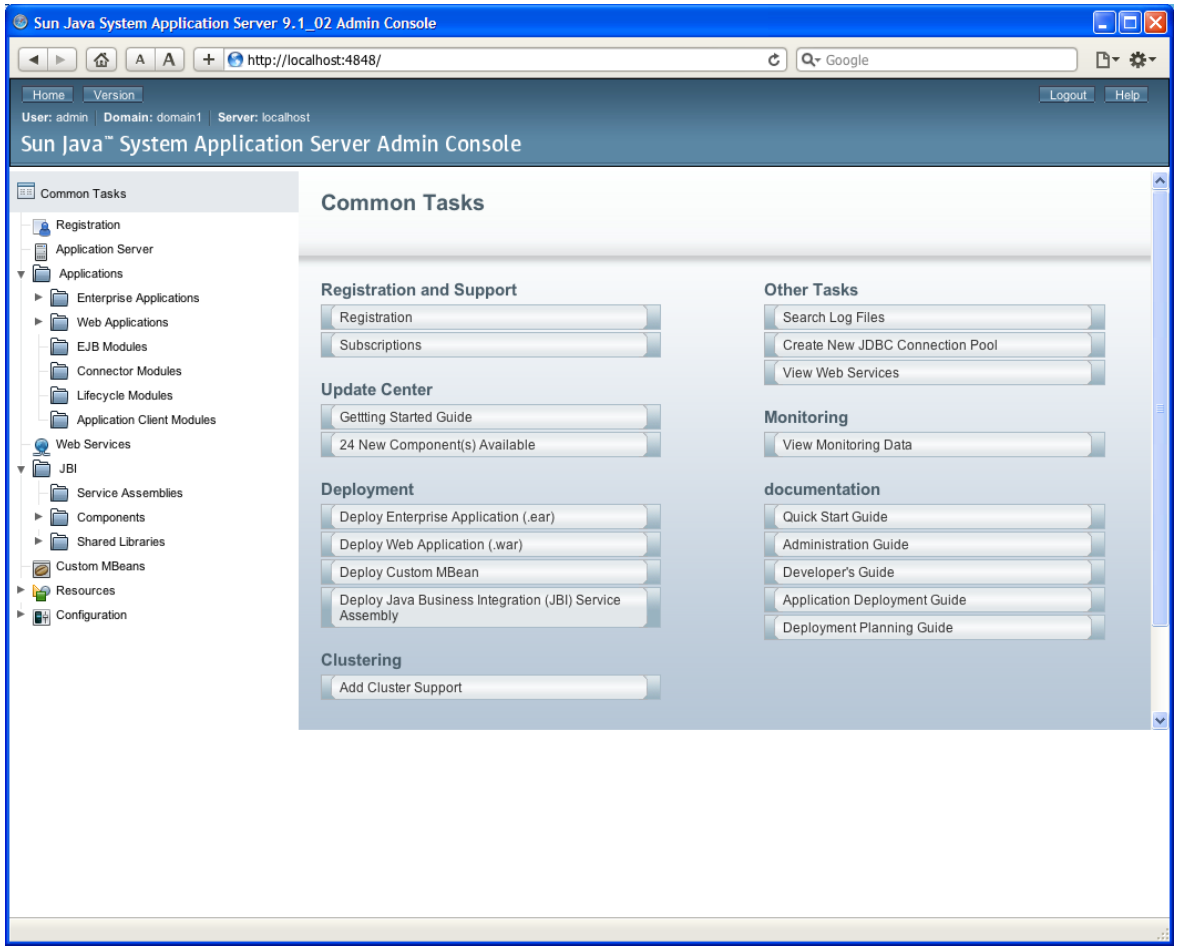
Сервер приложений
(на примере GlassFish 2.1)

- Содержит в себе наборы прикладных модулей, образующих приложения
- Включает в себя Web-сервер
- Содержит реализации первостепенных и вспомогательных служб, необходимых для функционирования приложений и сервера
- При запуске конфигурируется с помощью файлов настроек
- В процессе работы может конфигурироваться с помощью администраторской консоли

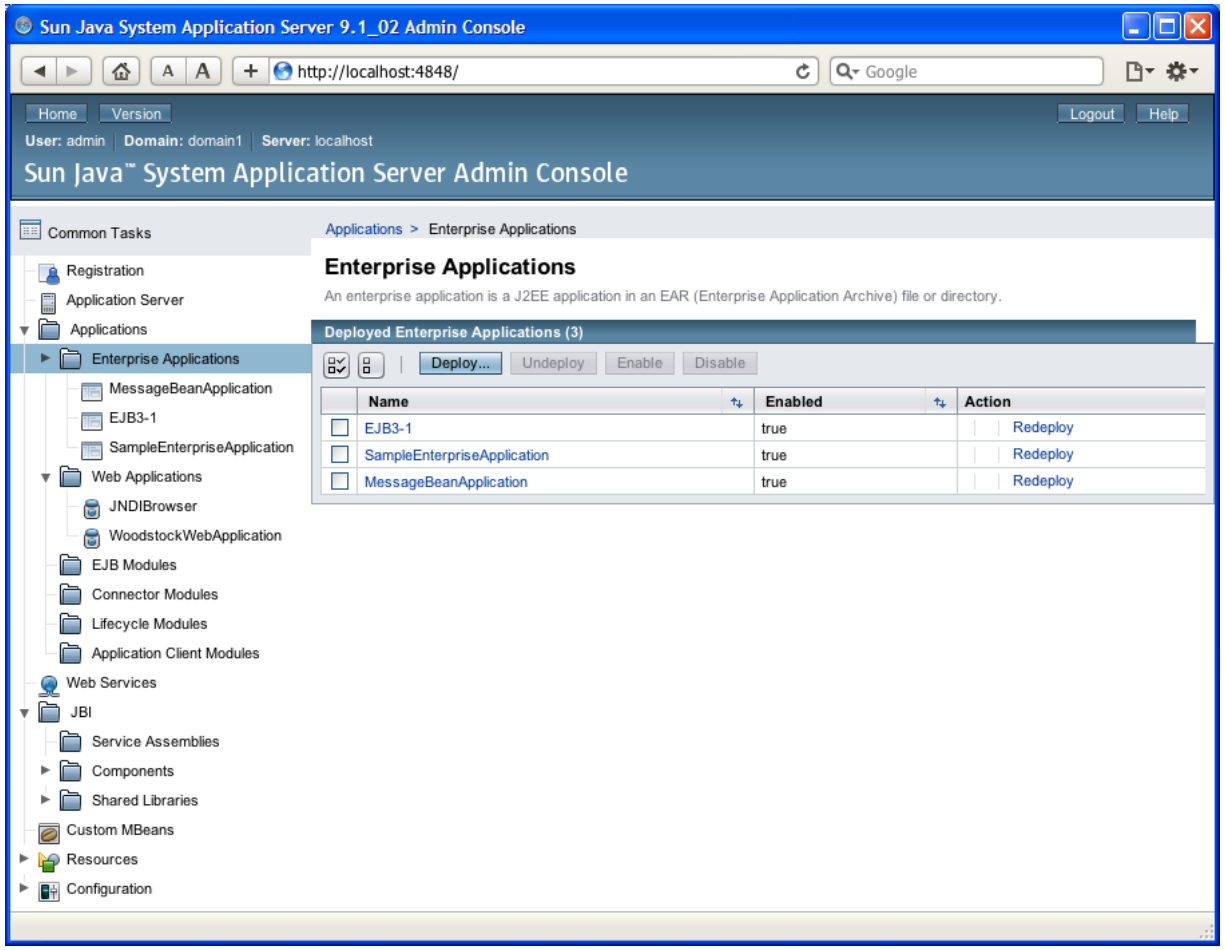
Вход в администраторскую консоль



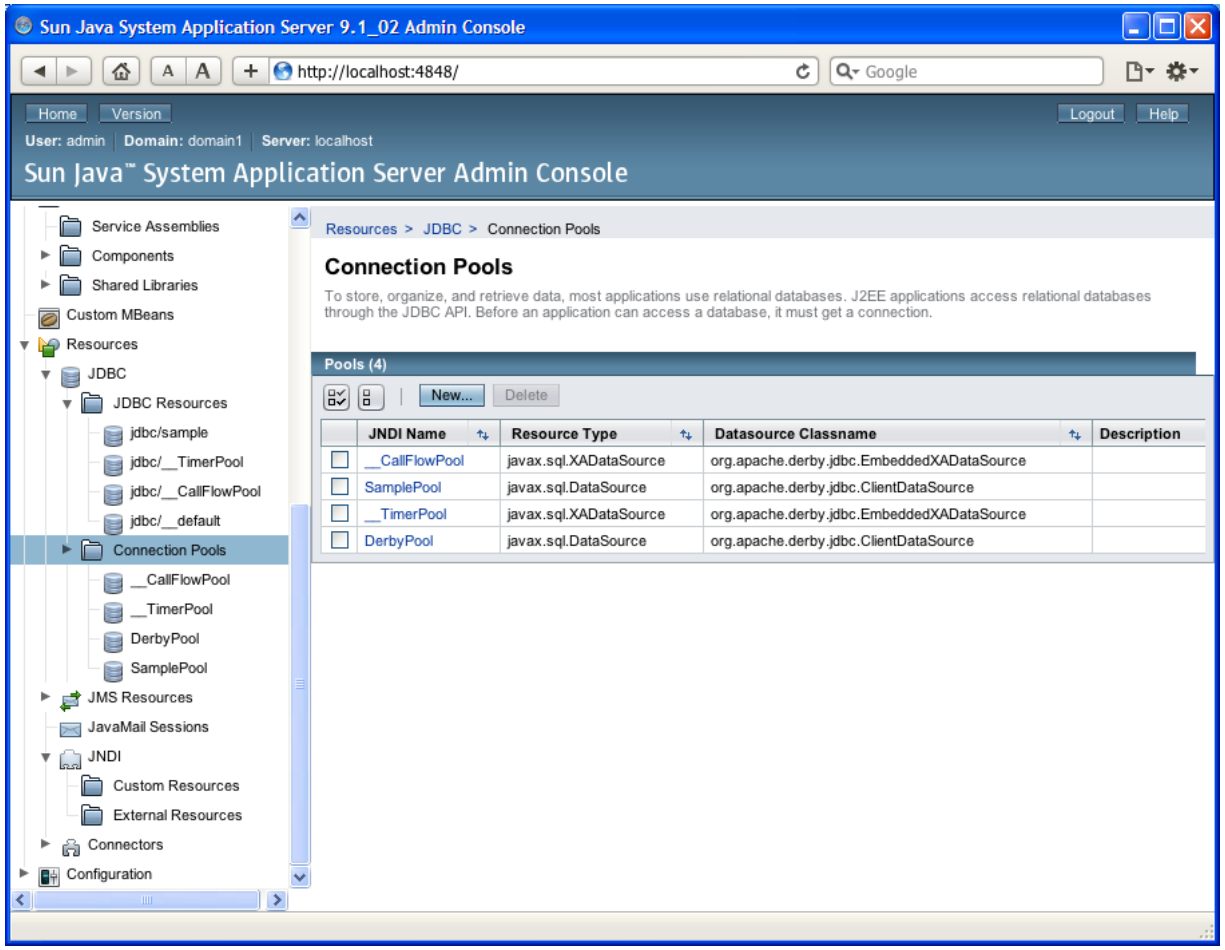
Общий вид администраторской консоли



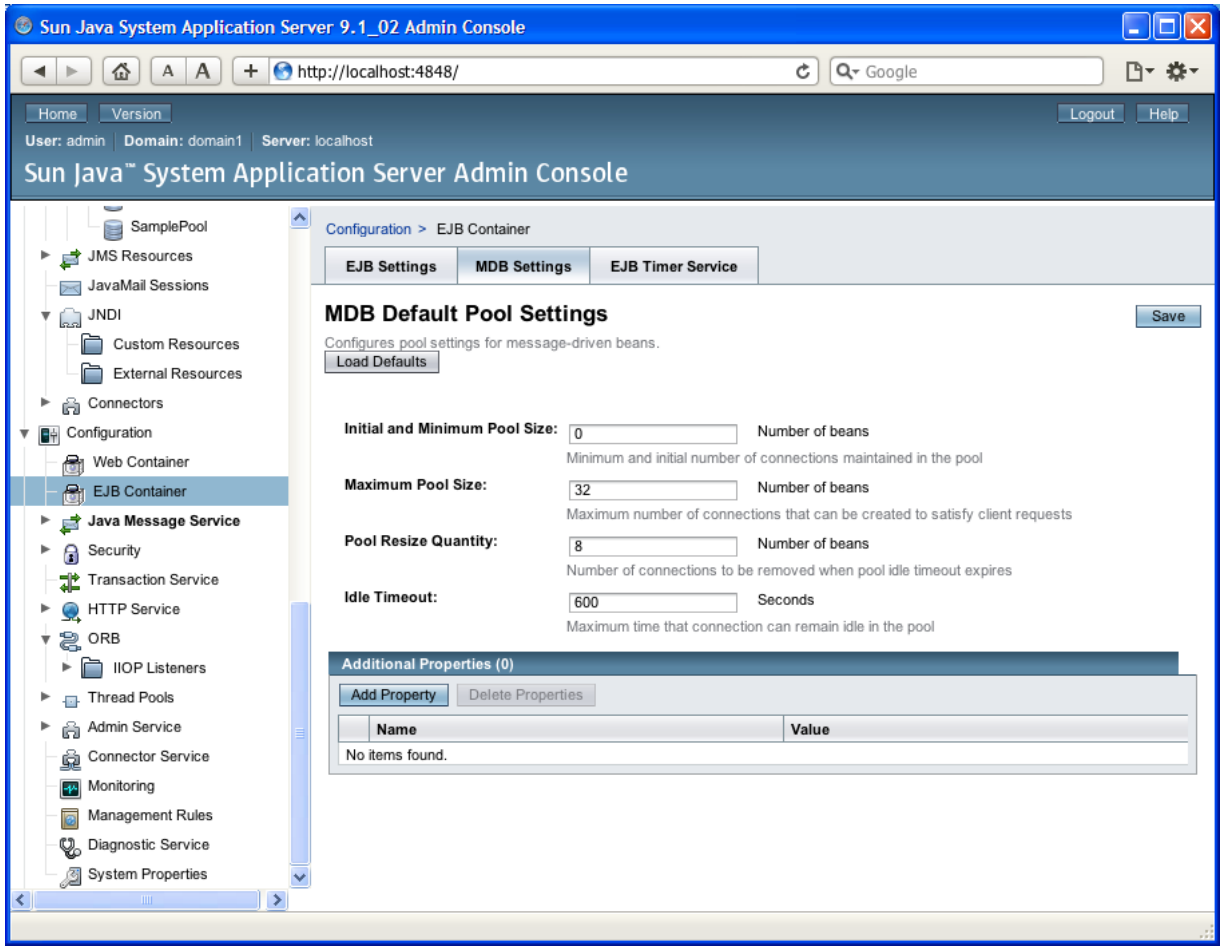
Приложения и модули



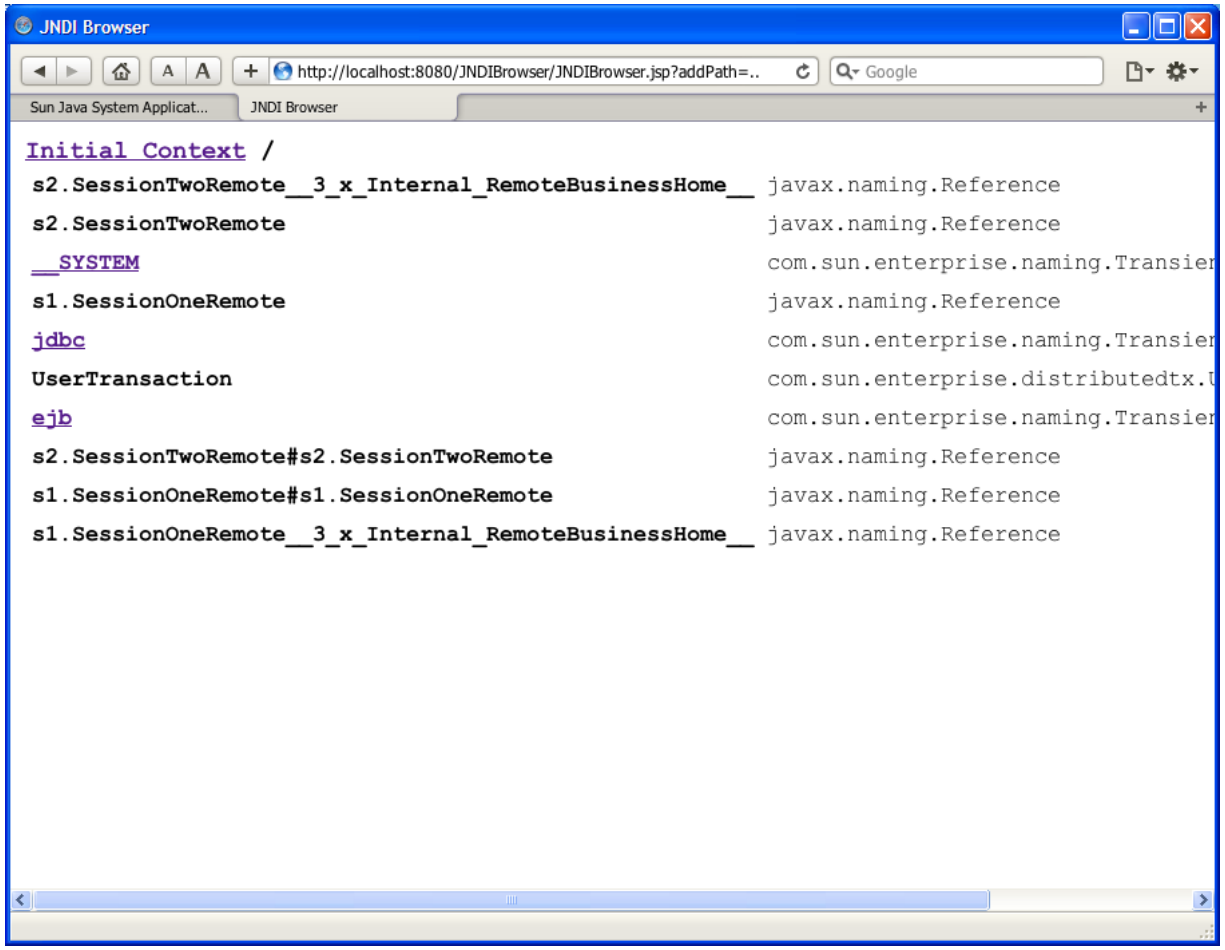
Вспомогательные элементы



Настройки сервера



Пример работы



Спасибо за внимание!

Дополнительные источники

- Цимбал, А.А. Технологии создания распределенных систем. Для профессионалов [Текст] / А. Цимбал, М. Аншина. – СПб. : Питер, 2003. – 576 с.
- Дейтел, Х.М. Технологии программирования на Java 2. Книга 3. Корпоративные системы, сервлеты, JSP, Web-сервисы [Текст] / Х.М. Дейтел, П.Дж. Дейтел, С.И. Сантари. – М. : Бином-пресс, 2003. – 672 с.
- Амриш, К. Разработка корпоративных Java-приложений с использованием J2EE и UML [Текст] / К. Амриш, Х.З. Ахмед. – М. : Издательский дом «Вильямс», 2002. – 272 с.
- JavaEE at Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javaee/overview/index.html, дата доступа: 14.10.2013.
- Java Platform, Enterprise Edition (Java EE) Technical Documentation [Электронный ресурс]. – Режим доступа: http://download.oracle.com/javaee/, доступ: 14.10.2013.

Вопросы к зачёту

по курсу «Технологии сетевого программирования»
направление 010400.62 «Прикладная математика и информатика»
направление 010900.62 «Прикладные математика и физика»

1. Понятие распределенного приложения, причины их создания.
2. Принципы построения распределенных систем.
3. Последствия распределённости приложений.
4. Требования к распределенным системам. Открытость.
5. Требования к распределенным системам. Безопасность.
6. Требования к распределенным системам. Масштабируемость.
7. Требования к распределенным системам. Обработка сбоев.
8. Требования к распределенным системам. Прозрачность.
9. Требования к распределенным системам. Управляемость.
10. Гетерогенность в распределенных приложениях. Промежуточное ПО.
11. Модели распределенных систем, модели архитектуры.
12. Архитектура и ее основные принципы.
13. Слои и ярусы в распределенных приложениях.
14. Основные парадигмы распределенного программирования. Требования к дизайну.
15. Модель «клиент-сервер».

16. Модель «пул серверов».
17. Модель с прокси-сервером.
18. Модель равноправных процессов.
19. Мобильный код и мобильные агенты.
20. Тонкие клиенты.
21. Модель OSI. Протоколы TCP и UDP. Клиент-серверные взаимодействия. Понятие порта.
22. Понятие сокета. Действия с сокетами и их особенности. Виды сокетов.
23. Структура пакета java.net. Классы Socket и ServerSocket, работа с ними.
24. Структура пакета java.net. Классы DatagramSocket и DatagramPacket, работа с ними.
25. Структура пакета java.net. Класс URL, работа с ним.
26. RMI: основные понятия и принцип действия.
27. RMI: архитектура и ее уровни, порядок вызова удаленного метода.
28. RMI: передача параметров, динамическая загрузка классов.
29. RMI: именование удаленных объектов.
30. RMI: порядок разработки серверной части приложения.
31. RMI: порядок разработки клиентской части приложения, политики безопасности.
32. RMI: разделение кода приложения, запуск приложений.

33. Язык HTML. Основные элементы языка.
34. HTML. Тип документа. Теги заголовка документа.
35. HTML. Тело документа. Управление отображением текста. Символьные мнемоники.
36. HTML. Ссылки, списки, изображение и таблицы.
37. HTML. Протокол HTTP и принципы функционирования форм.
38. HTML. Работа с формами, основные теги форм.
39. CSS. Задачи и основные принципы. Приоритеты стилей.
40. CSS. Селекторы и их виды, применение в html.
41. Язык XML. Основные элементы языка.
42. XML. Структура документа. Понятие правильного документа.
43. XML. Document Type Definition.
44. XML. XMLSchema.
45. XSL. XPath.
46. XML. DOM vs. SAX
47. JAXP. SAX.
48. JAXP. DOM.
49. JAXP. Запись документов.
50. JAXB. Основные принципы и аннотации.

51. Задачи, особенности и применение языка JavaScript, встраивание в документы и обработка браузером
52. Основы синтаксиса языка JavaScript
53. Функции в JavaScript
54. Объекты и работа с ними в JavaScript, особенности наследования
55. DOM и доступ к элементам из JavaScript
56. Обработка событий объектов DOM и динамическое изменение DOM средствами JavaScript
57. Сервлеты: назначение, жизненный цикл и особенности.
58. Сервлеты: объекты запроса и отклика.
59. Сервлеты: дескриптор развертывания, использование других web-ресурсов.
60. Сервлеты: сессии в сервлетах.
61. JSP: назначение и жизненный цикл страниц.
62. JSP: элементы сценариев.
63. JSP: директива page.
64. JSP: директива include.
65. JSP: стандартные доступные объекты.
66. JSP: JSP-документы и теги .
67. JSP Unified Expression Language: назначение, выражения и их виды.

68. JSP Unified Expression Language: литералы, операторы и объекты.
69. JSP Custom tags: назначение, порядок обработки.
70. JSP Custom tags: JSTL, его библиотеки и их предназначение.
71. JSF: назначение и возможности.
72. JSF: элементы приложения, их взаимодействие.
73. JSF: жизненный цикл обработки запросов.
74. JSF: подложенные бины.
75. JSF: конвертеры и валидаторы.
76. JSF: особенности навигации.
77. JNDI: назначение и структура.
78. JNDI: имена и их виды.
79. JNDI: контексты, сопоставление и разрешение имен.
80. JNDI: основные классы и интерфейсы.
81. JNDI: способы настройки.
82. JDBC: назначение и структура.
83. JDBC: виды драйверов.
84. JDBC: получение соединения с помощью класса DriverManager.
85. JDBC: получение соединения с помощью DataSource.
86. JDBC: интерфейс Connection, управление транзакциями.

87. JDBC: интерфейсы Statement и PreparedStatement.
88. JDBC: интерфейс ResultSet, типы данных.
89. EJB: назначение, преимущества.
90. EJB: виды, состав.
91. EJB: локальные и удаленные клиенты, базовые интерфейсы для компонентов.
92. EJB: порядок разработки компонентов и их особенности.
93. Session Beans: назначение, виды, случаи использования.
94. Stateful Session Beans: особенности, жизненный цикл, случаи использования.
95. Stateless Session Beans: особенности, жизненный цикл, случаи использования.
96. Singleton Session Beans: особенности, жизненный цикл, случаи использования.
97. Session Beans: класс компонента.
98. Session Beans: методы жизненного цикла.
99. Session Beans: методы бизнес-логики.
100. Session Beans: порядок обращения к экземпляру компонента, особенности обращения из JSP-страниц.
101. Dependency Injection. Общие принципы.
102. Dependency Injection. Аннотации и использование.

103. AJAX. Принципы и порядок работы.

104. AJAX. Виды передаваемых данных.

105. AJAX. Класс XMLHttpRequest.

106. Отладка web-приложений.

107. Основы работы с фреймворком jQuery.

108. ПО уровня предприятия.

109. JavaEE: назначение, состав и особенности.

110. JavaEE: компоненты и контейнеры, элементы JavaEE-приложений.

111. JavaEE: службы распределенных систем и архитектура серверной части приложений.

112. JavaEE: элементы серверного приложения и их разработчики.

113. JavaEE: структура серверных приложений, ejb-модули, war-модули, ear-модули.

Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Самарский государственный аэрокосмический университет имени академика С.П. Королева (национальный исследовательский университет)» (СГАУ)

АТТЕСТАЦИОННЫЕ ПЕДАГОГИЧЕСКИЕ ИЗМЕРИТЕЛЬНЫЕ МАТЕРИАЛЫ

для проверки остаточных знаний студентов

по дисциплине «Технологии сетевого программирования»

направление 010400.62 “Прикладная математика и информатика”

направление 010900.62 “Прикладные математика и физика”

1. Какой механизм должен применяться, если данные не должен прочитать никто, кроме получателя, даже если доступ к ним получит кто-то иной?

а) Аутентификация
б) Деморализация
в) Авторизация
г) Шифрование
д) Идентификация

2. Как называется программное обеспечение, чья цель состоит в том, чтобы скрывать неоднородность и обеспечивать удобную модель программирования для разработчиков?

а) Software
б) Middleware
в) Spyware
г) Interware
д) Proxyware
е) Hardware

3. Какой интерфейс должны расширять все интерфейсы удаленных объектов в RMI?

а) `UnicastRemoteObject`
б) `Deleted`
в) `RemoteObject`
г) `Remote`
д) `MulticastRemoteObject`

4. Как в службе имен называется процесс получения доступа к ресурсу по его имени?

а) Развязывание
б) Расширение
в) Разрешение
г) Запрещение
д) Запрашивание

5. Какой тип Java соответствует JDBC-типу VARCHAR?

а) `Character`
б) `StringBuffer`
в) `char`
г) `VarChar`
д) `String`
е) `StringBuilder`

6. В файлах с каким расширением обычно хранятся JavaEE-приложения?

а) tar
б) jar
в) zip
г) war
д) rar
е) app
ж) ear

7. Какое значение атрибута target тега ссылки в HTML позволяет открыть ссылку в новой вкладке (или новом окне)?

а) _blank
б) _parent
в) _top
г) _new
д) _window
е) _self

8. Выберите среди предложенных CSS-селектор для тега (без дополнительных условий):

```
а) a:hover {color:yellow;}
б) #paragraph1 {margin: 0;}
в) p:first-letter {font-size: 32pc;}
г) p {font-family: Garamond, serif;}
д) .note {color: red; background: yellow; font-weight: bold;}
```

9. Выберите характеристику, больше соответствующую модели DOM обработки XML-документов:

а) модель обработки событий;
б) многократная обработка документа;
в) используется мало памяти;
г) последовательный доступ к элементам документа.

10. Какой аннотацией снабжаются HTTP-сервлеты?

```
а) @Web
б) @Servlet
в) @WebServlet
г) @HTTPServlet
```

11. Какая директива позволяет указать страницу для обработки ошибок, возникших в ходе работы JSP-страницы?

```
а) catch
б) errorpage
в) error
г) page
д) exception
```

12. С какого символа должно начинаться выражение Expression Language, чтобы оно допускало изменение свойства управляемого компонента в JSF?

```
а) +
б) -
в) $
г) €
д) #
е) b
```

13. Какая аннотация позволяет внедрить зависимость на EJB-компонент согласно стандарту EJB3.1?

```
а) @Component
б) @Injection
в) @Bean
г) @Resource
д) @EJB
```

14. Какое максимальное количество бизнес-интерфейсов может быть объявлено для Session Stateless компонента?

а) 0
б) 1
в) 2
г) 4
д) 6
е) Количество не регламентировано

15. К каким языкам относится JavaScript?

а) компилируемые
б) интерпретируемые
в) гибридные
г) ингибидорные
д) катализаторные

16. Объекты какого класса предназначены для прослушивания входящих сетевых соединений?

а) Socket
б) Server
в) ServerSocket
г) URL

17. Какой из используемых в AJAX способов передачи данных обладает одновременно структурированностью и компактностью?

а) Простой текст
б) JSON
в) XML
г) HTML