ИТМО ВТ

Веб-программирование

Автор курса — Цопа Е.А.
2023/24 уч. год

1. Введение

ИТМО ВТ

Мир веб-программирования

- Большая часть современных приложений взаимодействует с внешним миром через сеть Интернет.
- Даже локальные приложения часто пишутся по «канонам» веб-программирования.
- Практически все современные веб-сайты – полноценные информационные системы.

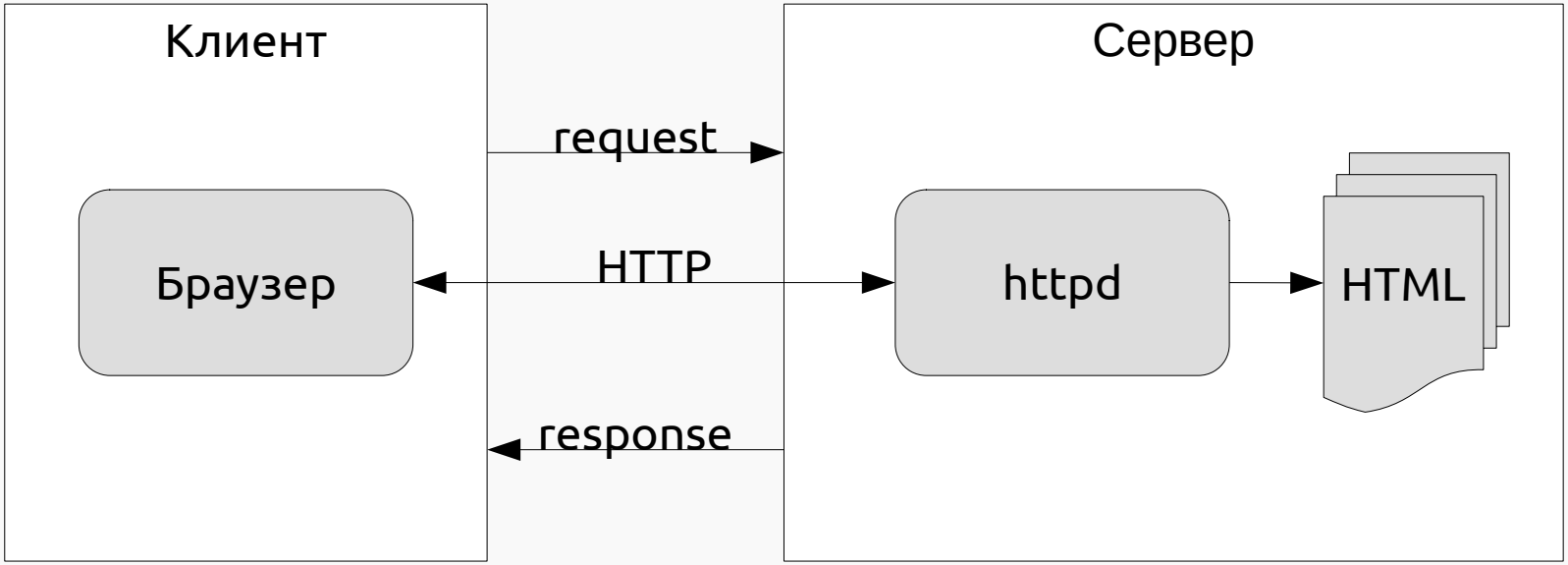
ИТМО ВТ

Архитектура интернет-приложений

- Много подходов и их реализаций — как на клиентской, так и на серверной стороне.
- Целесообразность подхода определяется сложностью разрабатываемого приложения и его областью использования.
- Независимо от выбранного подхода, «снизу» всё равно используются одни и те же стандарты и протоколы.

ИТМО ВТ

Сферическое интернет-приложение в вакууме



ИТМО ВТ

План курса (1)

	Раздел	Технологии
1	Введение	
2	Общие стандарты и протоколы сети Интернет	
2.1	Клиент-серверное взаимодействие	HTTP
2.2	Разметка страниц	HTML
2.3	Стилизация страниц	CSS, SCSS
2.4	Динамические сценарии на стороне клиента	JavaScript
2.5	Асинхронное клиент-серверное взаимодействие	DHTML, Long Polling, WebSocket

ИТМО ВТ

План курса (2)

	Раздел	Технологии
3	«Классические» интернет-приложения	
3.1	Серверные сценарии	CGI, FastCGI, Servlets
3.2	Шаблоны проектирования и архитектурные шаблоны	
3.3	Шаблонизация страниц	JSP, FreeMarker, ThymeLeaf
4	Rich Internet Applications	Java Server Faces
5	Архитектура корпоративных приложений	Java / Jakarta EE, Spring, Spring Web MVC
6	REST и SPA-фреймворки	React, Angular, Vue

2. Общие стандарты и протоколы сети Интернет

ИТМО ВТ

Стандарты и протоколы сети Интернет

- «Основа основ»:
  - Hypertext Transfer Protocol (HTTP) — предназначен для передачи гипертекста между клиентом и сервером.
  - Hypertext Markup Language (HTML) — язык разметки гипертекста.
- «Дополнения» к HTML:
  - Cascade StyleSheets (CSS) – язык описания внешнего вида HTML-документа.
  - JavaScript – язык для написания динамических сценариев, выполняемых на стороне клиента.

2.1. Клиент-серверное взаимодействие. Протокол HTTP.

ИТМО ВТ

Протокол HTTP

- Протокол прикладного уровня
- Основа — технология «клиент-сервер»
- Может быть использован в качестве «транспорта» для других протоколов прикладного уровня
- Основной объект манипуляции — *ресурс*, на который указывает *URI*
- Обмен сообщениями идёт по схеме «запрос-ответ»
- Stateless-протокол (состояние не сохраняется). Для реализации сессий используются cookies.

ИТМО ВТ

URI, URL и URN

- URI (Uniform Resource Identifier) — уникальный идентификатор ресурса — символьная строка, позволяющая идентифицировать ресурс.
- URL (Uniform Resource Locator) — URI, позволяющий определить местонахождение ресурса.
- URN (Uniform Resource Name) — URI, содержащий единообразное имя ресурса (не указывает на его местонахождение).

ИТМО ВТ

URI, URL и URN (продолжение)

- URI:
  `<схема>:<идентификатор-в-зависимости-от-схемы>`
- URL:
  `https://se.ifmo.ru/courses/web`
  `../task.shtml`
  `mailto:Joe.Bloggs@somedomain.com`
- URN:
  `urn:isbn:5170224575`
  `urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJUQCZO5C`

ИТМО ВТ

REST

- *Representational State Transfer* (передача состояния представления) – подход к архитектуре сетевых протоколов, обеспечивающих доступ к информационным ресурсам.
- Основные концепции:
  - Данные должны передаваться в виде небольшого числа стандартных форматов (HTML, XML, JSON).
  - Сетевой протокол должен поддерживать кеширование, не должен зависеть от сетевого слоя, не должен сохранять информацию о состоянии между парами «запрос-ответ».
- Антипод REST – подход, основанный на вызове удаленных процедур (*Remote Procedure Call – RPC*).

ИТМО ВТ

Структура запроса HTTP

- Стартовая строка:
  ```
  Метод URI HTTP/Версия
  GET /spip.html HTTP/1.1
  ```
- Заголовки:
  ```
  Host: se.ifmo.ru
  User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5)
  Gecko/2008050509 Firefox/3.6
  Accept: text/html
  Connection: close
  ```
- Тело сообщения

ИТМО ВТ

Структура ответа HTTP

- Стартовая строка:
  ```
  HTTP/Версия КодСостояния Пояснение
  HTTP/1.1 200 Ok
  ```
- Заголовки:
  ```
  Server: Apache/2.2.11 (Win32) PHP/5.3.0
  Last-Modified: Sat, 16 Jan 2010 21:16:42 GMT
  Content-Type: text/plain; charset=windows-1251
  Content-Language: ru
  ```
- Тело сообщения

ИТМО ВТ

Методы HTTP

- OPTIONS — определение возможностей сервера.
- GET — запрос содержимого ресурса.
- HEAD — аналог GET, но в ответе отсутствует тело.
- POST — передача данных ресурсу.
- PUT — загрузка содержимого запроса на указанный URI.

ИТМО ВТ

Коды состояния

- Состоят из 3-х цифр.
- Первая цифра — *класс состояния:*
  «1» — Informational — информационный;
  «2» — Success — успешно;
  «3» — Redirection — перенаправление;
  «4» — Client error — ошибка клиента;
  «5» — Server error — ошибка сервера.
- Примеры:

```
201 Webpage Created
403 Access allowed only for registered users
507 Insufficient Storage
```

ИТМО ВТ

Заголовки HTTP

- Формат:
  ключ:значение
- 4 группы:
  - General Headers — могут включаться в любое сообщение клиента и сервера. Пример — Cache-Control.
  - Request Headers — используются только в запросах клиента. Пример — Referer.
  - Response Headers — используются только в запросах сервера. Пример — Allow.
  - Entity Headers — сопровождают любую сущность сообщения. Пример — Content-Language.

ИТМО ВТ

Примеры сообщений HTTP

- Запрос клиента:

```
GET /iaps/labs HTTP/1.1
Host: cs.ifmo.ru
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5)
Gecko/2008050509 Firefox/3.6.14
Accept: text/html
Connection: close
```

- Ответ сервера:

```
HTTP/1.0 200 OK
Date: Wed, 02 Mar 2011 11:11:11 GMT
Server: Apache
X-Powered-By: PHP/5.2.4-2ubuntu5wm1
Last-Modified: Wed, 02 Mar 2011 11:11:11 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close
...HTML-код запрашиваемой страницы...
```

2.2. Разметка страниц. Язык HTML.

ИТМО ВТ

Что такое HTML

- Стандартный язык разметки документов в Интернете.
- Интерпретируется *браузером* и отображается в виде документа.
- Разработан в 1989-91 годах *Тимом Бернерсом-Ли*.
- Является частным случаем *SGML* (стандартного обобщённого языка разметки).
- Существует нотация *XHTML*, являющаяся частным случаем языка *XML*.

ИТМО ВТ

Браузеры

- *Браузер* — программа, отображающая HTML-документ в его отформатированном виде.
- Популярные браузеры:
  - Google Chrome
  - Mozilla Firefox
  - M$ Internet Explorer / Edge
  - Apple Safari
  - Opera

ИТМО ВТ

Структура HTML-документа

- Документ состоит из *элементов*.
- Начало и конец элемента обозначаются *тегами:*
  `<b>`**текст**`</b>`
- Теги могут быть пустыми:
  `<br>`
- Теги могут иметь *атрибуты*:
  `<a href="http://www.example.com">Здесь элемент содержит атрибут href.</a>`
- Элементы могут быть вложенными:

```
<b>
  Этот текст будет полужирным,
  <i>а этот - ещё и курсивным</i>
</b>
```

Структура HTML-документа (продолжение)

- Документ должен начинаться со строки объявления версии HTML:
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```
- Начало и конец документа обозначаются тегами `<html>` и `</html>`.
- Внутри этих тегов должны находиться заголовок (`<head>...</head>`) и тело документа (`<body>...</body>`).

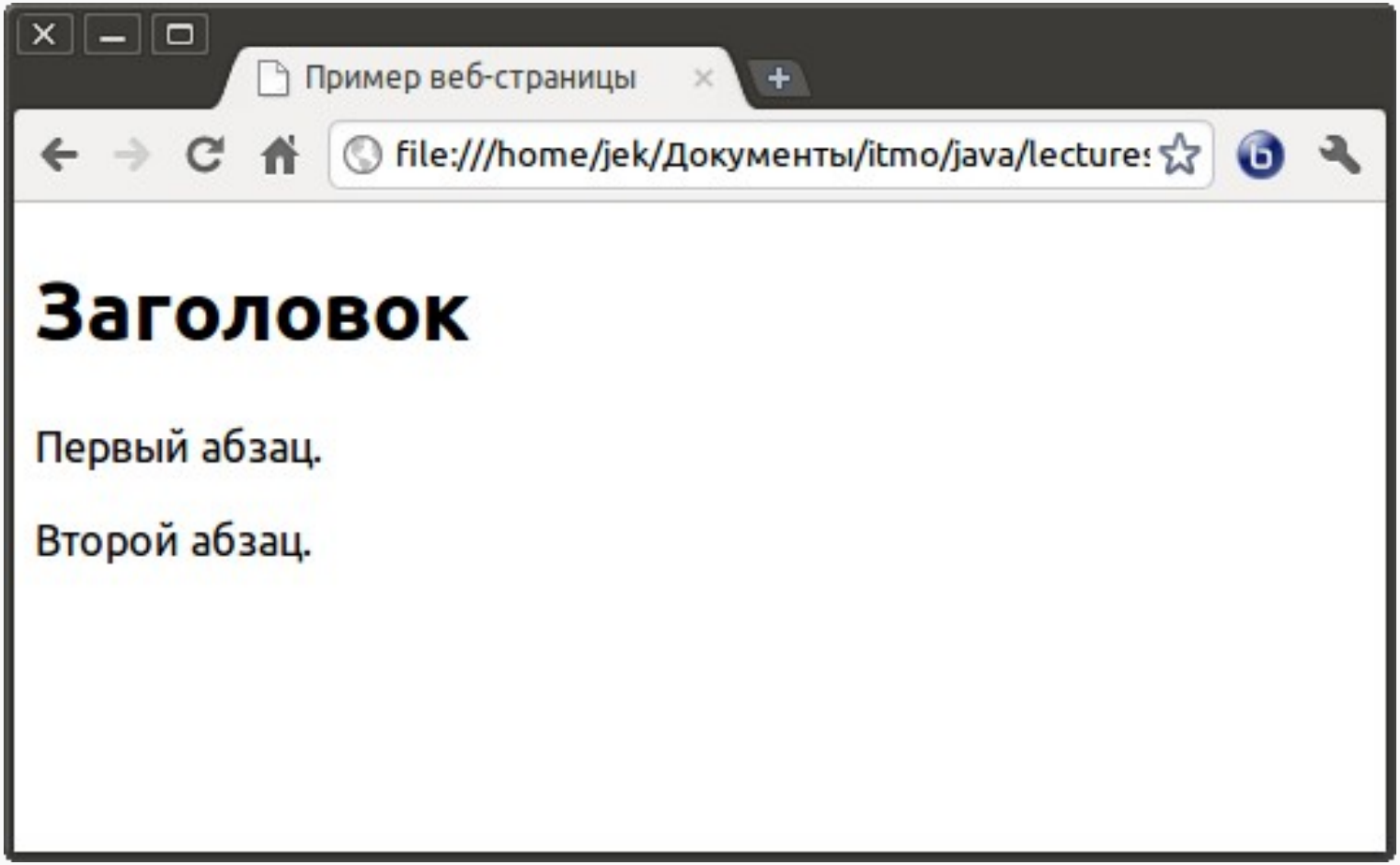
ИТМО ВТ

Пример HTML-документа

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html;
   charset=utf-8">
  <title>Пример веб-страницы</title>
 </head>
 <body>
  <h1>Заголовок</h1>
  <!-- Комментарий -->
  <p>Первый абзац.</p>
  <p>Второй абзац.</p>
 </body>
</html>
```

ИТМО ВТ

Пример HTML-документа (продолжение)



ИТМО ВТ

HTML-формы

- Предназначены для обмена данными между пользователем и сервером.
- Документ может содержать любое число форм, но одновременно на сервер может быть отправлена только одна из них.
- Вложенные формы запрещены.
- Границы формы задаются тегами `<form>...</form>`.
- Метод HTTP задаётся атрибутом method тега `<form>`:
  `<form method="GET" action="URL">...</form>`

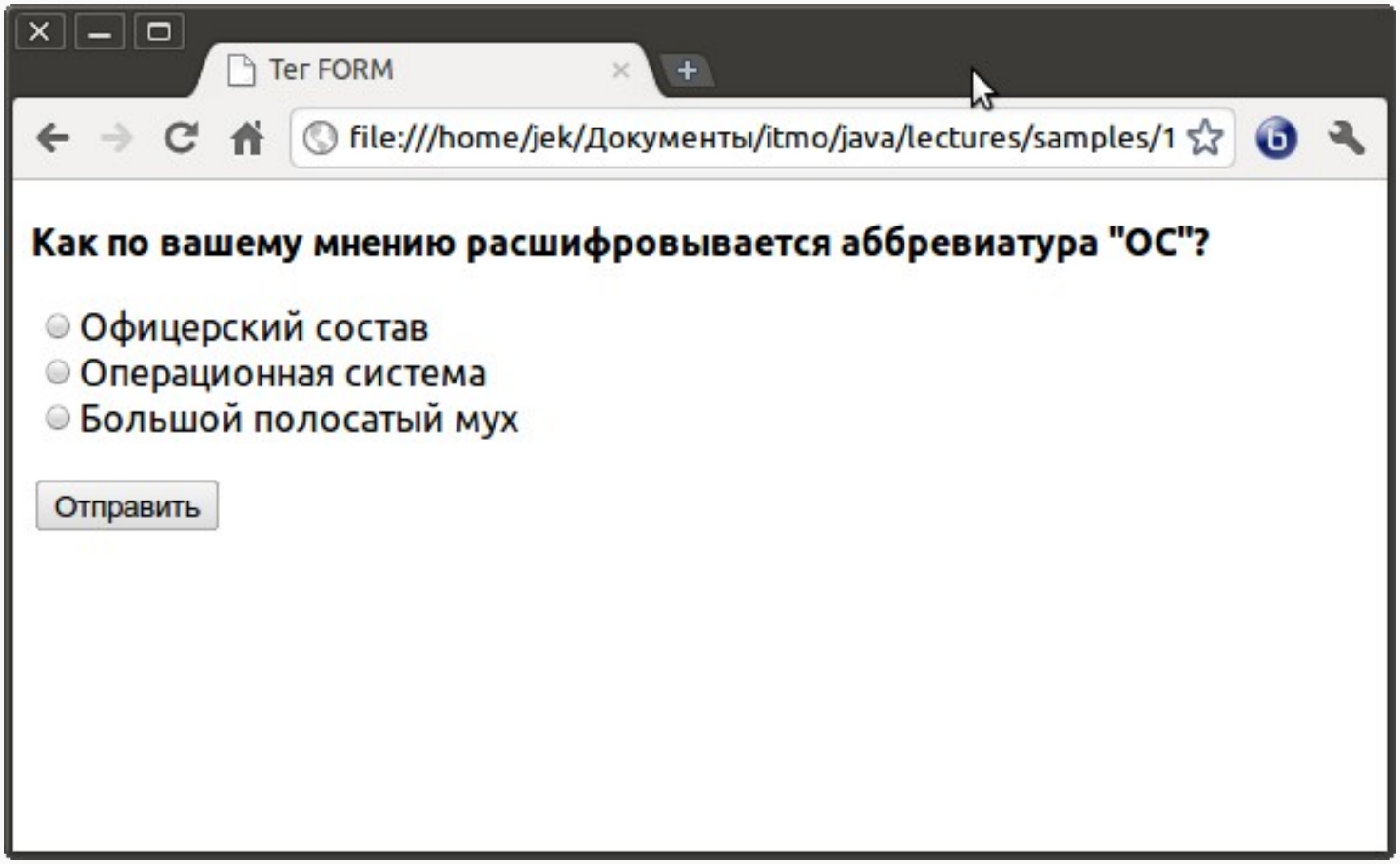
ИТМО ВТ

Пример HTML-формы

```
<form method="POST" action="handler.php">

  <p><b>Как по вашему мнению расшифровывается
           аббревиатура &quot;ОС&quot;?</b></p>

  <p><input type="radio" name="answer"
   value="a1">Офицерский состав<Br>

  <input type="radio" name="answer"
   value="a2">Операционная система<br>

  <input type="radio" name="answer"
   value="a3">Большой полосатый мух</p>

  <p><input type="submit"></p>

</form>
```

ИТМО ВТ

Пример HTML-формы (продолжение)

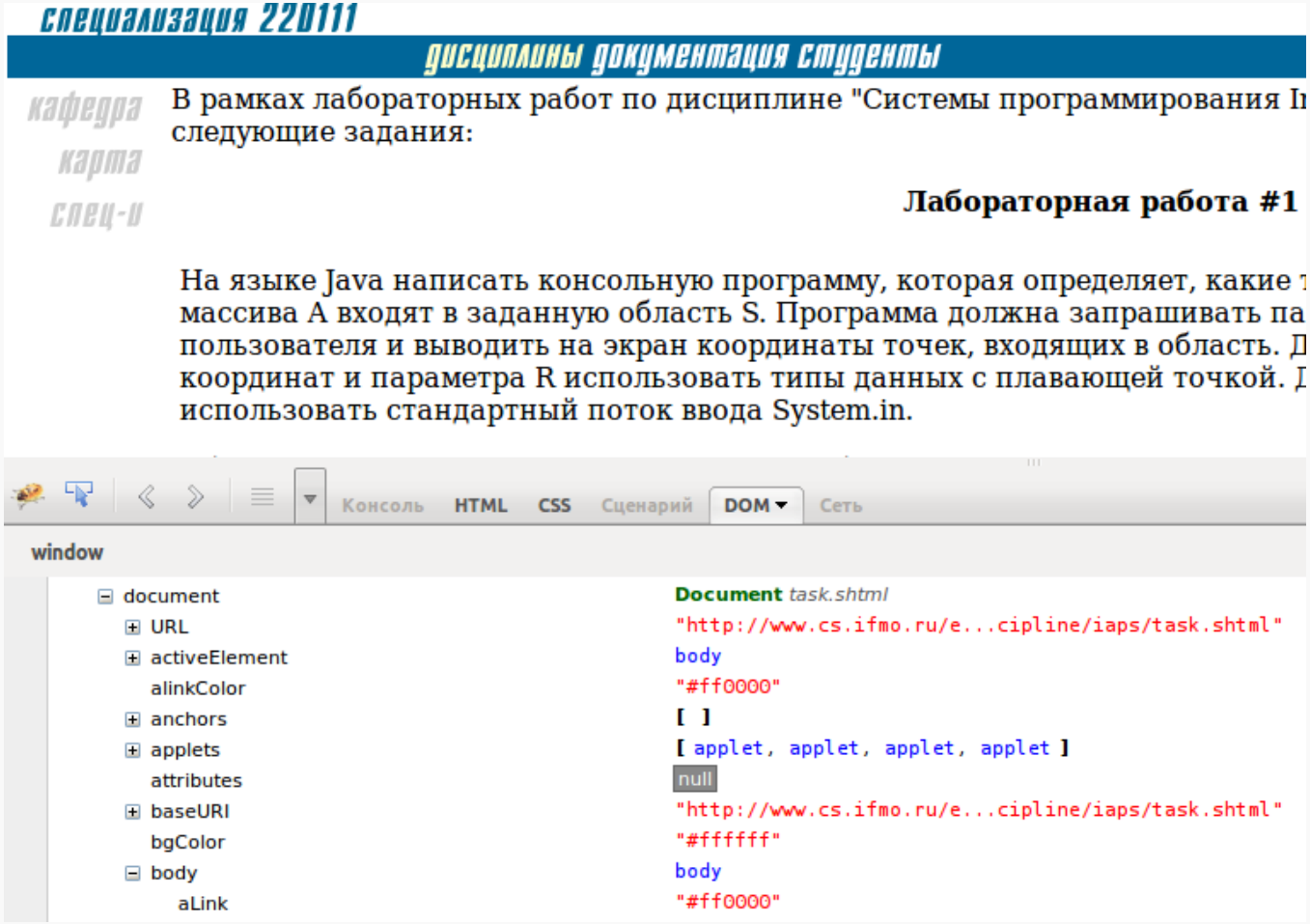


ИТМО ВТ

Объектная модель документа (DOM)

- DOM — это платформо-независимый интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-документов.
- Стандартизирована W3C.
- Документ в DOM представляет собой *дерево узлов*.
- Узлы связаны между собой отношением «родитель-потомок».
- Используется для динамического изменения страниц HTML.

Объектная модель документа (DOM, продожение)



ИТМО ВТ

HTML5

- Пятая версия спецификации языка HTML.
- Стандарт принят в 2014 г. (HTML4 – в 1999 г.).
- Много новых синтаксических особенностей, в первую очередь – для более удобного управления мультимедийным содержимым страницы.

ИТМО ВТ

HTML5: изменения в синтаксисе

- Больше не базируется на SGML (хотя всё ещё обратно совместим с HTML4).
- Новая вводная строка `<!DOCTYPE html>`.
- Новые мультимедийные теги `<audio>` и `<video>`.
- Семантические замены для универсальных блочных (`<div>`) и строчных (`<span>`) элементов – `<nav>`, `<footer>` и т. д.
- Поддержка Web Forms 2.0 – новые поля ввода `date/time`, `email`, новые атрибуты и т. д.

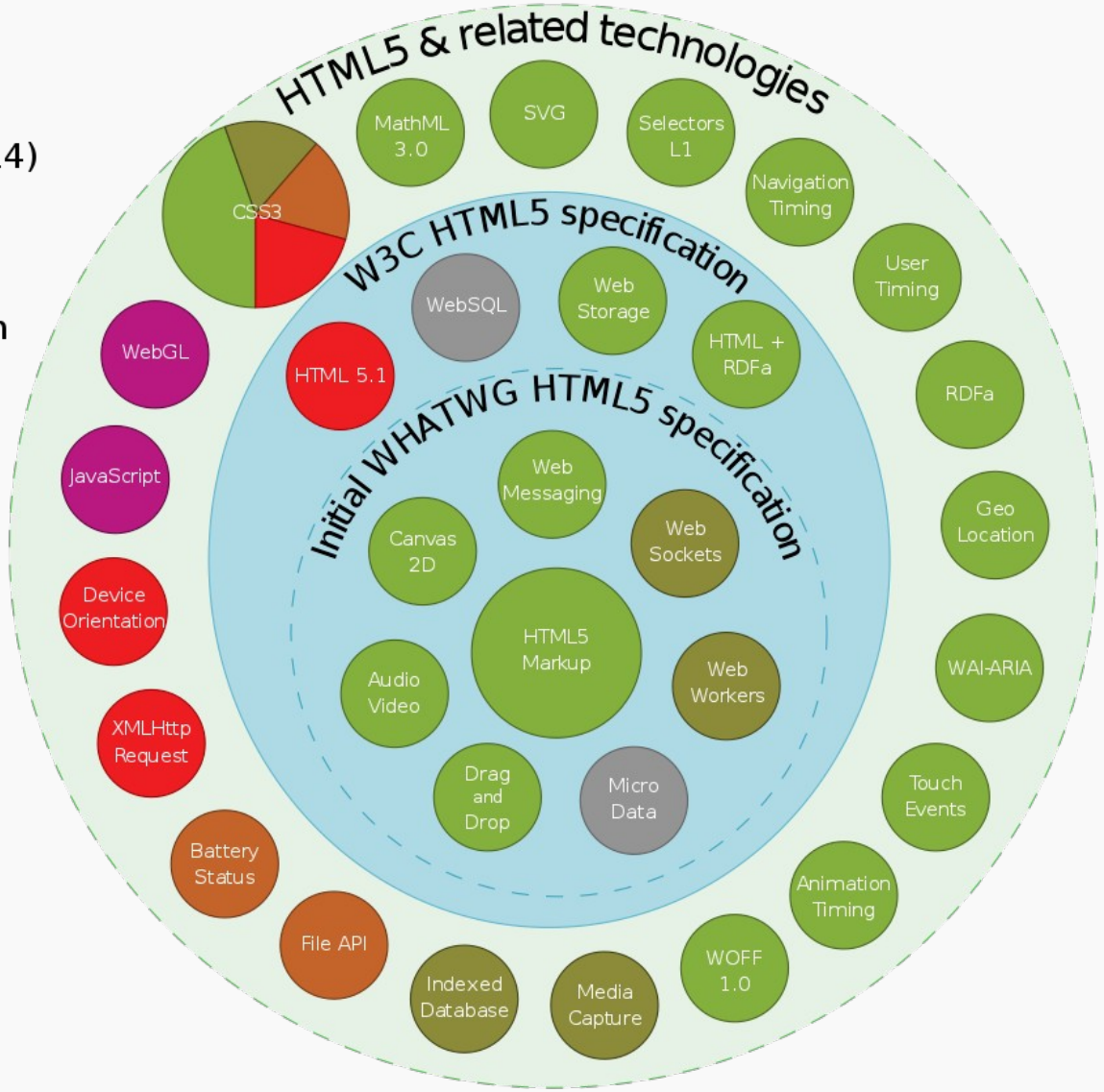
ИТМО ВТ

HTML5: изменения в API

HTML5

Taxonomy & Status (October 2014)

- Recommendation/Proposed
- Candidate Recommendation
- Last Call
- Working Draft
- Non-W3C Specifications
- Deprecated or inactive



ИТМО ВТ

HTML5: пример страницы

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Img Width Attribute</title>
  </head>
  <body>
    <img src="image.png" alt="Image" width="100" />
  </body>
</html>
```

ИТМО ВТ

HTML5: пример формы

```
<form>
  <label for="username">Username:</label>
  <input type="text" name="username" id="username" />
  <label for="password">Password:</label>
  <input type="password" name="password" id="password" />
  <input type="radio" name="gender" value="male" />Male<br />
  <input type="radio" name="gender" value="female" />Female<br />
  <input type="radio" name="gender" value="other" />Other
  <input list="Options" />
  <datalist id="Options">
    <option value="Option1"></option>
    <option value="Option2"></option>
    <option value="Option3"></option>
  </datalist>

  <input type="submit" value="Submit" />
  <input type="color" />
  <input type="checkbox" name="correct" value="correct" />Correct
</form>
```

2.3. Стилизация страниц. Основы CSS.

ИТМО ВТ

Что такое CSS

- CSS — технология описания внешнего вида документа, написанного языком разметки.
- Используется для задания цветов, шрифтов и других аспектов представления документа.
- Основная цель — разделение содержимого документа и его представления.
- Позволяет представлять один и тот же документ в различных методах вывода (например, обычная версия и версия для печати).

ИТМО ВТ

Источники CSS

- Авторские стили (информация стилей, предоставляемая автором страницы) в виде:
  - Inline-стилей — стиль элемента указывается в его атрибуте style.
  - Встроенных стилей — блоков CSS внутри самого HTML-документа.
  - Внешних таблиц стилей — отдельного файла .css.
- Пользовательские стили:
  - Локальный CSS-файл, указанный пользователем в настройках браузера, переопределяющий авторские стили.
- Стиль браузера:
  - Стандартный стиль, используемый браузером по умолчанию для представления элементов.

ИТМО ВТ

Структура CSS

- Таблица стилей состоит из набора *правил*.
- Каждое правило состоит из набора *селекторов* и *блока определений*:

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

- Пример:

```
div, td {
  background-color: red;
}
```

ИТМО ВТ

Приоритеты стилей

- Если к одному элементу «подходит» сразу несколько стилей, применён будет наиболее приоритетный.
- Приоритеты рассчитываются таким образом (от большего к меньшему):
  1. свойство задано при помощи `!important`;
  2. стиль прописан напрямую в теге;
  3. наличие идентификаторов (`#id`) в селекторе;
  4. количество классов (`.class`) и псевдоклассов (`:pseudoclass`) в селекторе;
  5. количество имён тегов в селекторе.
- Имеет значение относительный порядок расположения свойств — свойство, указанное позже, имеет приоритет.

ИТМО ВТ

Пример CSS

```
p {
  font-family: "Garamond", serif;
}

h2 {
  font-size: 110 %;
  color: red;
  background: white;
}

.note {
  color: red;
  background: yellow;
  font-weight: bold;
}
```

```
p#paragraph1 {
  margin: 0;
}

a:hover {
  text-decoration: none;
}

#news p {
  color: blue;
}
```

ИТМО ВТ

Пример страницы с CSS

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html;
charset=utf-8">
  <title>Заголовки</title>
  <style type="text/css">
   h1 { color: #a6780a; font-weight: normal; }
   h2 {
    color: olive;
    border-bottom: 2px solid black;
   }
  </style>
 </head>
 <body>
  <h1>Заголовок 1</h1>
  <h2>Заголовок 2</h2>
 </body>
</html>
```

ИТМО ВТ

LESS & Sass / SCSS

Языки стилей, позволяющие повысить уровень абстракции CSS-кода и упростить структуру таблиц стилей.

По сравнению с «обычным» CSS, имеются следующие особенности:

- Можно использовать переменные (константы и примеси).
- Можно использовать вложенные правила.
- Более мощные возможности по импорту, наследованию стилей.
- Поддержка математических операторов.

Браузеры могут не поддерживать LESS & Sass / SCSS-таблицы стилей — нужен специальный транслятор, который преобразует эти правила в «обычный» CSS.

LESS & Sass / SCSS (продолжение)

- LESS — CSS-like синтаксис:

```
.box-1 {
  color: #BADA55;
  .set-bg-color(#BADA55);
}
```

- SASS — Ruby-like синтаксис:

```
.my-element
    color= !primary-color
    width= 100%
    overflow= hidden
```

- SCSS — диалект SASS с CSS-like синтаксисом:

```
.my-element {
    color: $primary-color;
    width: 100%;
    overflow: hidden;
}
```

Константы

(SCSS)

```
//-- Font size ------
$md-font-size-h1: 24px;
$md-font-size-h2: 20px;
$md-font-size-h3: 16px;
$md-font-size-h4: 13px;
$md-font-size-h5: 12px;
$md-font-size-h6: 10px;

//-- Line height ------
$md-line-height-h1: 32px;
$md-line-height-h2: 28px;
$md-line-height-h3: 24px;
$md-line-height-h4: 24px;
$md-line-height-h5: 20px;
$md-line-height-h6: 20px;

//-- Font weight ------
$md-font-weight-regular: 400;

h1, h2, h3, h4, h5, h6 {
  font-weight: $md-font-weight-regular;
  margin: 0;
}

h1 {
  font-size: $md-font-size-h1;
  line-height: $md-line-height-h1;
}
```

```
h2 {
  font-size: $md-font-size-h2;
  line-height: $md-line-height-h2;
}

h3 {
  font-size: $md-font-size-h3;
  line-height: $md-line-height-h3;
}

h4 {
  font-size: $md-font-size-h4;
  line-height: $md-line-height-h4;
}

h5 {
  font-size: $md-font-size-h5;
  line-height: $md-line-height-h5;
}

h6{
  font-size: $md-font-size-h6;
  line-height: $md-line-height-h6;
}
```

ИТМО ВТ

Примеси

Код на SCSS:

```
@mixin border-radius($radius,$border,
$color) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
    border:$border solid $color
}

.box {
  @include border-radius(10px,1px,red);
}
```

«Обычный» CSS:

```
.box {
   -webkit-border-radius: 10px;
   -moz-border-radius: 10px;
   -ms-border-radius: 10px;
   border-radius: 10px;
   border: 1px solid red;
}
```

ИТМО ВТ

Вложенные стили

Код на SCSS:

```
#header {
    background: #FFFFFF;
    .error {
        color: #FF0000;
    }
    a {
        text-decoration:
            none;
        &:hover {
            text-decoration:
                underline;
        }
    }
}
```

Может быть преобразован в «обычный» CSS:

```
#header {
   background: #FFFFFF;
}
#header .error {
   color: #FF0000;
}
#header a {
   text-decoration: none;
}
#header a:hover {
   text-decoration:
underline;
}
```

ИТМО ВТ

Импорт стилей

```
// _reset.scss

html,
body,
ul,
ol {
  margin: 0;
  padding: 0;
}
```

```
// base.scss

@import 'reset';

body {
  font: 100% Helvetica, sans-serif;
  background-color: #efefef;
}
```

ИТМО ВТ

Наследование

```
.message {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  @extend .message;
  border-color: green;
}
```

```
.error {
  @extend .message;
  border-color: red;
}

.warning {
  @extend .message;
  border-color: yellow;
}
```

ИТМО ВТ

Математика

```
.container { width: 100%; }

article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}

aside[role="complementary"] {
  float: right;
  width: 300px / 960px * 100%;
}
```

ИТМО ВТ

Компиляция в CSS

Пример для Maven:

```
<!-- Sass compiler -->
<plugin>
    <groupId>org.jasig.maven</groupId>
    <artifactId>sass-maven-plugin</artifactId>
    <version>2.25</version>
    <executions>
        <execution>
            <phase>prepare-package</phase>
            <goals>
                <goal>update-stylesheets</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <resources>
            <resource>
                <!-- Set source and destination dirs -->
                <source>
                    <directory>${project.basedir}/src/main/webapp/sass</directory>
                </source>

<destination>${project.basedir}/src/main/webapp/sass_compiled</destination>
            </resource>
        </resources>
    </configuration>
</plugin>
```

2.4. Динамические сценарии на стороне клиента. Язык JavaScript.

JavaScript и клиентские сценарии

- JavaScript — объектно-ориентированный скриптовый язык программирования.
- Используется для придания интерактивности веб-страницам.
- Основные архитектурные черты:
  - динамическая типизация;
  - слабая типизация;
  - автоматическое управление памятью;
  - прототипное программирование;
  - функции как объекты первого класса.

ИТМО ВТ

Особенности синтаксиса

- Все идентификаторы регистрозависимы.
- В названиях переменных можно использовать буквы, подчёркивание, символ доллара, арабские цифры.
- Названия переменных не могут начинаться с цифры,
- Для оформления однострочных комментариев используются //, многострочные и внутристрочные комментарии начинаются с /* и заканчиваются */.

ИТМО ВТ

Структура языка

- Ядро (ECMAScript);
- Объектная модель браузера (Browser Object Model);
- Объектная модель документа (Document Object Model).

ИТМО ВТ

Особенности ECMAScript

- Встраиваемый расширяемый не имеющий средств ввода/вывода язык программирования.
- 5 примитивных типов данных — Number, String, Boolean, Null и Undefined.
- Объектный тип данных — Object.
- 15 различных видов инструкций.

ИТМО ВТ

Особенности ECMAScript (продолжение)

Блок не ограничивает область видимости переменной:

```
function foo() {
    var sum = 0;
    for (var i = 0; i < 42; i += 2) {
        var tmp = i + 2;
        sum += i * tmp;
    }
    for (var i = 1; i < 42; i += 2) {
        sum += i*i;
    }
    alert(tmp);
    return sum;
}

foo();
```

ИТМО ВТ

Особенности ECMAScript (продолжение)

Если переменная объявляется вне функции, то она попадает в глобальную область видимости:

```
var a = 42;

function foo() {
    alert(a);
}

foo();
```

ИТМО ВТ

Особенности ECMAScript (продолжение)

Функция — это тоже объект:

```
// объявление функции
function sum(arg1, arg2) {
    return arg1 + arg2;
}

// задание функции с помощью инструкции
var sum2 = function(arg1, arg2) {
    return arg1 + arg2;
};

// задание функции с использованием
// объектной формы записи
var sum3 = new Function("arg1", "arg2",
         "return arg1 + arg2;");
```

ИТМО ВТ

Объектная модель браузера

- BOM — прослойка между ядром и DOM.
- Основное предназначение — управление окнами браузера и обеспечение их взаимодействия.
- Специфична для каждого браузера.
- Каждое из окон браузера представляется объектом `window`:

```
var contentsWindow;
contentsWindow =
  window.open("http://cs.ifmo.ru","contents");
```

ИТМО ВТ

Объектная модель браузера (продолжение)

- Возможности BOM:
  - управление фреймами;
  - поддержка задержки в исполнении кода и зацикливания с задержкой;
  - системные диалоги;
  - управление адресом открытой страницы;
  - управление информацией о браузере;
  - управление информацией о параметрах монитора;
  - ограниченное управление историей просмотра страниц;
  - поддержка работы с HTTP cookie.

ИТМО ВТ

Объектная модель документа

- С помощью JavaScript можно производить следующие манипуляции:
  - получение узлов:
    `document.all("image1").outerHTML;`
  - изменение узлов;
  - изменение связей между узлами;
  - удаление узлов.

Встраивание в веб-страницы

- Внутри страницы:

```
<script type="text/javascript">
   alert('Hello, World!');
</script>
```

- Внутри тега:

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

- Отделение от разметки (используется DOM):

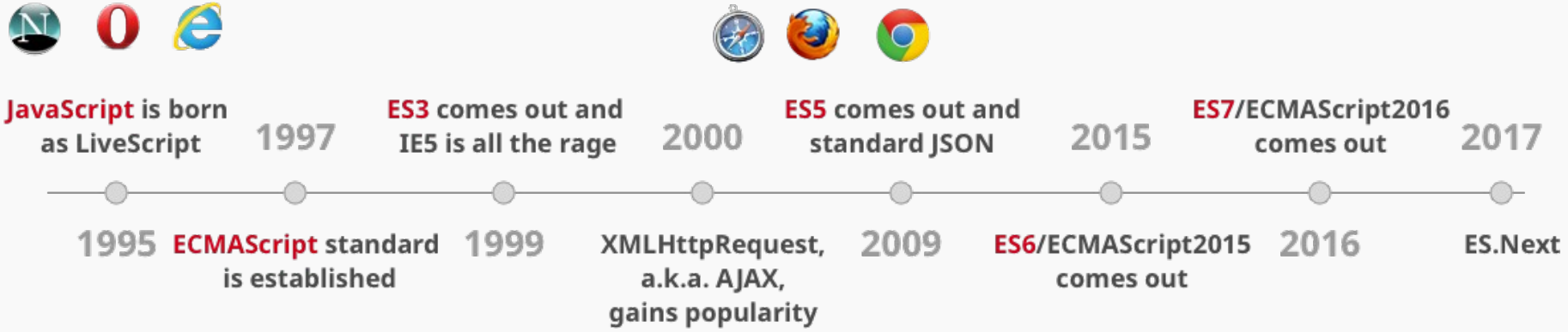
```
window.onload = function() {
    var linkWithAlert = document.getElementById("alertLink");
    linkWithAlert.onclick = function() {
        return confirm('Вы уверены?');
    };
};
...
<a href="delete.php" id="alertLink">Удалить</a>
```

- В отдельном файле:

```
<script type="text/javascript"
   src="http://Путь_к_файлу_со_скриптом"></script>
```

ИТМО ВТ

ES6 / ES2015+



- ES6 — новая версия языка ECMAScript, выпущенная в 2015 г.
- Добавляет в синтаксис языка множество новых возможностей.
- Поддерживается практически всеми современными браузерами (хотя с этим всё ещё могут быть нюансы).
- Для работы в старых браузерах может потребоваться специальная программа — *транспилер (transpiler)*.

ИТМО ВТ

Поддержка ES6 браузерами

		Compilers/polyfills							Desktop browsers														
	98%	56%	71%	71%	72%	50%	69%	17%	5%	11%	96%	96%	98%	98%	98%	98%	98%	98%	98%	98%	98%	98%	98%
Feature name	Current browser	Traceur	Babel 6 + core-js 2	Babel 7 + core-js 2	Babel 7 + core-js 3	Closure 2019.07	TypeScript + core-js 3	es6-shim	Konq 4.14[1]	IE 11	Edge 17	Edge 18	FF 60 ESR	FF 68 ESR	FF 69	FF 70 Beta	FF 71 Nightly	CH 75, OP 62	CH 76, OP 63	CH 77, OP 64	CH 78, OP 65	SF 12
proper tail calls (tail call optimisation)	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2
Syntax																						
default function parameters	7/7	4/7	4/7	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7
rest parameters	5/5	4/5	3/5	3/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5	5/5
spread syntax for iterable objects	15/15	15/15	13/15	13/15	14/15	11/15	14/15	0/15	0/15	0/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15	15/15
object literal extensions	6/6	6/6	6/6	6/6	6/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
for..of loops	9/9	9/9	9/9	9/9	9/9	6/9	9/9	0/9	0/9	0/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9	9/9
octal and binary literals	4/4	2/4	4/4	4/4	4/4	2/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4
template literals	7/7	6/7	6/7	6/7	6/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	7/7	6/7
RegExp "y" and "u" flags	6/6	4/6	4/6	4/6	4/6	0/6	0/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6
destructuring, declarations	22/22	20/22	21/22	21/22	21/22	20/22	21/22	0/22	0/22	0/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22	22/22
destructuring, assignment	24/24	23/24	24/24	24/24	24/24	22/24	24/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
destructuring, parameters	24/24	19/24	21/24	21/24	21/24	20/24	21/24	0/24	0/24	0/24	23/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
Unicode code point escapes	2/2	1/2	1/2	1/2	1/2	1/2	1/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2	2/2
Bindings																						
const	18/18	16/18	16/18	16/18	16/18	16/18	16/18	0/18	2/18	14/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18	18/18
let	14/14	12/14	12/14	12/14	12/14	12/14	12/14	0/14	0/14	12/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14	14/14
block-level function declaration[18]	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Functions																						
arrow functions	13/13	11/13	9/13	9/13	9/13	10/13	9/13	0/13	0/13	0/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13	13/13
class	24/24	17/24	19/24	19/24	19/24	14/24	19/24	0/24	0/24	0/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24	24/24
super	8/8	7/8	4/8	4/8	4/8	7/8	7/8	0/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8
generators	27/27	24/27	24/27	24/27	24/27	19/27	22/27	0/27	0/27	0/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27	27/27
Built-ins																						
typed arrays	46/46	0/46	45/46	45/46	46/46	0/46	46/46	0/46	8/46	16/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46	46/46
Map	19/19	14/19	19/19	19/19	19/19	15/19	19/19	15/19	0/19	8/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19
Set	19/19	14/19	19/19	19/19	19/19	15/19	19/19	15/19	0/19	8/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19	19/19

Block Scope Variables

- Два новых ключевых слова — `let` и `const`.
- ES5:

```
var x = 'outer';
function test(inner) {
  if (inner) {
    var x = 'inner'; // scope whole function
    return x;
  }
  return x; // gets redefined on line 4
}

test(false); // undefined
test(true); // inner
```

ИТМО ВТ

Ключевое слово let

- Позволяет объявить переменную, областью видимости которой является блок.
- ES6:

```
let x = 'outer';
function test(inner) {
  if (inner) {
    let x = 'inner'; // scope whole function
    return x;
  }
  return x; // gets redefined on line 4
}

test(false); // outer
test(true); // inner
```

IIFE (Immediately Invoked Function Expression)

ES5:

```
{
  var private = 1;
}

// 1
console.log(private);
```

ES5 & IIFE («костыли»):

```
(function(){
  var private2 = 1;
})();

// Uncaught ReferenceError
console.log(private2);
```

ES6:

```
{
  let private3 = 1;
}

// Uncaught ReferenceError
console.log(private3);
```

ИТМО ВТ

Ключевое слово const

Позволяет объявить константу:

```
// define MY_FAV as a constant and give it the value 7
const MY_FAV = 7;

// this will throw an error
MY_FAV = 20;

// will print 7
console.log('my favorite number is: ' + MY_FAV);

// trying to redeclare a constant throws an error
const MY_FAV = 20;

// the name MY_FAV is reserved for constant above,
// so this will fail too
var MY_FAV = 20;

// this throws an error too
let MY_FAV = 20;
```

Template Literals

ES5:

```
var first = 'Adrian';
var last = 'Mejia';
console.log('Your name is ' + first
    + ' ' + last + '.');
```

ES6:

```
const first = 'Adrian';
const last = 'Mejia';
console.log(`Your name is ${first} ${last}.`);
```

ИТМО ВТ

Деструктуризация

Деструктуризация (destructuring assignment) – особый синтаксис присваивания, при котором можно присвоить массив или объект сразу нескольким переменным, разбив его на части.

Пример — получение элемента из массива:

ES5:

```
var array = [1, 2, 3, 4];

var first = array[0];
var third = array[2];

console.log(first,
third); // 1 3
```

ES6:

```
const array = [1, 2, 3, 4];

const [first, ,third] =
array;

console.log(first,
third); // 1 3
```

ИТМО ВТ

Деструктуризация — обмен значениями

ES5:

```
var a = 1;
var b = 2;

var tmp = a;
a = b;
b = tmp;

// 2 1
console.log(a, b);
```

ES6:

```
let a = 1;
let b = 2;

[a, b] = [b, a];

console.log(a, b); // 2 1
```

ИТМО ВТ

Деструктуризация нескольких возвращаемых значений

ES5:

```
function margin() {
  var left=1, right=2, top=3, bottom=4;
  return { left: left, right: right, top: top, bottom: bottom };
}

var data = margin();
var left = data.left;
var bottom = data.bottom;

console.log(left, bottom); // 1 4
```

ES6:

```
function margin() {
  const left=1, right=2, top=3, bottom=4;
  return { left, right, top, bottom };
}

const { left, bottom } = margin();

console.log(left, bottom); // 1 4
```

ИТМО ВТ

Деструктуризация и сопоставление параметров

ES5:

```
var user = {firstName: 'Adrian', lastName: 'Mejia'};

function getFullName(user) {
  var firstName = user.firstName;
  var lastName = user.lastName;

  return firstName + ' ' + lastName;
}

console.log(getFullName(user)); // Adrian Mejia
```

ES6:

```
const user = {firstName: 'Adrian', lastName: 'Mejia'};

function getFullName({ firstName, lastName }) {
  return `${firstName} ${lastName}`;
}

console.log(getFullName(user)); // Adrian Mejia
```

ИТМО ВТ

Деструктуризация объекта

ES5:

```
function settings() {
  return { display: { color: 'red' }, keyboard: { layout: 'querty'} };
}

var tmp = settings();
var displayColor = tmp.display.color;
var keyboardLayout = tmp.keyboard.layout;

console.log(displayColor, keyboardLayout); // red querty
```

ES6:

```
function settings() {
  return { display: { color: 'red' }, keyboard: { layout: 'querty'} };
}

const { display: { color: displayColor }, keyboard: { layout:
keyboardLayout }} = settings();

console.log(displayColor, keyboardLayout); // red querty
```

ИТМО ВТ

Классы и объекты

В ES6 появился новый синтаксис описания и инициализации объектов:

ES5:

```
var Animal = (function () {
  function MyConstructor(name) {
    this.name = name;
  }
  MyConstructor.prototype.speak =
    function speak() {
    console.log(this.name +
      ' makes a noise.');
  };
  return MyConstructor;
})();

var animal =
  new Animal('animal');
// animal makes a noise.
animal.speak();
```

ES6:

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(this.name
      + ' makes a noise.');
  }
}

const animal =
  new Animal('animal');
// animal makes a noise.
animal.speak();
```

ИТМО ВТ

Наследование

Новые ключевые слова extends и super.

ES5:

```
var Lion = (function () {
  function MyConstructor(name){
    Animal.call(this, name);
  }

  // prototypal inheritance
  MyConstructor.prototype =
    Object.create(Animal.prototype);
  MyConstructor.prototype.constructor =
Animal;

  MyConstructor.prototype.speak =
  function speak() {
    Animal.prototype.speak.call(this);
    console.log(this.name + ' roars ');
  };
  return MyConstructor;
})();

var lion = new Lion('Simba');
lion.speak(); // Simba makes a noise.
// Simba roars.
```

ES6:

```
class Lion extends Animal {
  speak() {
    super.speak();
    console.log(this.name + ' roars
');
  }
}

const lion = new Lion('Simba');
lion.speak(); // Simba makes a noise.
// Simba roars.
```

Промисы (promises) вместо коллбэков

ES5:

```
function printAfterTimeout(string,
timeout, done){
  setTimeout(function(){
    done(string);
  }, timeout);
}

printAfterTimeout('Hello ', 2e3,
function(result){
  console.log(result);

  // nested callback
  printAfterTimeout(result +
    'Reader', 2e3,
    function(result){
    console.log(result);
  });
});
```

ES6:

```
function printAfterTimeout(string,
timeout){
  return new Promise(
    (resolve, reject) => {
    setTimeout(function(){
      resolve(string);
    }, timeout);
  });
}

printAfterTimeout('Hello ', 2e3)
  .then((result) => {
  console.log(result);
  return printAfterTimeout(result
    + 'Reader', 2e3);
}).then((result) => {
  console.log(result);
});
```

ИТМО ВТ

Стрелочные функции

ES5:

```
var _this = this; // need to hold a reference

$('.btn').click(function(event){
  _this.sendData(); // reference outer this
});

$('.input').on('change',function(event){
  this.sendData(); // reference outer this
}.bind(this)); // bind to outer this
```

ES6:

```
// this will reference the outer one
$('.btn').click((event) => this.sendData());

// implicit returns
const ids = [291, 288, 984];
const messages = ids.map(value => `ID is ${value}`);
```

ИТМО ВТ

Цикл с итератором

ES5:

```
// for
var array = ['a', 'b', 'c', 'd'];
for (var i = 0; i < array.length; i++) {
  var element = array[i];
  console.log(element);
}

// forEach
array.forEach(function (element) {
  console.log(element);
});
```

ES6:

```
// for ...of
const array = ['a', 'b', 'c', 'd'];
for (const element of array) {
    console.log(element);
}
```

ИТМО ВТ

Параметры по умолчанию

ES5:

```
function point(x, y, isFlag){
  x = x || 0;
  y = typeof(y) ===
    'undefined' ? -1 : y;
  isFlag =
    typeof(isFlag) ===
      'undefined' ?
        true : isFlag;
  console.log(x,y, isFlag);
}

point(0, 0) // 0 0 true
point(0, 0, false) // 0 0
false
point(1) // 1 -1 true
point() // 0 -1 true
```

ES6:

```
function point(x = 0, y = -1,
isFlag = true){
  console.log(x,y,isFlag);
}

point(0, 0) // 0 0 true
point(0, 0, false) // 0 0 false
point(1) // 1 -1 true
point() // 0 -1 true
```

ИТМО ВТ

Rest-параметры

Аналог vararg в Java.

ES5:

```
function printf(format) {
  var params = [].slice.call(arguments, 1);
  console.log('params: ', params);
  console.log('format: ', format);
}

printf('%s %d %.2f', 'adrian', 321, Math.PI);
```

ES6:

```
function printf(format, ...params) {
  console.log('params: ', params);
  console.log('format: ', format);
}

printf('%s %d %.2f', 'adrian', 321, Math.PI);
```

ИТМО ВТ

Операция spread

ES5:

```
Math.max.apply(Math,
[2,100,1,6,43]) // 100

var array1 =
  [2,100,1,6,43];
var array2 =
  ['a', 'b', 'c', 'd'];
var array3 =
  [false, true, null,
    undefined];

console.log(array1
  .concat(array2,
array3));
```

ES6:

```
Math.max(...[2,100,1,6,43])
// 100

const array1 =
  [2,100,1,6,43];
const array2 =
  ['a', 'b', 'c', 'd'];
const array3 =
  [false, true, null,
    undefined];

console.log([...array1,
  ... array2, ...array3]);
```

2.5. Асинхронное клиент-серверное взаимодействие. DHTML и AJAX

ИТМО ВТ

DHTML

- Dynamic HTML — способ создания интерактивного веб-сайта, использующий сочетание:
  - статичного языка разметки HTML;
  - выполняемого на стороне клиента скриптового языка JavaScript;
  - CSS (каскадных таблиц стилей);
  - DOM (объектной модели документа).

Пример страницы DHTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
<html>

<head>
  <title>Заголовок страницы</title>
  <script type="text/javascript">
   window.onload= function () {
    myObj = document.getElementById("navigation");
    // .... какой-то код
   }
  </script>
</head>

<body>
  <div id="navigation">
  </div>
</body>
</html>
```

ИТМО ВТ

Что такое AJAX

- *AJAX* (*Asynchronous Javascript and XML*) — подход к построению интерактивных пользовательских интерфейсов веб-приложений.
- Основан на «фоновом» обмене данными браузера с веб-сервером.
- При обмене данными между клиентом и сервером веб-страница не перезагружается полностью.

ИТМО ВТ

Основные принципы AJAX

- Использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью, например:
  - с использованием XMLHttpRequest;
  - через динамическое создание дочерних фреймов;
  - через динамическое создание тега <script>.
- Использование DHTML для динамического изменения содержания страницы.

ИТМО ВТ

XMLHttpRequest

- XMLHTTP (XMLHttpRequest, XHR) — набор API, позволяющий осуществлять HTTP-запросы к серверу без необходимости перезагружать страницу.
- Данные можно пересылать в виде XML, JSON, HTML или просто неструктурированным текстом.
- При пересылке используется текстовый протокол HTTP и потому данные должны передаваться в виде текста.

XMLHttpRequest (пример)

```
var req;

function loadXMLDoc(url) {
    req = null;
    if (window.XMLHttpRequest) {
        try {
            req = new XMLHttpRequest();
        } catch (e){}
    } else if (window.ActiveXObject) {
        try {
            req = new ActiveXObject('Msxml2.XMLHTTP');
        } catch (e){
            try {
                req = new ActiveXObject('Microsoft.XMLHTTP');
            } catch (e){}
        }
    }

    if (req) {
        req.open("GET", url, true);
        req.onreadystatechange = processReqChange;
        req.send(null);
    }
}
```

ИТМО ВТ

XMLHttpRequest (пример, продолжение)

```
function processReqChange() {
  try { // Важно!
    // только при состоянии "complete"
    if (req.readyState == 4) {
        // для статуса "OK"
        if (req.status == 200) {
            // обработка ответа
        } else {
            alert("Не удалось получить данные:\n" +
                req.statusText);
        }
    }
  }

  catch( e ) {
      // alert('Ошибка: ' + e.description);
      // В связи с багом XMLHttpRequest в Firefox
    // приходится отлавливать ошибку
  }
}
```

ИТМО ВТ

Преимущества и недостатки AJAX

- Преимущества:
  - экономия трафика;
  - уменьшение нагрузки на сервер;
  - ускорение реакции интерфейса;
- Недостатки:
  - отсутствие интеграции со стандартными инструментами браузера;
  - динамически загружаемое содержимое недоступно поисковикам;
  - старые методы учёта статистики сайтов становятся неактуальными;
  - усложнение проекта;
  - требуется включенный JavaScript в браузере.

ИТМО ВТ

Протокол WebSocket

- *WebSocket* — протокол полнодуплексной связи поверх TCP-соединения, предназначенный для обмена сообщениями между браузером и веб-сервером в режиме реального времени.
- Позволяет серверу отправлять данные браузеру без дополнительного запроса со стороны клиента.
- Обмен данными ведётся через отдельное TCP-соединение.
- Поддерживается всеми современными браузерами (даже IE).
- Альтернатива — AJAX + Long Polling.

Протокол WebSocket (продолжение)

```
<script>

   var webSocket = new WebSocket('ws://localhost/echo');

   webSocket.onopen = function(event) {
      alert('onopen');
      webSocket.send("Hello Web Socket!");
   };

   webSocket.onmessage = function(event) {
      alert('onmessage, ' + event.data);
      webSocket.close();
   };

   webSocket.onclose = function(event) {
      alert('onclose');
   };

</script>
```

ИТМО ВТ

Библиотека jQuery

- JS-библиотека, предназначенная для разработки DHTML и AJAX-приложений.
- Упрощает доступ к элементам DOM с помощью кучи разных способов.
- Упрощает и унифицирует (для разных браузеров) реализацию AJAX.
- Упрощает добавление визуальных эффектов.
- Ключевым элементом API является функция (объект) `$` и её синоним `jQuery`.

ИТМО ВТ

AJAX на jQuery

Без jQuery (и без кроссбраузерности):

```
req = new XMLHttpRequest();
req.open("POST", "some.php", true);
req.onreadystatechange = processReqChange;
req.send(null);
if (req.readyState == 4) {
    if (req.status == 200) {
        alert( "Data Saved " );
    }
}
```

С jQuery:

```
$.ajax({
  type: "POST",
  url: "some.php",
  data: {name: 'John', location: 'Boston'},
  success: function(msg){
    alert( "Data Saved: " + msg );
  }
});
```

jQuery: взаимодействие с DOM

```
$("div.test")
    .add("p.quote")
    .addClass("blue")
    .slideDown("slow");


$("a").click(function() {
    alert("Hello world!");
});

$( "div.demo-container" )
  .html( "<p>All new content. </p>" );
```

ИТМО ВТ

jQuery: работа с событиями

```
$("#foo").bind( "mouseenter mouseleave", function() {
  $( this ).toggleClass( "entered" );
});


$( document ).ready(function() {
  $( "#foo" ).bind( "click", function( event ) {
    alert( "The mouse cursor is at (" +
      event.pageX + ", " + event.pageY +
      ")" );
  });
});

$( "#other" ).click(function() {
  $( ".target" ).change();
});



$( "#target" ).click(function() {
  alert( "Handler for .click() called." );
});
```

ИТМО ВТ

jQuery: эффекты и анимация

```
$( "#clickme" ).click(function() {
  $( "#book" ).animate({
    opacity: 0.25,
    left: "+=50",
    height: "toggle"
  }, 5000, function() {
    // Animation complete.
  });
});

$( "#foo" ).slideUp( 300 )
  .delay( 800 ).fadeIn( 400 );

$( "#clickme" ).click(function() {
  $( "#book" ).slideDown( "slow", function() {
    // Animation complete
  });
});
```

ИТМО ВТ

SuperAgent

API для реализации AJAX:

```
request
   .post('/api/pet')
   .send({ name: 'Manny', species: 'cat' })
   .set('X-API-Key', 'foobar')
   .set('Accept', 'application/json')
   .end(function(err, res){
     if (err || !res.ok) {
       alert('Oh no! Error');
     } else {
       alert('yay got ' + JSON.stringify(res.body));
     }
   });
```

3. «Классические» интернет-приложения

ИТМО ВТ

Интернет-приложения

- Интернет-приложение — это сайт с той или иной динамической функциональностью на стороне сервера.
- Интернет-приложение осуществляет вызов программ на стороне сервера, к примеру:
  - Браузер отправляет на веб-сервер запрос на получение HTML-формы.
  - Веб-сервер формирует HTML-форму и возвращает её браузеру.
  - Браузер отправляет на сервер новый запрос с данными из HTML-формы.
  - Веб-сервер делегирует обработку данных из формы какой-либо программе на стороне сервера.

ИТМО ВТ

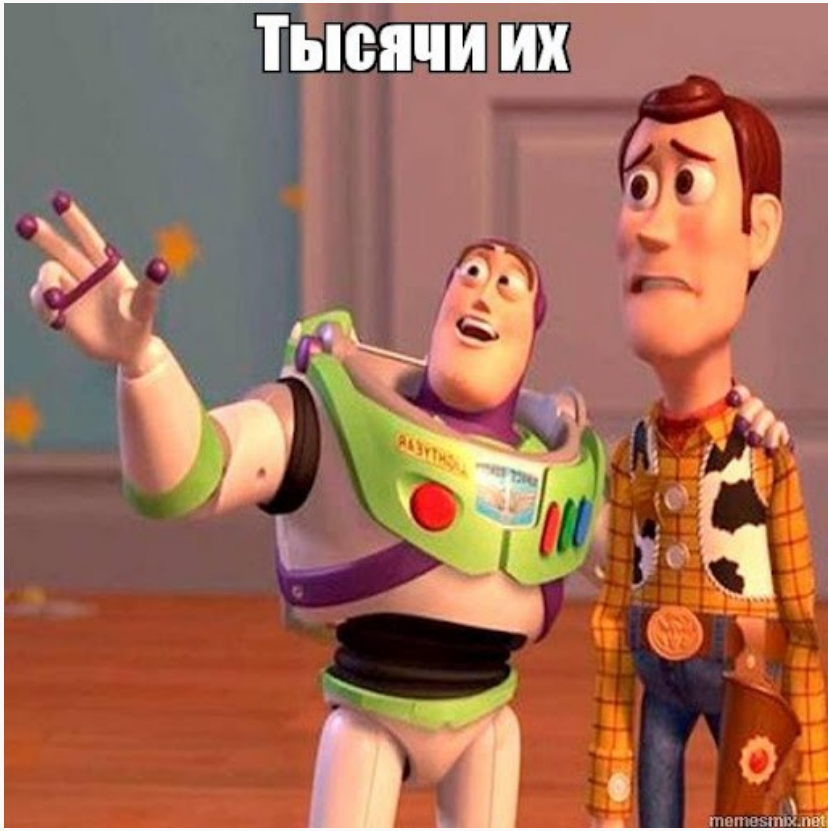
Особенности архитектуры

- Динамический контент формируют серверные сценарии.
- Разметка страниц задаётся с помощью шаблонов.
- Клиент-серверное взаимодействие реализуется либо «вручную», либо с помощью низкоуровневого фреймворка а-ля JQuery.
- Активно используются архитектурные паттерны, за их реализацию отвечает сам программист.

ИТМО ВТ

Технологии для создания веб-приложений

- HTML over HTTP
- Common Gateway Interface (CGI)
- FastCGI
- PHP
- Servlets
- JavaServer Pages (JSP)
- JavaServer Faces

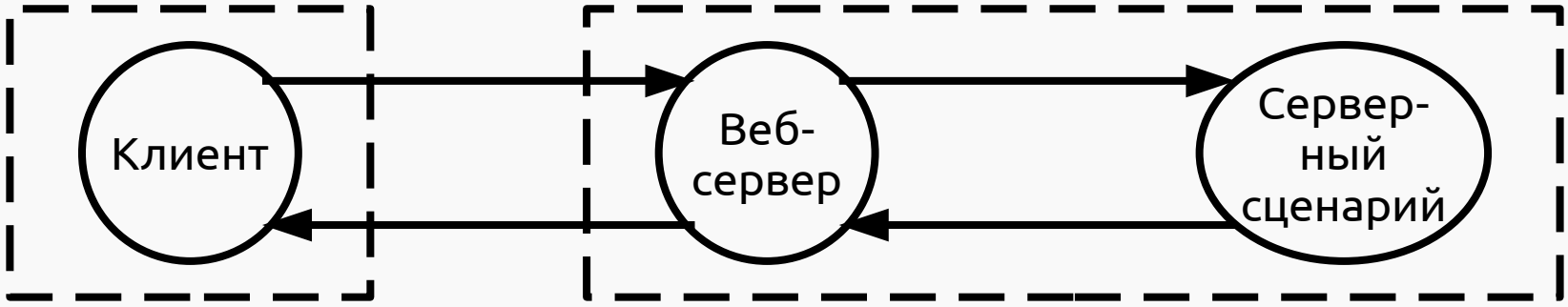


3.1. Серверные сценарии

ИТМО ВТ

Серверные сценарии

- Программы, вызываемые на сервере для формирования динамического контента.
- Веб-сервер «делегирует» им на обработку запрос и «транслирует» сформированный ими ответ клиенту.
- Могут использоваться вместе со «статикой», могут – сами по себе.



3.1.1. CGI & FastCGI

ИТМО ВТ

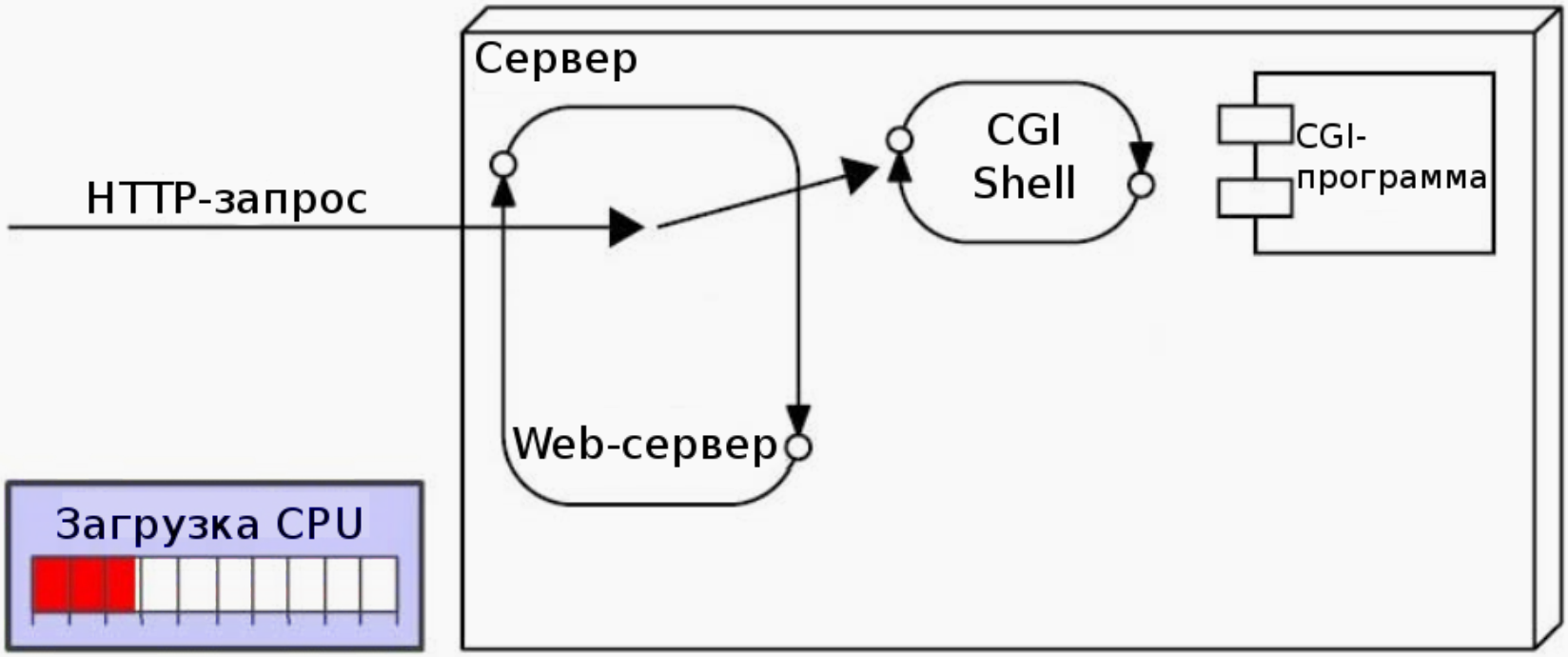
Common Gateway Interface

- CGI — простейший механизм вызова пользователем программ на стороне сервера.
- Данные отправляются программе посредством HTTP-запроса, формируемого веб-браузером.
- То, какая именно программа будет вызвана, обычно определяется URL запроса.
- Каждый запрос обрабатывается отдельным процессом CGI-программы.
- Взаимодействие программы с веб-сервером осуществляется через stdin и stdout.

ИТМО ВТ

Выполнение CGI-сценариев

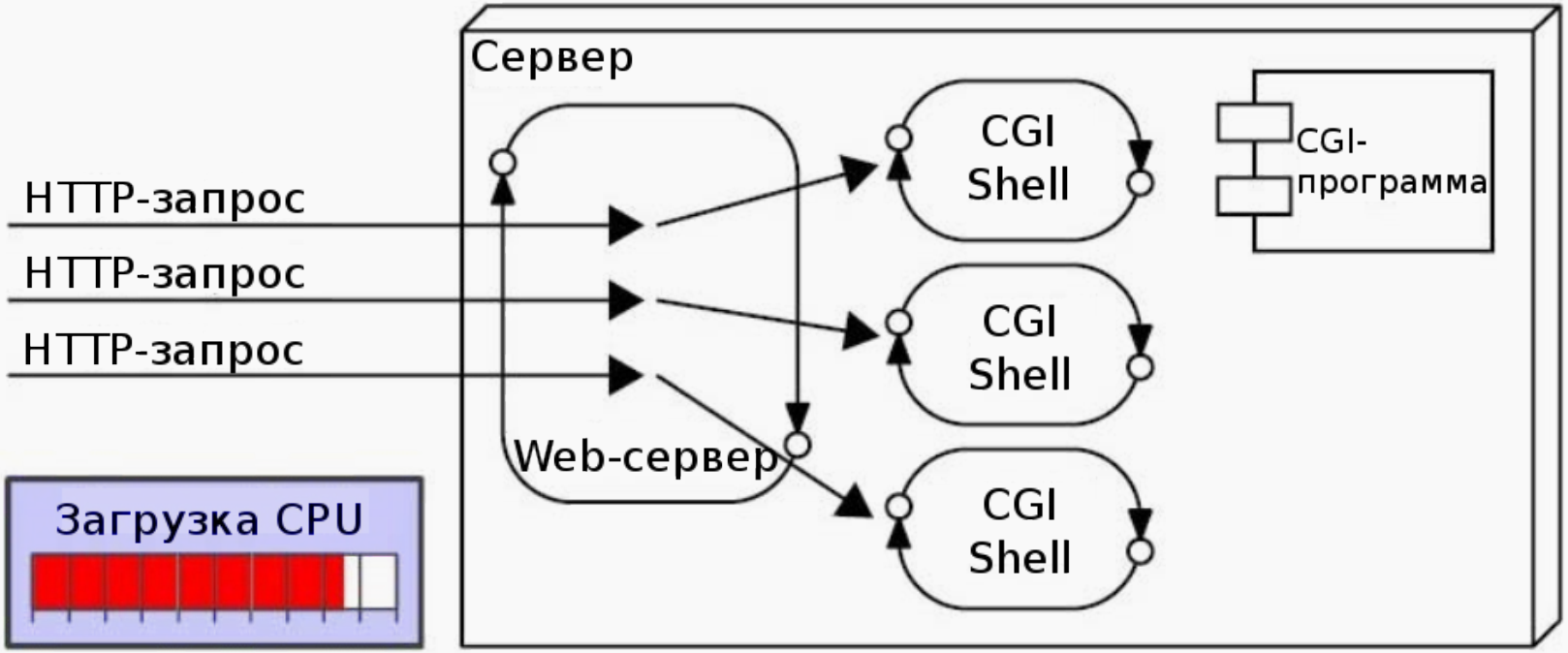
Один запрос:



ИТМО ВТ

Выполнение CGI-сценариев (продолжение)

Параллельная обработка нескольких запросов:



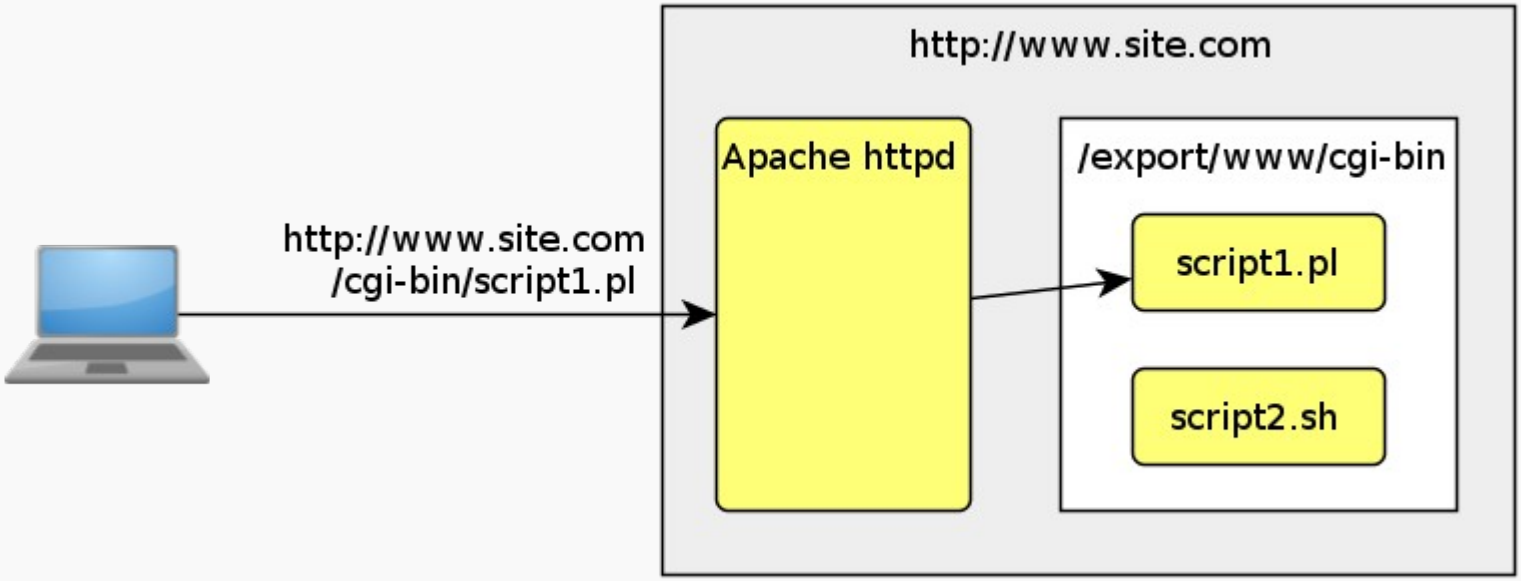
Пример реализации CGI-сценария

```
#include <stdio.h>

int main(void) {
   printf("Content-Type:
      text/html;charset=UTF-8\n\n");

   printf("<HTML>\n");
   printf("<HEAD>\n");
   printf("<TITLE>Hello, World!</TITLE>\n");
   printf("</HEAD>\n");
   printf("<BODY>\n");
   printf("<H1>Hello, World</H1>\n");
   printf("</BODY>\n");
   printf("</HTML>\n");

   return 0;
}
```

Конфигурация веб-сервера

Apache (httpd.conf):

```
ScriptAlias /cgi-bin/ "/opt/www/cgi-bin/"
```



ИТМО ВТ

Достоинства и недостатки CGI-сценариев

- Достоинства:
  - Программы могут быть написаны на множестве языков программирования.
  - «Падение» CGI-сценария не приводит к «падению» всего сервера.
  - Исключены конфликты при параллельной обработке нескольких запросов.
  - Хорошая поддержка веб-серверами.
- Недостатки:
  - Высокие накладные расходы на создание нового процесса.
  - Плохая масштабируемость.
  - Слабое разделение уровня представления и бизнес-логики.
  - Могут быть платформо-зависимыми.

ИТМО ВТ

FastCGI

- Развитие технологии CGI.
- Все запросы могут обрабатываться одним процессом CGI-программы (фактическая реализация определяется программистом).
- Веб-сервер взаимодействует с процессом через UNIX Domain Sockets или TCP/IP (а не через stdin и stdout).

3.1.2. Серверные сценарии на PHP

ИТМО ВТ

Язык PHP

- PHP (*PHP: Hypertext Preprocessor*) — скриптовый язык, часто используемый для написания веб-приложений.
- Первая версия разработана в 1994 г. Расмусом Лердорфом (Rasmus Lerdorf).
- Распространяется по лицензии с открытым исходным кодом.

ИТМО ВТ

Синтаксис PHP

- Интерпретатор выполняет код, находящийся внутри ограничителей:

```
<?php
  echo 'Hello, world!';
?>
```

- Имена переменных начинаются с символа «$»:

```
$hello = 'Hello, world!';
```

- Инструкции разделяются символом «;»:

```
$a = 'Hello '; $b = 'world!';
$c = $b + $a;
```

Синтаксис PHP (продолжение)

- Весь текст вне ограничителей оставляется интерпретатором без изменений:

```
<html>
 <head>
  <title>Тестируем PHP</title>
 </head>
 <body>
  <?php echo 'Hello, world!'; ?>
 </body>
</html>
```

ИТМО ВТ

Типы данных

- PHP — язык с динамической типизацией; при объявлении переменных их тип не указывается.
- 6 скалярных типов данных — integer, float, double, boolean, string и NULL. Диапазоны числовых типов зависят от платформы.
- 3 нескалярных типа — ресурс (например, дескриптор файла), массив и объект.
- 4 псевдотипа — mixed, number, callback и void.

ИТМО ВТ

Суперглобальные массивы (Superglobal Arrays)

Предопределённые массивы, имеющие глобальную область видимости:

- `$_GLOBALS` — массив всех глобальных переменных.
- `$_SERVER` — параметры, которые ОС передаёт серверу при его запуске.
- `$_ENV` — переменные среды ОС.

ИТМО ВТ

Суперглобальные массивы (продолжение)

- `$_GET`, `$_POST` — параметры GET- и POST-запроса:

```
<?php
   echo 'Привет, '
      .htmlspecialchars($_GET["name"])
      .'!';
?>
```

- `$_FILES` — сведения об отправленных методом POST файлах.
- `$_COOKIE` — массив cookies.
- `$_REQUEST` — содержит элементы из массивов `$_GET`, `$_POST`, `$_COOKIE` и `$_FILES`.
- `$_SESSION` — данные HTTP-сессии.

ИТМО ВТ

Поддержка ООП

- Полная поддержка появилась в PHP5.
- Реализованы все основные механизмы ООП — инкапсуляция, полиморфизм и наследование.
- Поля и методы могут быть приватными (private), публичными (public) и защищёнными (protected).
- Можно объявлять финальные и абстрактные методы и классы (аналогично Java).
- Множественное наследование не поддерживается, но есть интерфейсы и механизм особенностей (traits).
- Объекты передаются по ссылке.
- Обращение к константам, статическим свойствам и методам класса осуществляется с помощью конструкции «::».

ИТМО ВТ

Пример PHP-класса

```
class C1 extends C2 implements I1, I2
{
  private $a;
  protected $b;

  function __construct($a, $b)
  {
    parent::__construct($a, $b);
    $this->a = $a;
    $this->b = $b;
  }

  public function plus()
  {
    return $this->a + $this->b;
  }
/* ............... */
}

$d = new C1(1, 2);
echo $d->plus(); // 3
```

ИТМО ВТ

PHP Traits

Трейты — инструментарий для повторного использования кода. Появился в PHP 5.4.

```
<?php
trait ezcReflectionReturnInfo {
    function getReturnType() { /*1*/ }
    function getReturnDescription() { /*2*/ }
}

class ezcReflectionMethod extends ReflectionMethod {
    use ezcReflectionReturnInfo;
    /* ... */
}

class ezcReflectionFunction extends ReflectionFunction {
    use ezcReflectionReturnInfo;
    /* ... */
}
?>
```

ИТМО ВТ

Конфигурация и варианты использования интерпретатора PHP

- Конфигурационные параметры хранятся в файле php.ini.
- Можно подключать дополнительные *модули*, расширяющие возможности языка (например, добавляющие поддержку взаимодействия с СУБД).
- Способы использования интерпретатора PHP:
  - С помощью SAPI / ISAPI (например, mod_php для Apache).
  - С помощью CGI / FastCGI.
  - Через интерфейс командной строки.

3.1.3. Сервлеты

ИТМО ВТ

Платформа Java EE

- Набор стандартов и спецификаций для создания корпоративных приложений на Java.
- Спецификации Java EE реализуются серверами приложений:
  - Apache Tomcat
  - Sun / Oracle GlassFish
  - BEA / Oracle WebLogic
  - IBM WebSphere
  - RedHat JBoss

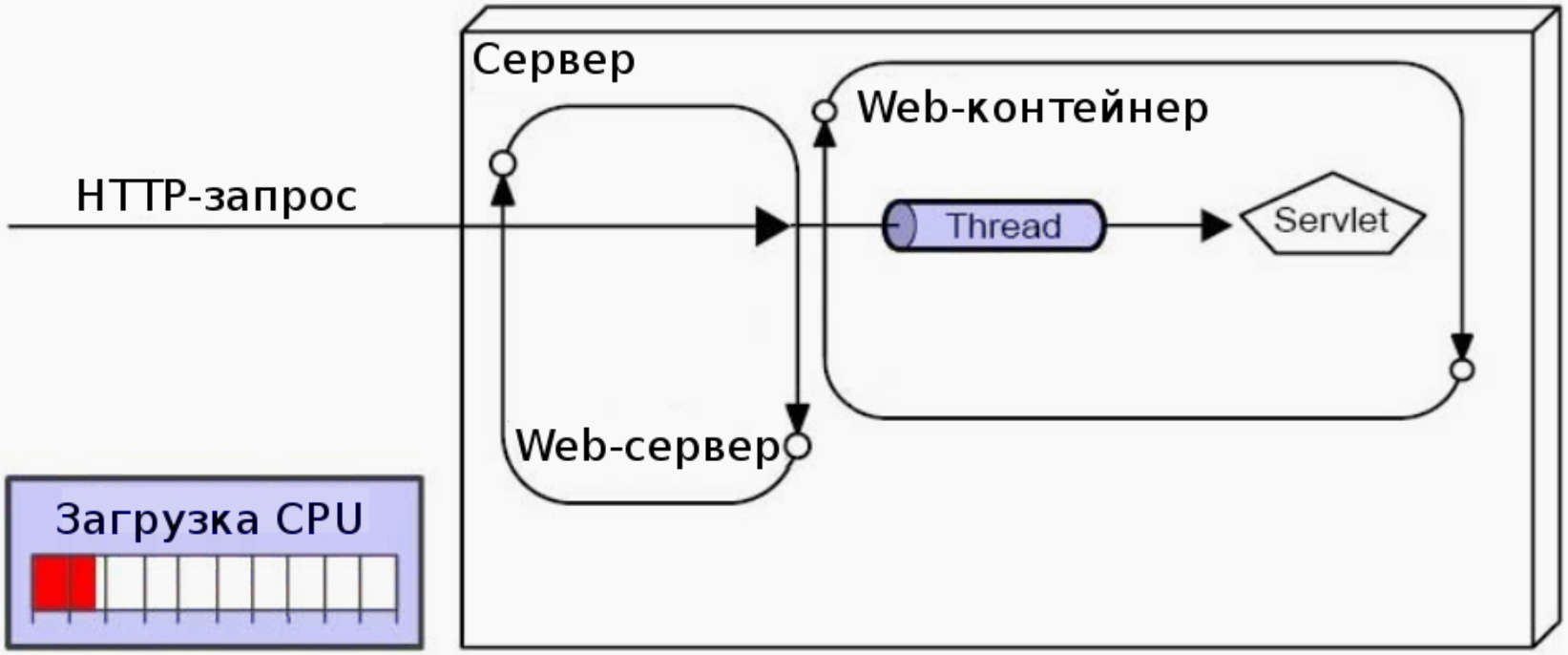
ИТМО ВТ

Java Servlets

- Сервлеты — это серверные сценарии, написанные на Java.
- Жизненным циклом сервлетов управляет веб-контейнер (он же контейнер сервлетов).
- В отличие от CGI, запросы обрабатываются в отдельных потоках (а не процессах) на веб-контейнере.

Обработка HTTP-запросов сервлетом

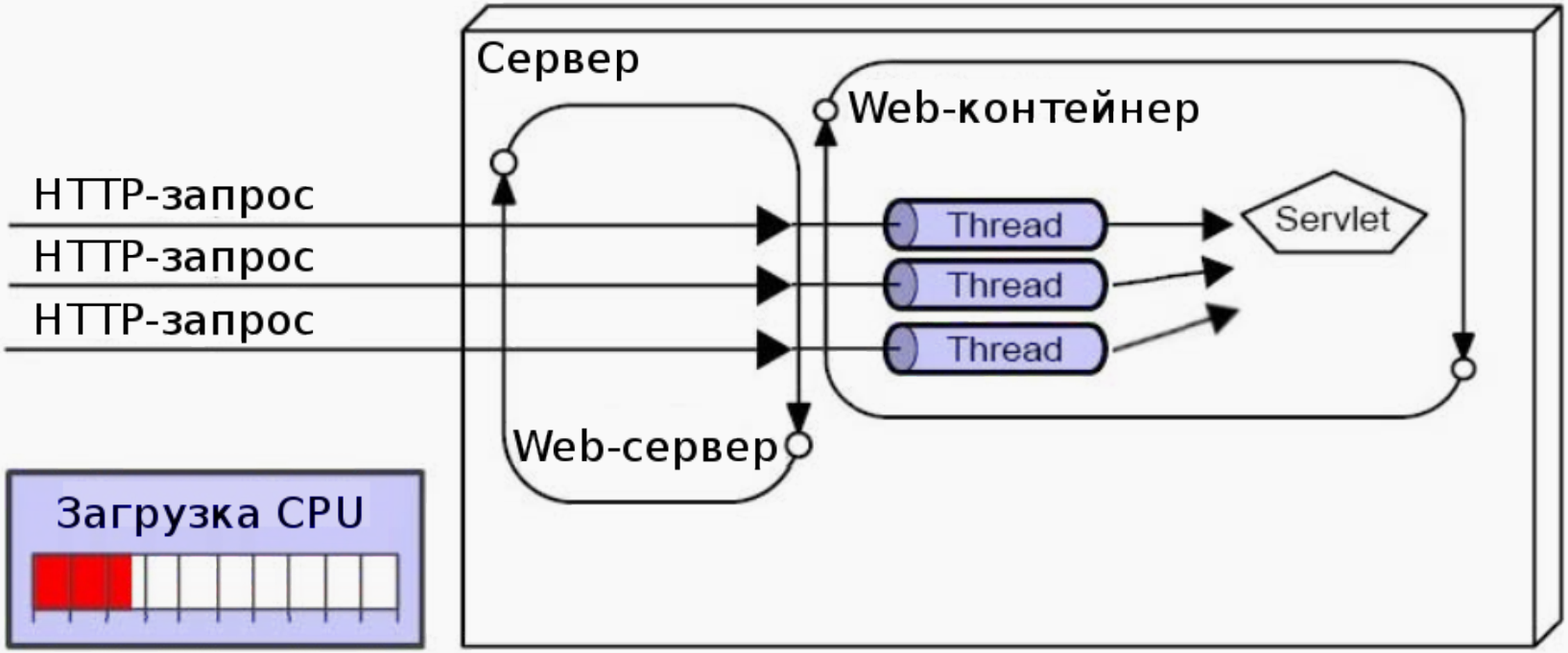
Один запрос:



ИТМО ВТ

Обработка HTTP-запросов сервлетом (продолжение)

Параллельная обработка нескольких запросов:



ИТМО ВТ

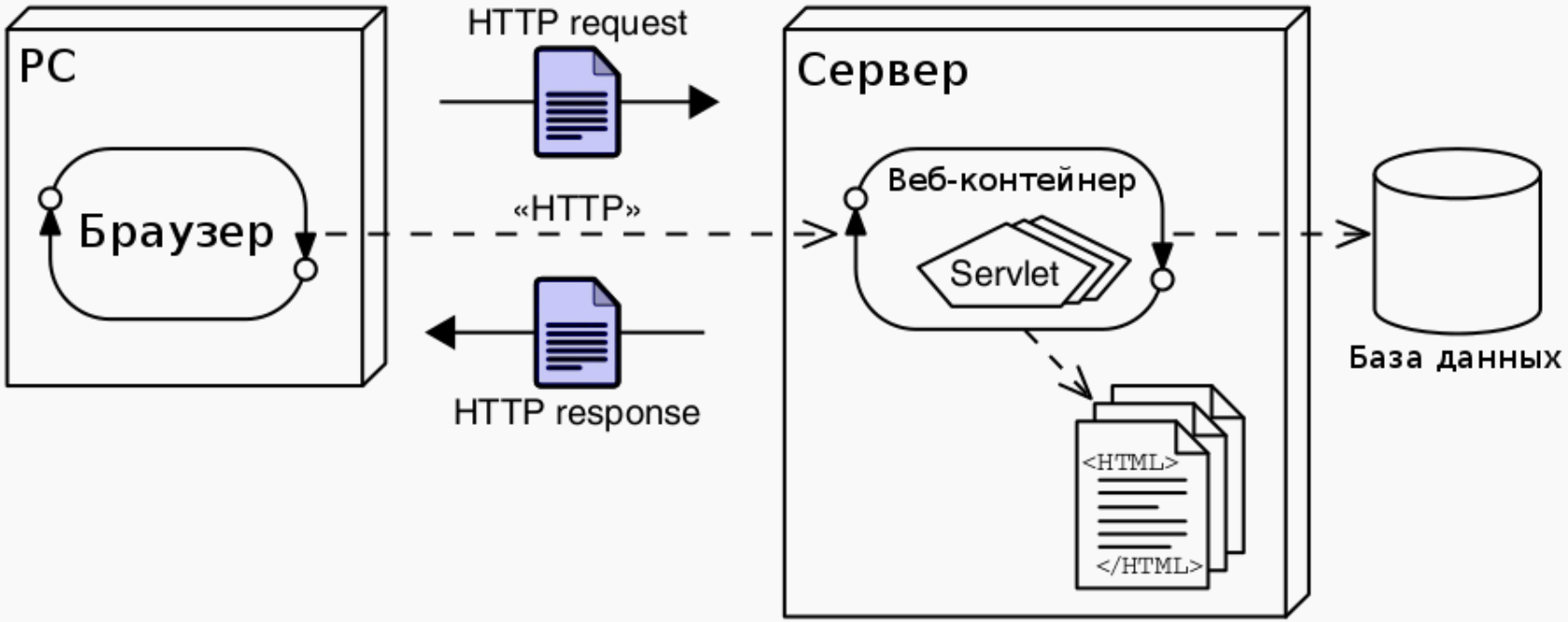
Преимущества и недостатки сервлетов

- Преимущества сервлетов:
  - Выполняются быстрее, чем CGI-сценарии.
  - Хорошая масштабируемость.
  - Надёжность и безопасность (реализованы на Java).
  - Платформенно-независимы.
  - Множество инструментов мониторинга и отладки.
- Недостатки сервлетов:
  - Слабое разделение уровня представления и бизнес-логики.
  - Возможны конфликты при параллельной обработке запросов.

3.1.4. Разработка сервлетов

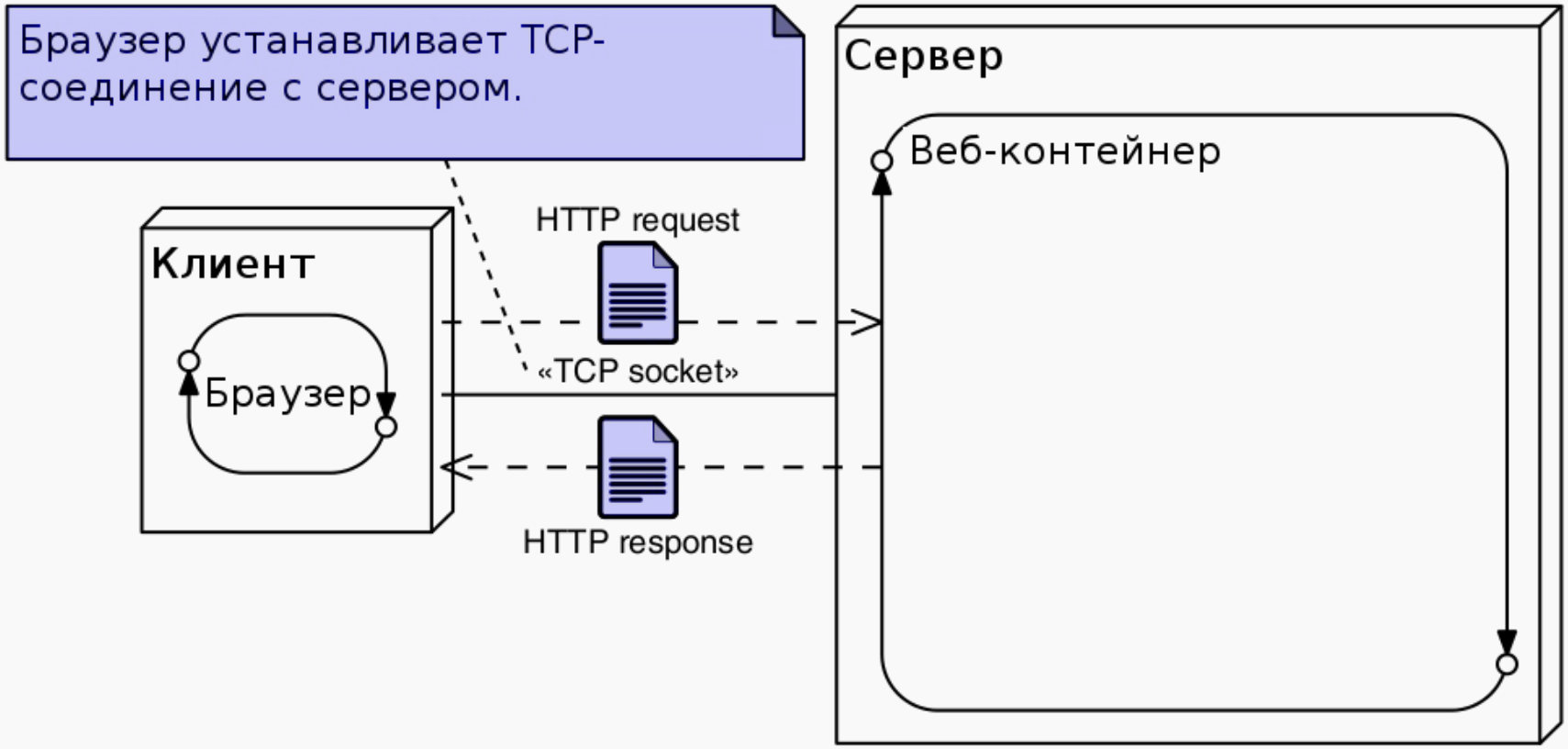
ИТМО ВТ

Архитектура веб-контейнера



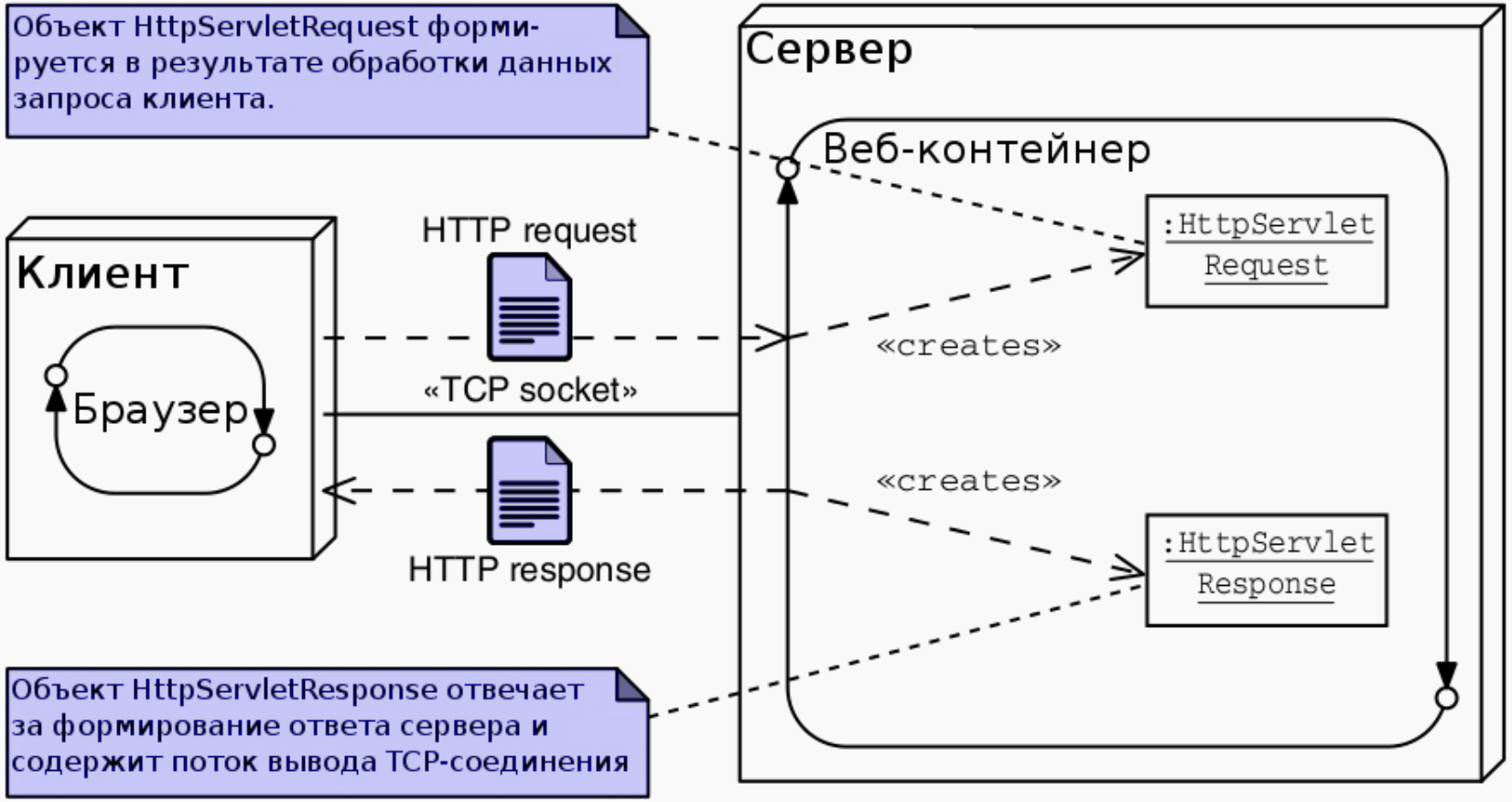
Обработка HTTP-запроса

1. Браузер формирует HTTP-запрос и отправляет его на сервер.



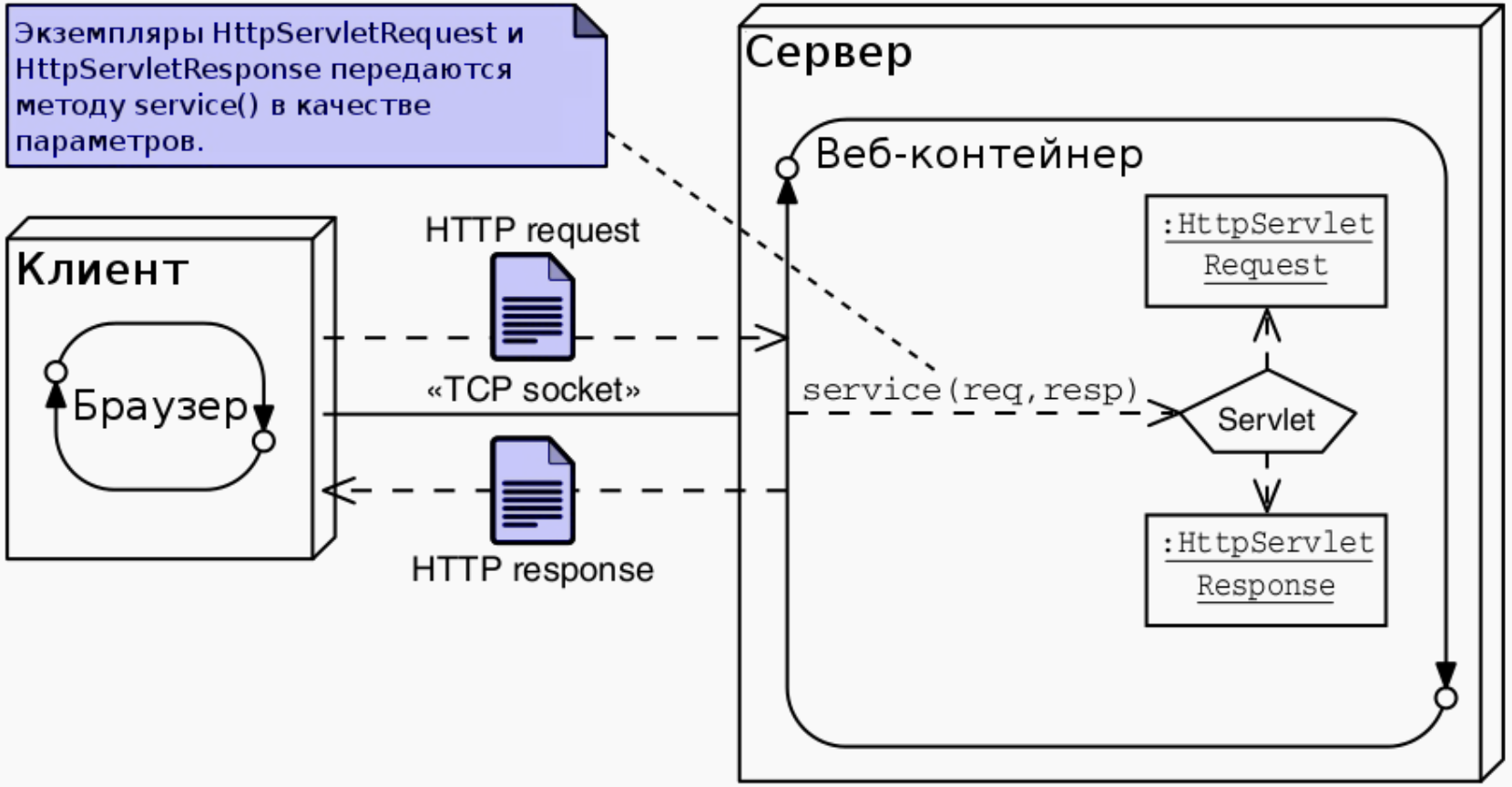
Обработка HTTP-запроса (продолжение)

2. Веб-контейнер создаёт объекты `HttpServletRequest` и `HttpServletResponse`.



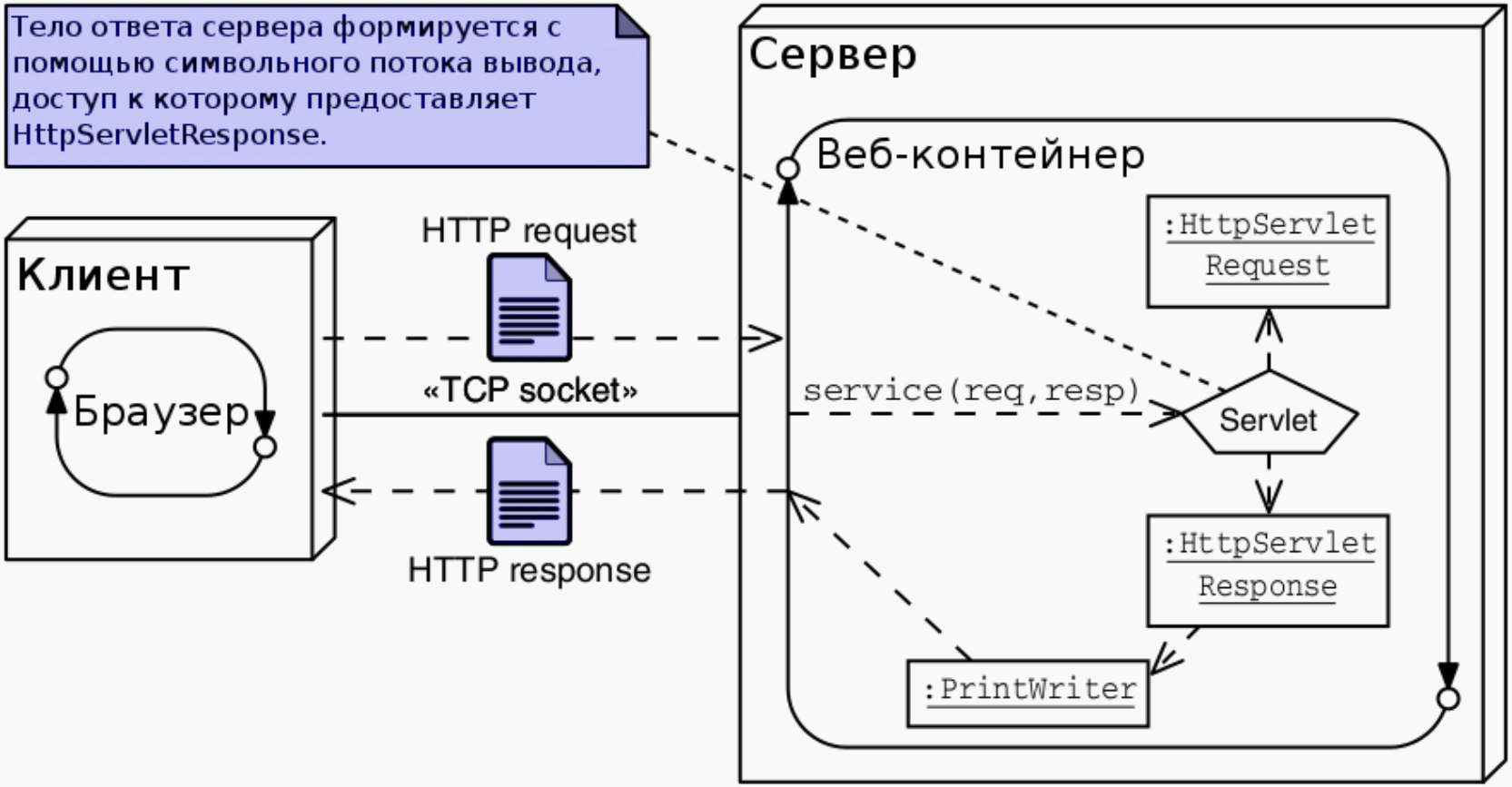
Обработка HTTP-запроса (продолжение)

3. Веб-контейнер вызывает метод `service` сервлета.



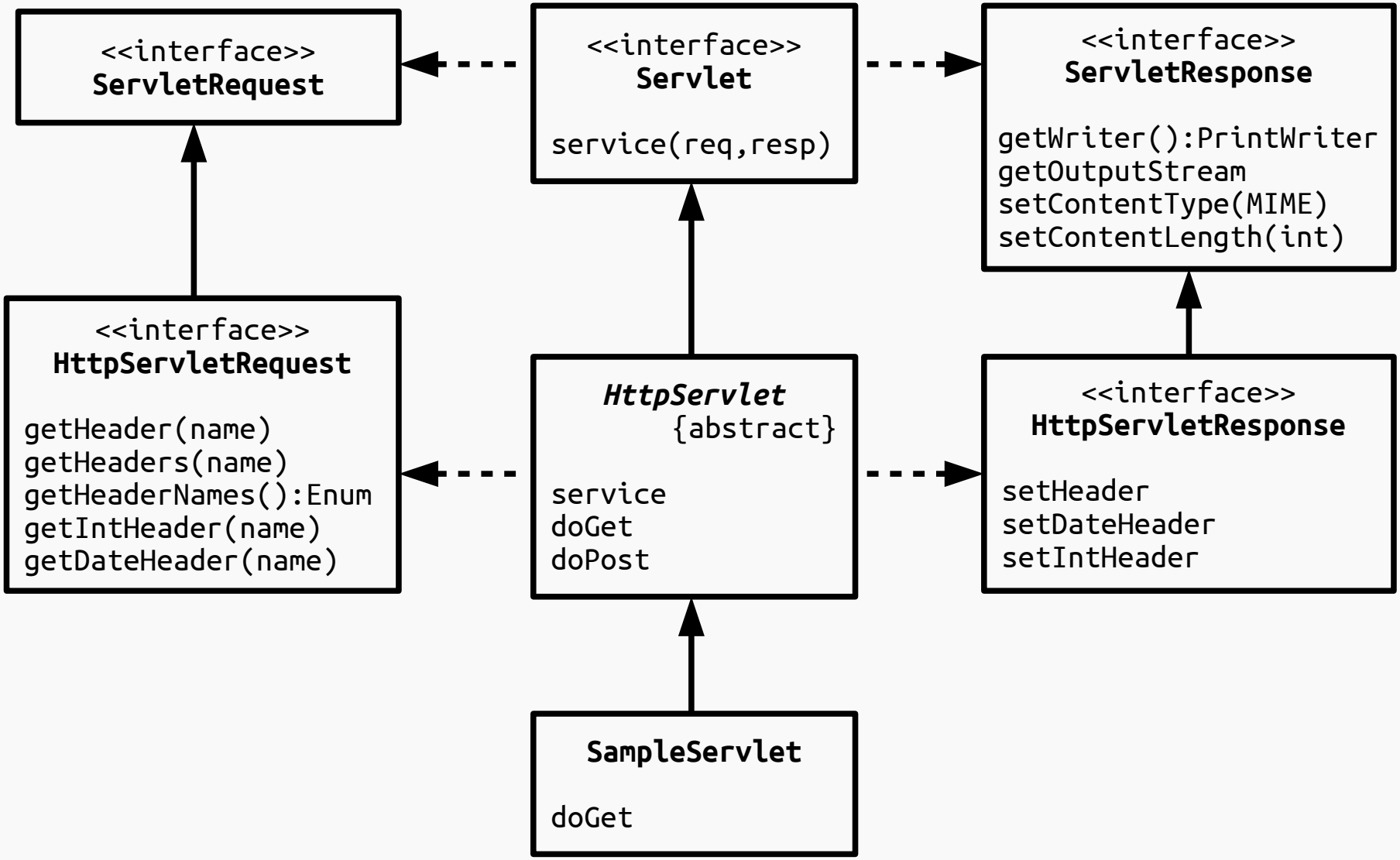
Обработка HTTP-запроса (продолжение)

4. Сервлет формирует ответ и записывает его в поток вывода `HttpServletResponse`.



ИТМО ВТ

HttpServlet API



ИТМО ВТ

Пример сервлета

```
package sample.servlet;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Support classes
import java.io.IOException;
import java.io.PrintWriter;

public class SampleServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException {

        // Заголовок страницы
        String pageTitle = “Пример сервлета”;
```

Пример сервлета (продолжение)

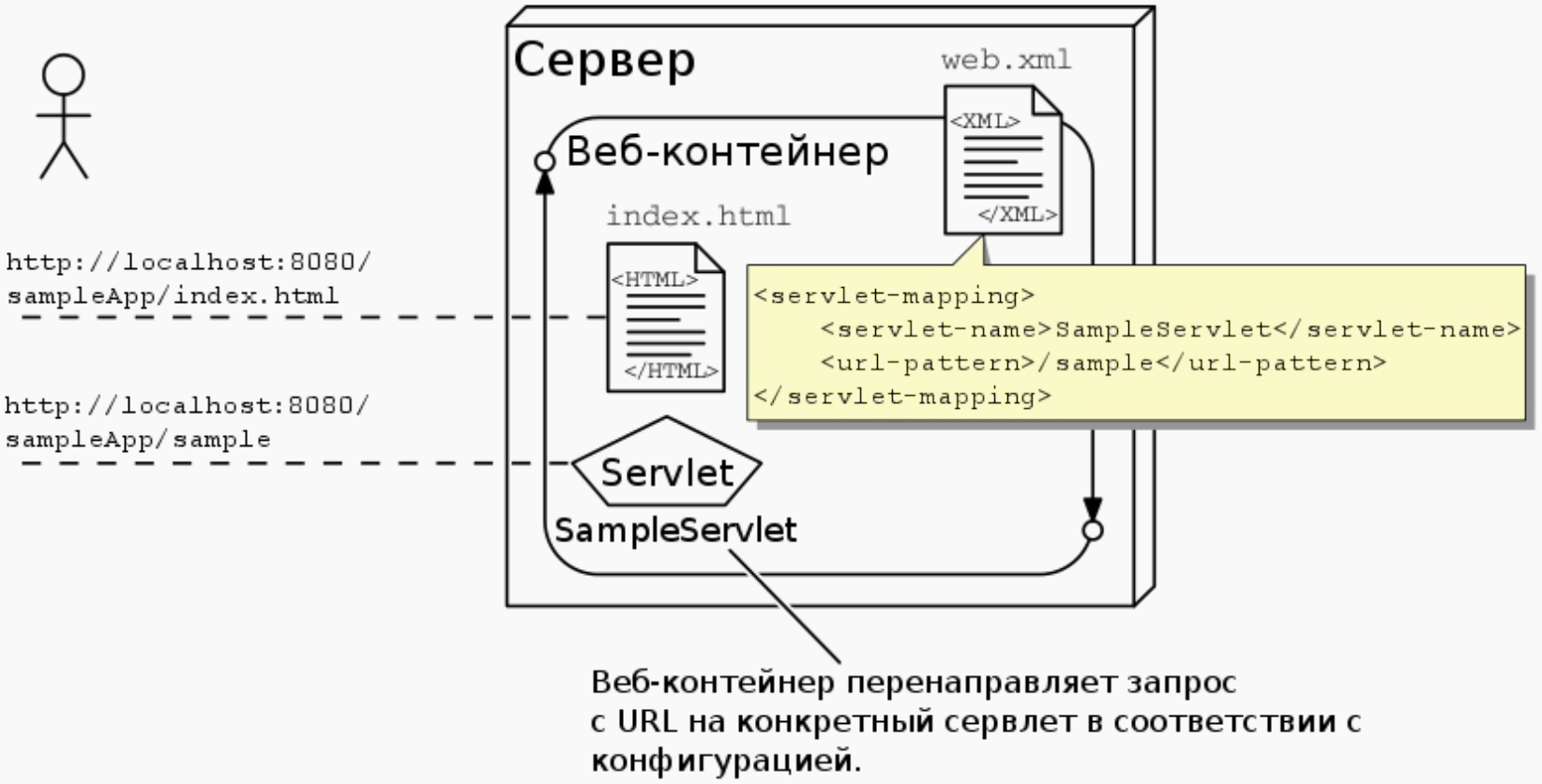
```
        // Content Type
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        // Формируем HTML
        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + pageTitle + "</title>");
        out.println("</head>");
        out.println("<body bgcolor='white'>");
        out.println("<h3>" + pageTitle + "</h3>");
        out.println("<p>");
        out.println("Hello, world!");
        out.println("</p>");
        out.println("</body>");
        out.println("</html>");
    }

}
```

ИТМО ВТ

Конфигурация сервлета

Веб-контейнер создаёт один (строго) экземпляр сервлета на каждую запись в дескрипторе.

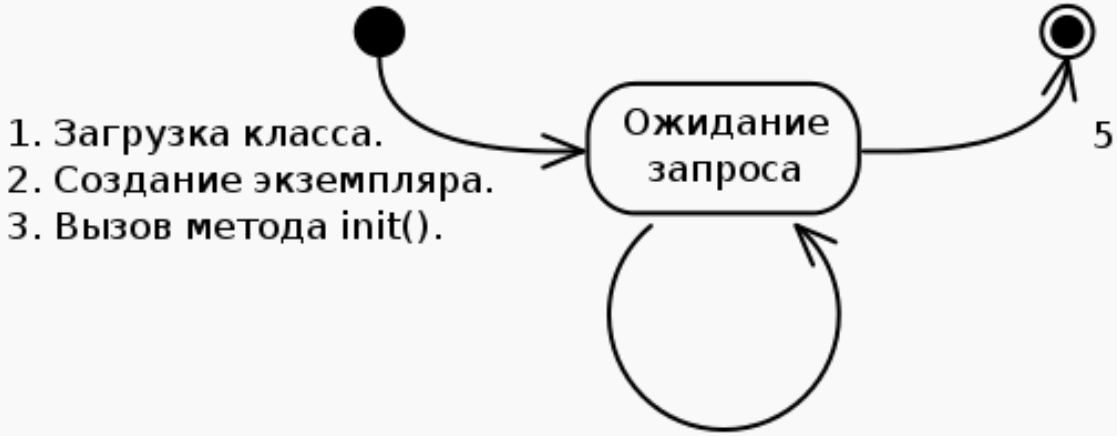
Конфигурация сервлета (продолжение)

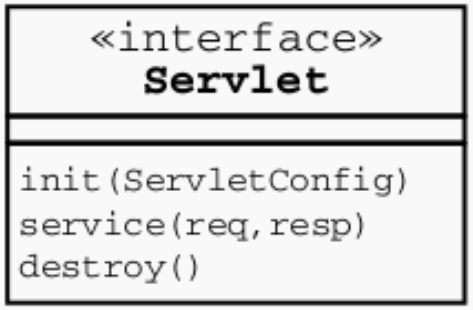


ИТМО ВТ

Жизненный цикл сервлета

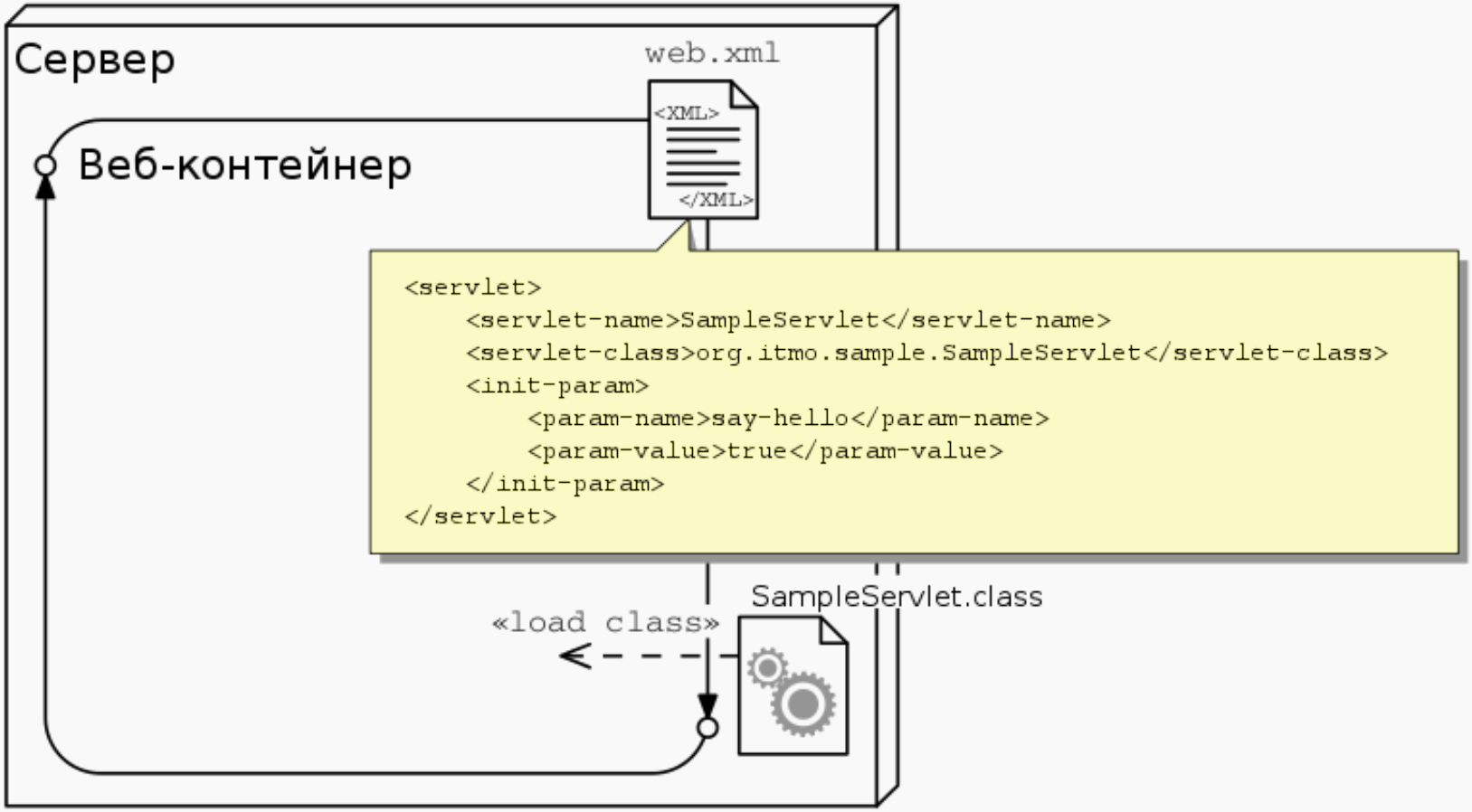
- Жизненным циклом сервлета управляет веб-контейнер.
- Методы, управляющие жизненным циклом, должен вызывать *только* веб-контейнер.





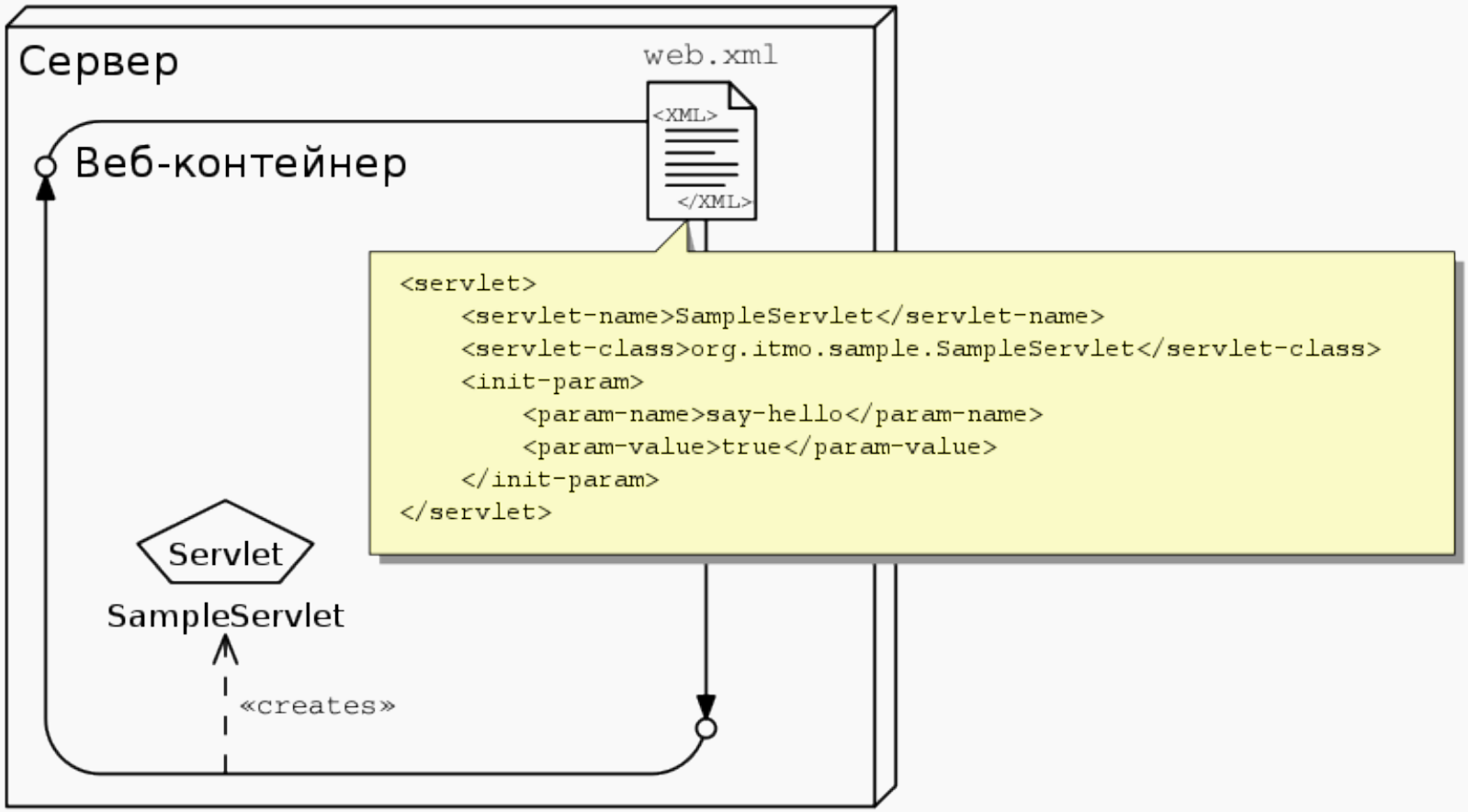
Жизненный цикл сервлета (продолжение)

1. Загрузка класса сервлета.
Проверяются пути: /WEB-INF/classes/, WEB-INF/lib/*.jar, стандартные классы Java SE и классы веб-контейнера.



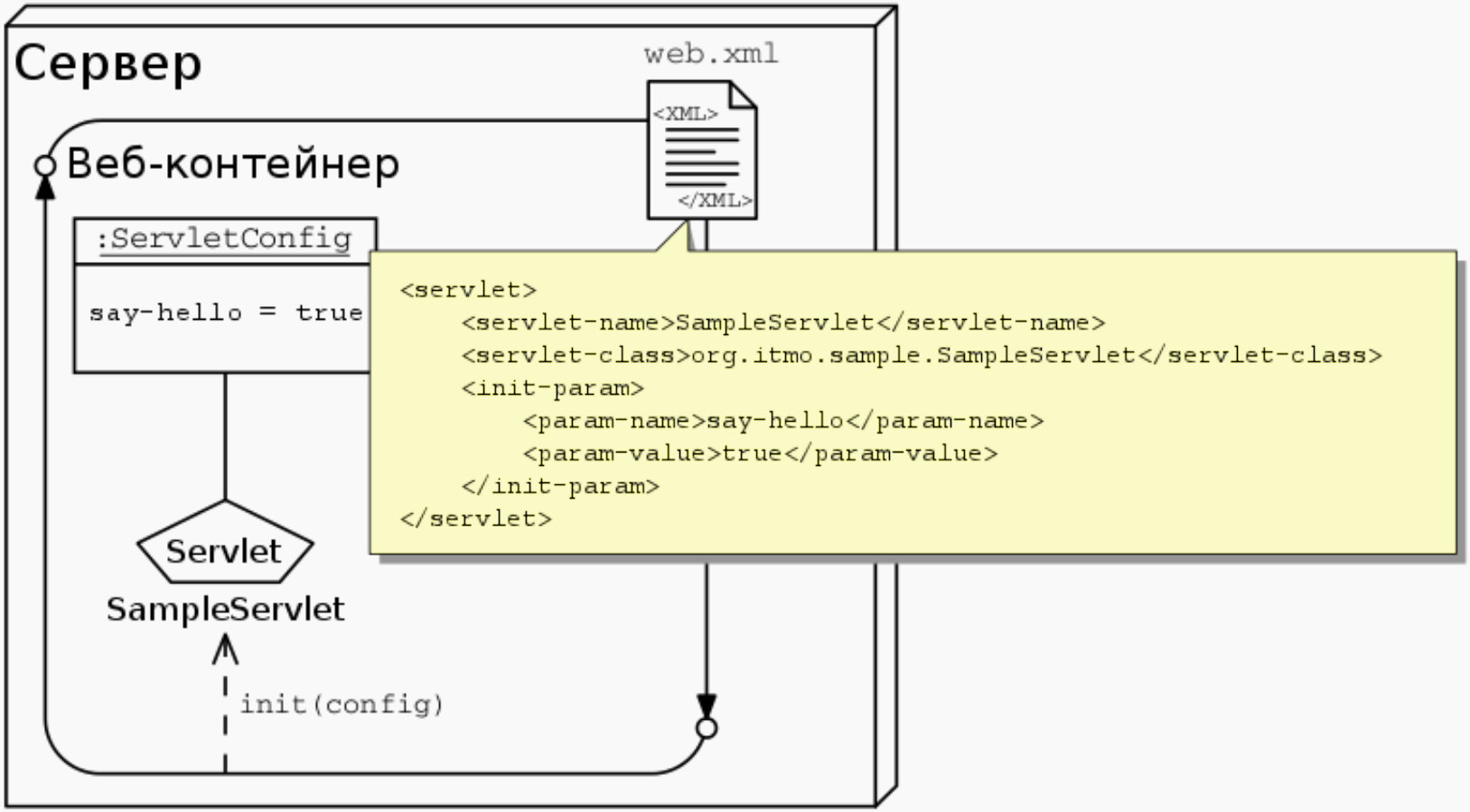
Жизненный цикл сервлета (продолжение)

2. Создание экземпляра сервлета.



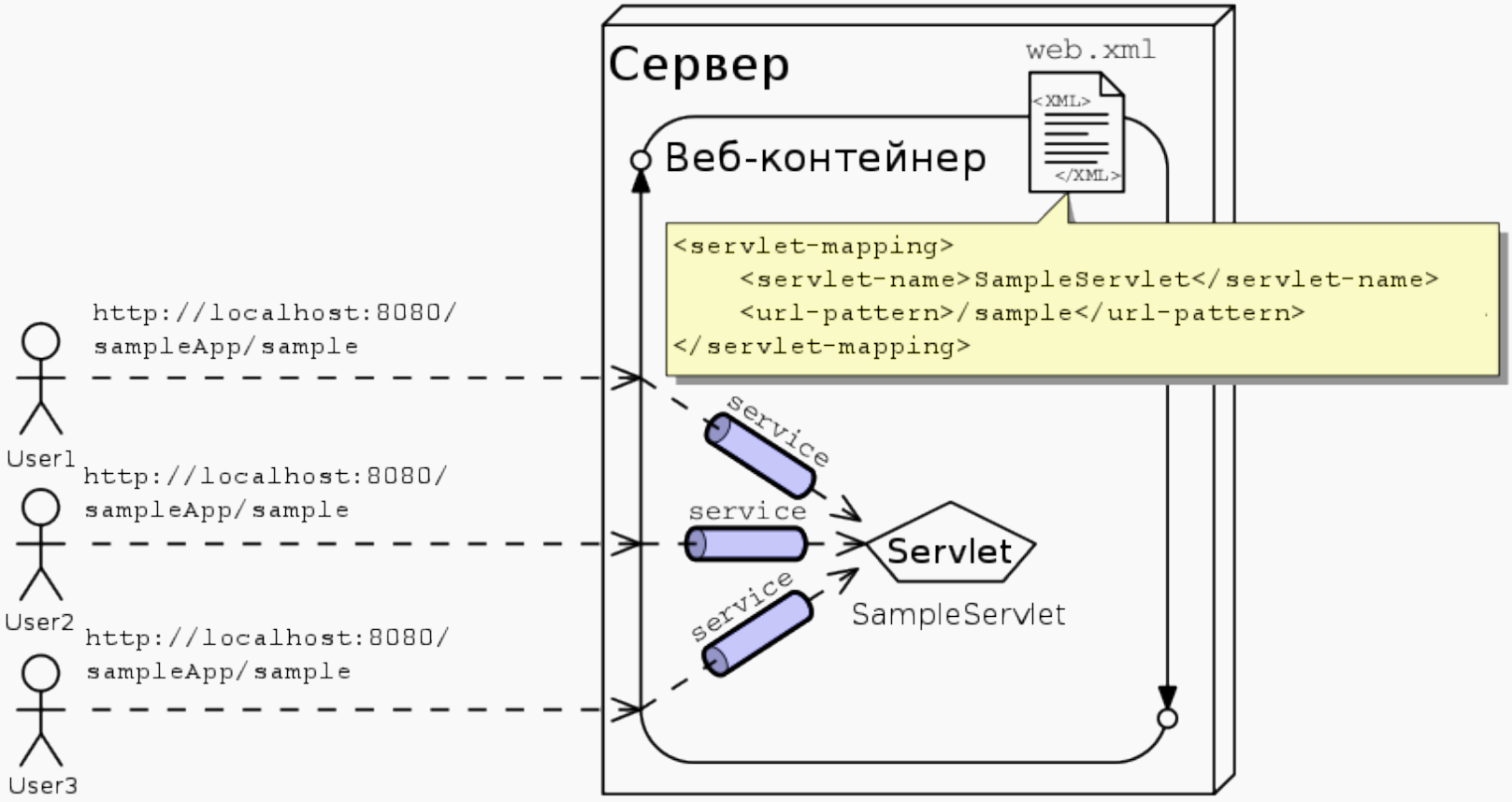
Жизненный цикл сервлета (продолжение)

3. Вызов метода init.



Жизненный цикл сервлета (продолжение)

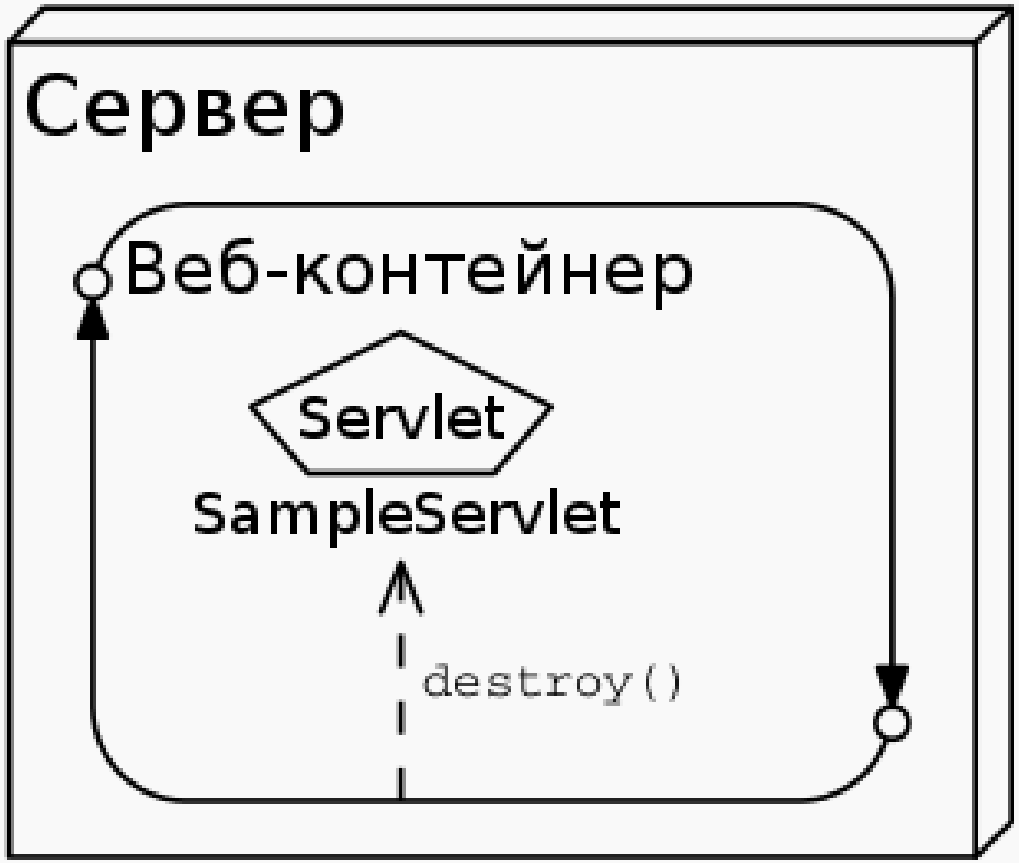
4. Обработка HTTP-запросов.



ИТМО ВТ

Жизненный цикл сервлета (продолжение)

5. Вызов метода destroy.



ИТМО ВТ

Контекст сервлетов

- API, с помощью которого сервлет может взаимодействовать со своим контейнером.
- Доступ к методам осуществляется через интерфейс `javax.servlet.ServletContext`.
- У всех сервлетов внутри приложения общий контекст.
- В контекст можно помещать общую для всех сервлетов информацию (методы `getAttribute` и `setAttribute`).
- Если приложение — распределённое, то на каждом экземпляре JVM контейнером создаётся свой контекст.

Контекст сервлетов (продолжение)

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

public class DemoServlet extends HttpServlet{
    public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException {

        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

        //creating ServletContext object
        ServletContext context=getServletContext();

        //Getting the value of the initialization parameter
        // and printing it
        String driverName=context.getInitParameter("dname");
        pw.println("driver name is="+driverName);

        pw.close();
    }
}
```

ИТМО ВТ

HTTP-сессии

- HTTP — stateless-протокол.
- `javax.servlet.HttpSession` — интерфейс, позволяющий идентифицировать конкретного клиента (браузер) при обработке множества HTTP-запросов от него.
- Экземпляр `HttpSession` создаётся при первом обращении клиента к приложению и сохраняется некоторое (настраиваемое) время после последнего обращения.
- Идентификатор сессии либо помещается в cookie, либо добавляется к URL. Если удалить этот идентификатор, то сервер не сможет идентифицировать клиента и создаст новую сессию.
- В экземпляр `HttpSession` можно помещать общую для этой сессии информацию (методы `getAttribute` и `setAttribute`).
- Сессия «привязана» к конкретному приложению; у разных приложений — разные сессии.
- В распределённом окружении обеспечивается сохранение целостности данных в HTTP-сессии (независимо от количества экземпляров JVM).

HTTP-сессии (продолжение)

```
public class SimpleSession extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, java.io.IOException {

    response.setContentType("text/html");
    java.io.PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();

    out.println("<html>");
    out.println("<head>");
    out.println("<title>Simple Session Tracker</title>");
    out.println("</head>");
    out.println("<body>");

    out.println("<h2>Session Info</h2>");
    out.println("session Id: " + session.getId() + "<br><br>");
    out.println("The SESSION TIMEOUT period is "
        + session.getMaxInactiveInterval() + " seconds.<br><br>");
    out.println("Now changing it to 20 minutes.<br><br>");
    session.setMaxInactiveInterval(20 * 60);
    out.println("The SESSION TIMEOUT period is now "
        + session.getMaxInactiveInterval() + " seconds.");

    out.println("</body>");
    out.println("</html>");
  }
}
```

ИТМО ВТ

Диспетчеризация запросов сервлетами

- Сервлеты могут делегировать обработку запросов другим ресурсам (сервлетам, JSP и HTML-страницам).
- Диспетчеризация осуществляется с помощью реализаций интерфейса `javax.servlet.RequestDispatcher`.
- Два способа получения `RequestDispatcher` — через `ServletRequest` (абсолютный или относительный URL) и `ServletContext` (только абсолютный URL).
- Два способа делегирования обработки запроса — `forward` и `include`.

Диспетчеризация запросов сервлетами (продолжение)

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

public class MyServlet extends HttpServlet{
   public void doGet(HttpServletRequest req,
                     HttpServletResponse res)
      throws ServletException,IOException {

      RequestDispatcher dispatcher =
         request.getRequestDispatcher("index.jsp");
      dispatcher.forward( request, response );
   }
}
```

ИТМО ВТ

Фильтры запросов

- Фильтры позволяют осуществлять пред- и постобработку запросов до и после передачи их ресурсу (сервлету, JSP или HTML-странице).
- Пример предобработки — допуск к странице только авторизованных пользователей.
- Пример постобработки — запись в лог времени обработки запроса.
- Реализуют интерфейс `javax.servlet.Filter`.
- Ключевой метод — `doFilter`.
- Метод `doFilter` класса `FilterChain` передаёт управление следующему фильтру или целевому ресурсу; таким образом, возможна реализация последовательностей фильтров, обрабатывающих один и тот же запрос.

Фильтры запросов (продолжение)

```
import javax.servlet.*;

public class MyFilter implements Filter{

    public void init(FilterConfig arg0) throws ServletException {}

    public void doFilter(ServletRequest req, ServletResponse resp,
        FilterChain chain) throws IOException, ServletException {

        PrintWriter out=resp.getWriter();
        out.print("filter is invoked before");

        chain.doFilter(req, resp);//sends request to next resource

        out.print("filter is invoked after");
    }

    public void destroy() {}
}
```

ИТМО ВТ

Конфигурация фильтров

```
<web-app>

    <servlet>
        <servlet-name>s1</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>s1</servlet-name>
        <url-pattern>/servlet1</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>f1</filter-name>
        <filter-class>MyFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>f1</filter-name>
        <url-pattern>/servlet1</url-pattern>
    </filter-mapping>

</web-app>
```

3.2. Шаблоны проектирования в веб-приложениях

ИТМО ВТ

Основные понятия

- **Шаблон проектирования** или **паттерн —** повторимая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста (© Wikipedia).
- Описывает подход к решению типовой задачи.
- Одну и ту же задачу часто можно решить с использованием разных шаблонов.
- Существует много литературы с описанием различных шаблонов проектирования.

ИТМО ВТ

Зачем нужны паттерны

- Позволяют избежать «типовых» ошибок при разработке типовых решений.
- Позволяют кратко описать подход к решению задачи — программистам, знающим шаблоны, проще обмениваться информацией.
- Легче поддерживать код — его поведение более предсказуемо.

ИТМО ВТ

GoF-паттерны

- Описаны в книге 1994 г. *«Design Patterns: Elements of Reusable Object-Oriented Software»* («Приёмы объектно-ориентированного проектирования. Паттерны проектирования»).
- Авторы — *Эрих Гамма* (Erich Gamma), *Ричард Хелм* (Richard Helm), *Ральф Джонсон* (Ralph Johnson), *Джон Влиссидс* (John Vlissides) — Gang of Four (GoF, «Банда Четырёх»).
- В книге описаны 23 классических шаблона проектирования.

ИТМО ВТ

Порождающие GoF-паттерны

- Abstract Factory — Абстрактная фабрика.
- Builder — Строитель.
- Factory Method — Фабричный метод.
- Prototype — Прототип.
- Singleton — Одиночка.

ИТМО ВТ

Пример порождающего GoF-паттерна

Singleton (одиночка):

- Гарантирует, что у класса есть *только один экземпляр*, и предоставляет к нему глобальную точку доступа.
- Можно пользоваться *экземпляром* класса (в отличие от статических методов).

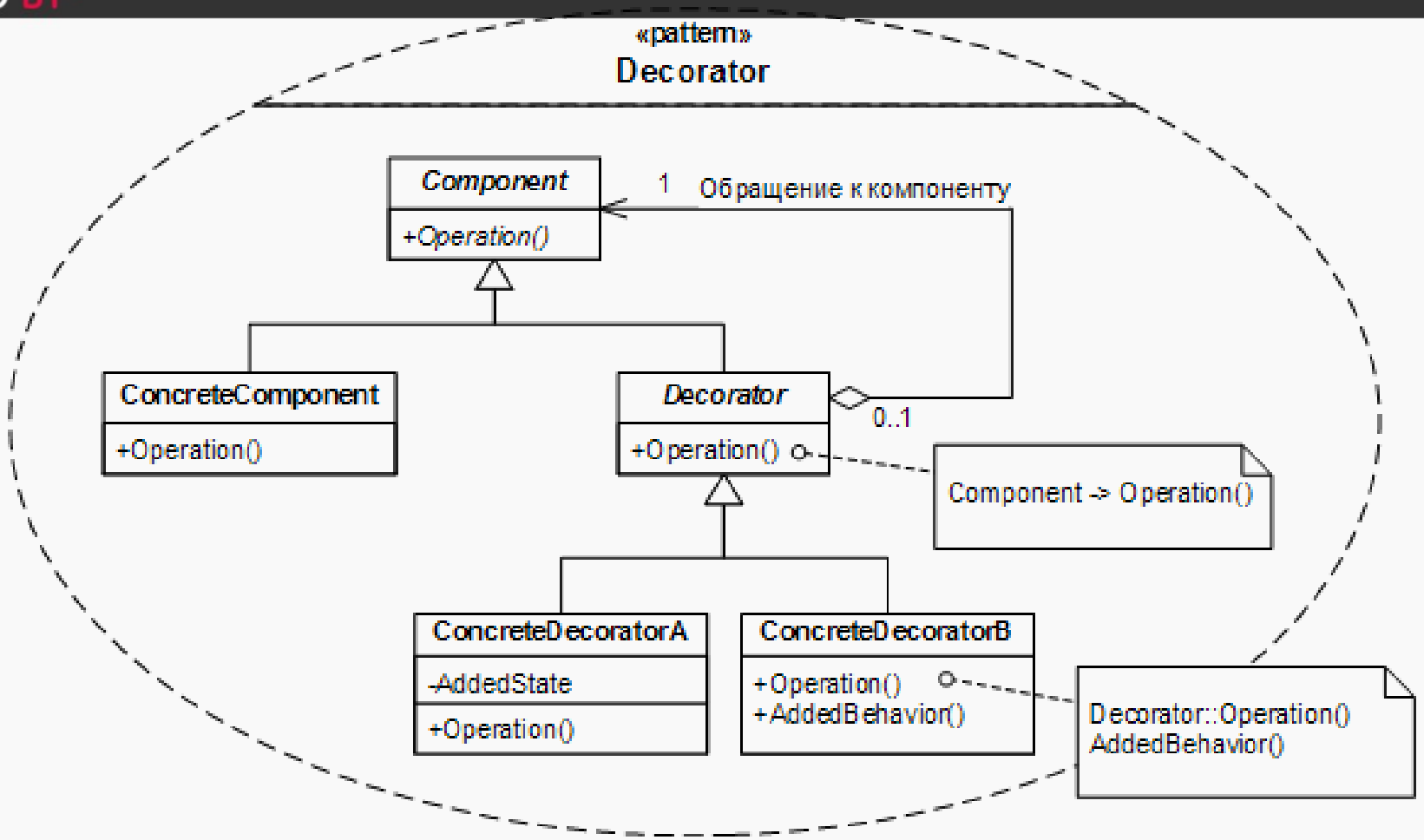
ИТМО ВТ

Структурные GoF-паттерны

- Adapter — Адаптер.
- Bridge — Мост.
- Composite — Компоновщик.
- Decorator — Декоратор.
- Facade — Фасад.
- Flyweight — Приспособленец.
- Proxy — Заместитель.

ИТМО ВТ

Пример структурного GoF-паттерна



Decorator (декоратор) — позволяет динамически подключать дополнительное поведение к объекту без использования наследования.

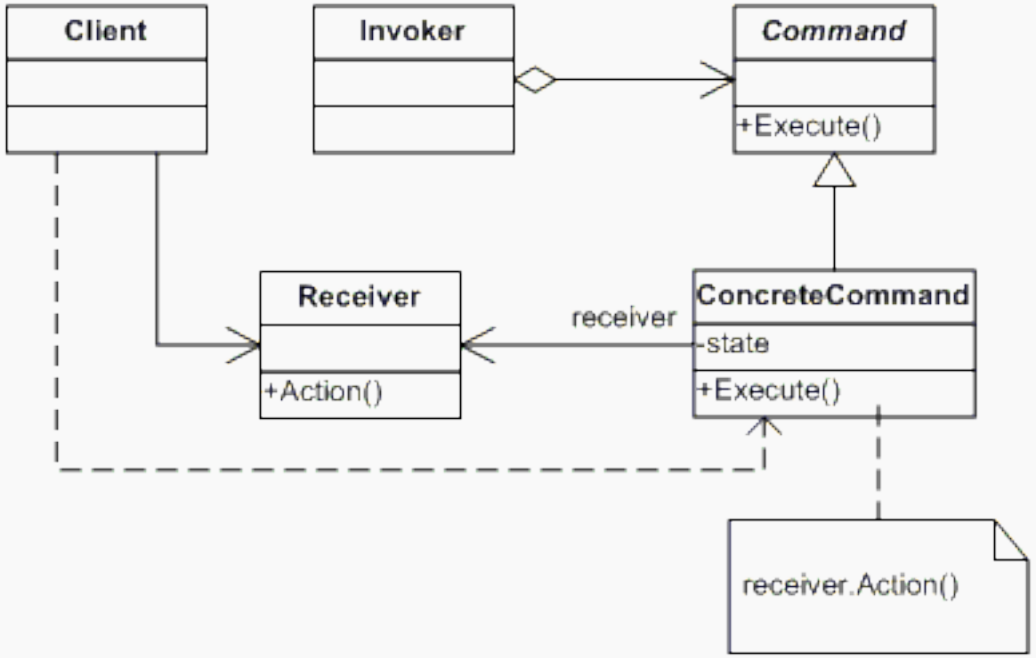
ИТМО ВТ

Поведенческие GoF-паттерны

- Chain of responsibility — Цепочка обязанностей.
- Command — Команда.
- Interpreter — Интерпретатор.
- Iterator — Итератор.
- Mediator — Посредник.
- Memento — Хранитель.
- Observer — Наблюдатель.
- State — Состояние.
- Strategy — Стратегия.
- Template — Шаблонный метод.
- Visitor — Посетитель.

ИТМО ВТ

Пример поведенческого GoF-паттерна



Command (команда) — команда передаётся с помощью специального объекта, который заключает в себе само действие (т. е. логику) и его параметры.

ИТМО ВТ

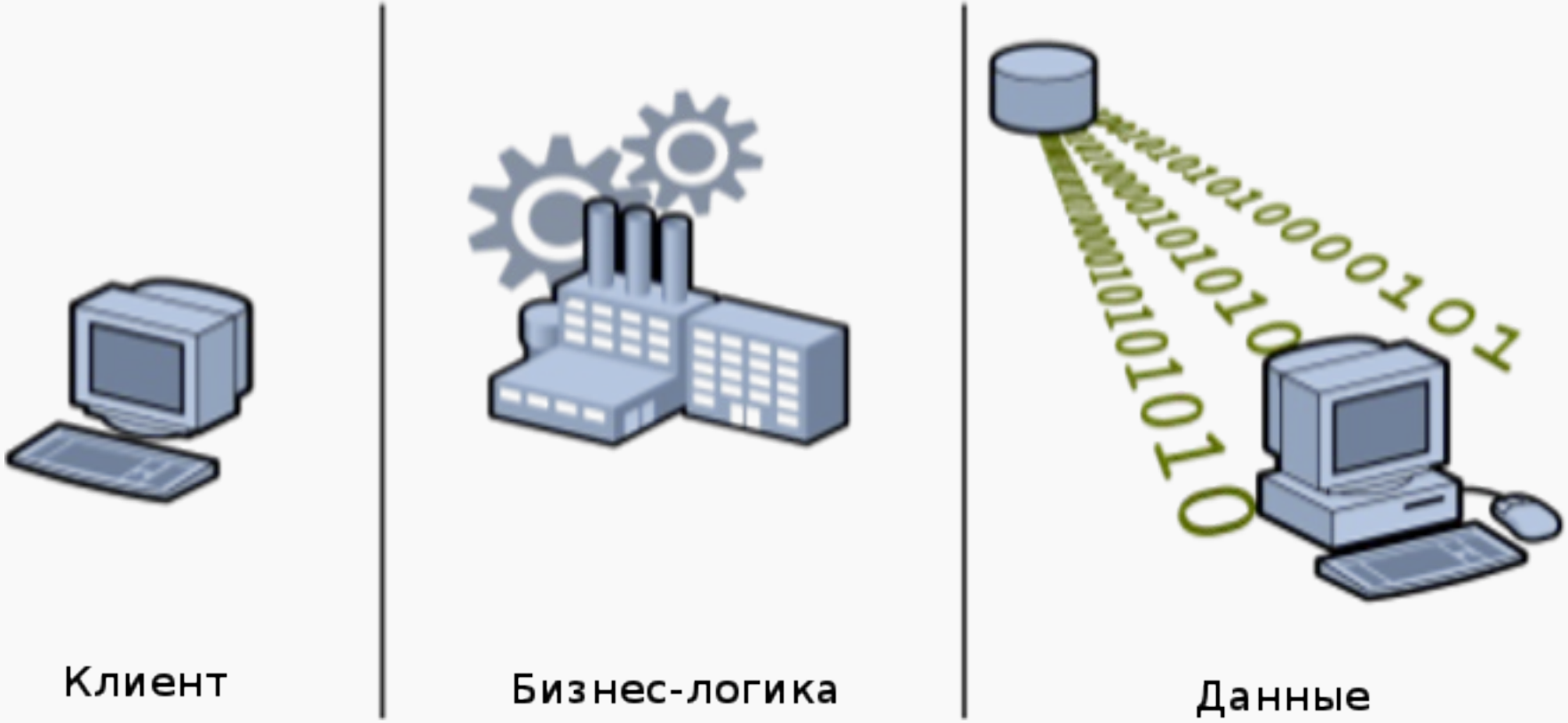
Архитектурные шаблоны

- Более высокий уровень по сравнению с шаблонами проектирования.
- Описывают архитектуру всей системы или приложения.
- Обычно имеют дело не с отдельными классами, а с целыми компонентами или модулями.
- Компоненты и модули могут быть построены с использованием различных шаблонов проектирования.

ИТМО ВТ

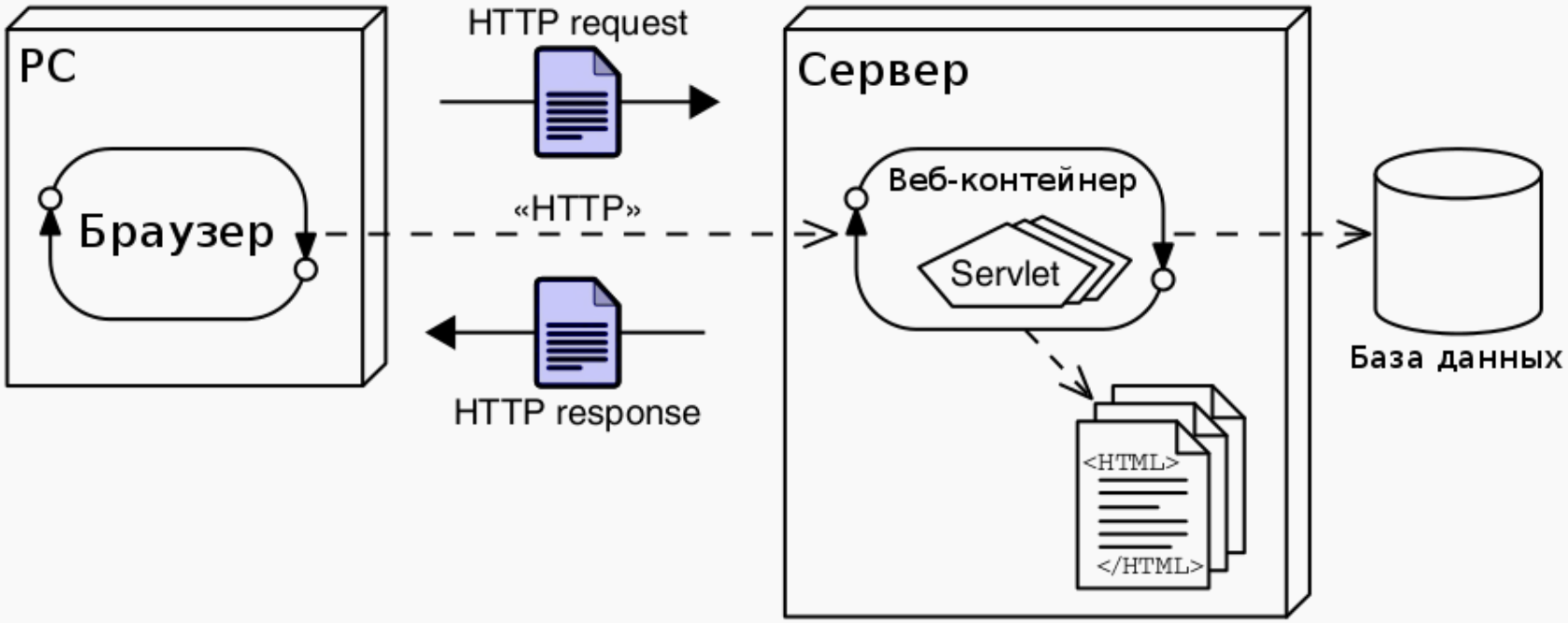
Архитектура веб-приложений

3 уровня архитектуры:



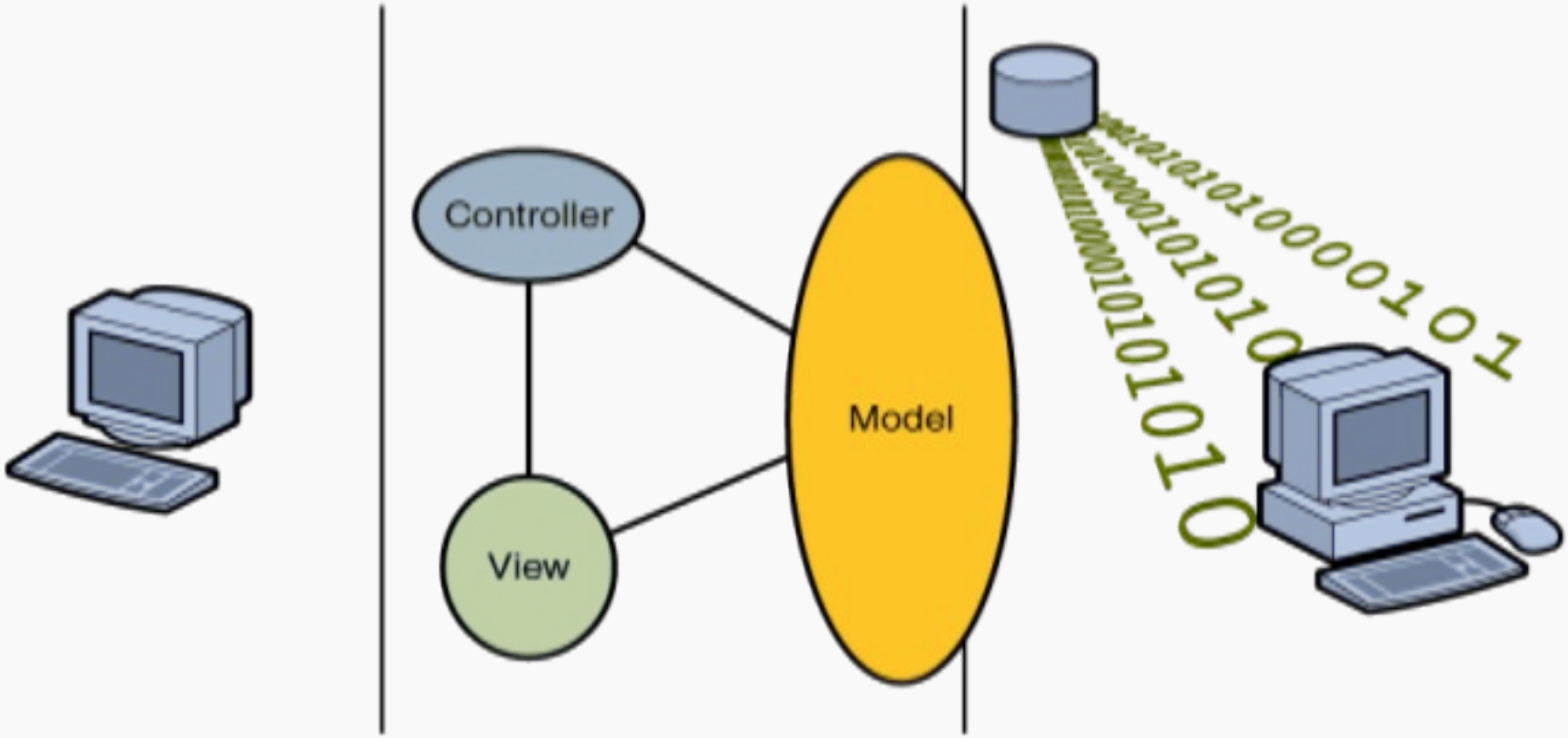
Архитектура Model 1 (Java)

- Предназначена для проектирования приложений небольшого масштаба и сложности.
- За обработку данных и представления отвечает *один и тот же* компонент (сервлет или JSP).



ИТМО ВТ

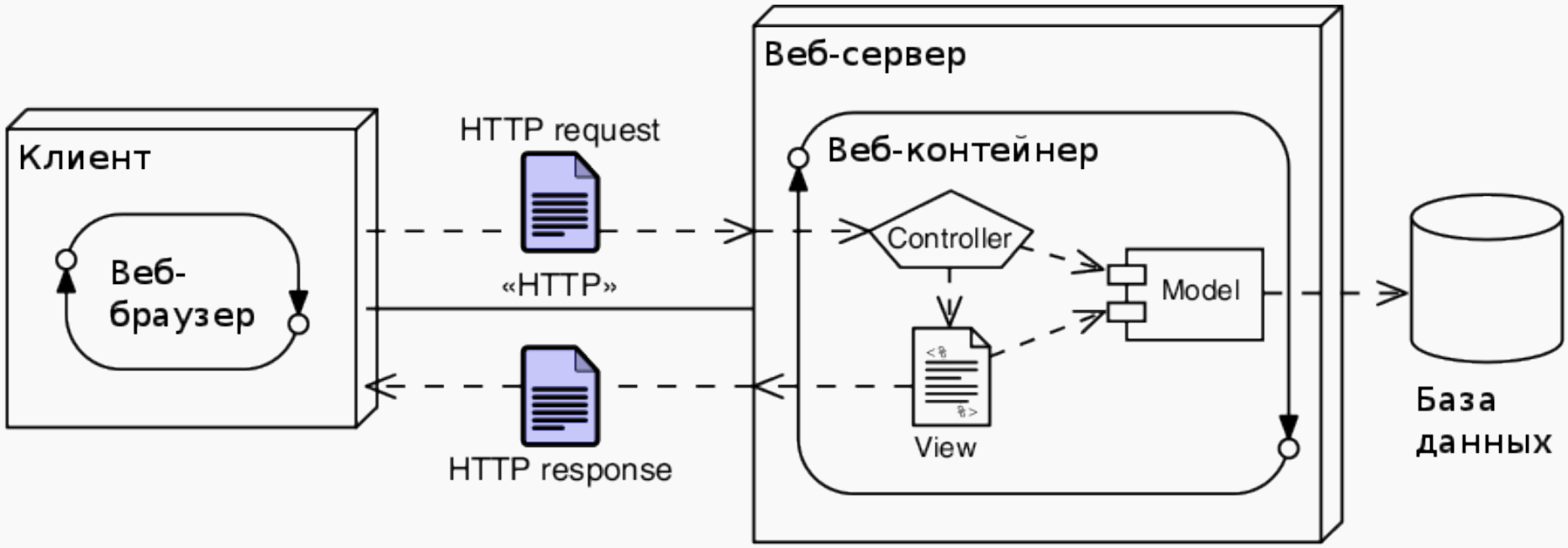
Шаблон MVC



ИТМО ВТ

Архитектура Model 2 (Java)

- Предназначена для проектирования достаточно сложных веб-приложений.
- За обработку и представление данных отвечают *разные* компоненты (сервлеты и JSP).



ИТМО ВТ

Как реализовать Model 2

- Вручную (сервлеты + какой-нибудь шаблонизатор + какая-нибудь бизнес-логика).
- Использовать фреймворк:
  - Apache Struts.
  - Apache Velocity.
  - JavaServer Faces (в составе Java EE).
  - Spring Web MVC.

3.3. Шаблонизация страниц

ИТМО ВТ

Зачем оно нужно

- Делать разметку страницы с помощью серверного сценария неудобно.
- Интерфейс приложения обычно состоит из типовых повторяющихся элементов.
- В больших проектах разработкой логики и интерфейсов обычно занимаются разные люди.

ИТМО ВТ

Инструменты шаблонизации страниц

- JavaServer Pages.
- FreeMarker Template Engine (FTL).
- Thymeleaf.
- Velocity.

…тысячи их!

3.3.1. JavaServer Pages

ИТМО ВТ

JavaServer Pages

- Страницы JSP — это текстовые файлы, содержащие статический HTML и JSP-элементы.
- JSP-элементы позволяют формировать динамическое содержимое.
- При загрузке в веб-контейнер страницы JSP транслируются компилятором (jasper) в сервлеты.
- Позволяют отделить бизнес-логику от уровня представления (если их комбинировать с сервлетами).

ИТМО ВТ

Преимущества и недостатки JSP

- Преимущества:
  - Высокая производительность — транслируются в сервлеты.
  - Не зависят от используемой платформы — код пишется на Java.
  - Позволяют использовать Java API.
  - Простые для понимания — структура похожа на обычный HTML.
- Недостатки:
  - Трудно отлаживать, если приложение целиком основано на JSP.
  - Возможны конфликты при параллельной обработке нескольких запросов.

Сервлеты и JSP

```
public class HelloServlet extends HttpServlet {

    private static final String DEFAULT_NAME = “World”;

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException {
        generateResponse(request, response);
    }

    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException {
        generateResponse(request, response);
    }

    public void generateResponse(HttpServletRequest
request,
        HttpServletResponse response) throws IOException {
        String name = reqest getParameter(“name”);
```

Сервлеты и JSP (продолжение)

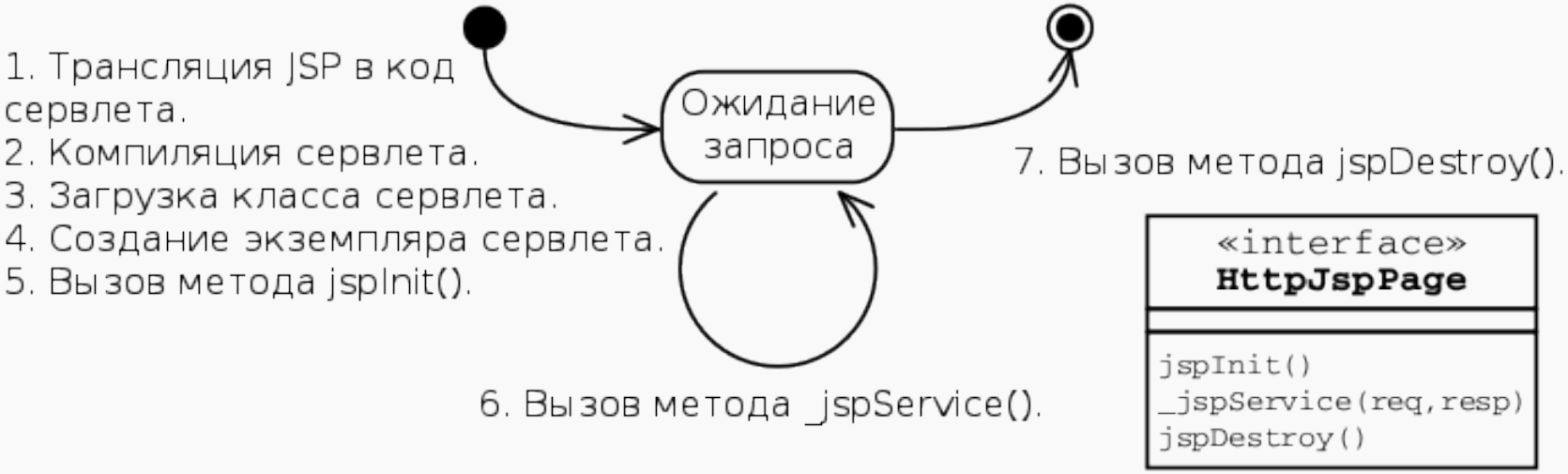
```
        if ( (name == null) || (name.length() == 0) ) {
            name = DEFAULT_NAME;
        }
        response.setContentType(“text/html”);
        PrintWriter out = response.getWriter();
        out.println(“<HTML>”);
        out.println(“<HEAD>”);
        out.println(“<TITLE>Hello Servlet</TITLE>”);
        out.println(“</HEAD>”);
        out.println(“<BODY BGCOLOR=’white’>”);
        out.println(“<B>Hello, “ + name + “</B>”);
        out.println(“</BODY>”);
        out.println(“</HTML>”);
        out.close();
    }
}
```

Сервлеты и JSP (продолжение)

```
<%! private static final String DEFAULT_NAME = “World”;
%>
<html>
<head>
<title>Hello JavaServer Page</title>
</head>
<%-- Determine the specified name (or use default) --%>
<%
    String name = request.getParameter(“name”);
    if ( (name == null) || (name.length() == 0) ) {
        name = DEFAULT_NAME;
    }
%>
<body bgcolor=’white’>
<b>Hello, <%= name %></b>
</body>
</html>
```

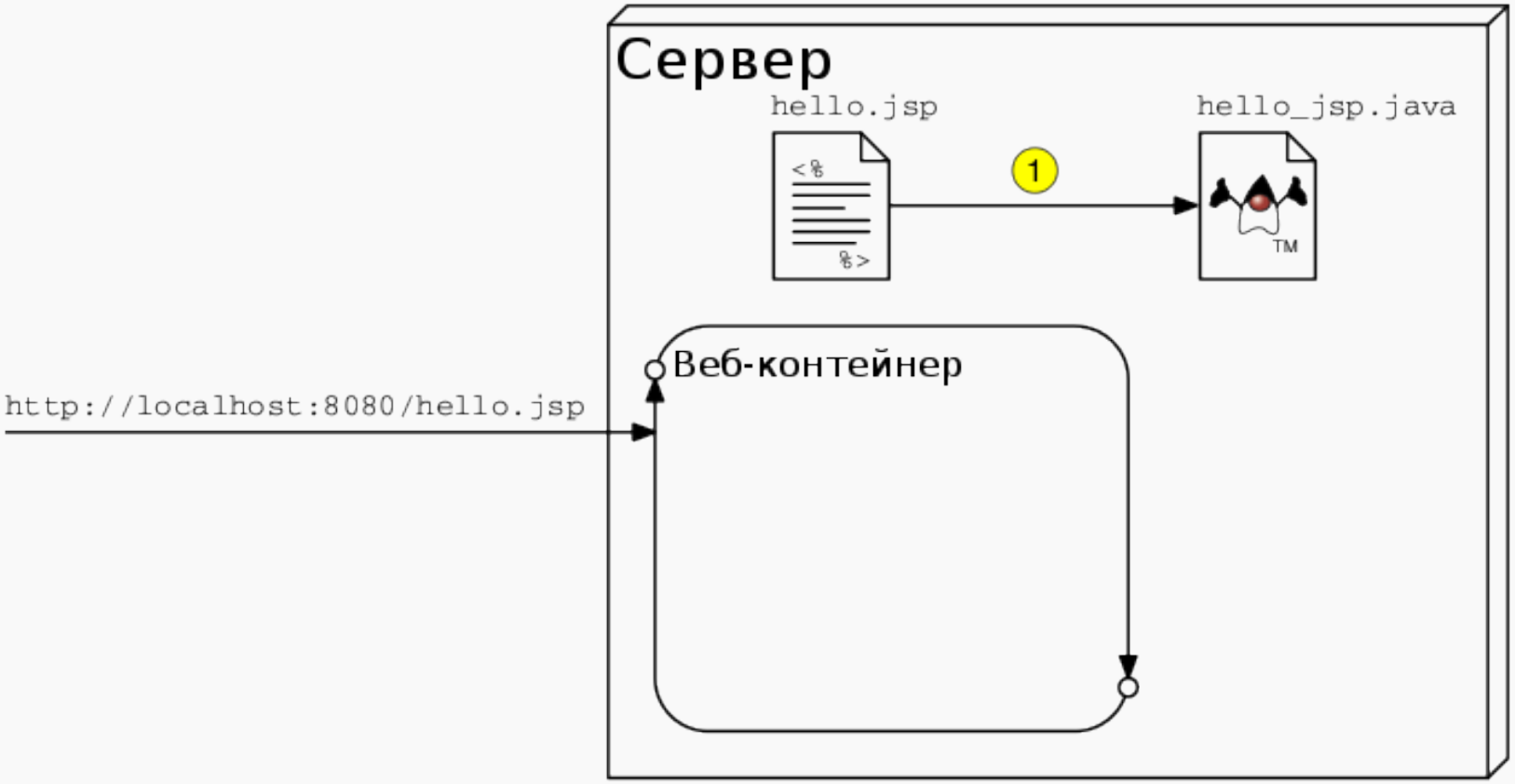
ИТМО ВТ

Жизненный цикл JSP



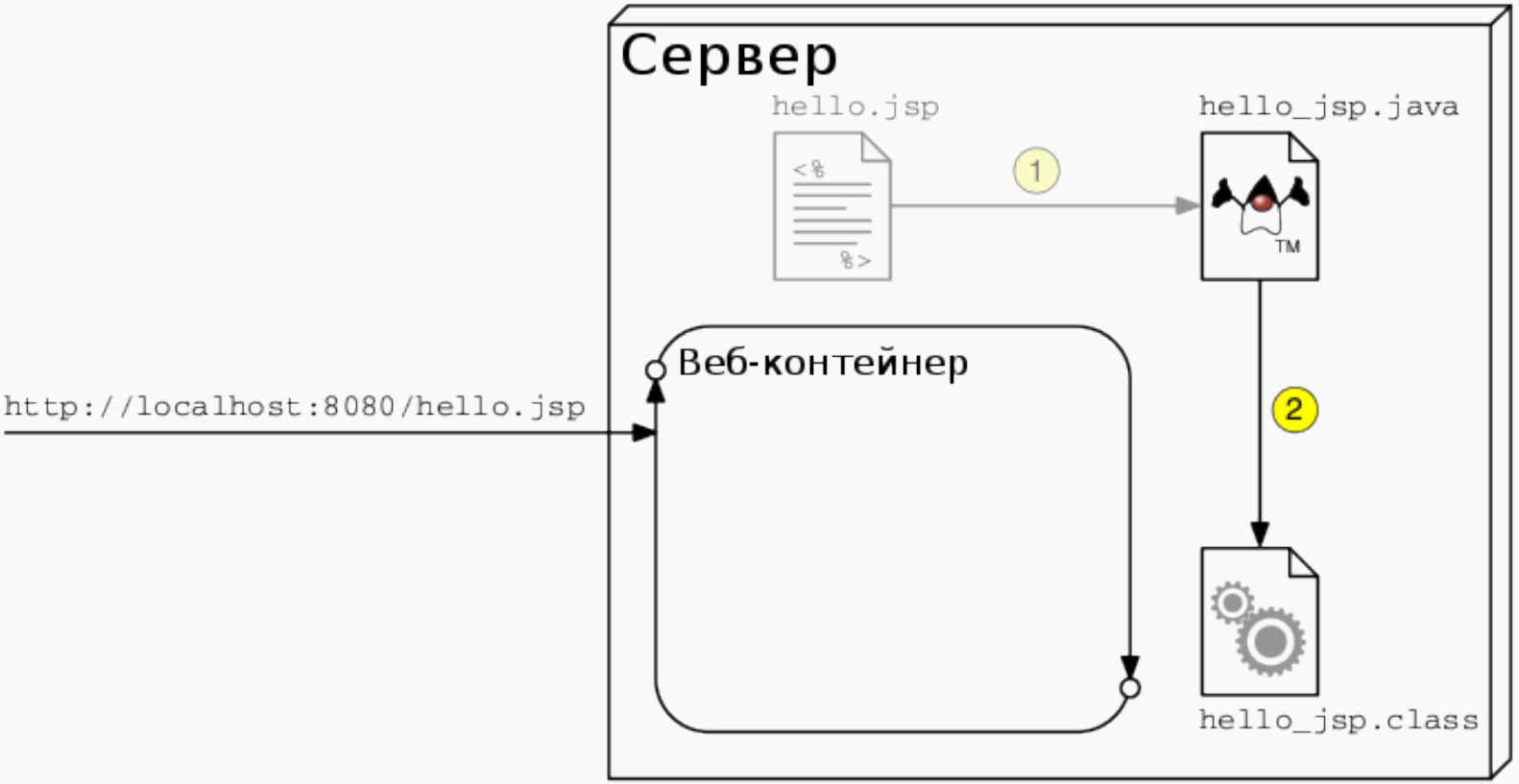
Жизненный цикл JSP (продолжение)

1. Трансляция.



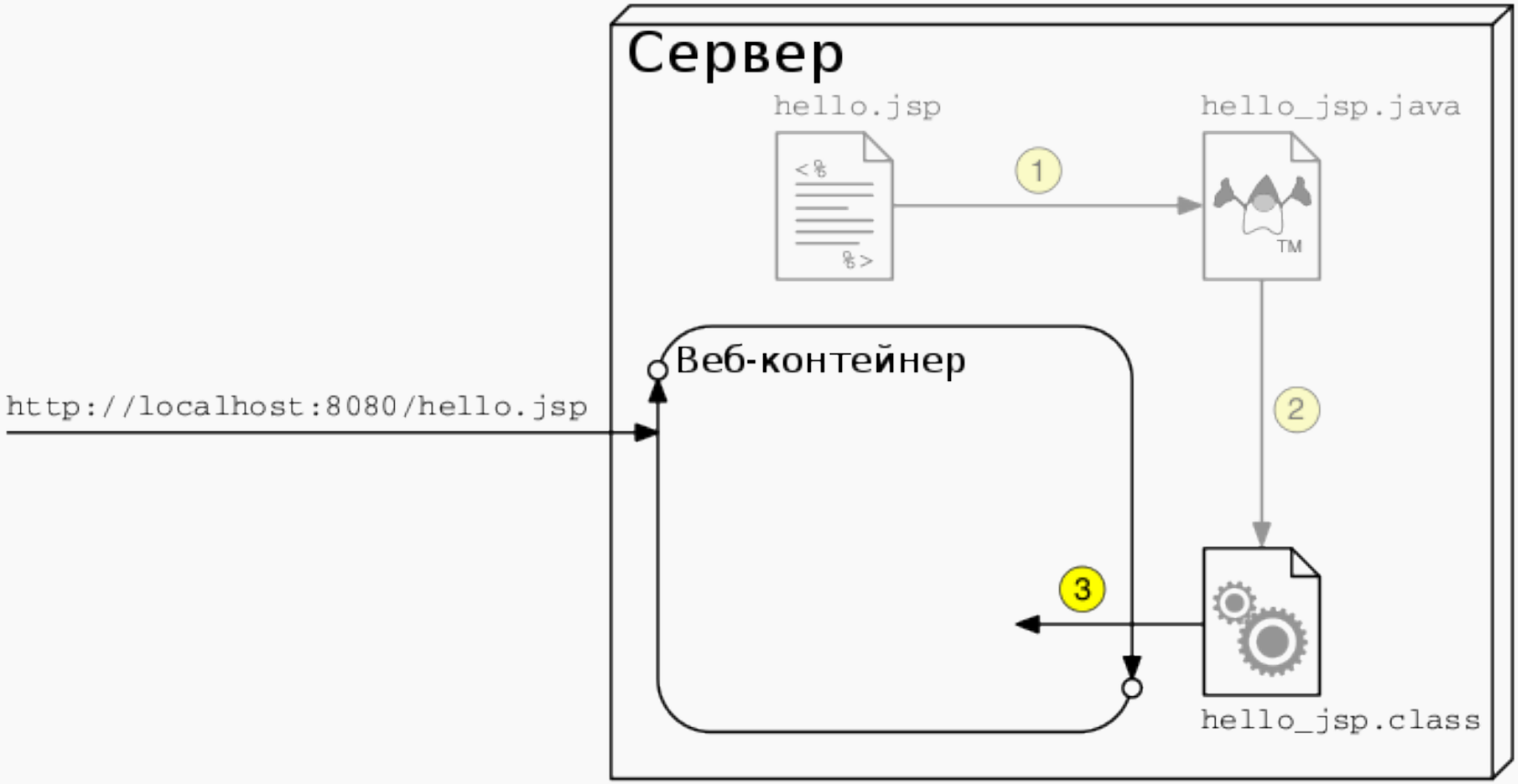
Жизненный цикл JSP (продолжение)

2. Компиляция сервлета.



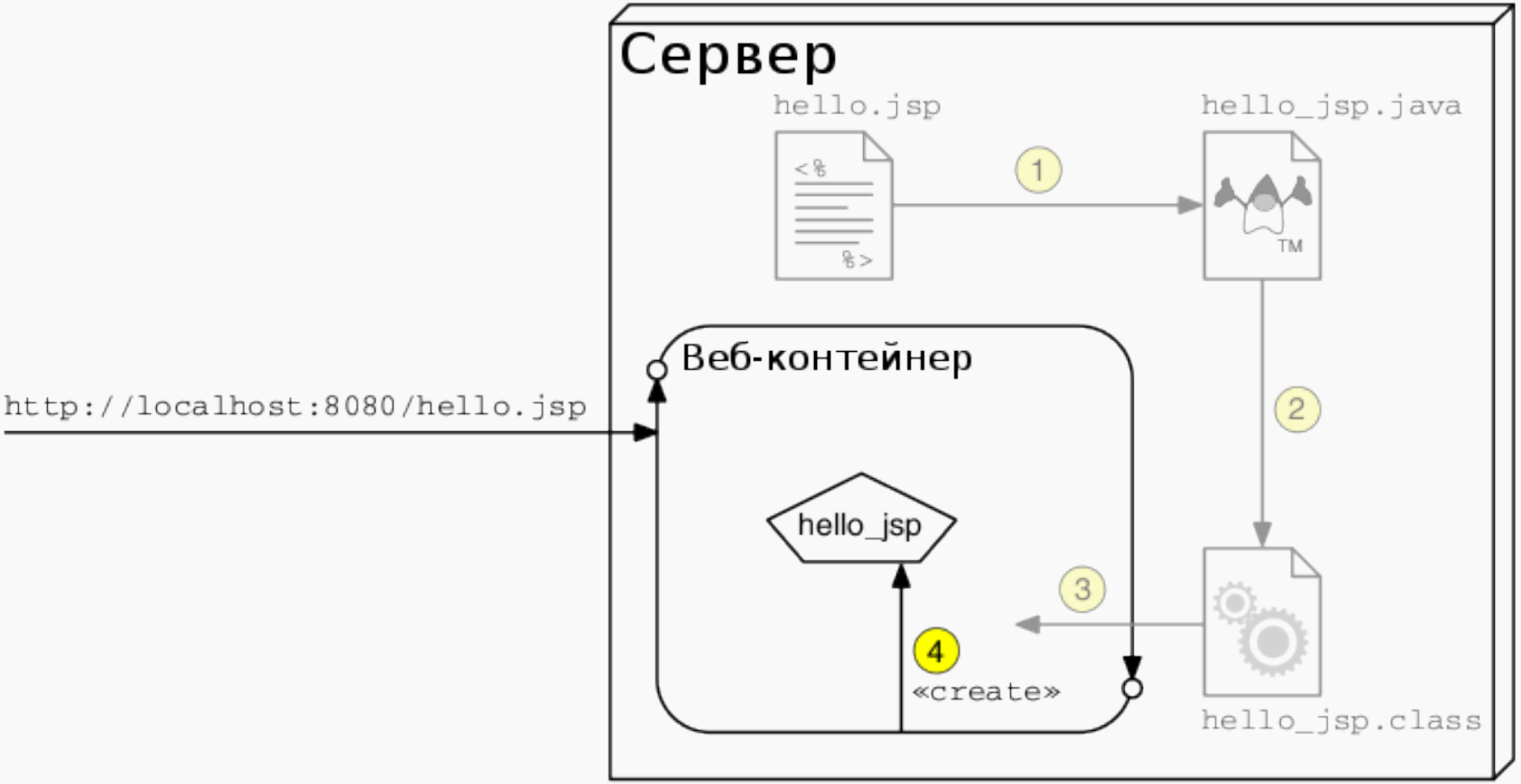
Жизненный цикл JSP (продолжение)

3. Загрузка сервлета веб-контейнером.



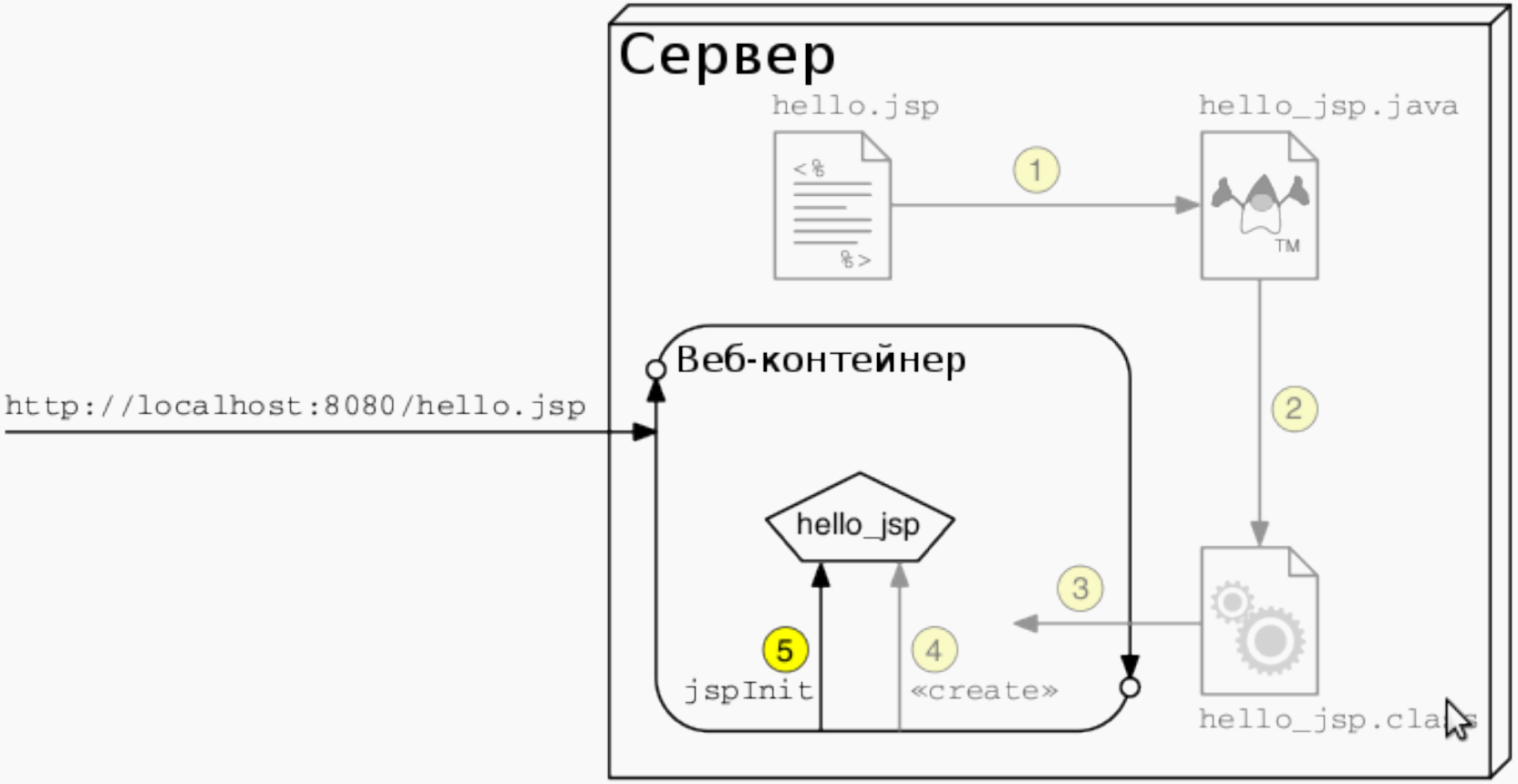
Жизненный цикл JSP (продолжение)

4. Создание веб-контейнером экземпляра сервлета.



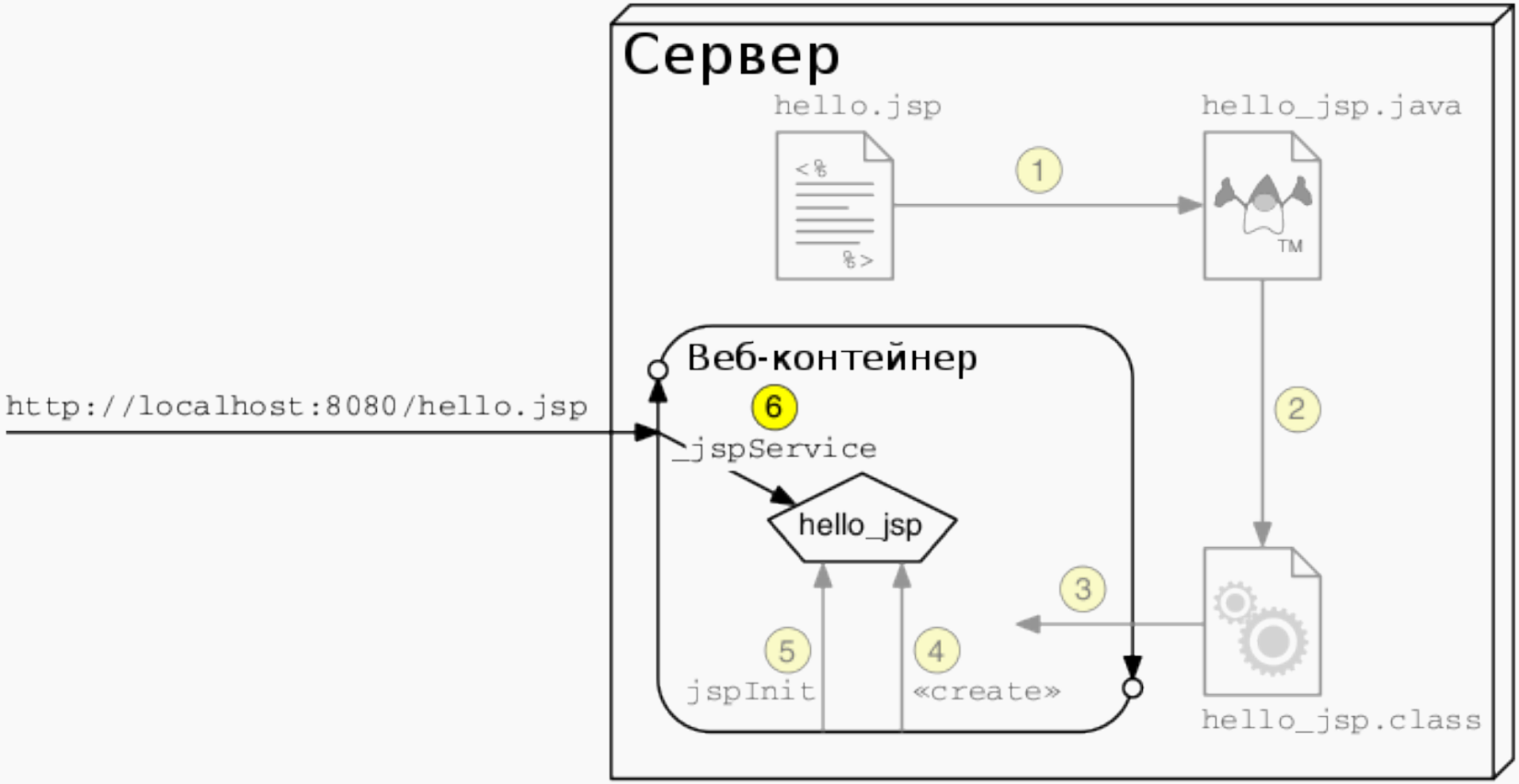
Жизненный цикл JSP (продолжение)

5. Инициализация сервлета.



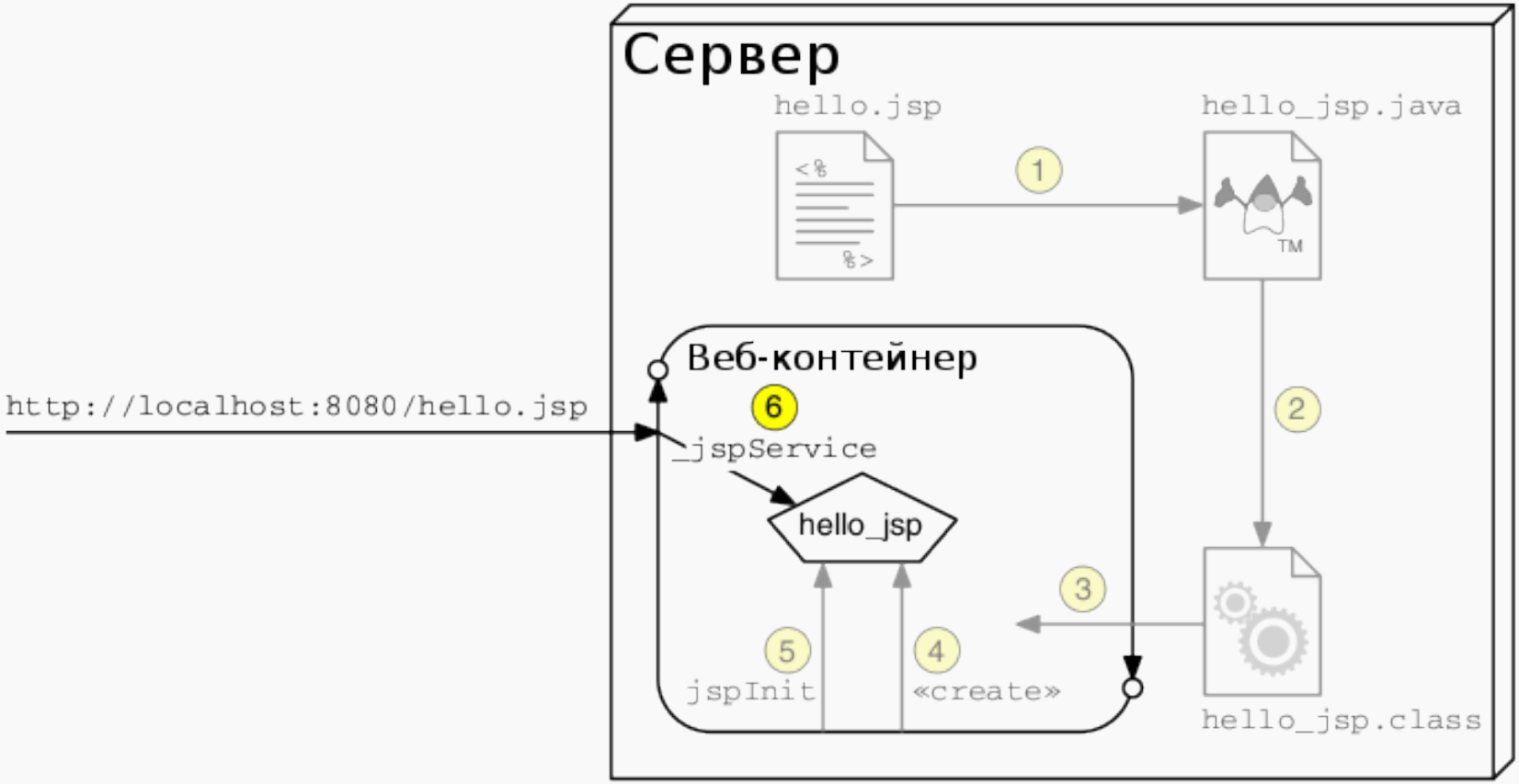
Жизненный цикл JSP (продолжение)

6. Обработка запросов.



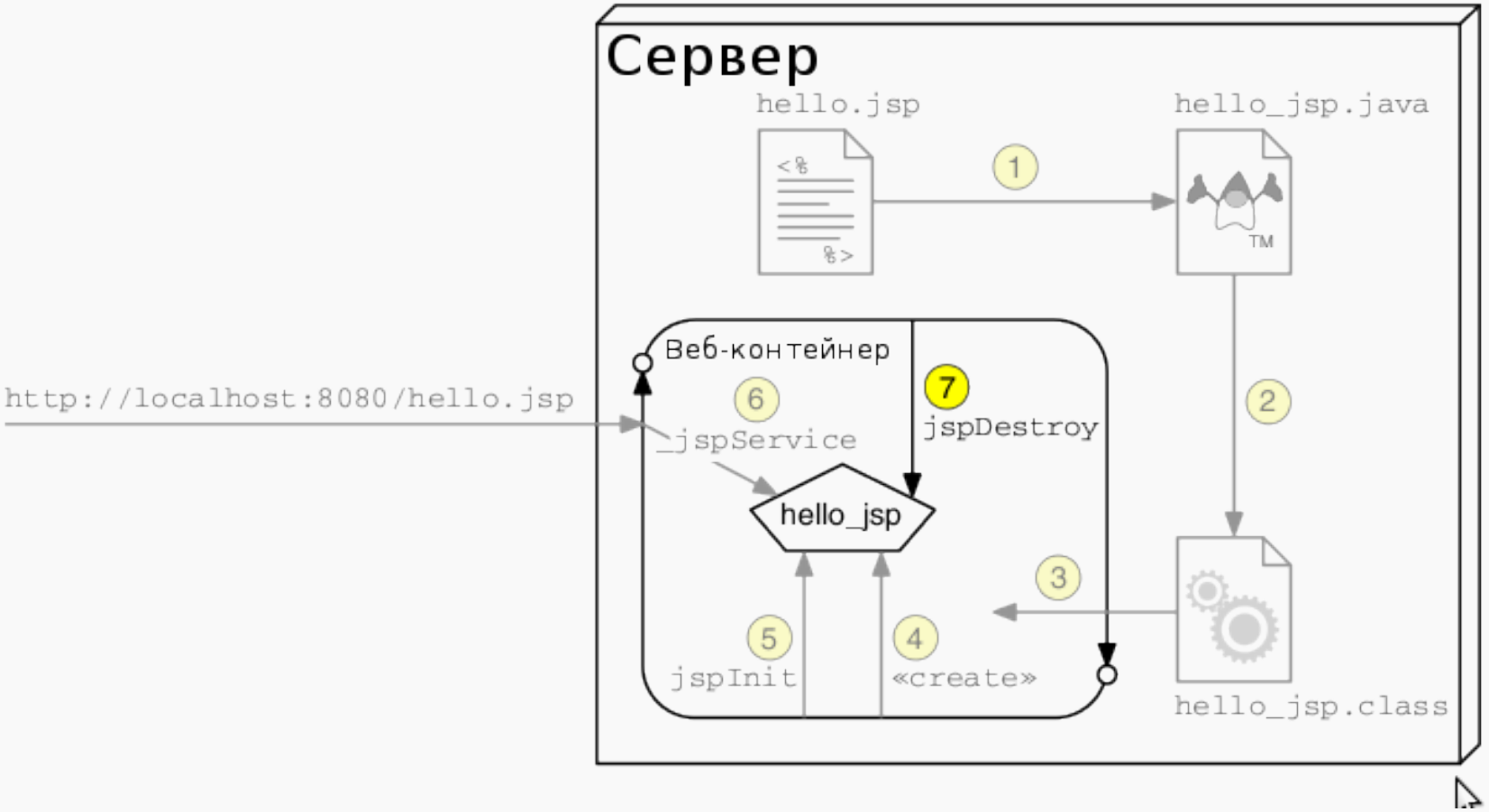
Жизненный цикл JSP (продолжение)

6. Обработка запросов.



Жизненный цикл JSP (продолжение)

7. Вызов метода jspDestroy.



ИТМО ВТ

JSP-элементы

- 2 варианта синтаксиса — на базе HTML и XML.
- Обозначаются тегами <% %> (HTML-вариант):

```
<html>
<%-- scripting element --%>
</html>
```

- Существует 5 типов JSP-элементов:
  - Комментарий — <%-- Comment --%>;
  - Директива — <%@ directive %>;
  - Объявление — <%! decl %>;
  - Скриптлет — <% code %>;
  - Выражение — <%= expr %>.

ИТМО ВТ

Комментарии

Поддерживаются 3 типа комментариев:

- HTML-комментарии:

```
<!-- This is an HTML comment.
     It will show up in the response. -->
```

- JSP-комментарии:

```
<%-- This is a JSP comment.
     It will only be seen in the JSP code.
     It will not show up in either the servlet code
     or the response.
--%>
```

- Java-комментарии:

```
<%
    /* This is a Java comment.
       It will show up in the servlet code.
       It will not show up in the response. */
%>
```

ИТМО ВТ

Директивы

Управляют процессом трансляции страницы в сервлет.

- Синтаксис:

```
<%@ DirectiveName [attr=”value”]* %>
```

- Примеры:

```
<%@ page session=”false” %>
<%@ include file=”incl/copyright.html” %>
```

ИТМО ВТ

Объявления

Позволяют объявлять поля и методы:

- Синтаксис:

```
<%! JavaClassDeclaration %>
```

- Примеры:

```
<%!
public static final String DEFAULT_NAME = “World”;
%>

<%!
public String getName(HttpServletRequest request) {
    return request.getParameter(“name”);
}
%>

<%! int counter = 0; %>
```

ИТМО ВТ

Скриптлеты

Позволяют задать Java-код, который будет выполняться при обработке запросов (при вызове метода `_jspService`).

- Синтаксис:

```
<% JavaCode %>
```

- Примеры:

```
<% int i = 0; %>

<% if ( i > 10 ) { %>
    I am a big number
<% } else { %>
    I am a small number
<% } %>
```

ИТМО ВТ

Выражения

Позволяют вывести результат вычисления выражения.

- Синтаксис:

```
<%= JavaExpression %>
```

- Примеры:

```
<B>Ten is <%= (2 * 5) %></B>
Thank you, <I><%= name %></I>, for registering
for the soccer league.
The current day and time is: <%= new
java.util.Date() %>
```

ИТМО ВТ

Предопределённые переменные

В процессе трансляции контейнер добавляет в метод `_jspService` ряд объектов, которые можно использовать в скриптлетах и выражениях:

Имя переменной	Класс
`application`	`javax.servlet.ServletContext`
`config`	`javax.servlet.ServletConfig`
`exception`	`java.lang.Throwable`
`out`	`javax.servlet.jsp.JspWriter`
`page`	`java.lang.Object`
`PageContext`	`javax.servlet.jsp.PageContext`
`request`	`javax.servlet.ServletRequest`
`response`	`javax.servlet.ServletResponse`
`session`	`javax.servlet.http.HttpSession`

ИТМО ВТ

Предопределённые переменные (продолжение)

- Exception — используется только на страницах-перенаправлениях с информацией об ошибках (Error Pages).
- Page — API для доступа к экземпляру класса сервлета, в который транслируется JSP.
- PageContext — контекст JSP-страницы.

ИТМО ВТ

Директива *Page*

- Позволяет задавать параметры, используемые контейнером при управлении жизненным циклом страницы.
- Обычно расположена в начале страницы.
- На одной странице может быть задано несколько директив page с разными указаниями контейнеру.
- Синтаксис:
  `<%@ page attribute="value" %>`

ИТМО ВТ

Атрибуты директивы *Page*

Атрибут	Для чего нужен
`buffer`	Задаёт параметры буферизации и размер буфера для потока вывода сервлета.
`autoFlush`	Указывает, автоматически ли выгружается содержимое буфера при его переполнении.
`contentType`	Позволяет задать Content Type и кодировку страницы.
`errorPage`	Позволяет задать страницу, на которую будет осуществлено перенаправление при возникновении Runtime Exception.
`isErrorPage`	Указывает, является ли текущая страница Error Page.
`extends`	Позволяет задать имя родительского класса, от которого будет наследоваться сервлет.

ИТМО ВТ

Атрибуты директивы *Page* (продолжение)

Атрибут	Для чего нужен
`import`	Импорт классов или пакетов.
`info`	Задаёт строку, которую будет возвращать метод `getServletInfo()`.
`isThreadSafe`	Если `isThreadSafe == false`, то контейнер блокирует параллельную обработку нескольких запросов страницей.
`language`	Позволяет задать язык программирования, на котором пишутся скриптовые элементы на странице (по умолчанию — Java).
`session`	Указывает контейнеру, создавать ли ему предопределённую переменную `session`.
`isELIgnored`	Указывает, вычисляются EL-выражения контейнером, или нет.
`isScriptingEnabled`	Указывает, обрабатываются ли скриптовые элементы.

ИТМО ВТ

JSP Actions

- XML-элементы, позволяющие управлять поведением сервлета.
- Синтаксис:
  `<jsp:action_name attribute="value" />`

JSP Action	Для чего нужен
`jsp:include`	Включает в страницу внешний файл *во время обработки запроса*.
`jsp:useBean`	Добавляет на страницу экземпляр Java Bean с заданным контекстом.
`jsp:getProperty` `jsp:setProperty`	Получение и установка свойств Java Bean.
`jsp:forward`	Перенаправление на другую страницу.

ИТМО ВТ

Конфигурация JSP

- Задаётся в дескрипторе развёртывания (web.xml).
- Находится внутри элемента `jsp-config`.
- Пример:

```
<jsp-config>
   <property name="initial-capacity"
             value="1024" >
</jsp-config>
```

ИТМО ВТ

JSP Standard Tag Library (JSTL)

- Расширение JSP, добавляющее возможность использования дополнительных тегов, решающих типовые задачи.
- Примеры задач:
  - Условная обработка.
  - Создание циклов, вывод массивов / коллекций.
  - Поддержка интернационализации.
- Рекомендуется использовать их вместе с EL вместо скриптлетов.

ИТМО ВТ

Теги JSTL

```
// Основные теги создания циклов, определения условий,
// вывода информации на страницу и т. д.
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>

// Теги для работы с XML-документами
<%@ taglib prefix="x"
    uri="http://java.sun.com/jsp/jstl/xml" %>

// Теги для работы с базами данных
<%@ taglib prefix="s"
    uri="http://java.sun.com/jsp/jstl/sql" %>

// Теги для форматирования и интернационализации
// информации (i10n и i18n)
<%@ taglib prefix="f"
    uri="http://java.sun.com/jsp/jstl/fmt" %>
```

ИТМО ВТ

JSP Expression Language (EL)

- Расширение JSP, позволяющее удобно работать с JavaBeans-компонентами без написания кода на Java.
- Позволяет использовать на страницах арифметические и логические выражения.
- Поддерживается «из коробки», можно отключить в настройках конкретной страницы и / или приложения.
- Пример использования:

```
<jsp:text>
    Box Perimeter is:
    ${2*box.width + 2*box.height}
</jsp:text>
```

Пример использования JSTL + EL

```
<%@ taglib prefix="c"
    uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <head>
        <title>Пример тега <c:if> библиотеки JSTL</title>
    </head>
    <body>
        <c:set var="salary" scope="session"
                value="${23400*2}"/>
        <c:if test="${salary > 45000}">
            <p>Salary = <c:out value="${salary}"/><p>
        </c:if>
    </body>
</html>
```

3.3.2. FreeMarker Template Engine

ИТМО ВТ

Apache FreeMarker

- Компилирующий обработчик шаблонов.
- Написан на Java.
- Разработчик – Apache Software Foundation, первая версия вышла в 2000 г.
- Свободное ПО, распространяется по лицензии BSD.

ИТМО ВТ

Особенности и возможности

Поддерживает «джентльменский набор» возможностей по созданию шаблонов:

- условия;
- циклы;
- присваивание значений переменным;
- арифметические операции;
- операции со строками;
- инструменты форматирования;
- макросы и функции;
- подключение внешних шаблонов;
- экранирование символов.

ИТМО ВТ

Пример шаблона

```
<html>
<head>
    <title>Welcome!</title>
</head>
<body>
    <h1>Welcome ${user}!</h1>
    <p>Our latest product:
    <a href="${latestProduct.url}">
        ${latestProduct.name}
    </a>!
</body>
</html>
```

ИТМО ВТ

Синтаксис шаблонов

Шаблон может содержать следующие элементы:

- Статический HTML.
- Обращения к модели данных:

```
Welcome ${user}!
```

- Директивы:

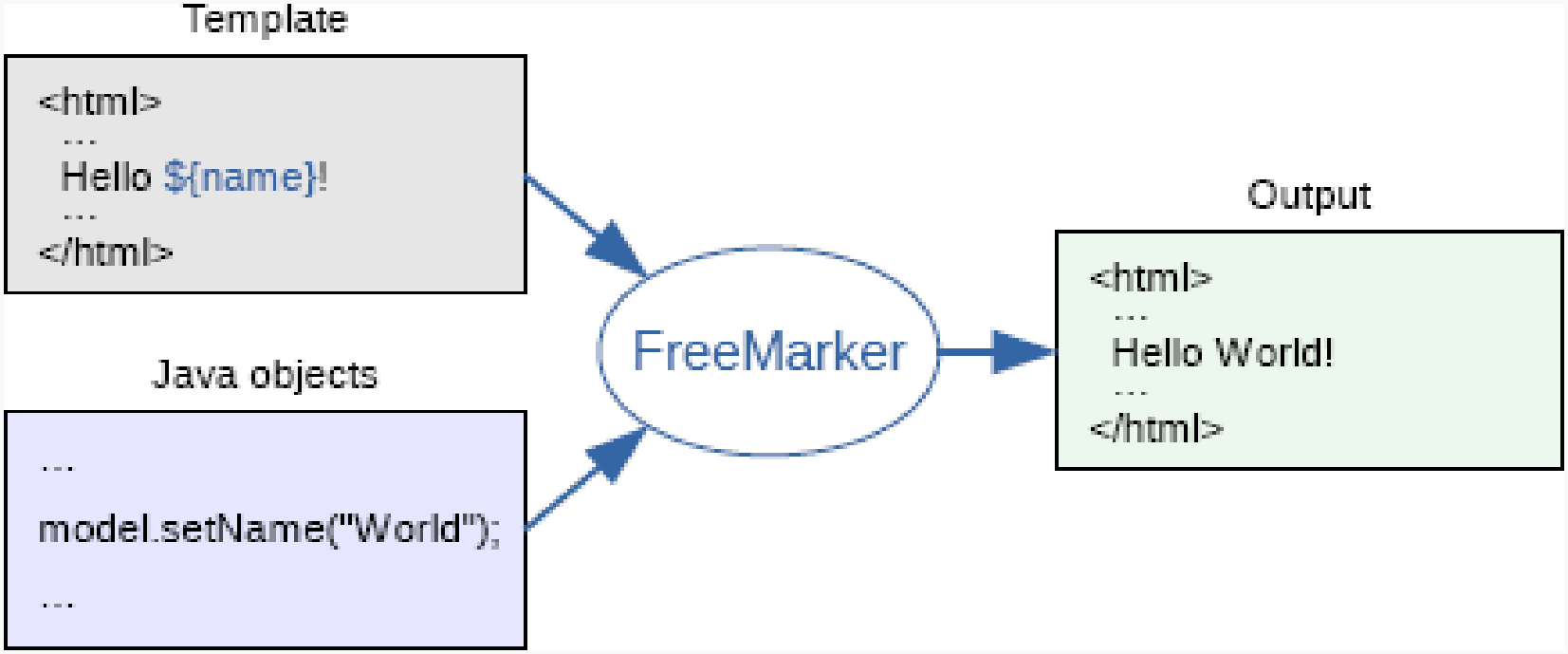
```
<#if animals.python.price != 0>
    Pythons are not free today!
</#if>
```

- Вызовы встроенных функций:

```
animals?filter(it -> it.protected)
```

ИТМО ВТ

Принцип работы FreeMarker



ИТМО ВТ

Модель данных FTL (FTL data-model)

- Древовидная объектная структура, данные из которой шаблон использует при формировании HTML.
- Элементы дерева – Java Beans.
- Сложность иерархии может быть любой.
- При выводе в HTML все объекты преобразуются в строки.

Пример модели данных

```
(root)
  |
  +- user = "Big Joe"
  +- latestProduct
      |
      +- url = "products/greenmouse.html"
      +- name = "green mouse"
```

```
// «Корневой» объект. Может быть JavaBean, может быть Map.
Map<String, Object> root = new HashMap<>();

// Кладём строку "user" в «корневой» объект
root.put("user", "Big Joe");

// Создаём объект "latestProduct". Здесь использован Java Bean,
// но Map'ы можно также вкладывать друг в друга.
Product latest = new Product();
latest.setUrl("products/greenmouse.html");
latest.setName("green mouse");
// Кладём "latestProduct" в «корневой» объект.
root.put("latestProduct", latest);
```

ИТМО ВТ

Использование в проектах

1) Скачиваем `freemarker.jar`.
2) Создаём конфигурацию.
3) Создаём шаблон.
4) Создаём модель данных.
5) Компилируем шаблон с нужными данными:

```
Writer out =
 new OutputStreamWriter(System.out);
temp.process(root, out);
```

3.3.3. Thymeleaf

Thymeleaf

- Компилирующий обработчик шаблонов.
- Универсальный – может использоваться как в веб-приложениях, так и для решения общих задач шаблонизации чего-либо.
- Написан на Java.
- Свободное ПО, распространяется по лицензии Apache 2.0.
- Интеграция «из коробки» со Spring Framework.

ИТМО ВТ

Особенности Thymeleaf

- «Выходной» формат – XML, XHTML и HTML5. Также поддерживаются JS, CSS и Plain Text.
- Не привязан к Servlet API – может работать как «онлайн», так и «оффлайн».
- Построен на модульной системе. Модули называются *диалектами (dialects)*.
- Поддержка возможностей i18n и l10n.
- Есть встроенный кеш скомпилированных шаблонов.

ИТМО ВТ

Процессоры и диалекты

- Thymeleaf – модульный движок. Модуль Thymeleaf называется *диалектом (dialect)*.
- Диалект состоит из одного или нескольких *процессоров (processor)*.
- Процессор – объект, который применяет некоторую логику к формируемому на основе шаблона артефакту.
- «Из коробки» Thymeleaf содержит *стандартный диалект (Standard Dialect)*, которого достаточно для решения большинства типовых задач.

ИТМО ВТ

Стандартный диалект

- Содержит набор процессоров, предназначенных для решения типовых задач.
- Большая часть процессоров стандартного диалекта – *процессоры атрибутов (Attribute Processors)*.
- Процессоры атрибутов обрабатывают дополнительные («нестандартные») атрибуты тегов:

```
<input type="text" name="userName"
       value="James Carrot"
       th:value="${user.name}" />
```

- Благодаря этому шаблоны Thymeleaf обычно можно тестировать в браузере – он просто игнорирует нестандартные атрибуты.

ИТМО ВТ

Выражения

- Значения атрибутов присваиваются путём вычисления *выражений (expressions)*.
- Выражения, поддерживаемые стандартным диалектом, называются *стандартными выражениями*.
- В стандартном диалекте Thymeleaf реализована поддержка 5 видов выражений:
  - `${...}` – Variable expressions.
  - `*{...}` – Selection expressions.
  - `#{...}` – Message (i18n) expressions.
  - `@{...}` – Link (URL) expressions.
  - `~{...}` – Fragment expressions.

ИТМО ВТ

Пример шаблона

```
<table>
  <thead>
    <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="prod : ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price,1,2)}">
        0.99
      </td>
    </tr>
  </tbody>
</table>
```

ИТМО ВТ

Variable Expressions

- Выражения на языке OGNL (Object Graph Navigation Language).
- Позволяют обращаться к переменной из контекста Thymeleaf.
- В случае использования совместно со Spring – позволяют обращаться к атрибутам модели:

```
<span th:text="${book.author.name}">
```

```
((Book)context.getVariable("book"))
              .getAuthor()
              .getName()
```

- Могут использоваться с более сложными конструкциями (условные выражения, перечисления и т. д.):

```
<li th:each="book : ${books}">
```

ИТМО ВТ

Selection Expressions

Позволяют обратиться к выбранному ранее объекту вместо обращения к контексту.

```
<div th:object="${book}">
  …
  <span th:text="*{title}">...</span>
  …
</div>
```

```
{
  final Book selection =
       (Book) context.getVariable("book");
  output(selection.getTitle());
}
```

ИТМО ВТ

Message (i18n) Expressions

- Позволяют обращаться к сообщениям из файлов локализации (`.properties`):

```
#{main.title}
```

- Помимо обращения по ключу, можно использовать параметры, в т.ч., вычисленные с помощью Variable Expressions:

```
#{message.entrycreated(${entryId})}
```

- Могут использоваться в любых элементах, предполагающих вывод локализованного текста:

```
<table>
  ...
  <th th:text="#{header.address.city}">...</th>
  <th th:text="#{header.address.country}">...</th>
  ...
</table>
```

- В сложных случаях могут целиком вычисляться с помощью Variable Expressions:

```
#{${config.adminWelcomeKey}(${session.user.name})}
```

Link (URL) Expressions

- Позволяют формировать URL внутри контекста приложения:

```
<a th:href="@{/order/list}">...</a>

<a href="/myapp/order/list;jsessionid=23fa31abd41ea093">
    ...
</a>
```

- Также могут принимать аргументы:

```
<a th:href="@{/order/details(id=${orderId},type=${orderType})}">...</a>

<a href="/myapp/order/details?id=23&amp;type=online">...</a>
```

- Могут быть относительными (внутри контекста приложения), привязанными к контексту сервера, привязанными к протоколу или абсолютными:

```
<a th:href="@{../documents/report}">...</a>

<a th:href="@{~/contents/main}">...</a>

<a th:href="@{//static.mycompany.com/res/initial}">...</a>

<a th:href="@{http://www.mycompany.com/main}">...</a>
```

ИТМО ВТ

Fragment Expressions

Позволяют компоновать шаблоны из фрагментов:

```
<div th:insert="~{commons :: main}">
    ...
</div>
```

Фрагменты могут использоваться несколько раз, передаваться другим шаблонам в качестве аргументов и т.д.:

```
<div th:with="frag=~{footer :: #main/text()}">
  <p th:insert="${frag}">
</div>
```

ИТМО ВТ

Использование в приложениях

Необходимо проинициализировать объекты `TemplateEngine` и `TemplateResolver`:

```
...
private final TemplateEngine templateEngine;
...


public GTVGApplication(final ServletContext servletContext) {

    super();

    ServletContextTemplateResolver templateResolver =
            new ServletContextTemplateResolver(servletContext);

    // HTML is the default mode, but we set it anyway for better understanding of code
    templateResolver.setTemplateMode(TemplateMode.HTML);
    // This will convert "home" to "/WEB-INF/templates/home.html"
    templateResolver.setPrefix("/WEB-INF/templates/");
    templateResolver.setSuffix(".html");
    // Template cache TTL=1h. If not set, entries would be cached until expelled
    templateResolver.setCacheTTLMs(Long.valueOf(3600000L));

    // Cache is set to true by default. Set to false if you want templates to
    // be automatically updated when modified.
    templateResolver.setCacheable(true);

    this.templateEngine = new TemplateEngine();
    this.templateEngine.setTemplateResolver(templateResolver);

    ...

}
```

ИТМО ВТ

Использование в приложениях на базе Spring

- Есть интеграция «из коробки» – библиотеки `thymeleaf-spring3` и `thymeleaf-spring4`.
- Эти библиотеки позволяют использовать Thymeleaf-шаблоны вместо JSP в приложениях на базе Spring.
- Для интеграции в состав Spring-приложений реализован специальный диалект Thymeleaf – SpringStandard.

4. Rich Internet Applications

ИТМО ВТ

Концепция RIA

- RIA – интернет-приложения, предназначенные для выполнения задач, обычно выполняемых десктопными приложениями.
- Впервые термин упомянут компанией Macromedia в 2002 г.
- Изначально применялся к «тяжёлым» технологиям а-ля Flash и Silverlight, требовавшим для работы отдельные плагины.
- В настоящее время термин достаточно «размыт», но чаще всего применяется к компонентно-ориентированным server-side веб-фреймворкам (gwt, JSF).

ИТМО ВТ

Преимущества и недостатки RIA

Преимущества:

- Приложения пишутся в стиле, похожем на стиль написания десктопных приложений.
- Высокий уровень абстракции => меньше «рутинной» работы программиста.
- Много готовых компонентов.

Недостатки:

- Высокий уровень абстракции => тяжелее «спуститься» на уровень протокола (а это часто бывает надо!)
- Сложность архитектуры фреймворков => сложно сделать что-то, не предусмотренное их разработчиками.
- Идеология «ломает» парадигму веб-страниц.

4.1. JavaServer Faces

ИТМО ВТ

JavaServer Faces

- JSF — фреймворк для разработки веб-приложений.
- Входит в состав платформы Java EE.
- Основан на использовании *компонентов*.
- Для отображения данных используются JSP или XML-шаблоны (*facelets*).

ИТМО ВТ

Достоинства JSF

- Чёткое разделение бизнес-логики и интерфейса (фреймворк реализует шаблон MVC).
- Управление обменом данными на уровне компонент.
- Простая работа с событиями на стороне сервера.
- Доступность нескольких реализаций от различных компаний-разработчиков.
- Расширяемость (можно использовать дополнительные наборы компонентов).
- Широкая поддержка со стороны интегрированных средств разработки (IDE).

ИТМО ВТ

Недостатки JSF

- Высокоуровневый фреймворк — сложно реализовывать не предусмотренную авторами функциональность.
- Сложности с обработкой GET-запросов (устранены в JSF 2.0).
- Сложность разработки собственных компонентов.

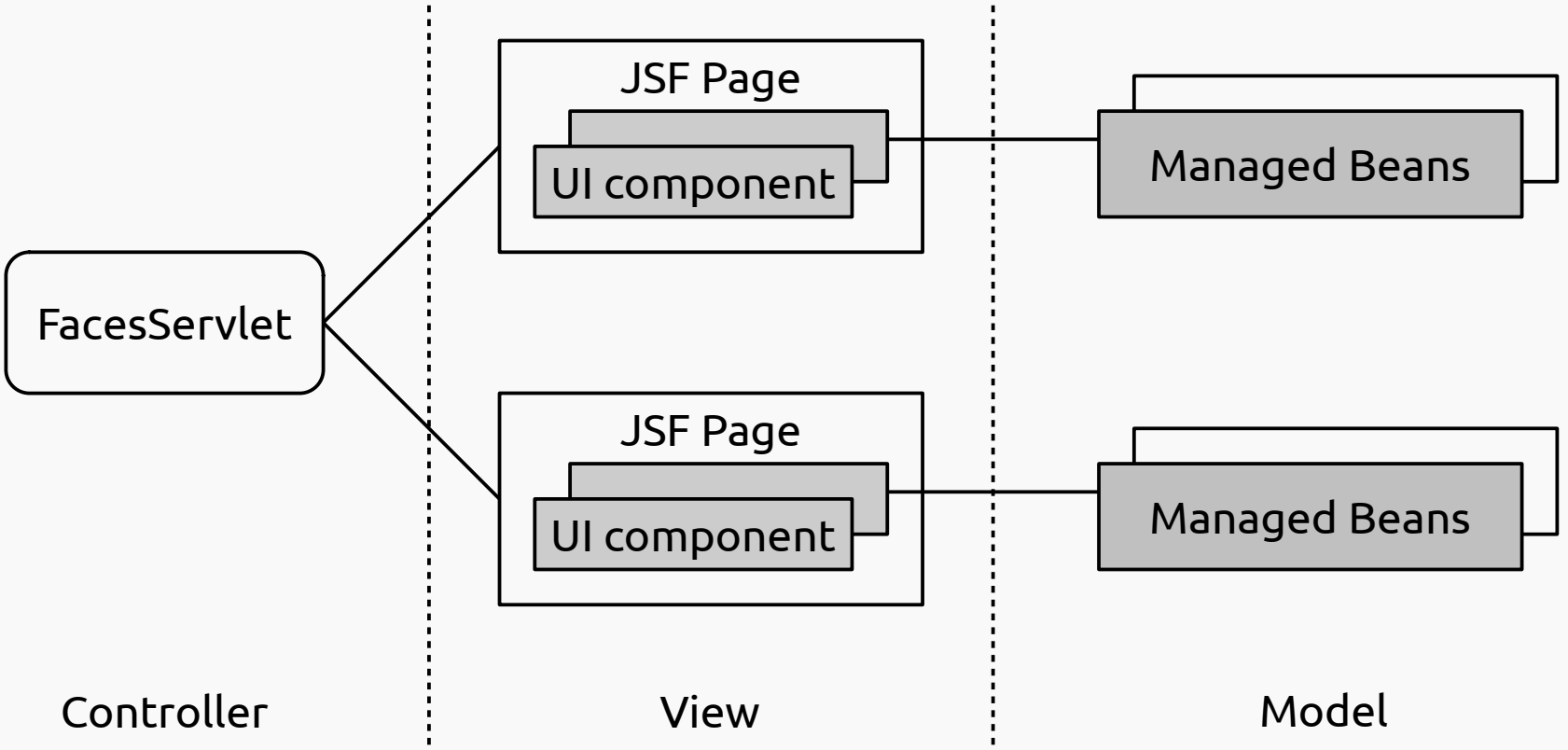
ИТМО ВТ

Структура JSF-приложения

- JSP или XHTML-страницы, содержащие компоненты GUI.
- Библиотеки тегов.
- Управляемые бины.
- Дополнительные объекты (компоненты, конвертеры и валидаторы).
- Дополнительные теги.
- Конфигурация — faces-config.xml (опционально).
- Дескриптор развёртывания — web.xml.

ИТМО ВТ

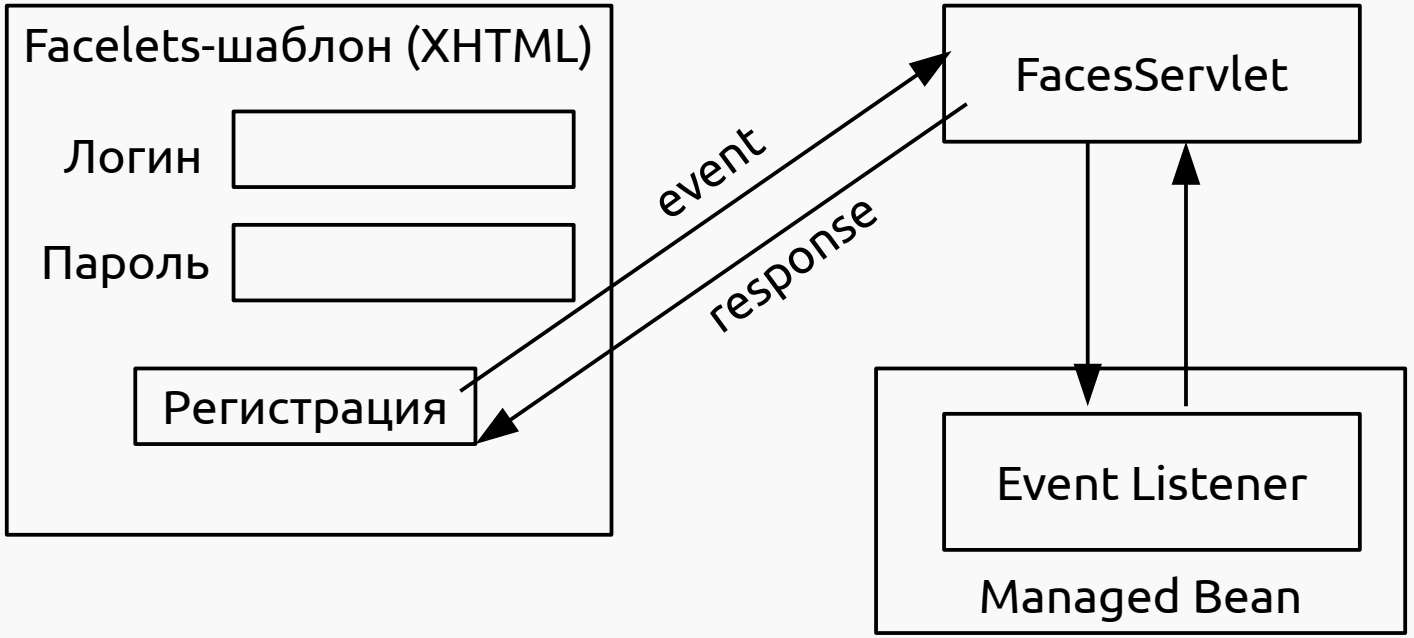
MVC-модель JSF



ИТМО ВТ

FacesServlet

- Обрабатывает запросы с браузера.
- Формирует объекты-события и вызывает методы-слушатели.



ИТМО ВТ

Конфигурация FacesServlet

Конфигурация задаётся в web.xml:

```
<!-- Faces Servlet -->
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>
    javax.faces.webapp.FacesServlet
  </servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<!-- Faces Servlet Mapping -->
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

ИТМО ВТ

Страницы и компоненты UI

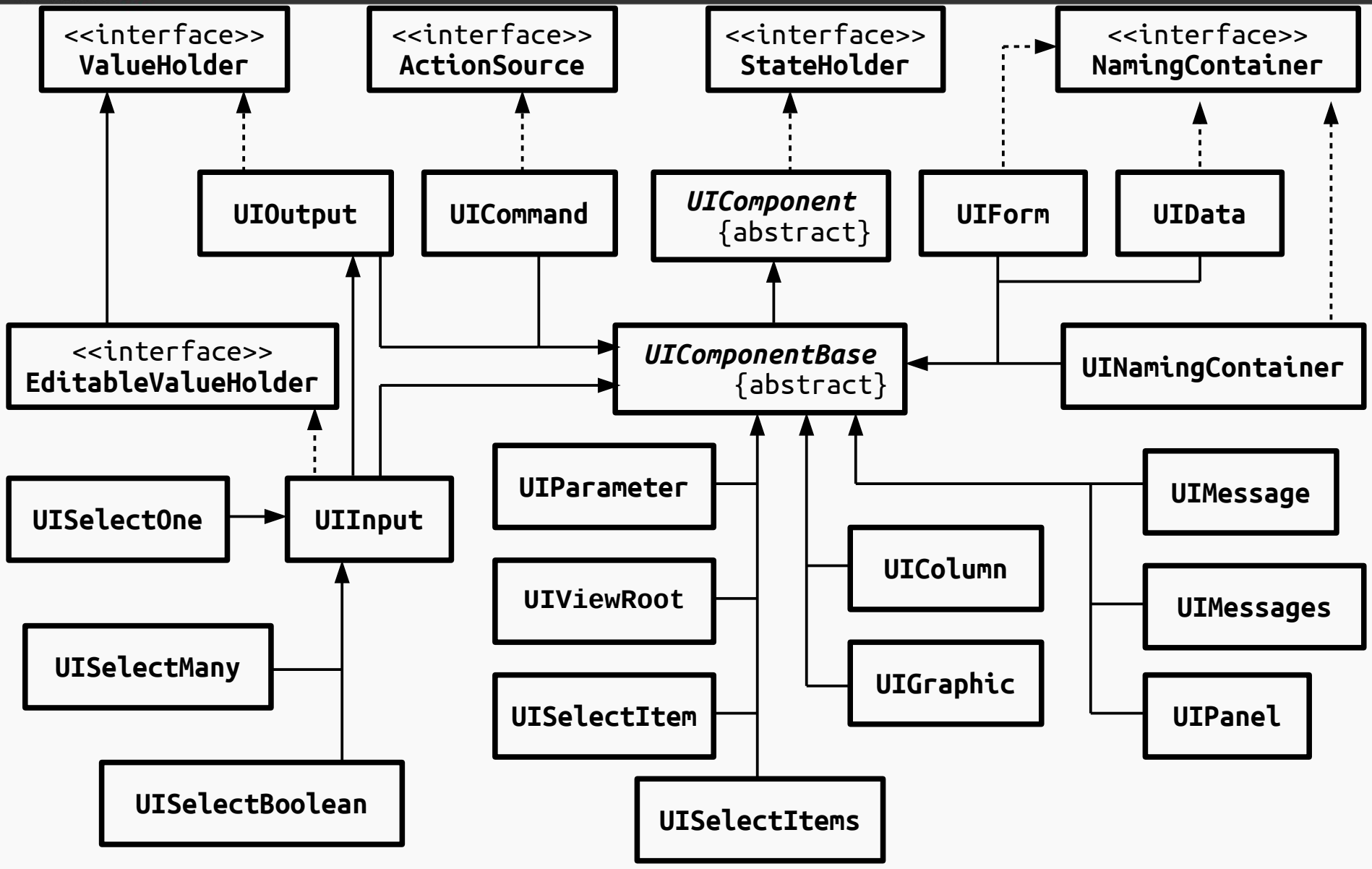
- Интерфейс строится из компонентов.
- Компоненты расположены на Facelets-шаблонах или страницах JSP.
- Компоненты реализуют интерфейс `javax.faces.component.UIComponent`.
- Можно создавать собственные компоненты.
- Компоненты на странице объединены в древовидную структуру — *представление*.
- Корневым элементов представления является экземпляр класса `javax.faces.component.UIViewRoot`.

Пример страницы JSF (Facelets)

```
<?xml version="1.0" encoding="UTF-8"?>
<!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:f="http://java.sun.com/jsf/core"
     xmlns:h="http://java.sun.com/jsf/html">
     <h:body>
      <h3>JSF 2.0 + Ajax Hello World Example</h3>
      <h:form>
         <h:inputText id="name"
value="#{helloBean.name}"></h:inputText>
         <h:commandButton value="Welcome Me">
            <f:ajax execute="name" render="output" />
         </h:commandButton>
         <h2>
          <h:outputText id="output"
value="#{helloBean.sayWelcome}" />
        </h2>
      </h:form>
    </h:body>
</html>
```

ИТМО ВТ

Иерархия компонентов JSF



Навигация между страницами JSF

- Реализуется экземплярами класса `NavigationHandler`.
- Правила задаются в файле faces-config.xml:

```
<navigation-rule>
   <from-view-id>/pages/inputname.xhtml</from-view-
id>
   <navigation-case>
      <from-outcome>sayHello</from-outcome>
      <to-view-id>/pages/greeting.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <to-view-id>/pages/goodbye.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>
```

- Пример перенаправления на другую страницу:

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

ИТМО ВТ

Управляемые бины

- Содержат параметры и методы для обработки данных с компонентов.
- Используются для обработки событий UI и валидации данных.
- Жизненным циклом управляет JSF Runtime Envronment.
- Доступ из JSF-страниц осуществляется с помощью элементов EL.
- Конфигурация задаётся в faces-config.xml (JSF 1.X), либо с помощью аннотаций (JSF 2.0).
- Вместо них могут использоваться CDI-бины, EJB или бины Spring.

ИТМО ВТ

Пример управляемого бина

```
package org.itmo.sample;

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

@ManagedBean
@SessionScoped
public class HelloBean implements Serializable {

    private static final long serialVersionUID = 1L;
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSayWelcome(){
        if("".equals(name) || name == null){ //check if null?
            return "";
        }else{
            return "Ajax message : Welcome " + name;
        }
    }
}
```

ИТМО ВТ

Контекст (scope) управляемых бинов

- Задаётся через faces-config.xml или с помощью аннотаций.
- 6 вариантов конфигурации:
  - `@NoneScoped` — контекст не определён, жизненным циклом управляют другие бины.
  - `@RequestScoped` (применяется по умолчанию) — контекст — запрос.
  - `@ViewScoped` (JSF 2.0) — контекст — страница.
  - `@SessionScoped` — контекст — сессия.
  - `@ApplicationScoped` — контекст — приложение.
  - `@CustomScoped` (JSF 2.0) — бин сохраняется в Map; программист сам управляет его жизненным циклом.

ИТМО ВТ

Конфигурация управляемых бинов

Способ 1 — через faces-config.xml:

```
<managed-bean>
  <managed-bean-name>customer</managed-bean-name>
  <managed-bean-class>CustomerBean</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>areaCode</property-name>
    <value>#{initParam.defaultAreaCode}</value>
  </managed-property>
</managed-bean>
```

Конфигурация управляемых бинов (продолжение)

Способ 2 (JSF 2.0) — с помощью аннотаций:

```
@ManagedBean(name="customer")
@RequestScoped
public class CustomerBean {

   ...

@ManagedProperty(value="#{initParam.defaultAreaCode}"
    name="areaCode")
  private String areaCode;
  ...
}
```

ИТМО ВТ

Доступ к управляемым бинам со страниц приложения

Осуществляется с помощью EL-выражений:

```
...
<h:inputText value="#{user.name}"
             validator="#{user.validate}" />
...
<h:inputText binding="#{user.nameField}" />
...
<h:commandButton action="#{user.save}"
                 value="Save" />
...
```

ИТМО ВТ

CDI-бины

- Универсальные компоненты уровня бизнес-логики.
- Появились в Java EE 6, копируют концепции, реализованные в Spring.
- Общая идея – «отвязаться» от конкретного фреймворка при создании бизнес-логики внутри приложения.
- В большинстве случаев их можно использовать вместо JSF Managed Beans и EJB.
- По реализации очень похожи на JSF Managed Beans.

ИТМО ВТ

Использование CDI-бинов

```
@Named("bb")
@SessionScoped
public class BookBean {
    { ... }
}

@Path("/book")
public class BookEndpoint {
    @Inject
    private @Named("bb") BookBean bookBean;

    @GET
    public List<Book> getAllBooks() { ... }
}
```

ИТМО ВТ

Enterprise Java Beans

- Компоненты уровня бизнес-логики для «кровавого энтерпрайза».
- Два основных вида – Session Beans и Message-driven Beans.
- Session Beans похожи на JSF Managed Beans и CDI-бины, но обеспечивают много дополнительных возможностей.
- Session Beans могут быть «прозрачно» использованы в JSF.
- Message-driven Beans предназначены для асинхронного выполнения задач и в JSF не используются.

ИТМО ВТ

Пример Statless Session Bean

```
package converter.ejb;

import java.math.BigDecimal;
import javax.ejb.*;

@Stateless
public class ConverterBean {
    private BigDecimal yenRate = new BigDecimal("83.0602");
    private BigDecimal euroRate = new BigDecimal("0.0093016");

    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }

    public BigDecimal yenToEuro(BigDecimal yen) {
        BigDecimal result = yen.multiply(euroRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }
}
```

ИТМО ВТ

Spring Beans

- Универсальные компоненты уровня бизнес-логики на платформе Spring.
- Могут быть использованы в JSF путём несложной конфигурации последнего в `faces-config.xml`:

```
<el-resolver>
  org.springframework.web.jsf.el
    .SpringBeanFacesELResolver
</el-resolver>
```

Использование Spring Beans в JSF

```
public interface UserManagementDAO {
    boolean createUser(
                    String newUserData);
}

@Named("registration")
@RequestScoped
public class RegistrationBean
                    implements Serializable {
    @Inject
    UserManagementDAO theUserDao;
}
```

ИТМО ВТ

Конвертеры данных

- Используются для преобразования данных компонента в заданный формат (дата, число и т. д.).
- Реализуют интерфейс `javax.faces.convert.Converter`.
- Существуют стандартные конвертеры для основных типов данных.
- Можно создавать собственные конвертеры.

ИТМО ВТ

Назначение конвертеров

- Автоматическое (на основании типа данных):
```
<h:inputText value="#{user.age}"/>
```
- С помощью атрибута converter:
```
<h:inputText
    converter="#{javax.faces.DateTime}"/>
```
- С помощью вложенного тега:
```
<h:outputText value="#{user.birthDay}">
  <f:converter
   converterId="#{javax.faces.DateTime}"/>
</h:outputText>
```

ИТМО ВТ

Валидация данных JSF-компонентов

- Осуществляется перед обновлением значения компонента на уровне модели.
- Класс, осуществляющий валидацию, должен реализовывать интерфейс `javax.faces.validator.Validator`.
- Существуют стандартные валидаторы для основных типов данных.
- Можно создавать собственные валидаторы.

ИТМО ВТ

Способы валидации данных

- С помощью параметров компонента:

```
<h:inputText id="zip" size="10"
             value="#{customerBean.zip}"
             required="true">
</h:inputText>
<h:message for="zip"/>
```

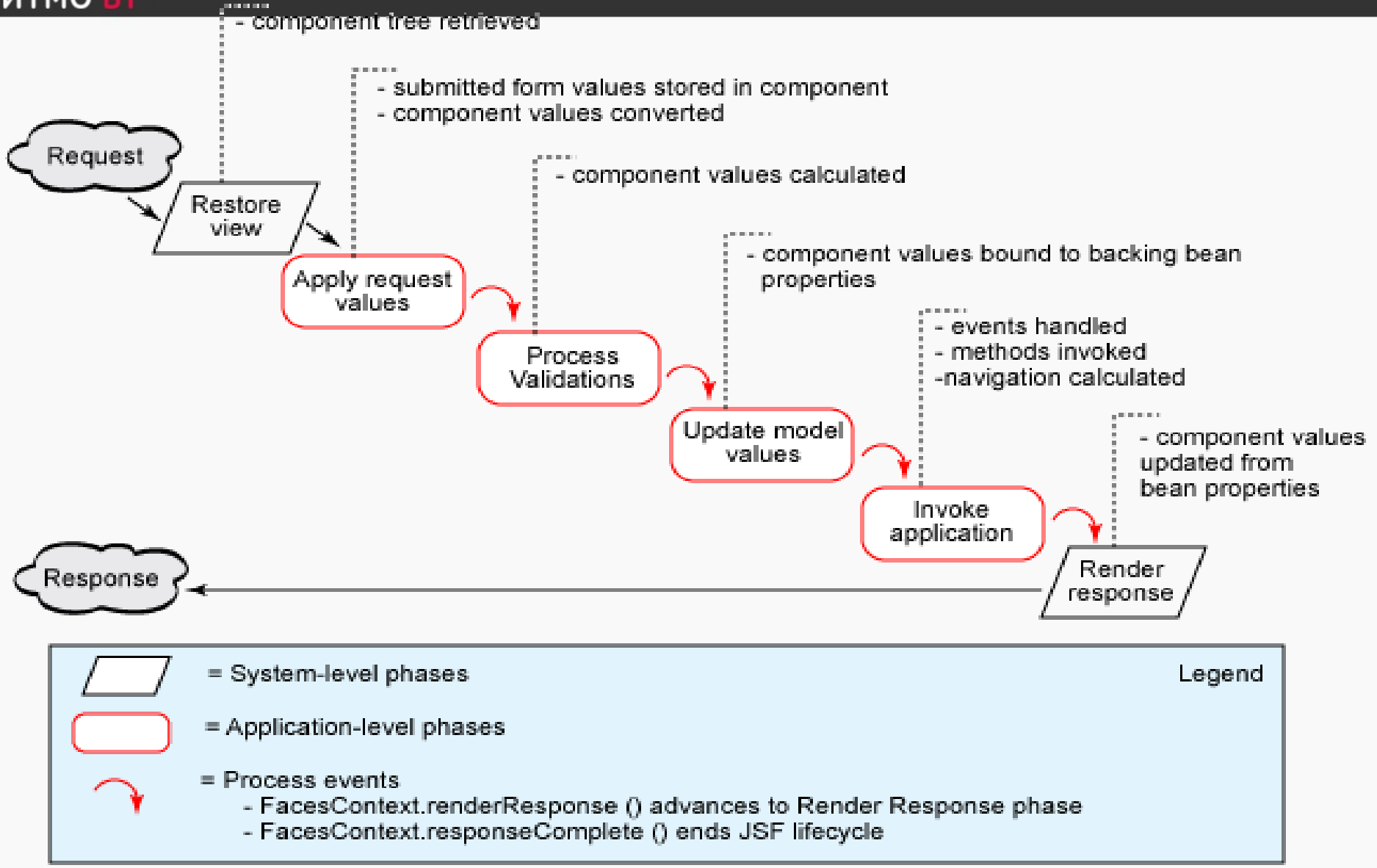
- С помощью вложенного тега:

```
<h:inputText id="quantity" size="4"
             value="#{item.quantity}">
  <f:validateLongRange minimum="1"/>
</h:inputText>
<h:message for="quantity"/>
```

- С помощью логики на уровне управляемого бина.

ИТМО ВТ

Обработка событий



Фаза формирования представления (Restore View Phase)

- JSF Runtime формирует представление (начиная с `UIViewRoot`):
  - Создаются объекты компонентов.
  - Назначаются слушатели событий, конвертеры и валидаторы.
  - Все элементы представления помещаются в `FacesContext`.
- Если это первый запрос пользователя к странице JSF, то формируется пустое представление.
- Если это запрос к уже существующей странице, то JSF Runtime синхронизирует состояние компонентов представления с клиентом.

ИТМО ВТ

Фаза получения значений компонентов (Apply Request Values Phase)

- На стороне клиента все значения хранятся в строковом формате — нужна проверка их корректности:
  - Вызывается конвертер в соответствии с типом данных значения.
- Если конвертация заканчивается успешно, значение сохраняется в *локальной переменной* компонента.
- Если конвертация заканчивается неудачно, создаётся сообщение об ошибке, которое помещается в `FacesContext`.

ИТМО ВТ

Фаза валидации значений компонентов (Process Validations Phase)

- Вызываются валидаторы, зарегистрированные для компонентов представления.
- Если значение компонента не проходит валидацию, формируется сообщение об ошибке, которое сохраняется в `FacesContext`.

ИТМО ВТ

Фаза обновления значений компонентов (Update Model Values Phase)

- Если данные валидны, то значение компонента обновляется.
- Новое значение присваивается *полю* объекта компонента.

ИТМО ВТ

Фаза вызова приложения (Invoke Application Phase)

- Управление передаётся слушателям событий.
- Формируются новые значения компонентов.

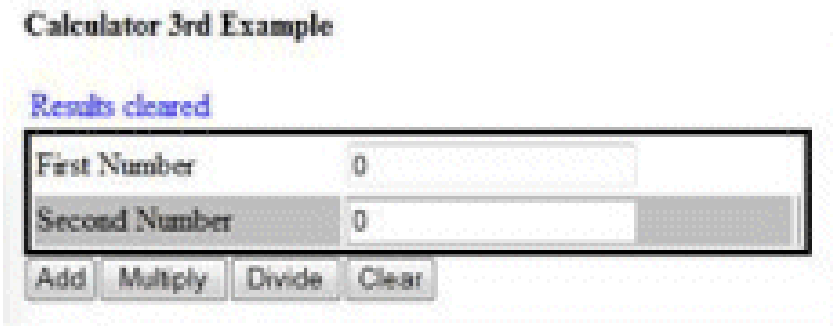
ИТМО ВТ

Фаза формирования ответа сервера (Render Response Phase)

- JSF Runtime обновляет представление в соответствии с результатами обработки запроса.
- Если это первый запрос к странице, то компоненты помещаются в иерархию представления.
- Формируется ответ сервера на запрос.
- На стороне клиента происходит обновление страницы.

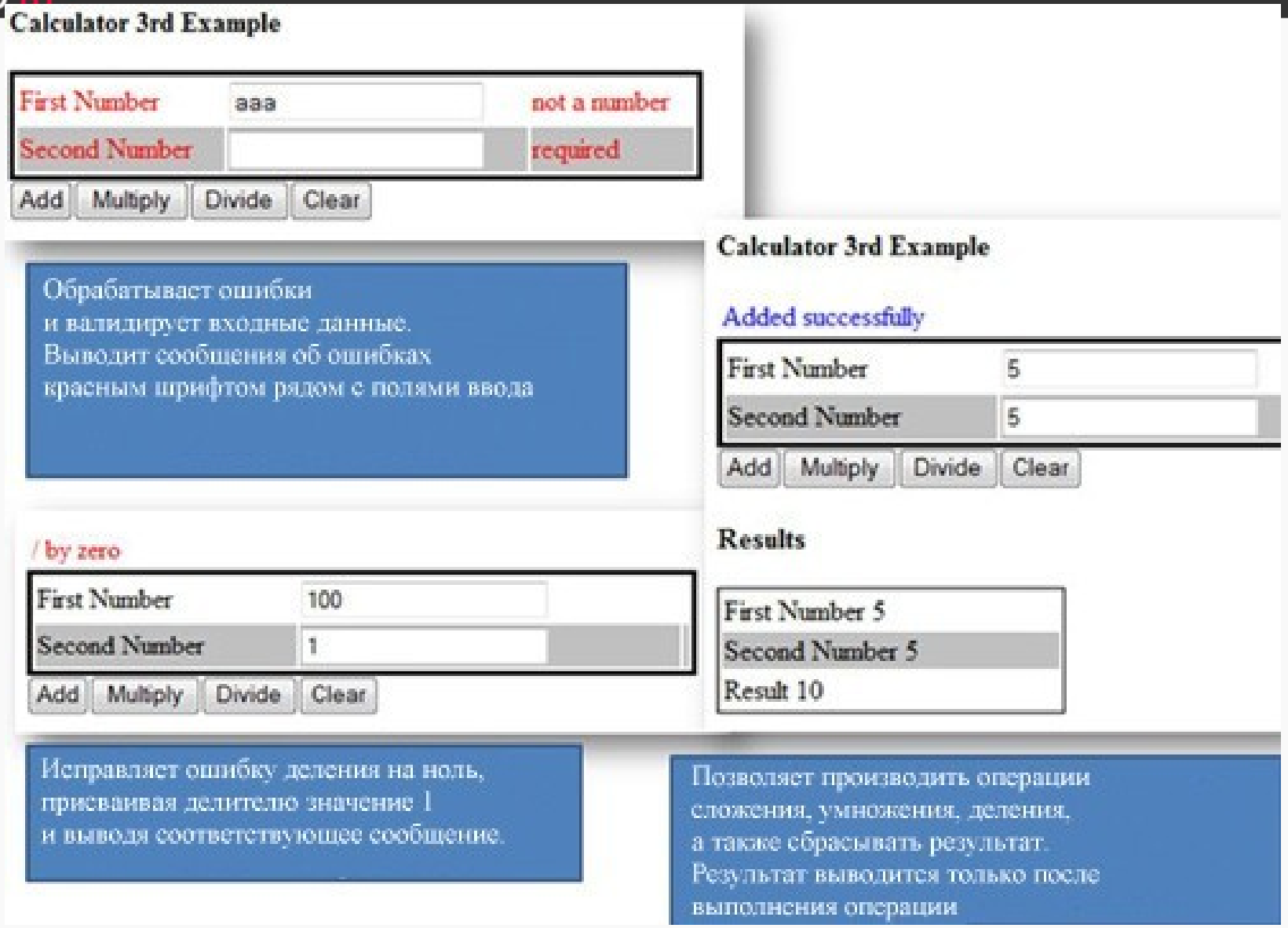
ИТМО ВТ

Пример JSF-приложения

Model	Controller	View
Calculator	Calculator controller	
- JSF-независимый слой - POJO - Бизнес логика	- JSF-зависимый слой - Выбор представления - Управление компонентами - Управление сообщениями	- Никакой логики - Никаких сравнений - Стили + компоненты GUI

Пример JSF-приложения (продолжение)

ИТМО ВТ



ИТМО ВТ

Пример JSF-приложения (продолжение)

Конфигурация web.xml:

```
...
<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>*.jsf</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
...
```

Пример JSF-приложения (продолжение)

```
package org.itmo.sample;

public class Calculator {

    /** Первый операнд */
    private int firstNumber = 0;

    /** Результат операции */
    private int result = 0;

    /** Второй операнд */
    private int secondNumber = 0;

    /** Сложение операндов */
    public void add() {
      result = firstNumber + secondNumber;
    }

    /** Перемножение операндов */
    public void multiply() {
      result = firstNumber * secondNumber;
    }

    /** Сброс результата */
    public void clear() {
      result = 0;
    }
```

Пример JSF-приложения (продолжение)

```
    /* ---------- свойства ------------- */

    public int getFirstNumber() {
      return firstNumber;
    }

    public void setFirstNumber(int firstNumber) {
      this.firstNumber = firstNumber;
    }

    public int getResult() {
      return result;
    }

    public void setResult(int result) {
      this.result = result;
    }

    public int getSecondNumber() {
      return secondNumber;
    }

    public void setSecondNumber(int secondNumber) {
      this.secondNumber = secondNumber;
    }

}
```

Пример JSF-приложения (продолжение)

Конфигурация faces-config.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config 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_1_2.xsd"
    version="1.2">
  <managed-bean>
    <managed-bean-name>calculator</managed-bean-name>
    <managed-bean-class>
      org.itmo.sample.Calculator
    </managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
  </managed-bean>
</faces-config>
```

Пример JSF-приложения (продолжение)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>

<!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">

<head>
  <title>Calculator Application</title>
</head>

<body>
<f:view>
  <h:form id="calcForm">
    <h4>Calculator</h4>
    <table>
      <tr>
        <td><h:outputLabel value="First Number" for="firstNumber" /></td>
        <td><h:inputText id="firstNumber"
          value="#{calculator.firstNumber}" required="true" /></td>
        <td><h:message for="firstNumber" /></td>
      </tr>
```

ИТМО ВТ

Пример JSF-приложения (продолжение)

```
      <tr>
      <td><h:outputLabel value="Second Number"
                         for="secondNumber" />
      </td>
      <td><h:inputText id="secondNumber"
                       value="#{calculator.secondNumber}"
                       required="true" /></td>
      <td><h:message for="secondNumber" /></td>
    </tr>
  </table>

  <div>
    <h:commandButton action="#{calculator.add}"
                     value="Add" />
    <h:commandButton action="#{calculator.multiply}"
                     value="Multiply" />
    <h:commandButton action="#{calculator.clear}"
                     value="Clear" immediate="true"/>
  </div>
</h:form>
```

Пример JSF-приложения (продолжение)

```
    <h:panelGroup rendered="#{calculator.result != 0}">
      <h4>Results</h4>
      <table>
        <tr><td>
          First Number  ${calculator.firstNumber}
        </td></tr>
        <tr><td>
          Second Number ${calculator.secondNumber}
        </td></tr>
        <tr><td>
          Result ${calculator.result}
        </td></tr>
      </table>
    </h:panelGroup>
</f:view>
</body>

</html>
```

5. Архитектура корпоративных приложений

ИТМО ВТ

Отличия корпоративных приложений от «обычных»

- Зависимость бизнеса от их работоспособности.
- Длительный жизненный цикл.
- Постоянные изменения требований.
- Жёсткие сроки внедрения изменений («эта функция нужна нам вчера!»).
- Сложность и разнородность – обычно состоят из многих подсистем.
- Большие команды разработчиков, часто – узкой специализации.

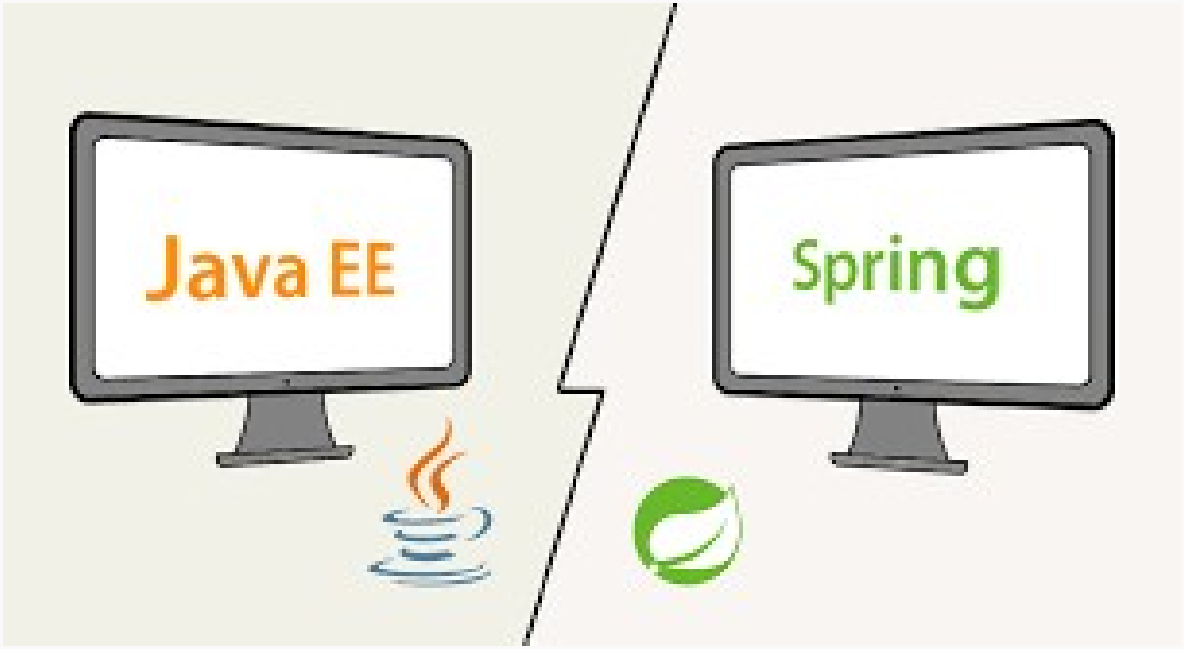
ИТМО ВТ

Требования к архитектуре

- Нужно отделять уровни архитектуры приложения друг от друга.
- Приложение должно строиться из «кубиков» => разные «кубики» смогут разрабатывать разные программисты.
- Нужны высокие масштабируемость и отказоустойчивость => хорошо бы, чтобы «кубики» не взаимодействовали друг с другом напрямую.

Платформы для разработки корпоративных приложений

- Spring: идейно «пляшет» от концепции CDI.
- Java / Jakarta EE: основа – концепции компонентов и контейнеров.



5.1. Java / Jakarta EE Full Profile

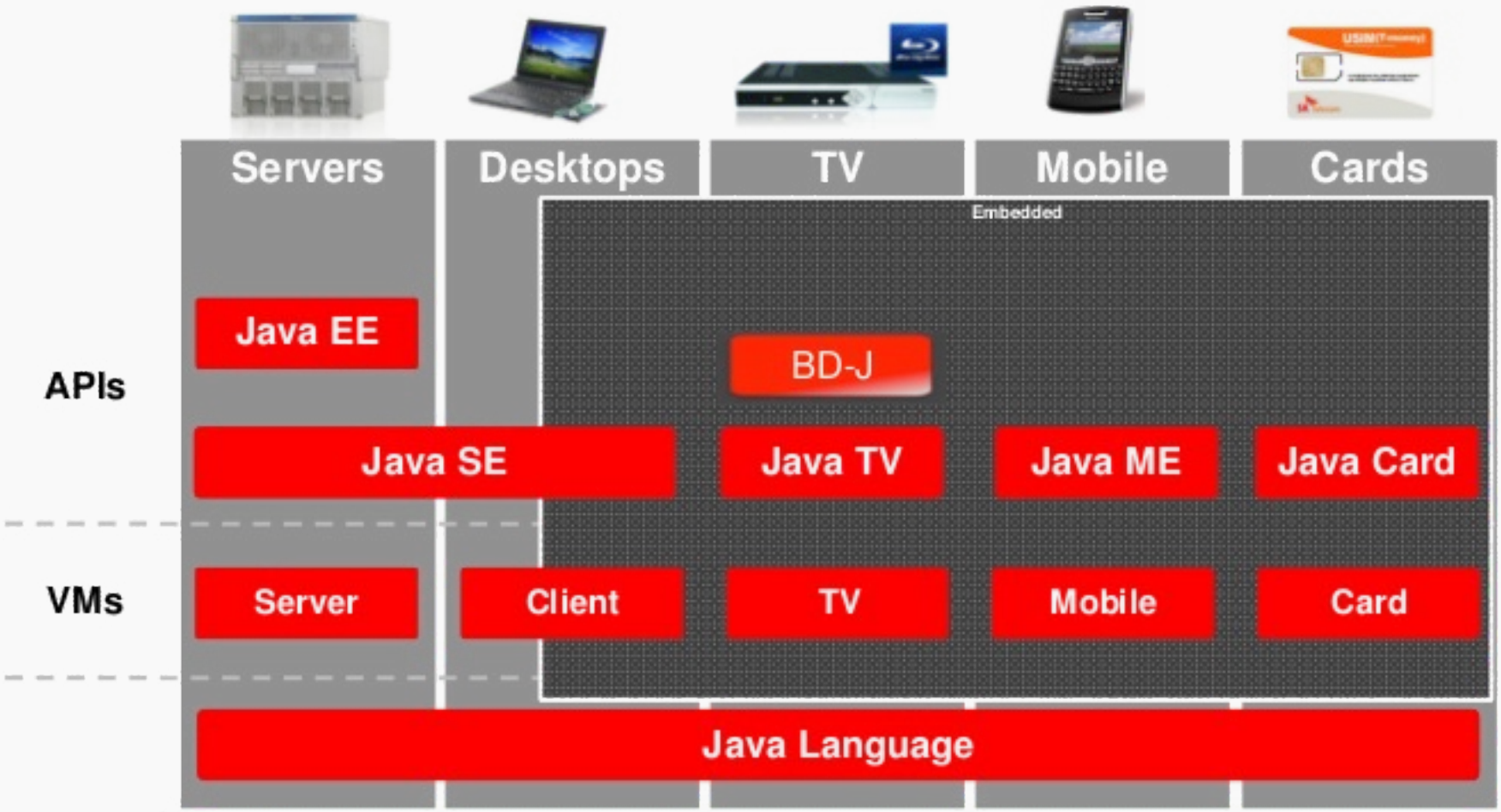
ИТМО ВТ

О платформе Jakarta EE

- «Надстройка» над Java SE – те же самые виртуальная машина и язык программирования, но дополнительные API.
- Первая версия вышла в 1999 г., актуальная (9.0) – в 2019 г.
- Другие названия – J2EE (до версии 1.4), Java EE – до версии 8.0.

ИТМО ВТ

Платформы Java



ИТМО ВТ

Основные концепции

- Приложения строятся из компонентов, работающих под управлением контейнеров.
- Используются следующие принципы:
  - Inversion of Control (IoC) + Contexts & Dependency Injection (CDI).
  - Location Transparency.

ИТМО ВТ

Принципы IoC и CDI

- IoC (применительно к Java EE):
  - Жизненным циклом компонента управляет контейнер (а не программист).
  - За взаимодействие между компонентами отвечает тоже контейнер.
- CDI — позволяет снизить (или совсем убрать) зависимость компонента от контейнера:
  - Не требуется реализации каких-либо интерфейсов.
  - Не нужны прямые вызовы API.
  - Реализуется через аннотации.

Пример CDI (JSF Managed Bean)

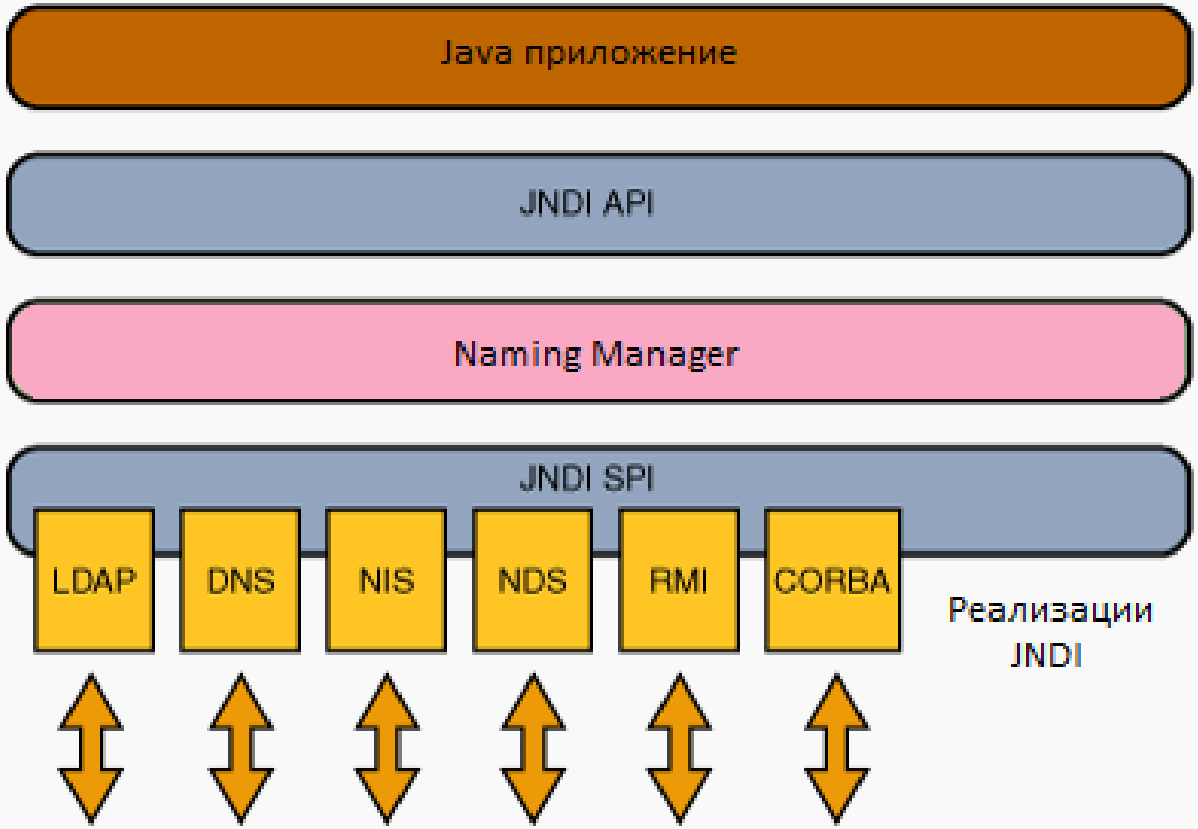
```
@ManagedBean(name="message")
@SessionScoped
public class MessageBean implements Serializable {
   //business logic and whatever methods
}
```

```
@ManagedBean
@SessionScoped
public class HelloBean implements Serializable {

   @ManagedProperty(value="#{message}")
   private MessageBean messageBean;

   //must povide the setter method
   public void setMessageBean(MessageBean messageBean) {
       this.messageBean = messageBean;
   }
}
```

ИТМО ВТ

Java Naming & Directory Interface (JNDI)

JNDI — это набор Java API, организованный в виде службы каталогов, который позволяет Java-клиентам открывать и просматривать данные и объекты по их именам (C) Wikipedia.



ИТМО ВТ

JNDI API

Два варианта использования JNDI:

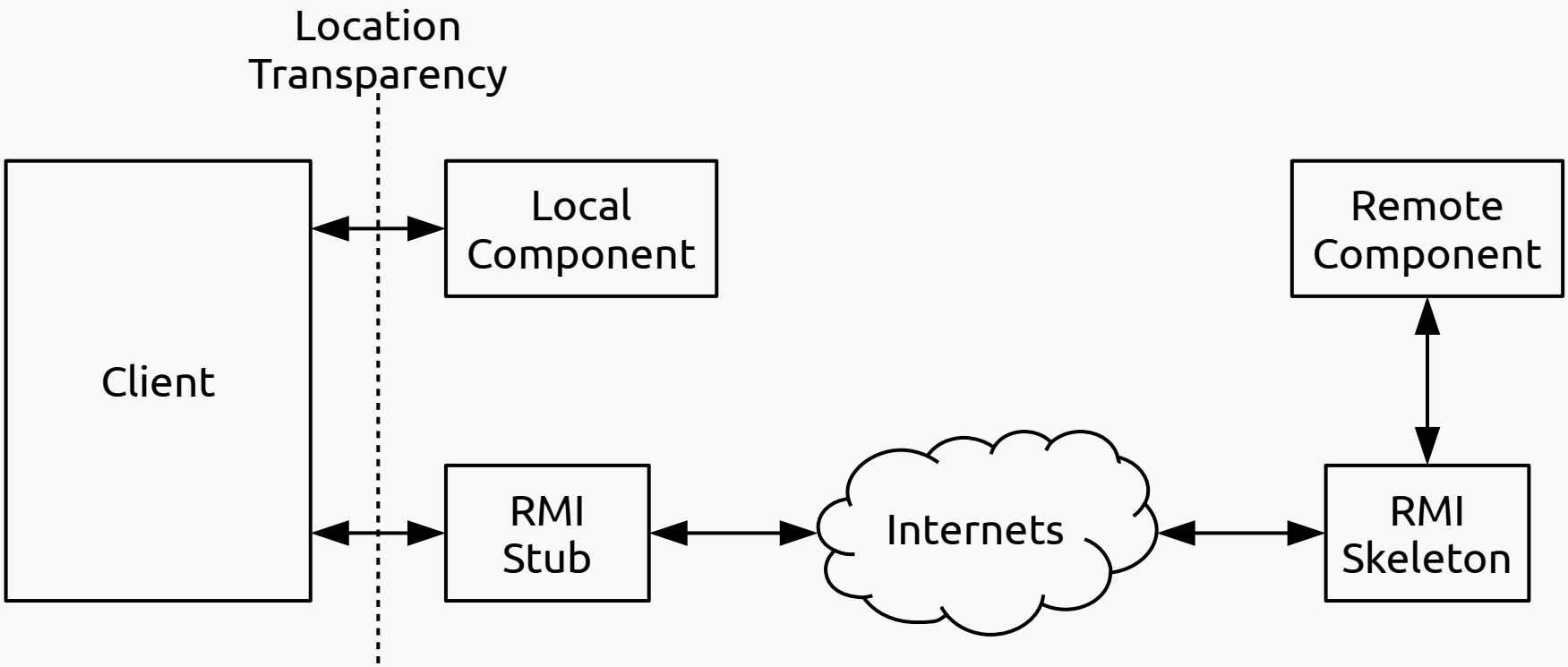
- CDI (аннотации) — работает только в managed компонентах.
- Прямой вызов API — работает везде.

```
// Пример получения ссылки на JDBC datasource.
DataSource dataSource = null;
try {
   // Инициализируем контекст по умолчанию.
   Context context = new InitialContext();
   dataSource =
      (DataSource) context.lookup("Database");
} catch (NamingException e) {
    // Ссылка не найдена
}
```

ИТМО ВТ

Принцип Location Transparency

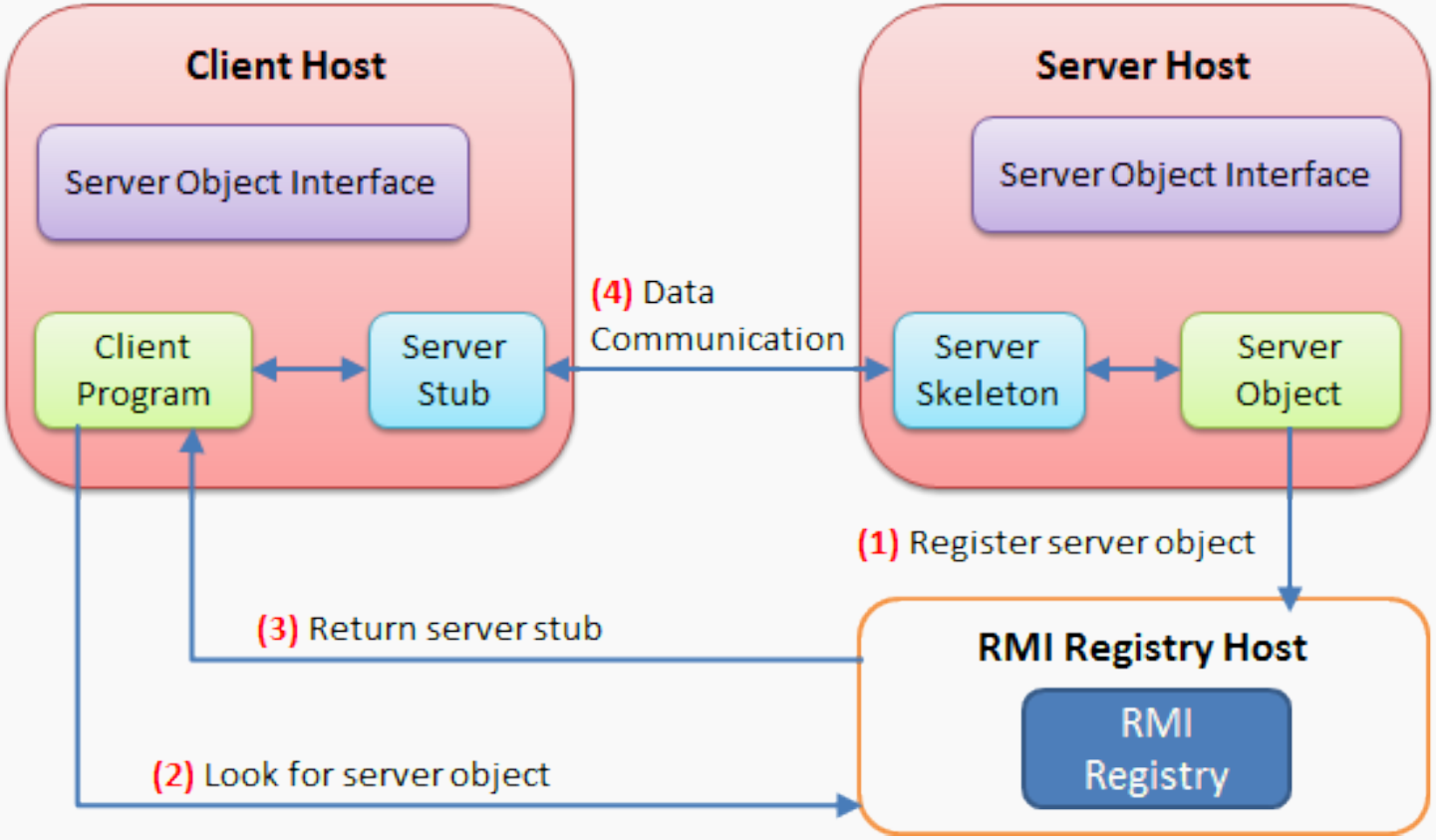
Благодаря CDI не важно, где физически расположен вызываемый компонент — за его вызов отвечает контейнер.



ИТМО ВТ

Remote Method Invocation

RMI — Java API, позволяющий вызывать методы удалённых объектов.



ИТМО ВТ

Особенности RMI

- В общем случае, объекты передаются по значению (копии).
- Передаваемые объекты должны быть Serializable.

Пример реализации RMI — сервер

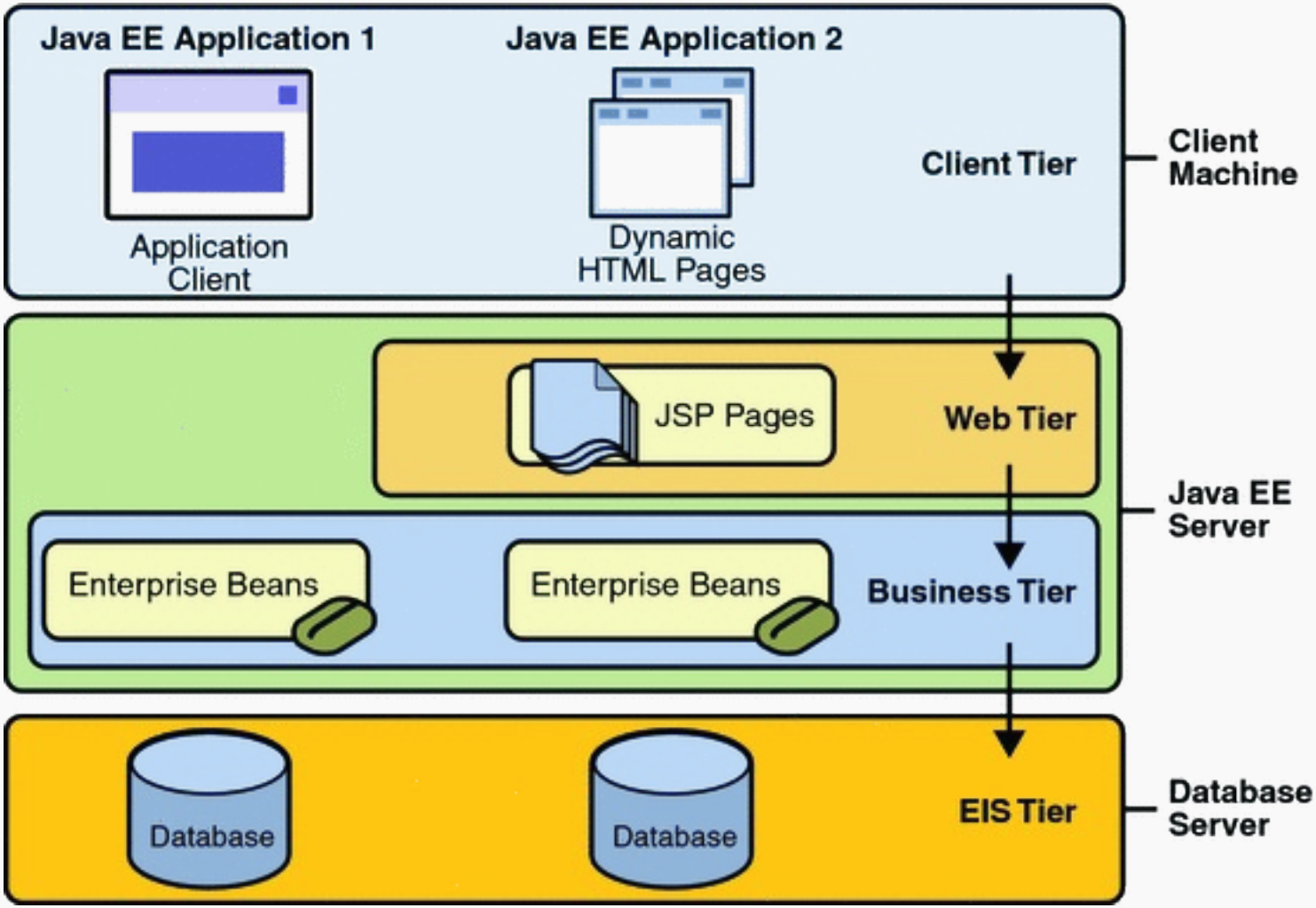
```
public class PrimeNumbersSearchServer implements ClientRegister {

   ...

   public static void main(String[] args) {
      PrimeNumbersSearchServer server =
                              new PrimeNumbersSearchServer();
      try {
         ClientRegister stub = (ClientRegister)
                        UnicastRemoteObject.exportObject(server, 0);
         Registry registry = LocateRegistry.createRegistry(12345);
         registry.bind("ClientRegister", stub);
         server.startSearch();
      } catch (Exception e) {
         System.out.println ("Error occured: " + e.getMessage());
         System.exit (1);
      }
   }
}
```

Пример реализации RMI — клиент

```
public class PrimeNumbersSearchClient implements PrimeChecker {

   ...

   public static void main(String[] args) {
      PrimeNumbersSearchClient client =
                                 new PrimeNumbersSearchClient();

      try {
         Registry registry = LocateRegistry.getRegistry(null, 12345);
         ClientRegister server =
                  (ClientRegister) registry.lookup("ClientRegister");
         PrimeChecker stub = (PrimeChecker)
                        UnicastRemoteObject.exportObject(client, 0);
         server.register(stub);
      } catch (Exception e) {
         System.out.println ("Error occured: " + e.getMessage());
         System.exit (1);
      }
   }
}
```

ИТМО ВТ

Java EE Application Tiers



ИТМО ВТ

Профили платформы Java EE

- Появились в Java EE 6.
- Позволяют сделать более «лёгкими» приложения, которым не нужен полный стек технологий Java EE.
- Существует только 2 профиля — Full и Web.
- Сервер приложений может реализовывать спецификации не всей платформы, а конкретного профиля.

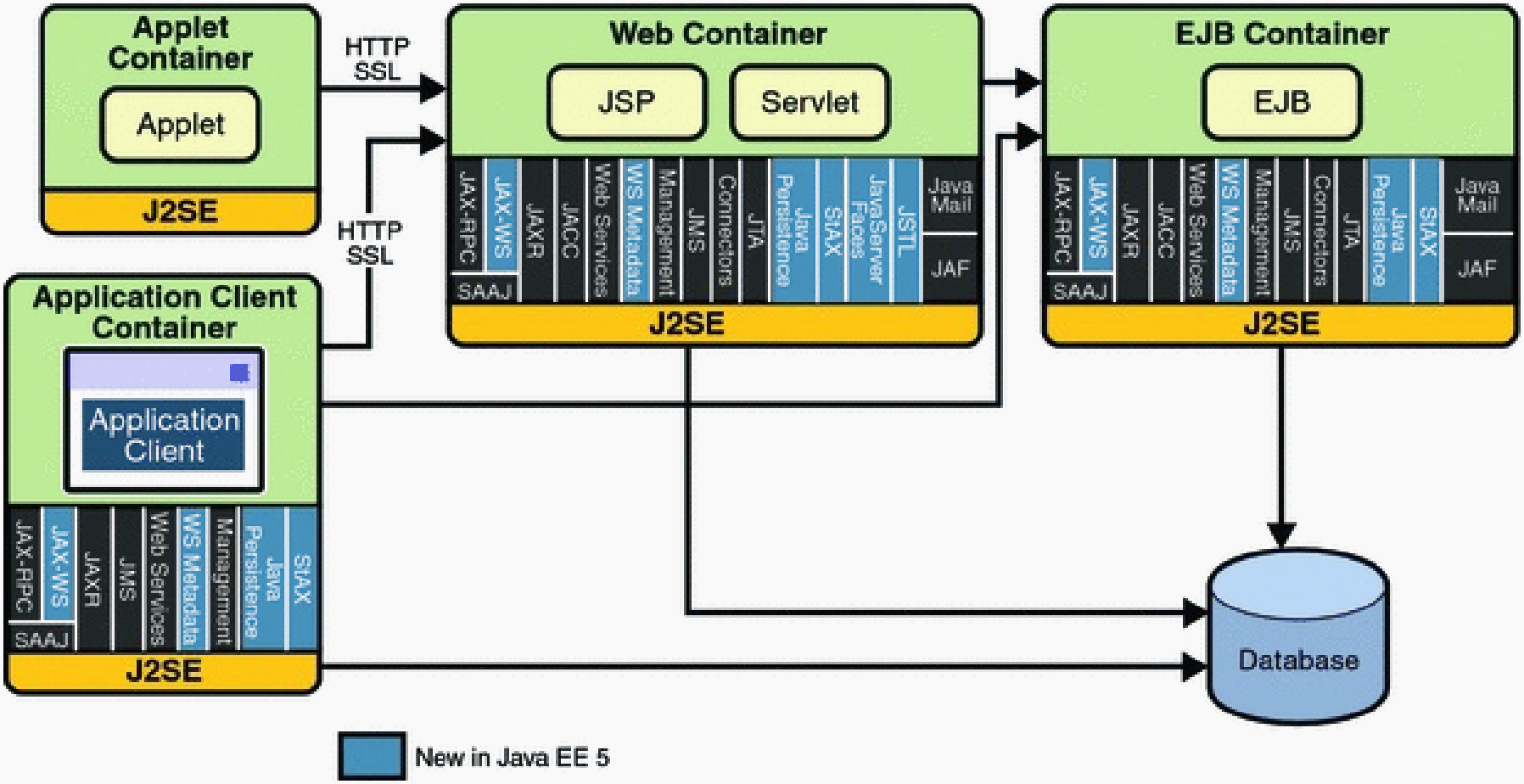
ИТМО ВТ

Java EE Full & Web Profiles

API	Web Profile	Full Profile		API	Web Profile	Full Profile
Servlet	+	+		JTA	+	+
JSP	+	+		JMS		+
JSTL	+	+		JavaMail		+
EL	+	+		JAX-WS		+
JSF	+	+		JAX-RS		+
CDI	+	+		JAXB		+
EJB Lite	+	+		JACC		+
EJB Full		+		JCA		+
JPA	+	+				

ИТМО ВТ

Архитектура приложения Java EE Full Profile



5.1.1. CDI Beans «по-серьёзному»

CDI Beans

- Максимально абстрактная реализация паттерна CDI в Java / Jakarta EE.
- Появились в Java EE 7, «клонируют» бины из Spring.
- Могут использоваться со всеми фреймворками Java / Jakarta EE.
- Конфигурируются аннотациями, основной пакет – `javax.enterprise.context`.
- Для CDI используется универсальная аннотация `@Inject`.
- В отличие от EJB, не обеспечивают горизонтальную масштабируемость «сами по себе» (но это часто и не надо!).

ИТМО ВТ

Пример CDI-бина.

```
package demos;
import javax.enterprise.context.*;

@RequestScoped
public class MyBean {
   public void doSomething() {
      //…
   }
}
```

ИТМО ВТ

Пример инъекции CDI-бина.

ProductOrder bean definition with "Product" property, "placeOrder" operation, and optional alias "order":

```
package demos;
@Named("order")
@RequestScoped
public class ProductOrder {
  private String productName;
  public String getProduct() {
    return productName;
  }
  public void setProduct(String name) {
    productName = name;
  }
  public String placeOrder() {
    //...
  }
}
```

Injection of the ProductOrder bean into JSF page using "order" alias:

```
<h:form>
 <h:inputText id="name" value="#{order.product}"/>
 <h:commandButton value="Ok" action="#{order.placeOrder}"/>
</h:form>
```

Injection of the ProductOrder bean into OrderManagement class:

```
package demos;
public class OrderManagement {
  @Inject;
  private ProductOrder productOrder;
  public void handleOrder() {
    String name = productOrder.getProduct();
    productOrder.placeOrder();
  }
}
```

ИТМО ВТ

CDI Bean Scope

- Аналог контекста (scope) управляемых бинов JSF.
- Определяет жизненный цикл бинов и их видимость друг для друга.
- Задаётся аннотацией.
- В спецификации определены 5 уровней контекста:
  - `@RequestScoped`.
  - `@SessionScoped`.
  - `@ApplicationScoped`.
  - `@ViewScoped` – из JSF, но работает и для CDI-бинов.
  - `@ConversationScoped`.
  - `@Dependent`.

ИТМО ВТ

Conversation Scope

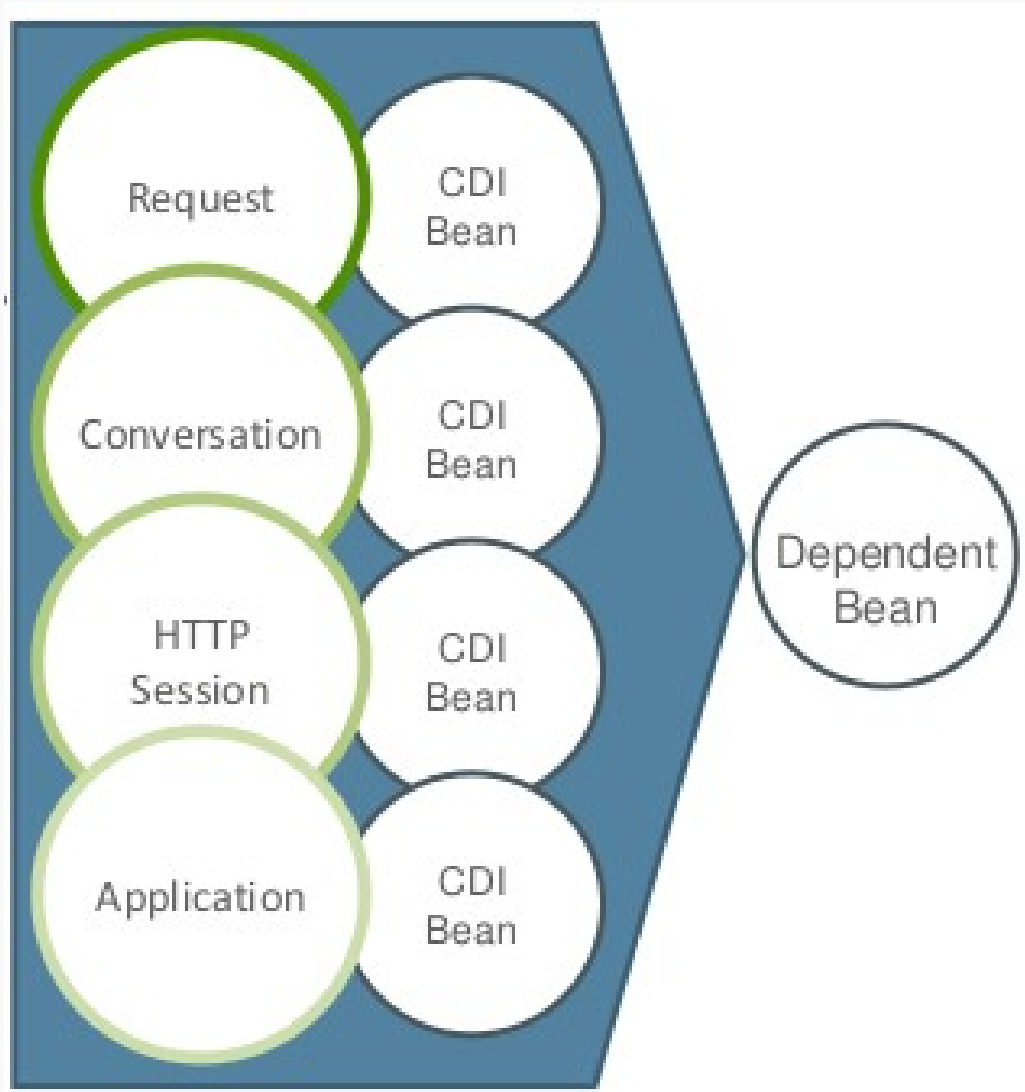
- Идейно похож на `@CustomScoped` из JSF – жизненным циклом компонента управляет программист.
- Управление осуществляется через инъекцию объекта `javax.enterprise.context.Conversation`.
- Пример использования:

```
@ConversationScoped
public class MyBean implements Serializable {

   @Inject
   private Conversation conversation;

   public void startConversation() {
         conversation.begin();
   }

   public void endConversation() {
       conversation.end();
   }
}
```

ИТМО ВТ

Dependent Scope

- Используется по умолчанию, может быть указан «вручную» аннотацией @Dependent.
- Идейно похож на @NoneScoped из JSF – жизненный циклом компонента определяется тем, где он был использован.



Именование бинов

Используется аннотация @Named.

```
@Named("ca")
@RequestScoped
public class CheckingAccount implements Account {...}

@Named("sa")
@RequestScoped
public class SavingsAccount implements Account {...}

@WebServlet
public class AccountServlet extends HttpServlet {
    @Inject
    private @Named("ca") Account account; // Реализацию определяет @Named
}

public class AccountApplication {
    @Inject
    private @Named("sa") Account account; // И здесь тоже
}

<!DOCTYPE html ...>
<html xmlns:h="http://java.sun.com/jsf/html">
    <h:form>
        <h:inputText value="#{ca.id}">
    </h:form>
</html>
```

ИТМО ВТ

@Alternative

Позволяет определять конкретную реализацию «на лету».

```
@Alternative
@RequestScoped
public class OnlineOrder implements Order {...}

@Alternative
@RequestScoped
public class StoreOrder implements Order {...}

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee "
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/beans_1_1.xsd "
    version="1.1" bean-discovery-mode="all">
    <alternatives>
        <class>demos.OnlineOrder</class>
    </alternatives>
</beans>

@WebServlet
public class OrderServlet extends HttpServlet {
    @Inject
    private Order order; // Реализация задаётся в beans.xml
}
```

ИТМО ВТ

«Фабрики» (Producer & Disposer Methods)

- Бины, которые управляют созданием новых экземпляров других бинов.
- Используют аннотации `@Produces` и `@Disposes`.
- Позволяют использовать в качестве CDI-бинов классы, не соответствующие паттерну Java Beans (например, имеющие только конструкторы с параметрами).

ИТМО ВТ

Пример фабрики

```
@RequestScoped
public class OrderFactory {
    @Produces
    @OrderTypeQualifier(OrderType.ONLINE)
    public Order getOnlineOrder() {
        return new OnlineOrder(...);
    }
    @Produces
    @OrderTypeQualifier(OrderType.STORE)
    public Order getStoreOrder() {
        return new StoreOrder(...);
    }
    public void disposeOrder(@Disposes
        @OrderTypeQualifier Order order){
        order.close();
    }
}
```

ИТМО ВТ

Перехватчики (Interceptors)

- Классы, реагирующие на определённые события жизненного цикла бинов.
- Чем-то похожи на фильтры запросов в сервлетах.
- Конкретный тип события задаётся аннотацией:
  - `@AroundInvoke` (используется чаще всего);
  - `@PostConstruct`;
  - `@PreDestroy`;
  - `@PrePassivate`;
  - `@PostActivate`.

ИТМО ВТ

Пример перехватчика

```
// Создаём свою аннотацию
@Inherited
@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD,TYPE})
public @interface Logging {}

// Реализуем логику перехвата событий заданного типа
@Logging
@Interceptor
@Priority(Interceptor.Priority.APPLICATION)
public class OrderInterceptor {
    @AroundInvoke // Перехватываем вызов метода
    public Object writeLog(InvocationContext ctx) throws Exception {
        logger.log(...);
        Object obj = ctx.proceed(); // Передаём управление в метод
        logger.log(...);
        return obj;
    }
}

// Используем перехватчик для записи в лог
@RequestScoped
public class OnlineOrder{
    @Logging
    public void addProduct(Product p){...}
}
```

ИТМО ВТ

События (Events)

- В спецификации CDI-beans описана гибкая модель работы с событиями.
- Каждый бин может определять свои дополнительные типы событий.
- Каждый бин может быть источником событий любых типов.
- Обработкой событий занимаются специальные методы-наблюдатели (аннотация `@Observes`).

ИТМО ВТ

Пример работы с событиями

```
// Бин-источник события
@RequestScoped
public class Order {
    @Inject Event<Product> addEvent;
    public void addProduct(Product p){
        ...
        addEvent.fire(p);
    }
}

// Бин-обработчик события
@RequestScoped
public class StockManagement {
    public void reserveProduct(
        @Observes Product p){...}
}
```

ИТМО ВТ

Стереотипы (Stereotypes)

- Аннотации, включающие в себя другие аннотации.
- Используются в больших и сложных приложениях, где есть много бинов, выполняющих схожие действия.
- Стереотип может задавать:
  - Контекст (scope) по-умолчанию.
  - Любое число назначенных перехватчиков.
  - Опционально – аннотацию `@Named`, специфицирующую имя, под которым бин будет виден из EL.
  - Опционально – аннотацию `@Alternative`, специфицирующую то, что бины внутри этого стереотипа являются альтернативами друг другу.

Пример использования стереотипа

```
// Задаём стереотип
@Alternative
@Stereotype
@Target(TYPE)
@Retention(RUNTIME)
@RequestScoped
public @interface Custom {}

// Используем стереотип
@Custom
public class Order {}
```

5.1.2. Обзор технологий Java / Jakarta EE

ИТМО ВТ

Enterprise Java Beans

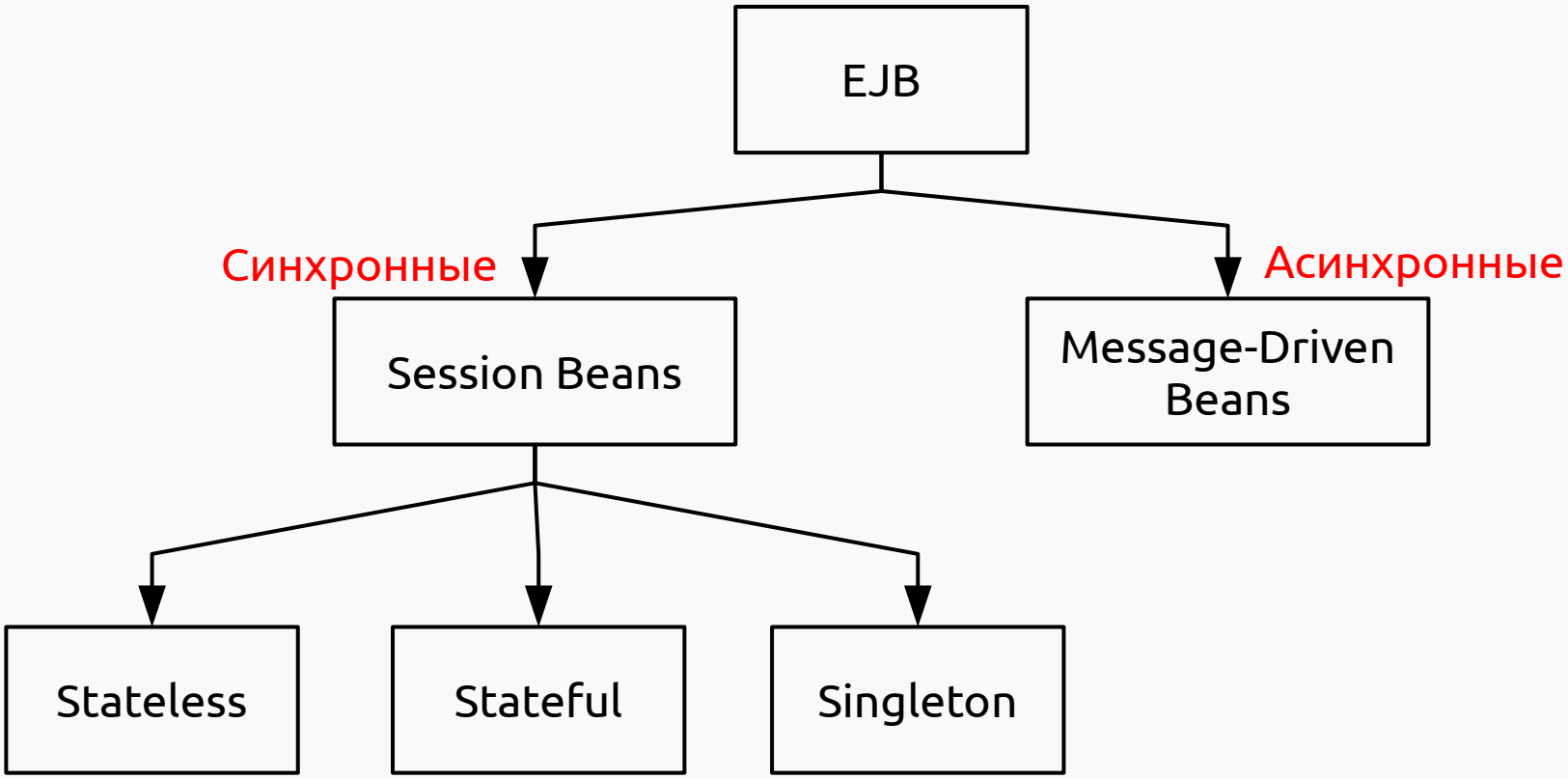
EJB — технология разработки серверных компонентов, реализующих бизнес-логику.

Особенности EJB:

- Возможность локального и удалённого доступа.
- Возможность доступа через JNDI или Dependency Injection.
- Поддержка распределённых транзакций (с помощью JTA).
- Поддержка событий.
- Жизненным циклом управляет EJB-контейнер (в составе сервера приложений).

ИТМО ВТ

Виды EJB



ИТМО ВТ

Веб-сервисы

- Позволяют организовать взаимодействие между сетевыми ресурсами по стандартизированному протоколу.
- Ресурсы могут работать на любой платформе.
- Данные «упаковываются» в XML и передаются по HTTP.
- Основные стандарты — SOAP, WSDL и WS-I.
- На платформе Java EE реализуются JAX-WS API и JAX-RS API.
- Основа SOA (Service Oriented Architecture).

ИТМО ВТ

Пример реализации веб-сервиса (JAX-WS)

```
package example;

import javax.jws.*;

@WebService
public class SayHello {

  @WebMethod
  public String getGreeting(String name){
    return "Hello " + name;
  }

}
```

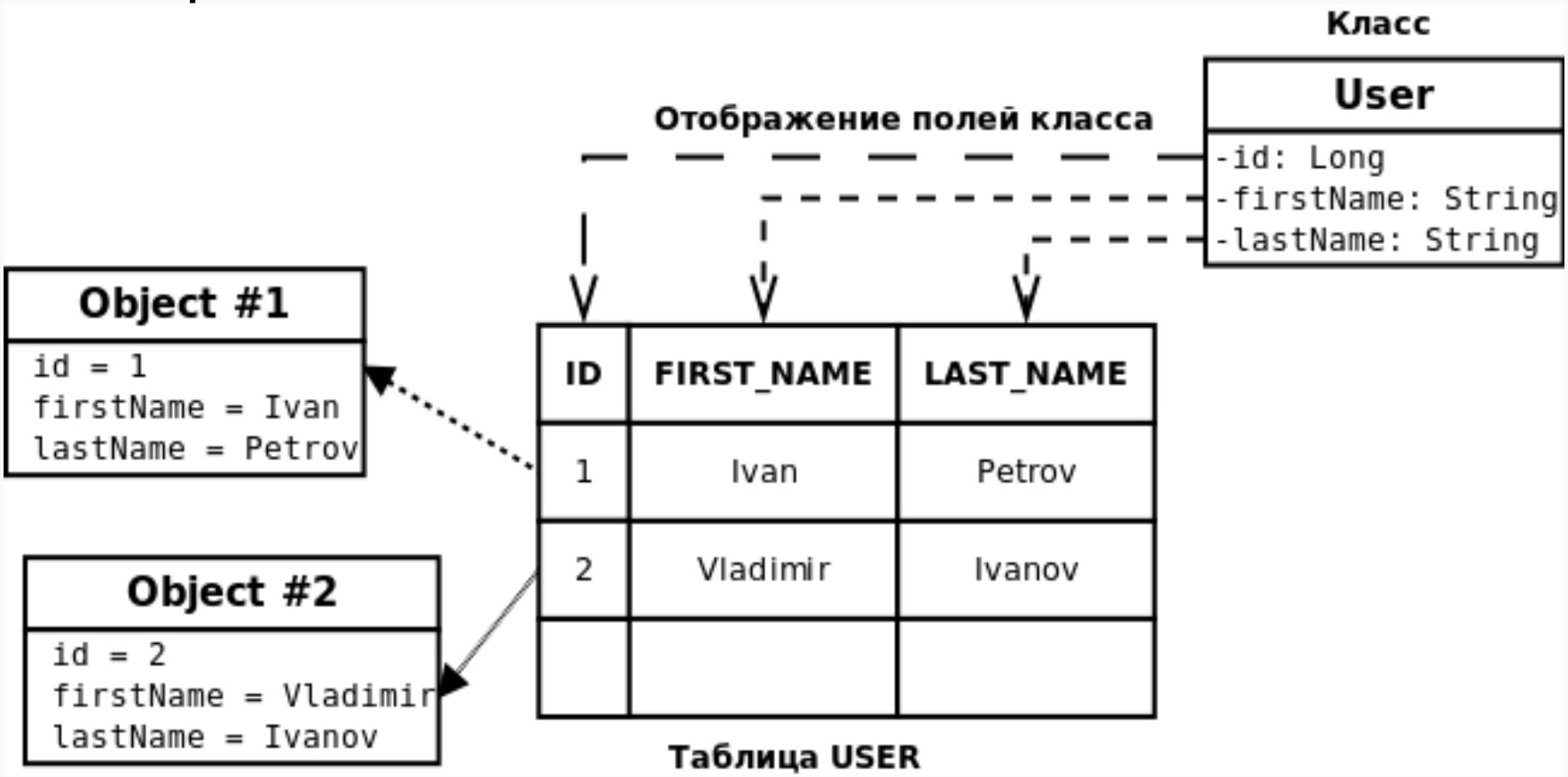
ИТМО ВТ

Пример реализации клиента веб-сервиса (JAX-WS)

```
import javax.xml.ws.WebServiceRef;

public class WSTest {

  public WSTest() { }

  public static void main(String[] args) {
    SayHelloService service = new SayHelloService();
    SayHello port = service.getSayHelloPort();
    System.out.println(port.sayHello(“Duke”));
  }
}
```

Объектно-реляционное отображение

ORM — Object/Relational Mapping — преобразование данных из объектной формы в реляционную и наоборот.



ИТМО ВТ

Как реализовать ORM на Java?

- JDBC;
- ORM-фреймворки (Hibernate, TopLink, …);
- Java Persistense API (JPA 2.0).

ИТМО ВТ

Hibernate ORM

ORM-фреймворк от Red Hat, разрабатывается с 2001 г.

Ключевые особенности:

- Таблицы БД описываются в XML-файле, либо с помощью аннотаций.
- 2 способа написания запросов — HQL и Criteria API.
- Есть возможность написания native SQL запросов.
- Есть возможность интеграции с Apache Lucene для полнотекстового поиска по БД (Hibernate Search).

ИТМО ВТ

EclipseLink ORM

ORM-фреймворк от Eclipse Foundation.

Ключевые особенности:

- Основан на кодовой базе Oracle TopLink.
- Является эталонной реализацией (reference implementation) для JPA.

ИТМО ВТ

Java Persistence API (JPA)

Java-стандарт (JSR 220, JSR 317), который определяет:

- как Java-объекты хранятся в базе;
- API для работы с хранимыми Java-объектами;
- язык запросов (JPQL);
- возможности использования в различных окружениях.

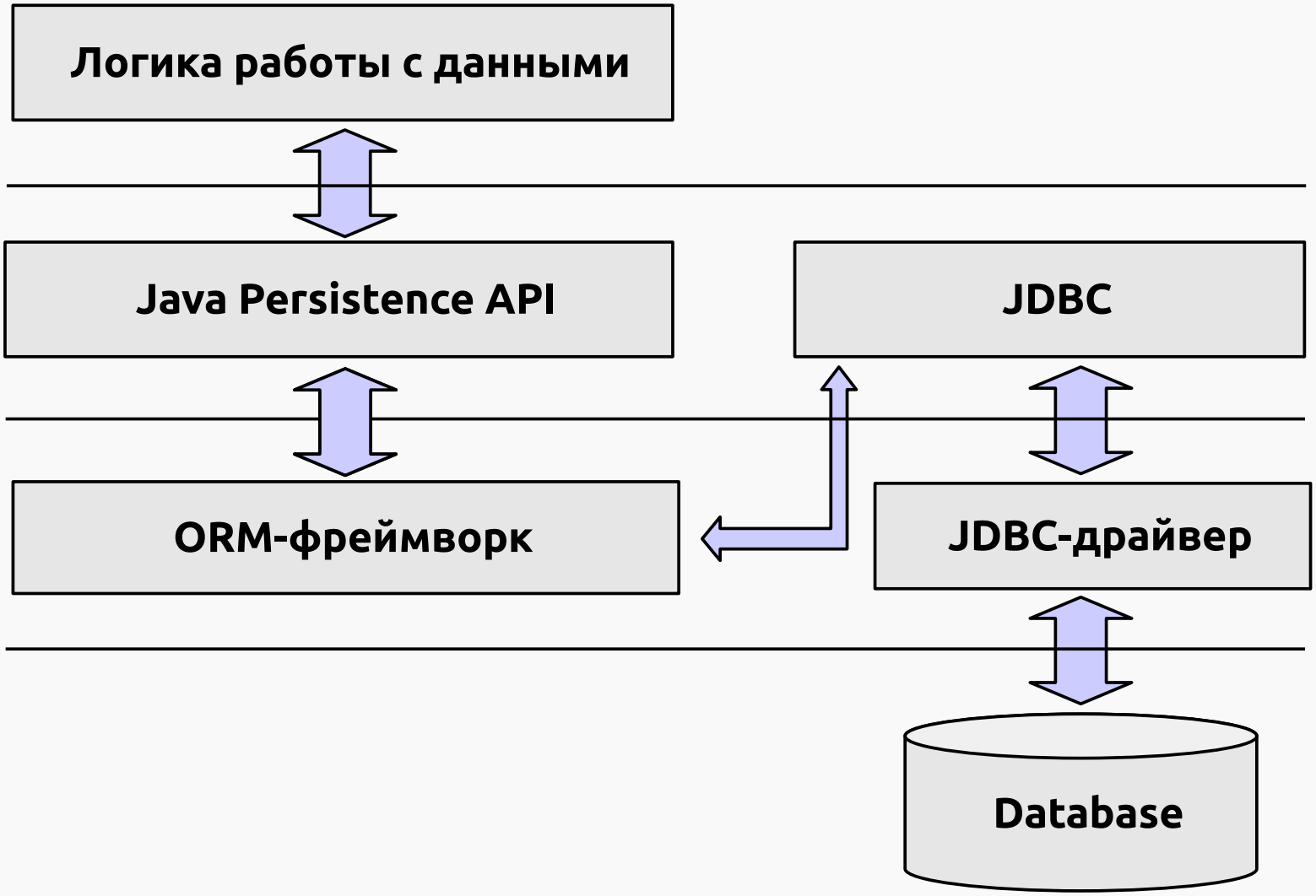
ИТМО ВТ

Что даёт использование JPA?

- Достижение лучшей переносимости.
- Упрощение кода.
- Сокращение времени разработки.
- Независимость от ORM-фреймворков.

ИТМО ВТ

Взаимодействие приложения с БД через JPA



5.2. Spring Framework

ИТМО ВТ

Что это такое

- Универсальный фреймворк для разработки приложений на Java (не только «кровавый энтерпрайз!»).
- Открытый исходный код, первая версия вышла в 2003 г.
- Реализует паттерн IoC и механизмы CDI.
- Активно использует инфраструктурные решения Java / Jakarta EE.
- «Фреймворк фреймворков».

spring®

ИТМО ВТ

«Идейные» отличия от Java EE

- «Базовая» концепция Java EE – разделение обязанностей между контейнером и компонентом; «базовая» концепция Spring – IoC / CDI.
- Контейнер в Java EE включает в себя приложение; приложение в Spring включает в себя контейнер.
- Java EE – спецификация; Spring – фреймворк.

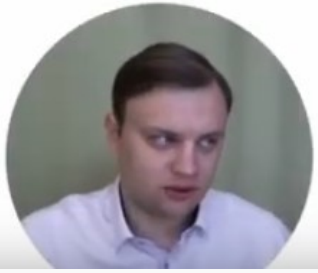
5.2.1. Архитектура Spring

ИТМО ВТ

Архитектура Spring

ITMO University

Introduction to Spring Framework



Pismak Alexey

5.2.2. Spring Web MVC

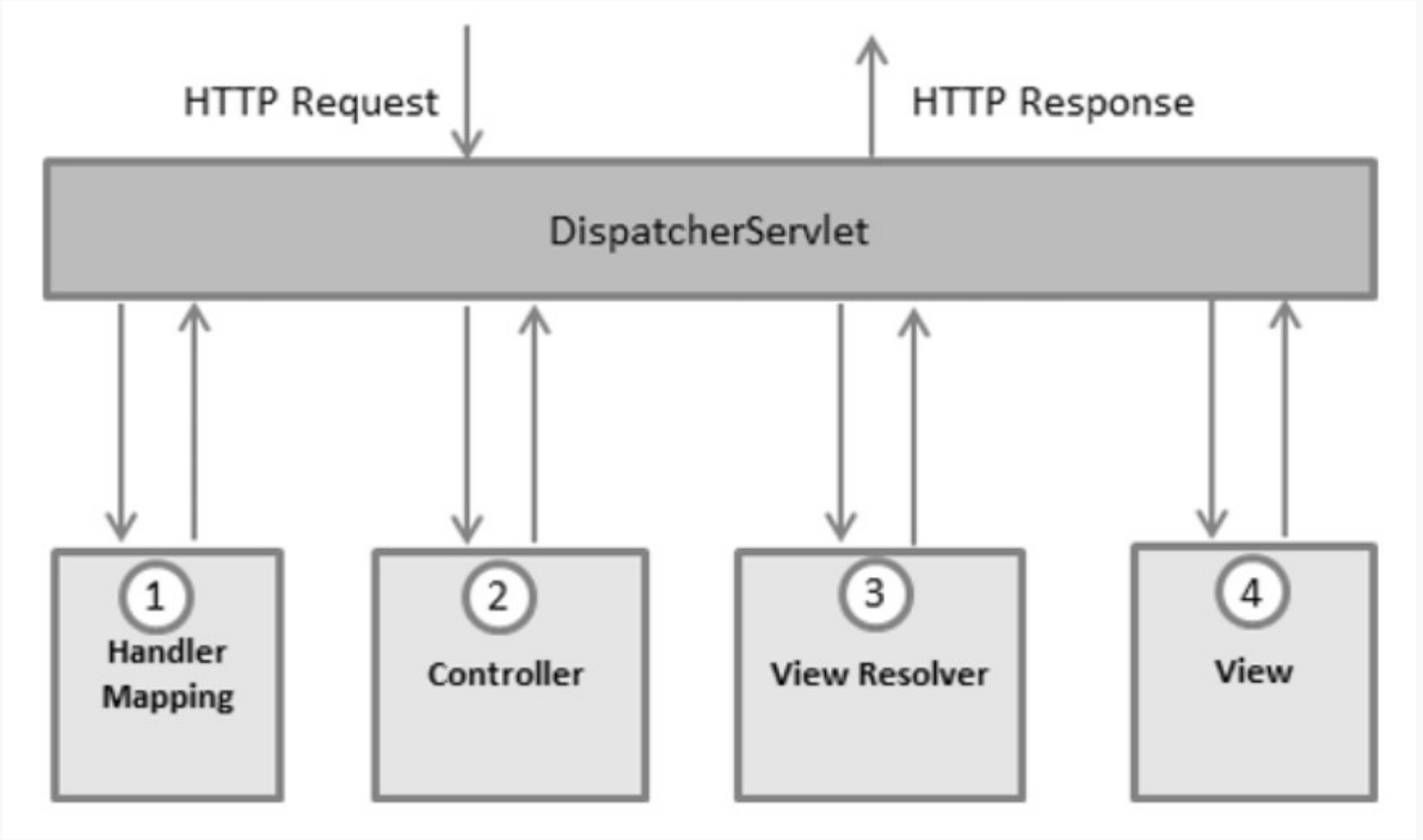
ИТМО ВТ

Общие моменты

- Spring Web MVC – “базовый” фреймворк в составе Spring для разработки веб-приложений.
- Основан на паттерне MVC (внезапно!)
- Back-end; универсальный, удобен для разработки REST API.
- На клиентской стороне интегрируется с популярными JS-фреймворками.
- Удобно интегрируется с Thymeleaf.

ИТМО ВТ

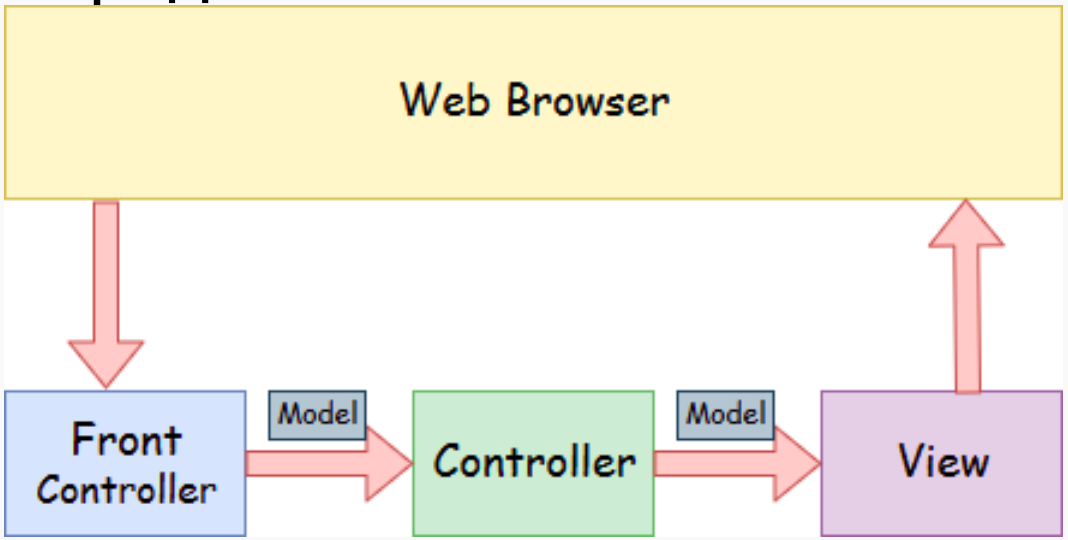
Архитектура Spring Web MVC



ИТМО ВТ

Из чего состоит приложение

- **Model** – инкапсулирует данные приложения (состоят из POJO или бинов).
- **View** – отвечает за отображение данных модели.
- **Controller** – обрабатывает запрос пользователя, создаёт соответствующую модель и передаёт её для отображения в представление.



ИТМО ВТ

Модель

- Хранит данные, необходимые для формирования представления.
- Сами по себе эти данные – обычные POJO.
- В общем случае, реализует интерфейс `org.springframework.ui.Model`.
- Есть «упрощённая» реализация, представляющая из себя Map – `org.springframework.ui.ModelMap`.

ИТМО ВТ

Работа с моделью

Model:

```
@GetMapping("/showViewPage")
public String passParametersWithModel(Model model) {
    Map<String, String> map = new HashMap<>();
    map.put("spring", "mvc");
    model.addAttribute("message", "Hello, World!");
    model.mergeAttributes(map);
    return "viewPage";
}
```

ModelMap:

```
@GetMapping("/printViewPage")
public String passParametersWithModelMap(ModelMap map) {
    map.addAttribute("welcomeMessage", "welcome");
    map.addAttribute("message", "Hello, World!");
    return "viewPage";
}
```

ИТМО ВТ

Контроллер

- Класс, который связывает модель с представлением, управляет состоянием модели.
- Помечается аннотацией `@Controller`.
- Класс или его методы могут быть помечены аннотациями, «привязывающими» их к определённым методам HTTP или URL.

ИТМО ВТ

Пример контроллера

```
@Controller
public class HelloController {
   @RequestMapping(value = "/hello",
            method = RequestMethod.GET)
   public String printHello(ModelMap
                                  model) {
      model.addAttribute("message",
          "Hello Spring MVC Framework!");
      return "hello";
   }
}
```

ИТМО ВТ

Представление

- Фреймворк не специфицирует жёстко технологию, на которой должно быть построено представление.
- Вариант «по-умолчанию» – JSP.
- Можно использовать Thymeleaf, FreeMarker, Velocity etc.
- Можно реализовать представление вне контекста Spring – целиком на JS.

ИТМО ВТ

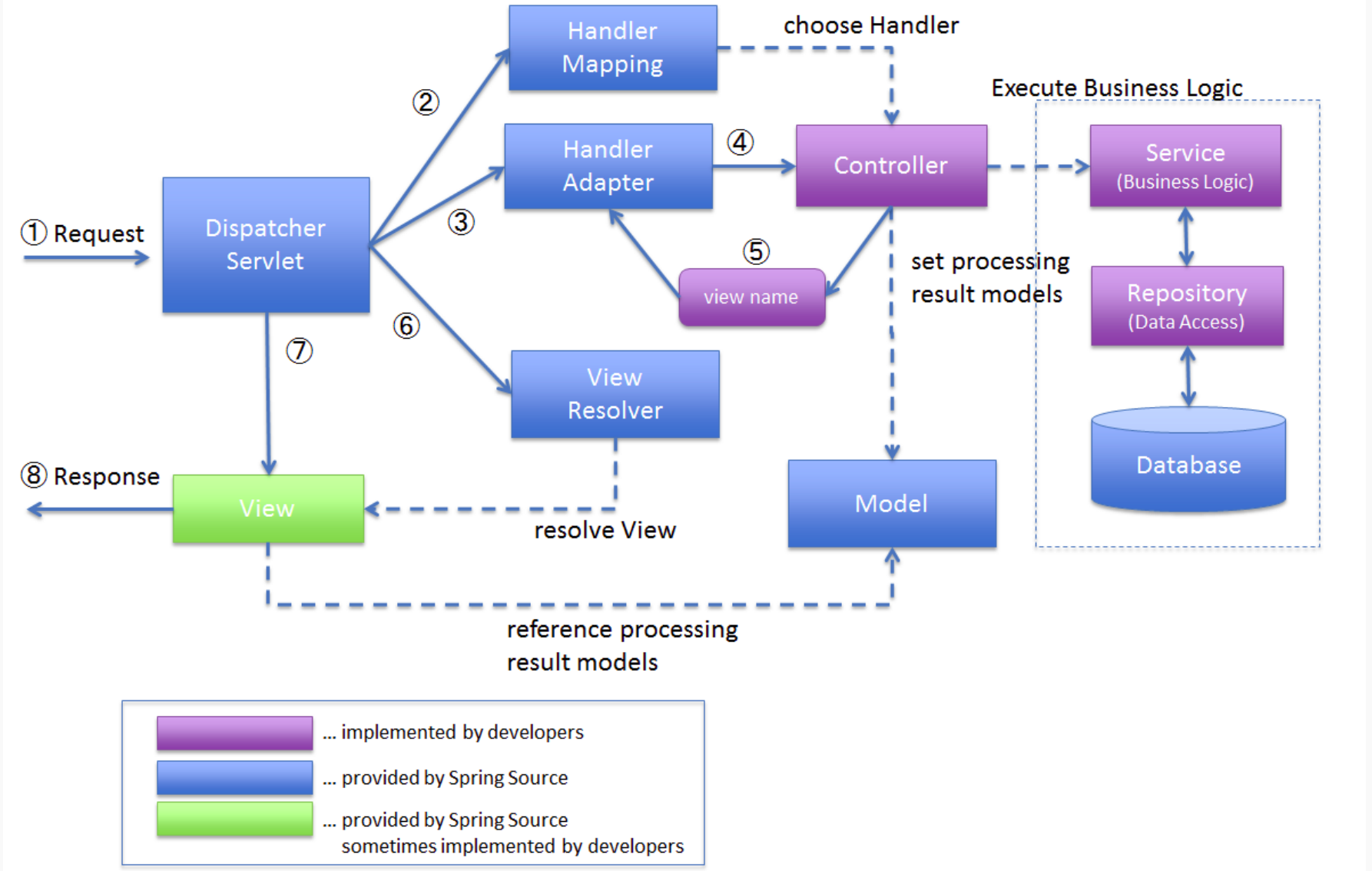
Пример представления

На JSP:

```
<html>
    <head>
        <title>Hello Spring MVC</title>
    </head>
    <body>
        <h2>${message}</h2>
    </body>
</html>
```

ИТМО ВТ

Обработка запроса



ИТМО ВТ

Dispatcher Servlet

- Обрабатывает все запросы и формирует ответы на них.
- Связывает между собой все элементы архитектуры Spring MVC.
- Обычный сервлет – конфигурируется в `web.xml`.

ИТМО ВТ

Конфигурация web.xml

```
<web-app id = "WebApp_ID" version = "2.4"
   xmlns = "http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://java.sun.com/xml/ns/j2ee
   http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Spring MVC Application</display-name>

   <servlet>
      <servlet-name>HelloWeb</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWeb</servlet-name>
      <url-pattern>*.jsp</url-pattern>
   </servlet-mapping>

</web-app>
```

ИТМО ВТ

Handler Mapping

- Механизм, позволяющий распределять запросы по различным обработчикам.
- Помимо «основного» Handler'а, в обработке запроса могут участвовать один или несколько «перехватчиков» (реализаций интерфейса `HandlerInterceptor`).
- Механизм в общем похож на сервлеты и фильтры.
- «Из коробки» программисту доступно несколько реализаций Handler Mapping.

ИТМО ВТ

Конфигурация Handler Mapping

На примере BeanNameUrlHandlerMapping:

```
<beans>
    <bean id="handlerMapping"
          class="o.s.w.s.h.BeanNameUrlHandlerMapping"/>

    <bean name="/editaccount.form"
          class="o.s.w.s.m.SimpleFormController">
        <property name="formView" value="account"/>
        <property name="successView"
                  value="account-created"/>
        <property name="commandName" value="account"/>
        <property name="commandClass"
                  value="samples.Account"/>
    </bean>
<beans>
```

ViewResolver

- Представление в Spring Web MVC может быть построено на разных технологиях.
- С каждым представлением сопоставляется его символическое имя.
- Преобразованием символических имён в ссылки на конкретные представления занимается специальный класс, реализующий интерфейс `org.springframework.web.servlet.ViewResolver`.
- Существует много реализаций ViewResolver для разных технологий построения представления.
- В одном приложении можно использовать несколько ViewResolver'ов.

ИТМО ВТ

Конфигурация ViewResolver'а

Пример для UrlBasedViewResolver:

```
<bean id="viewResolver"
    class="o.s.w.s.v.UrlBasedViewResolver">
    <property name="viewClass"
              value="o.s.w.s.v.JstlView"/>
    <property name="prefix"
              value="/WEB-INF/jsp/"/>
    <property name="suffix"
              value=".jsp"/>
</bean>
```

ИТМО ВТ

Объединение ViewResolver'ов в последовательность

```
<bean id="jspViewResolver"
      class="o.s.w.s.v.InternalResourceViewResolver">
    <property name="viewClass"
              value="o.s.w.s.v.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>

<bean id="excelViewResolver"
      class="o.s.w.s.v.XmlViewResolver">
    <property name="order" value="1"/>
    <property name="location" value="/WEB-INF/views.xml"/>
</bean>


<!-- in views.xml →
<beans>
    <bean name="report"
          class="o.s.e.ReportExcelView"/>
</beans>
```

ИТМО ВТ

Интеграция с Thymeleaf (1)

```
@Bean
@Description("Thymeleaf Template Resolver")
public ServletContextTemplateResolver templateResolver() {
    ServletContextTemplateResolver templateResolver =
                                  new ServletContextTemplateResolver();
    templateResolver.setPrefix("/WEB-INF/views/");
    templateResolver.setSuffix(".html");
    templateResolver.setTemplateMode("HTML5");
    return templateResolver;
}

@Bean
@Description("Thymeleaf Template Engine")
public SpringTemplateEngine templateEngine() {
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver());
    templateEngine.setTemplateEngineMessageSource(messageSource());
    return templateEngine;
}
```

ИТМО ВТ

Интеграция с Thymeleaf (2)

```
@Bean
@Description("Thymeleaf View Resolver")
public ThymeleafViewResolver viewResolver()
{
    ThymeleafViewResolver viewResolver =
                    new ThymeleafViewResolver();
    viewResolver
       .setTemplateEngine(templateEngine());
    viewResolver.setOrder(1);
    return viewResolver;
}
```

6. REST и SPA-фреймворки

ИТМО ВТ

Общие моменты

- Single Page Application – концепция, в соответствии с которой всё приложение «формально» – одна HTML-страница.
- За фактическую навигацию отвечает клиентский JavaScript – он «подменяет» URL и синхронизирует состояние с сервером.
- Клиент и сервер реализуются независимо, управляют своим состоянием независимо и взаимодействуют по REST.
- Формат передаваемых данных не специфицирован, но обычно это JSON.

6.1. REST back-end

ИТМО ВТ

Что это такое

- Серверная часть приложения, открытая «наружу» через REST API.
- Интерфейс обычно специфицирован (или даже автодокументирован – см., например, Swagger).
- Может соответствовать принципам RESTful (но это не обязательно!)
- Управляет бизнес-логикой и взаимодействием с хранилищем данных.
- В мире Java обычно строится на JAX-RS или Spring Web MVC (чаще всего).

REST Back-End на Spring Web MVC

- Обычные контроллеры Spring Web MVC.
- Server-side front-end обычно отсутствует – весь интерфейс строится отдельно на JS.
- Требуется отдельная защита от несанкционированного доступа – см., например, Spring Security.
- Требуется автоматическая сериализация / десериализация данных.

ИТМО ВТ

Создание контроллера

```
@Controller
@RequestMapping("/hello")
public class HelloController {
    @RequestMapping(method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Возвращаемое представление

Атрибуты модели

ИТМО ВТ

Request Body & Response Body

```java
@Controller
@RequestMapping("/post")
public class ExamplePostController {

    @Autowired
    ExampleService exampleService;

    @PostMapping("/response")
    @ResponseBody
    public ResponseTransfer postResponseController(
      @RequestBody LoginForm loginForm) {
        return new ResponseTransfer("Thanks For Posting!!!");
    }
}
```

ResponseTransfer будет сериализован в JSON

LoginForm будет десериализован из JSON

ИТМО ВТ

@GetMapping

```
@GetMapping("/books")
public void book() {
        //
}

/* these two mappings are identical */

@RequestMapping(value = "/books", method = RequestMethod.GET)
public void book2() {

}
```

Есть аналогичные аннотации для Post, Put, Delete и Patch

@RequestParam

```
@PostMapping("/users")
/* First Param is optional */
public User createUser(
    @RequestParam(required = false)
      Integer age,
    @RequestParam String name) {
      // does not matter
}
```

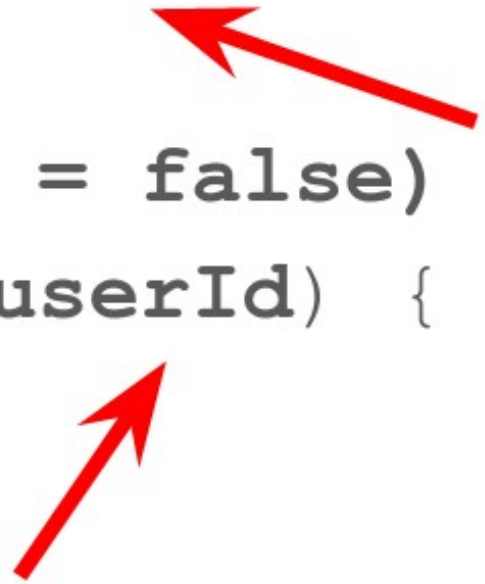
Автоматическая сериализация параметров запроса

```
@PostMapping("/users")
/* Spring преобразует userDto
автоматически, если в классе есть getters
and setters */
public User createUser(UserDto userDto) {
    //
}
```

ИТМО ВТ

@PathVariable

```
@GetMapping("/users/{userId}")
public User getUser(
   @PathVariable(required = false)
                    String userId) {
    //...
    return user;
}
```



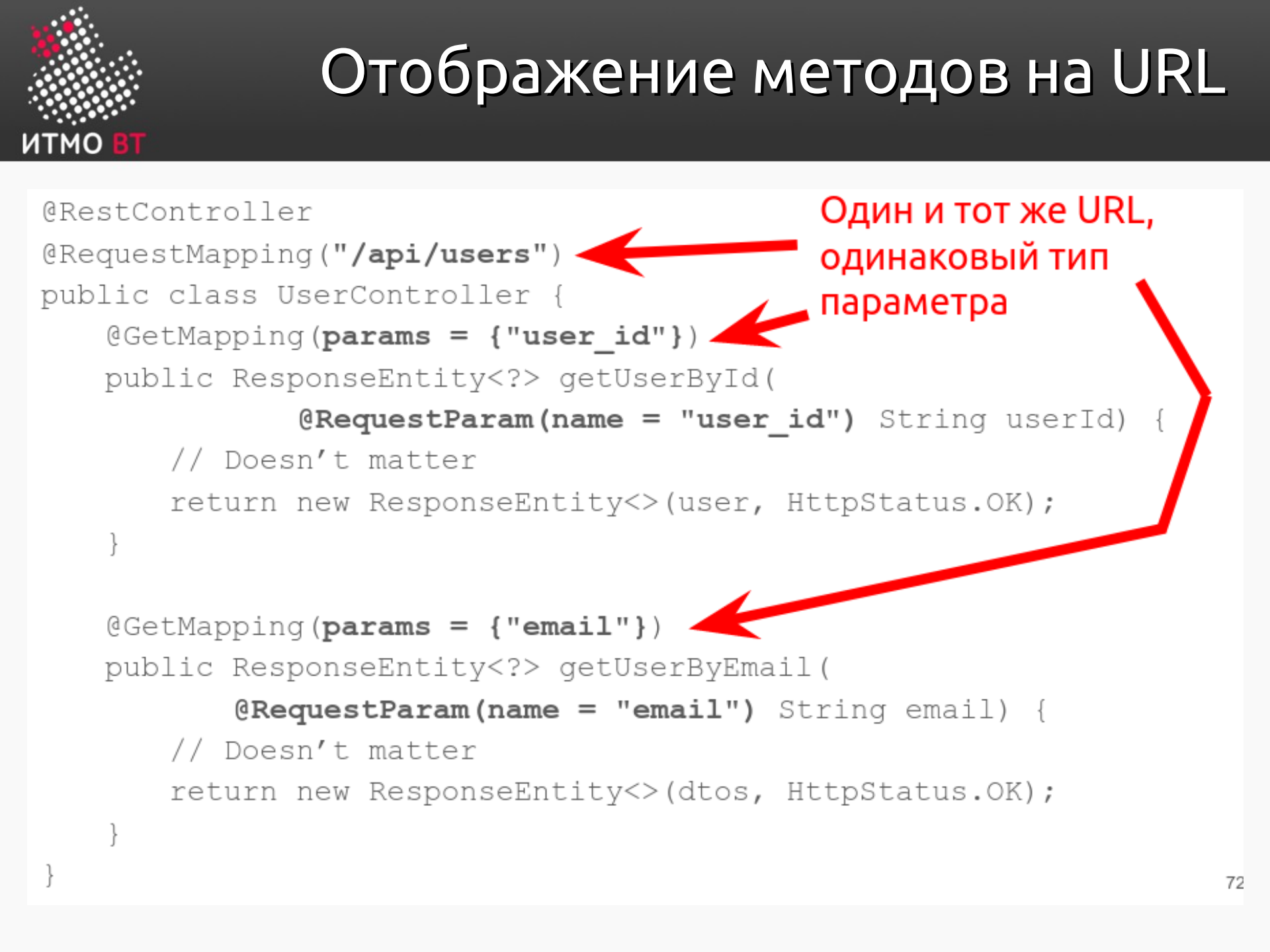
ИТМО ВТ

Автоматическая сериализация параметров URL

```
@GetMapping("/users/{userId}/{userName}")
public User getUser(UserDto userDto) {
    /* Автоматически присвоит значения
       свойствам “userId” и “userName” */
    return user;
}
```

Отображение методов на URL

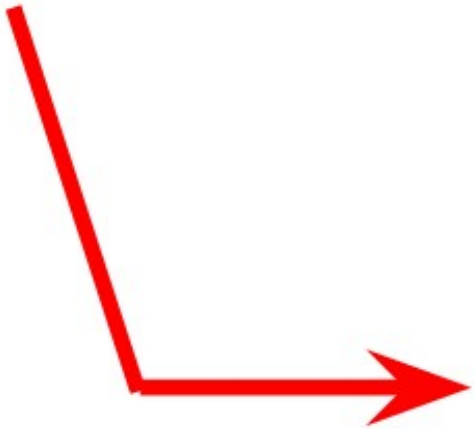
```
@RestController
@RequestMapping("/api/users")
public class UserController {
    @GetMapping(params = {"user_id"})
    public ResponseEntity<?> getUserById(
            @RequestParam(name = "user_id") String userId) {
        // Doesn't matter
        return new ResponseEntity<>(user, HttpStatus.OK);
    }

    @GetMapping(params = {"email"})
    public ResponseEntity<?> getUserByEmail(
            @RequestParam(name = "email") String email) {
        // Doesn't matter
        return new ResponseEntity<>(dtos, HttpStatus.OK);
    }
}
```



@RestController

```java
@Controller
@RequestMapping("books")
public class SimpleBookController {

    @GetMapping("/{id}", produces = "application/json")
    public @ResponseBody Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

@Controller
+
@ResponseBody
=
@RestController



```java
@RestController
@RequestMapping("books-rest")
public class SimpleBookRestController {

    @GetMapping("/{id}", produces = "application/json")
    public Book getBook(@PathVariable int id) {
        return findBookById(id);
    }

    private Book findBookById(int id) {
        // ...
    }
}
```

ИТМО ВТ

Accept Header

- Клиент сообщает серверу, какой формат ответа он хочет получить от контроллера.
- Используется заголовок `Accept`:

```
GET http://localhost:8080
                /transactions/{userid}
Accept: application/json
```

Ожидаемый формат ответа

ИТМО ВТ

HTTP Message Converter (1)

- Spring сам не умеет в сериализацию / маршалинг.
- Сериализация / маршалинг реализуются сторонними библиотеками.
- HttpMessageConverter – адаптер для сторонних библиотек.
- Содержит 4 метода – `canRead(MediaType)`, `canWrite(MediaType)`, `read(Object, InputStream, MediaType)` и `write(Object, OutputStream, MediaType)`.

HTTP Message Converter (2)

Есть готовые конвертеры “из коробки”:

```
Static {
    ClassLoader classLoader =
         AllEncompassingFormHttpMessageConverter
                                .class.getClassLoader();
    jaxb2Present = ClassUtils.isPresent(
                "javax.xml.bind.Binder", classLoader);
    jackson2Present = /*...*/
    jackson2XmlPresent = /*...*/
    jackson2SmilePresent = /*...*/
    gsonPresent = /*...*/
    jsonbPresent = /*...*/

}
```

6.2. SPA front-end

ИТМО ВТ

Общие моменты

- Фреймворков очень много (JS-бинго!), знать все – невозможно.
- Все базируются на схожих принципах, уровень абстракции различается.
- Наиболее популярные сейчас (осень 2021 г.) – React («библиотека, а не фреймворк»), Angular, Vue.

6.2.1. React

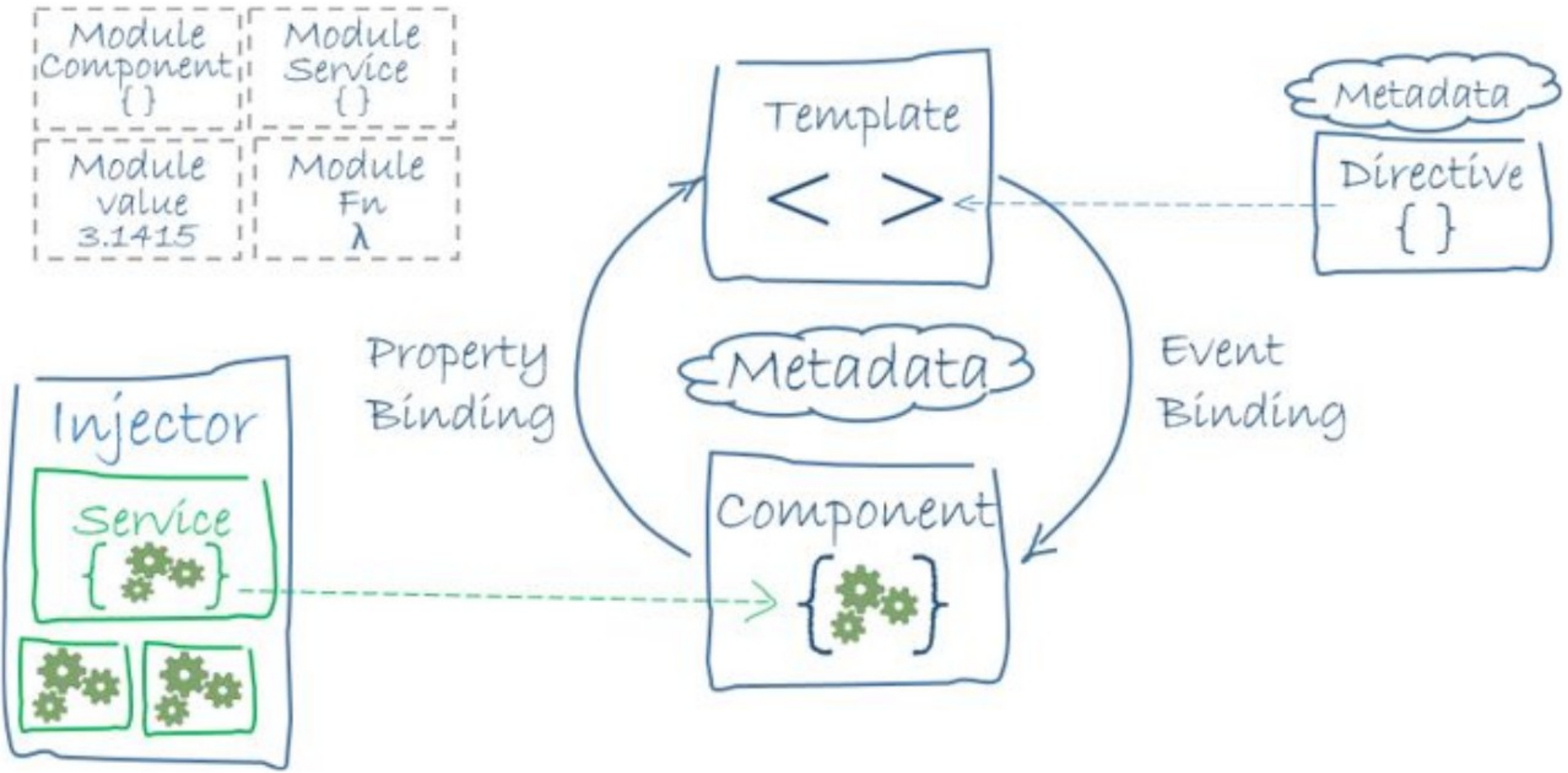
React

…см. видео на YouTube.

6.2.2. Angular

ИТМО ВТ

Angular



…см. отдельную презентацию.

6.2.3. Vue.js

ИТМО ВТ

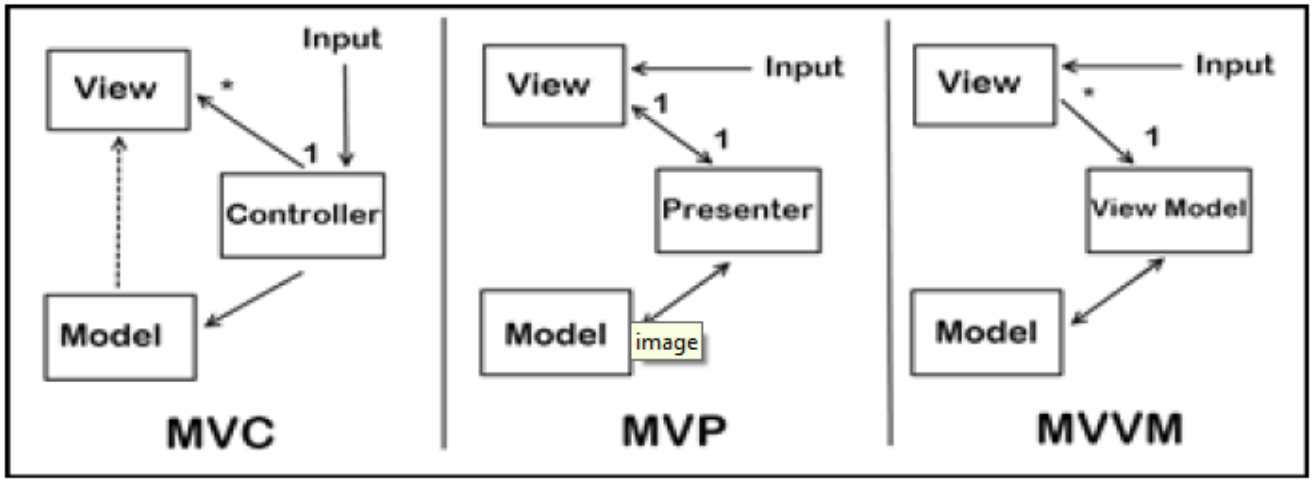
О Vue.js

- JS-фреймворк с открытым исходным кодом (лицензия MIT) для разработки UI.
- Первая версия вышла в 2014 г., актуальная версия (3.2) – в августе 2021 г.
- Построен на архитектуре MVVM; может быть использован для разработки SPA в реактивном стиле.
- Реактивность сводится к тому, что представление в меняется по мере изменения модели.
- Есть документация на русском языке.

ИТМО ВТ

MVVM, MVP и MVC

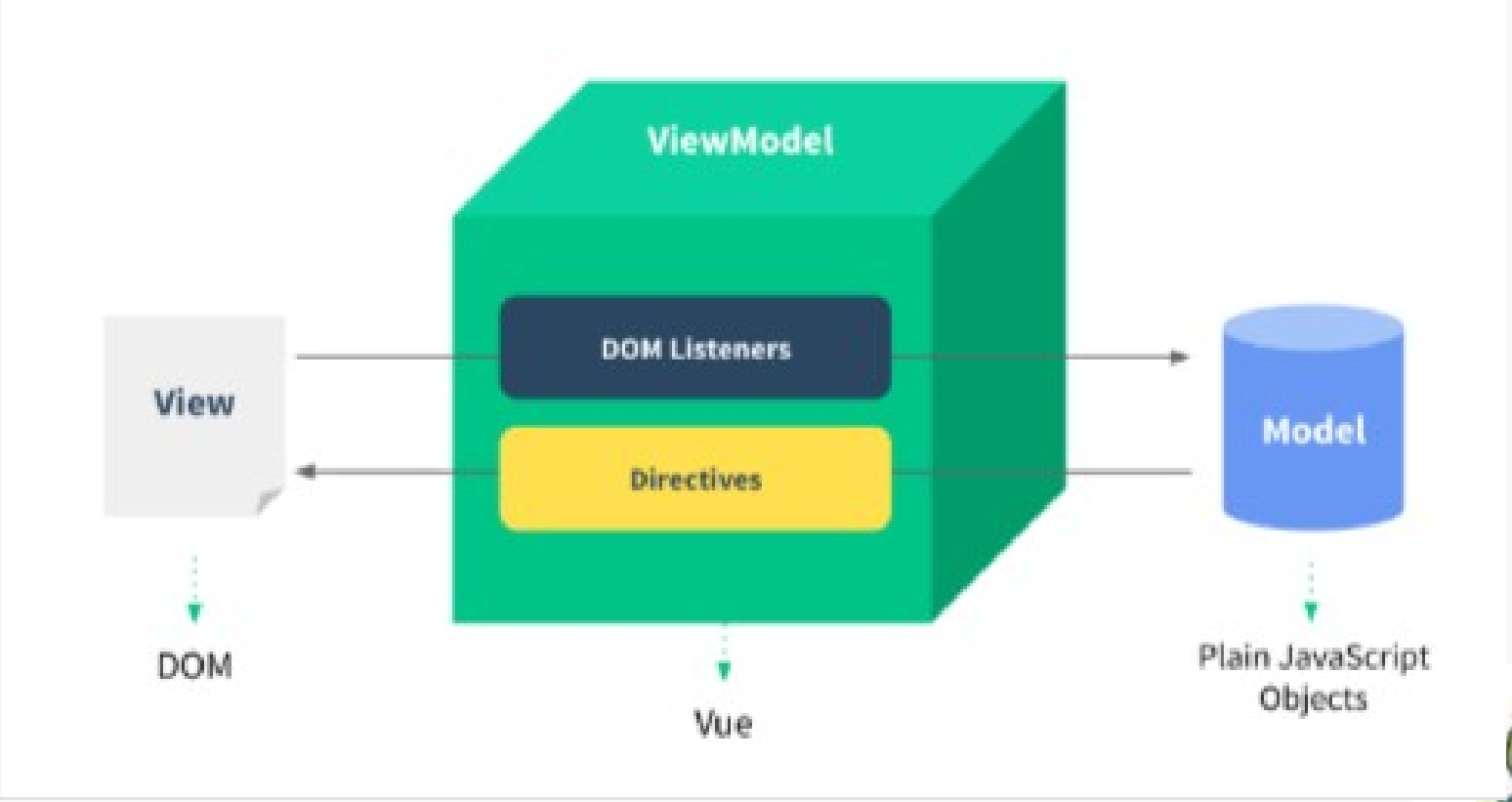
Difference between MVP, MVVM, and MVC.



MVC	MVP	MVVM
The Input is directed at the controller	The Input begins with View	The input begins with model
Many-to-one relation between Controller and View	One-to-one mapping between view and its associated presenter	One-to-may mapping between various and one ViewModel
View does not have any knowledge about the controller	The View hold reference to the presenter; Presenter also reacting to the events being triggered from the View,so presenter does aware of the View	The View holds reference to the ViewModel;View Model has no information about the View.Here different technology can share viewModel (i.e) WCF and Silverlight can share same ViewModel.
Controller does pass the Model to the View,so View has knowledge about the controller	View is not Model aware	View is not Model aware

ИТМО ВТ

Архитектура приложения на Vue.js



ИТМО ВТ

Структура приложения: экземпляр

- Любое приложение имеет, как минимум, один центральный экземпляр.
- Для каждого файла HTML возможно любое количество экземпляров.
- Экземпляр привязывается к узлу DOM с помощью свойства `el`:

```
var vm = new Vue({
  el: "body",
  data: {
    message: "Привет Мир!",
    items: [
      "это",
      "и",
      "есть",
      "Array/Массив"
    ]
  }
});
```

ИТМО ВТ

Структура приложения: компоненты

- Позволяют расширить функциональность экземпляров.
- В отличие от экземпляров, не привязываются к узлам HTML, а содержат собственную разметку:

```
// Определение компонента и глобальная регистрация
Vue.component('my-component', {
  template: '<div> это новый компонент < / div>'
})

// Создание экземпляра
new Vue({
  el: '#example'
})

<!-- HTML -->
<div id="example">
  <my-component></my-component>
</div>
```

Структура приложения: директивы

- Позволяют выполнять различные операции, например, итерацию по массиву или включение элементов в DOM по условию.
- В разметке представляют собой атрибуты тегов:

```
<div id="conditional-rendering">
  <span v-if="seen">Сейчас меня видно</span>
</div>

const ConditionalRendering = {
  data() {
    return { seen: true }
  }
}

Vue.createApp(ConditionalRendering)
   .mount('#conditional-rendering')
```

ИТМО ВТ

Жизненный цикл компонентов

Можно привязывать обработчики к фазам и событиям жизненного цикла компонентов:

```
new Vue({
  // Вызывается, когда компонент будет видно,
  // например, через v-if vue или маршрутизатор.
  mounted: function () {
    console.log('Этот компонент был ' +
                'интегрирован в DOM ' + Date.now());
    this.$next(() => console.log('Теперь ' +
               'компонент готов к работе.'))
  },
  //Вызывается, когда компонент удаляется из DOM.
  destroyed: function() {
    removeListener(XY);
  }
})
```

Маршрутизация и навигация

```
<div id="app">
  <router-view></router-view>
</div>

...
<script>
  ...
  const User = {
    template: '<div>User {{ $route.params.id }}</div>'
  };

  const router = new VueRouter({
    routes: [
      { path: '/user/:id', component: User }
    ]
  });
  ...
</script>
```

ИТМО ВТ

Экосистема

- Официальные инструменты для разработчика:
  - Devtools – браузерный плагин для отладки приложений.
  - Vue CLI – консольные утилиты разработчика.
  - Vue Loader – загрузчик на базе webpack, позволяющий создавать компоненты Vue в формате Single-File Components (SFCs)
- Официальные библиотеки:
  - Vue Router – библиотека для реализации маршрутизации и навигации в SPA.
  - Vuex – библиотека управления состоянием на базе Flux.
  - Vue Server Renderer – инструментарий для Server-Side Rendering.

Приложение 1. Google Web Toolkit

ИТМО ВТ

GWT

- By Google.
- На нём сделан Gmail! (нет).
- OpenSource (Apache 2.0).
- Проектируем веб как десктоп.
- Весь код пишется на Java — можно попытаться не верстать.
- Если всё-таки нужно верстать — это можно сделать, но довольно больно.

ИТМО ВТ

Архитектура приложений

- Клиентский код пишется на Java и транслируется в HTML и JavaScript.
- «Точка входа» в приложение — класс, реализующий интерфейс `com.google.gwt.core.client.EntryPoint`.
- Конфигурация приложения задаётся в дескрипторе `AppName.gwt.xml`.
- Для создания «каркаса» приложения можно использовать IDE, либо утилиту GWT ApplicationCreator:

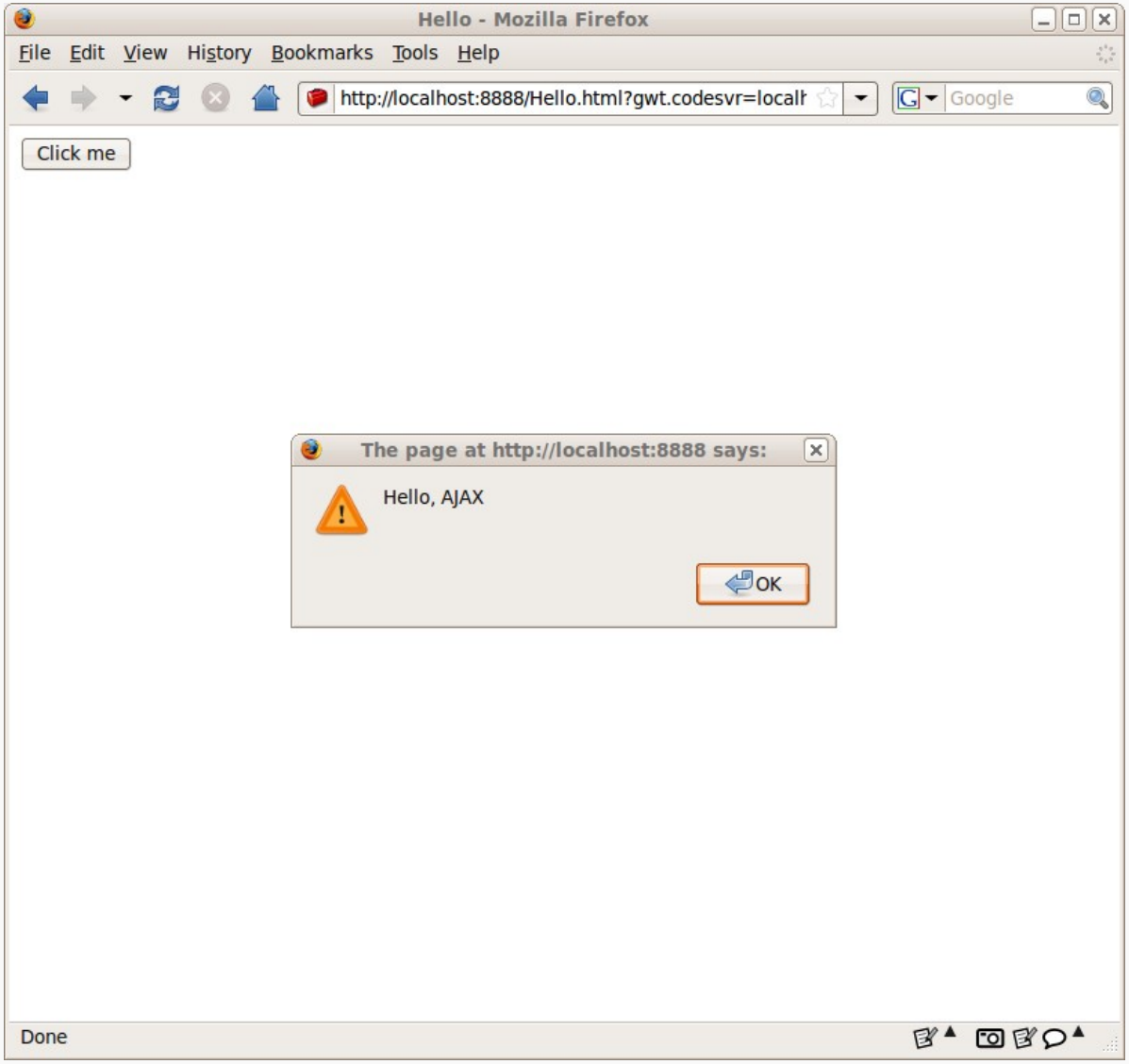
```
user@host:~$ ./webAppCreator -out MyWebApp \
> com.mycompany.mywebapp.MyWebApp
```

- Несколько способов организации клиент-серверного взаимодействия — GWT RPC, JSON via HTTP, Cross-Site Requests.
- В сложных приложениях часто используется паттерн Model-View-Presenter (MVP).

HelloWorld на GWT

```
package com.google.gwt.sample.hello.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.Widget;

/**
 * Hello World application.
 */
public class Hello implements EntryPoint {

  public void onModuleLoad() {
    Button b = new Button("Click me", new ClickHandler() {
      public void onClick(ClickEvent event) {
        Window.alert("Hello, AJAX");
      }
    });

    RootPanel.get().add(b);
  }
}
```

HelloWorld на GWT (продолжение)



ИТМО ВТ

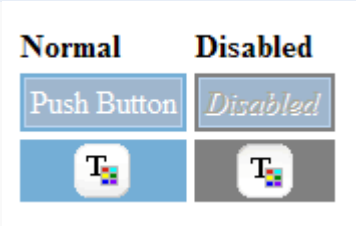
Виджеты и события

- Виджет (Widget) — базовый элемент интерфейса на GWT.
- Все виджеты наследуются от базового класса `com.google.gwt.user.client.ui.Widget`.
- Панели и шаблоны разметки (`Panel`, см. дальше) — это тоже виджеты.
- Виджеты могут формировать события (`Event`), которые обрабатываются обработчиками (`Handler`).
- Внешний вид виджетов можно менять с помощью обычного CSS.
- Можно (и часто бывает нужно!) создавать собственные виджеты.

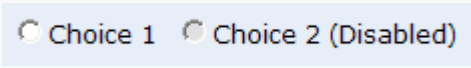
Примеры виджетов

- Button

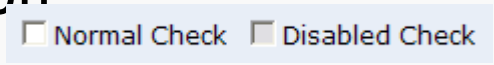
- PushButton



- RadioButton

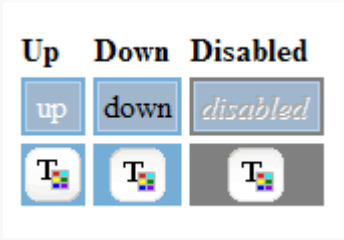


- CheckBox

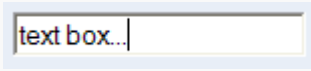


- DatePicker

- ToggleButton

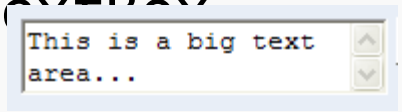


- TextBox

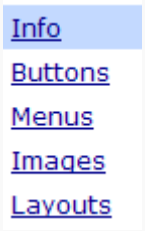


- PasswordTextBox

- TextArea



- HyperLink



ИТМО ВТ

Работа с виджетами

```
public class ButtonExample implements EntryPoint {

  public void onModuleLoad() {
    // Make a new button that does something when you
    // click it.
    Button b = new Button("Jump!", new ClickHandler() {
      public void onClick(ClickEvent event) {
        Window.alert("How high?");
      }
    });

    // Add it to the root panel.
    RootPanel.get().add(b);
  }
}
```

ИТМО ВТ

События и обработчики

- Модель похожа на Swing.
- Виджеты умеют порождать события (их номенклатура зависит от конкретного виджета).
- Обработкой событий занимаются классы-обработчики (Handler'ы).
- Обработчики подписываются на определённые типы событий с определённых компонентов.

События и обработчики (продолжение)

```
public class HandlerExample extends Composite implements ClickHandler {
  private FlowPanel fp = new FlowPanel();
  private Button b1 = new Button("Button 1");
  private Button b2 = new Button("Button 2");

  public HandlerExample() {
    initWidget(fp);
    fp.add(b1);
    fp.add(b2);
    b1.addClickHandler(this);
    b2.addClickHandler(this);
  }

  public void onClick(ClickEvent event) {
    // note that in general, events can have sources that are not Widgets.
    Widget sender = (Widget) event.getSource();

    if (sender == b1) {
      // handle b1 being clicked
    } else if (sender == b2) {
      // handle b2 being clicked
    }
  }
}
```

ИТМО ВТ

Разметка страниц

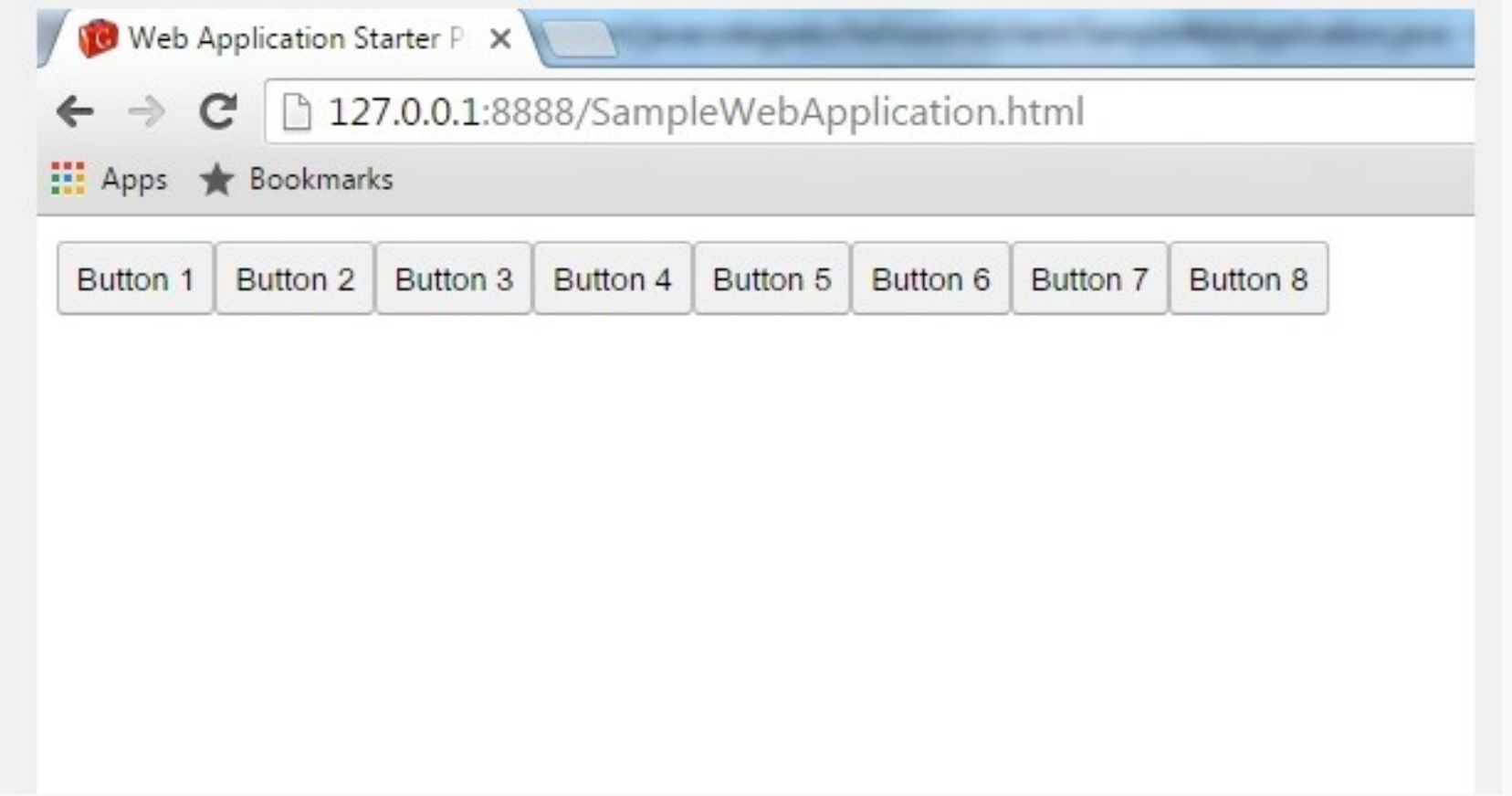
- Делается с помощью панелей — виджетов, наследующихся от `com.google.gwt.user.client.ui.Panel`.
- Панели могут вкладываться друг в друга (а-ля Swing).
- «Корневая» панель — экземпляр класса `com.google.gwt.user.client.ui.RootPanel`.

Разметка страниц (продолжение)

```
public void onModuleLoad() {

    FlowPanel flowPanel = new FlowPanel();

    // Add buttons to flow Panel
    for(int i = 1;  i <= 8; i++){
        Button btn = new Button("Button " + i);
        flowPanel.add(btn);
    }

    // Add the Flow Panel to the root panel.
    RootPanel.get().add(flowPanel);
}
```

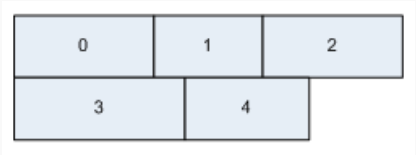
ИТМО ВТ

Разметка страниц (продолжение)



Basic Panels

- `RootPanel` — «корневая» панель страницы.
- `FlowPanel` —



- `HTMLPanel` — позволяет сверстать HTML руками и расположить там виджеты на заданных позициях.
- `FormPanel` — объединяет виджеты внутри общей HTML-формы.
- `ScrollPanel` — блок с прокруткой.
- `PopupPanel` и `DialogBox` —

- `Grid` и `FlexTable` —

sender	email
markboland05	mark@example.com
Hollie Voss	hollie@example.com
boticario	boticario@example.com
Emerson Milton	emerson@example.com
Healy Colette	healy@example.com

ИТМО ВТ

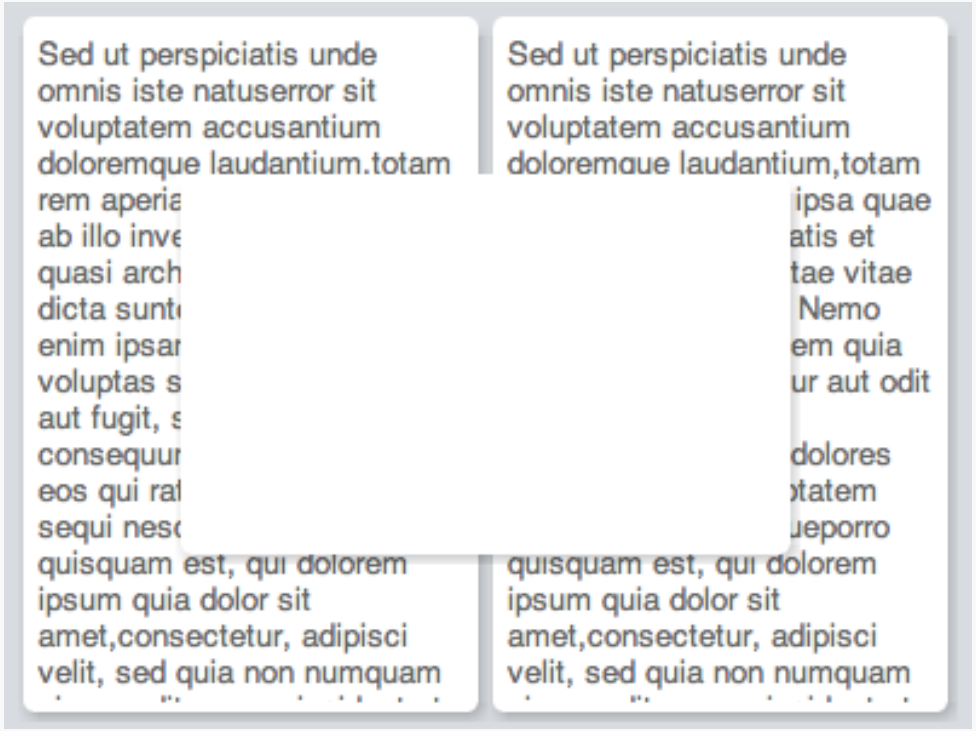
Layout Panels

- Появились в GWT 2.0 для более удобной [адаптивной] вёрстки.
- Базовый класс — `com.google.gwt.layout.client.Layout`, описывает «абстрактный» шаблон.
- Можно создавать свои шаблоны.

Layout Panels (продолжение)

- `RootLayoutPanel` — «корневая» панель, singleton (аналог `RootPanel`).
- `LayoutPanel` — «базовая» панель:

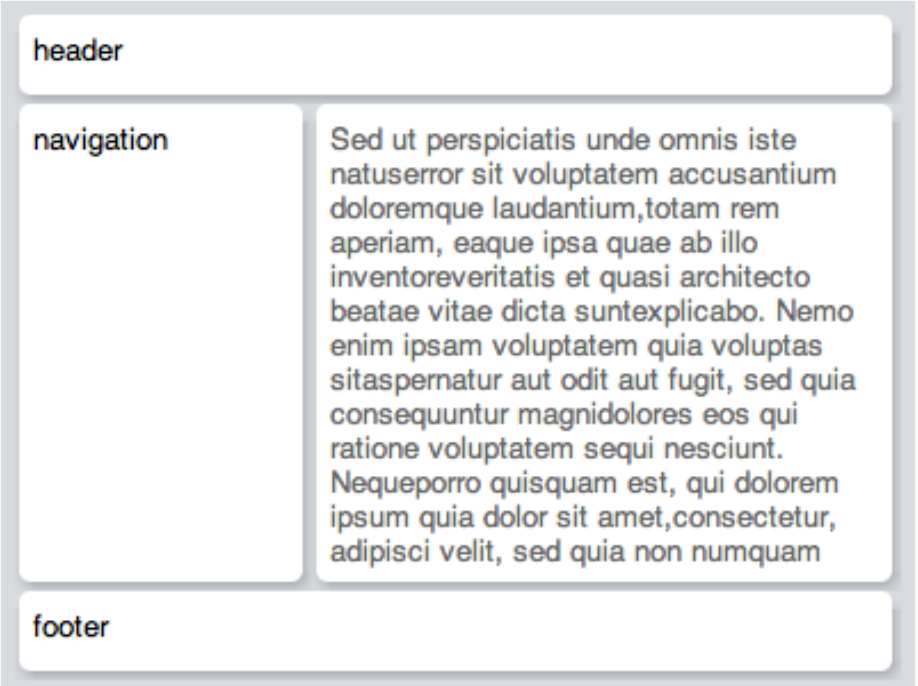
```
Widget child0, child1, child2;
LayoutPanel p = new LayoutPanel();
p.add(child0); p.add(child1); p.add(child2);

p.setWidgetLeftWidth(child0,
  0, PCT,
  50, PCT);  // Left panel
p.setWidgetRightWidth(child1,
  0, PCT,
  50, PCT);  // Right panel

p.setWidgetLeftRight(child2,
  5, EM,
  5, EM);   // Center panel
p.setWidgetTopBottom(child2,
  5, EM,
  5, EM);
```



Layout Panels (продолжение)

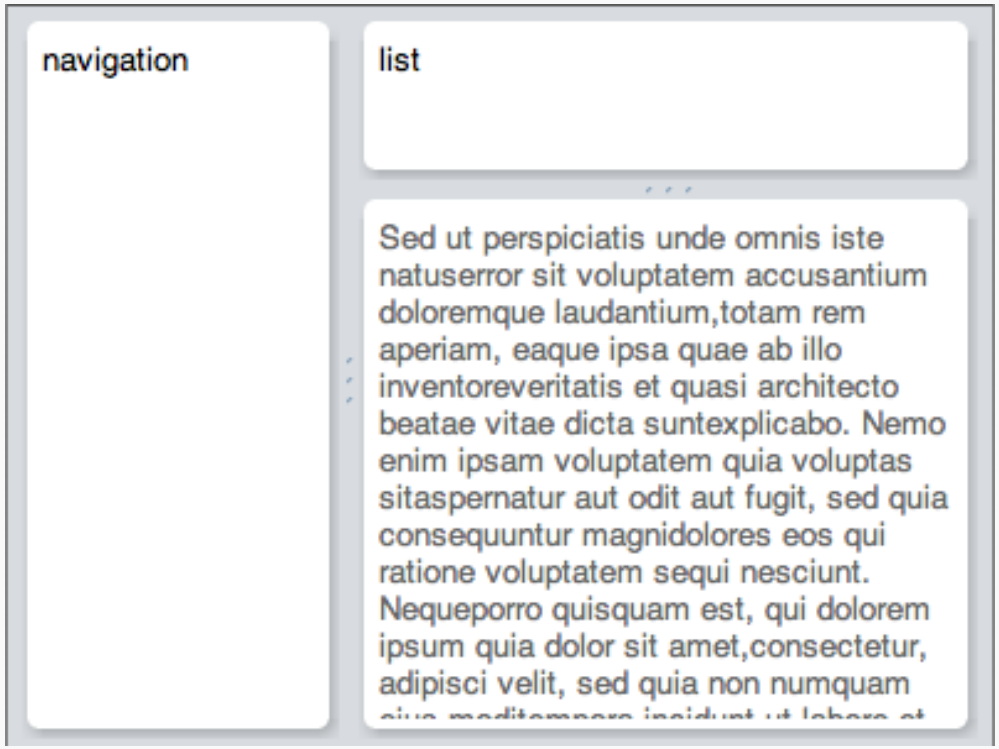
DockLayoutPanel – аналог BorderLayout в Swing:

```
DockLayoutPanel p = new DockLayoutPanel(Unit.EM);
p.addNorth(
  new HTML("header"),
  2);
p.addSouth(
  new HTML("footer"),
  2);
p.addWest(
  new HTML("navigation"),
  10);
p.add(new HTML(content));
```



Layout Panels (продолжение)

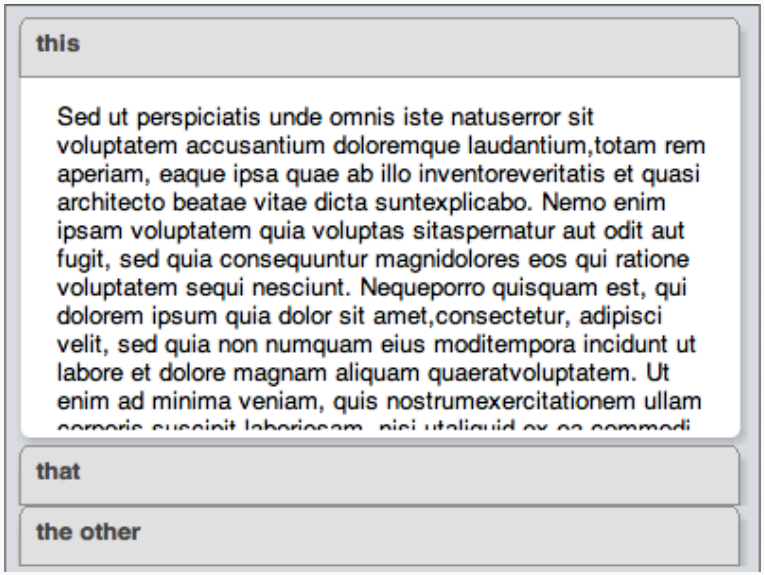
`SplitLayoutPanel` — `DockLayoutPanel` с движущимися разделителями:



Layout Panels (продолжение)

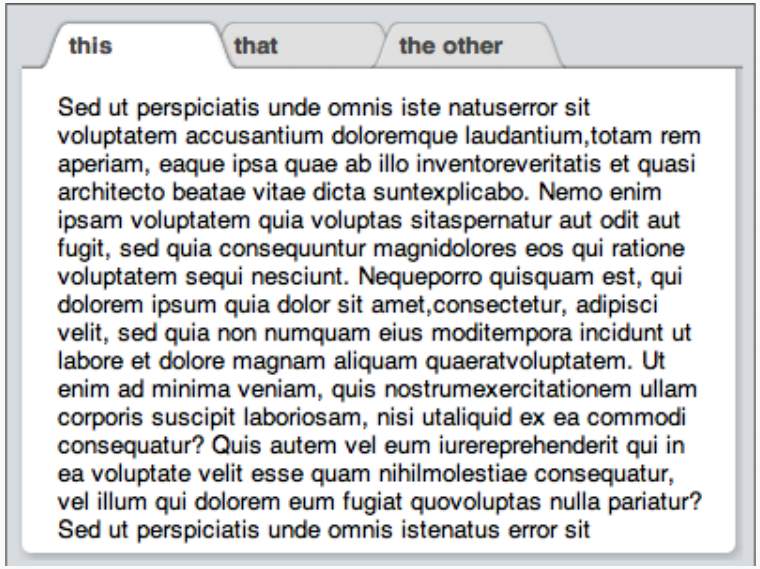
- StackLayoutPanel:

```
StackLayoutPanel p
  = new StackLayoutPanel(Unit.EM);
p.add(new HTML("this content"),
  new HTML("this"), 4);
p.add(new HTML("that content"),
  new HTML("that"), 4);
p.add(new HTML("the other content"),
  new HTML("the other"), 4);
```



- TabLayoutPanel:

```
TabLayoutPanel p =
  new TabLayoutPanel(1.5, Unit.EM);
p.add(new HTML("this content"),
  "this");
p.add(new HTML("that content"),
  "that");
p.add(new HTML("the other content"),
  "the other");
```



CSS

- Используется обычный CSS и темы оформления.
- Тему можно задать в конфигурационном файле:

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='stockwatcher'>
  <!-- Inherit the core Web Toolkit stuff. -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.  You can change   -->
  <!-- the theme of your GWT application by uncommenting   -->
  <!-- any one of the following lines.   -->
  <inherits name='com.google.gwt.user.theme.standard.Standard'/>
  <!-- <inherits name="com.google.gwt.user.theme.chrome.Chrome"/> -->
  <!-- <inherits name="com.google.gwt.user.theme.dark.Dark"/> -->

  <!-- Other module inherits  -->

  <!-- Specify the app entry point class. -->
  <entry-point
    class='com.google.gwt.sample.stockwatcher.client.StockWatcher'/>
</module>
```

- Можно создавать собственные темы.

ИТМО ВТ

Взаимодействие с сервером

- Весь написанный код разметки транслируется в HTML + CSS + JS.
- Взаимодействие с сервером нужно описывать «вручную» (в отличие от Vaadin — см. далее).
- 3 способа взаимодействия с сервером:
  - Remote Procedure Calls (GWT RPC).
  - JSON Data via HTTP.
  - Cross-Site Requests for JSONP.

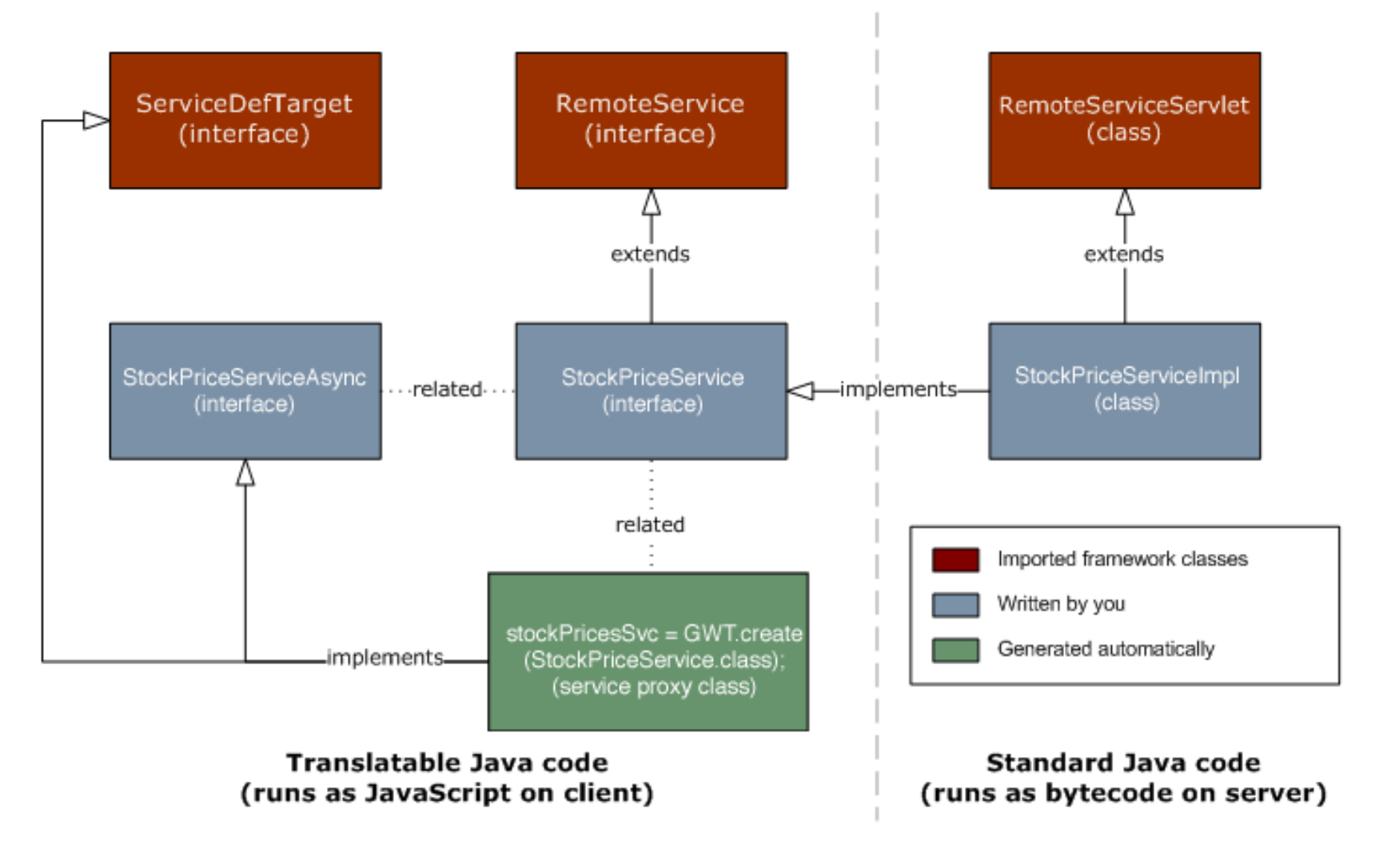
ИТМО ВТ

GWT RPC

- Классическая реализация RPC — клиент (GWT) вызывает сервис на сервере.
- Сервисы GWT RPC базируются на сервлетах.
- Клиентский код использует автоматически сгенерированный класс-заглушку (proxy class).
- GWT Runtime управляет сериализацией и десериализацией объектов (аргументов и возвращаемых значений).

ИТМО ВТ

Архитектура GWT RPC



ИТМО ВТ

GWT RPC: интерфейс и реализация сервиса

```
package com.google.gwt.sample.stockwatcher.client;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("stockPrices")
public interface StockPriceService extends RemoteService {
      StockPrice[] getPrices(String[] symbols);
}
```

```
package com.google.gwt.sample.stockwatcher.server;

import com.google.gwt.sample.stockwatcher.client.StockPrice;
import com.google.gwt.sample.stockwatcher.client.StockPriceService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.util.Random;

public class StockPriceServiceImpl extends RemoteServiceServlet implements StockPriceService {
    private static final double MAX_PRICE = 100.0; // $100.00
    private static final double MAX_PRICE_CHANGE = 0.02; // +/- 2%

    public StockPrice[] getPrices(String[] symbols) {
        Random rnd = new Random();

        StockPrice[] prices = new StockPrice[symbols.length];
        for (int i=0; i<symbols.length; i++) {
            double price = rnd.nextDouble() * MAX_PRICE;
            double change = price * MAX_PRICE_CHANGE * (rnd.nextDouble() * 2f - 1f);

            prices[i] = new StockPrice(symbols[i], price, change);
        }

        return prices;
  }
}
```

ИТМО ВТ

GWT RPC: интерфейс и реализация сервиса

```
package com.google.gwt.sample.stockwatcher.client;

import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("stockPrices")
public interface StockPriceService extends RemoteService {
      StockPrice[] getPrices(String[] symbols);
}
```

```
package com.google.gwt.sample.stockwatcher.server;

import com.google.gwt.sample.stockwatcher.client.StockPrice;
import com.google.gwt.sample.stockwatcher.client.StockPriceService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.util.Random;

public class StockPriceServiceImpl extends RemoteServiceServlet implements StockPriceService {
    private static final double MAX_PRICE = 100.0; // $100.00
    private static final double MAX_PRICE_CHANGE = 0.02; // +/- 2%

    public StockPrice[] getPrices(String[] symbols) {
        Random rnd = new Random();

        StockPrice[] prices = new StockPrice[symbols.length];
        for (int i=0; i<symbols.length; i++) {
            double price = rnd.nextDouble() * MAX_PRICE;
            double change = price * MAX_PRICE_CHANGE * (rnd.nextDouble() * 2f - 1f);

            prices[i] = new StockPrice(symbols[i], price, change);
        }

        return prices;
    }
}
```

ИТМО ВТ

GWT RPC: клиент

```
(...)
private void refreshWatchList() {
    // Initialize the service proxy.
    if (stockPriceSvc == null) {
      stockPriceSvc = GWT.create(StockPriceService.class);
    }

    // Set up the callback object.
    AsyncCallback<StockPrice[]> callback = new AsyncCallback<StockPrice[]>() {
      public void onFailure(Throwable caught) {
        // TODO: Do something with errors.
      }

      public void onSuccess(StockPrice[] result) {
        updateTable(result);
      }
    };

    // Make the call to the stock price service.
    stockPriceSvc.getPrices(stocks.toArray(new String[0]), callback);
}
(...)
```

ИТМО ВТ

JSON via HTTP

- GWT-приложение формирует HTTP-запросы на сервер и обрабатывает получаемый в ответ на них JSON.
- В этом случае реализация сервера может быть любой.
- Спецификация JSON задаётся с помощью класс `JavaScriptObject`.
- Для формирования и обработки запроса используются классы `RequestBuilder` и `RequestCallback`.

JSON via HTTP: спецификация JSON

```
package com.google.gwt.sample.stockwatcher.client;

import com.google.gwt.core.client.JavaScriptObject;

class StockData extends JavaScriptObject {
  (...)

  // Overlay types always have protected, zero argument constructors.
  protected StockData() {}

  // JSNI methods to get stock data.
  public final native String getSymbol(); // { return this.symbol; }
  public final native double getPrice();  // { return this.price; }
  public final native double getChange(); // { return this.change; }

  // Non-JSNI method to return change percentage.
  public final double getChangePercent() {
    return 100.0 * getChange() / getPrice();
  }
}
```

ИТМО ВТ

JSON via HTTP: формирование запроса и обработка ответа

```
RequestBuilder builder = new RequestBuilder(RequestBuilder.GET, url);

try {
  Request request = builder.sendRequest(null, new RequestCallback() {
    public void onError(Request request, Throwable exception) {
      displayError("Couldn't retrieve JSON");
    }

    public void onResponseReceived(Request request, Response response) {
      if (200 == response.getStatusCode()) {
        updateTable(JsonUtils.
          <JsArray<StockData>>safeEval(response.getText()));
      } else {
        displayError("Couldn't retrieve JSON ("
          + response.getStatusText() + ")");
      }
    }
  });
} catch (RequestException e) {
  displayError("Couldn't retrieve JSON");
}
```

ИТМО ВТ

Cross-Site Requests for JSONP

- HTTP-запросы отправляются на разные сервисы в разных доменах.
- Приходится решать проблему с ограничениями SOP (Same Origin Policy).
- Для формирования запросов используется класс `JsonpRequestBuilder`.

ИТМО ВТ

Ограничения SOP и пути его обхода

- Браузеры запрещают вызывать из JS сервисы на «внешних» доменах.
- Пути обхода этого ограничения:
  - Настроить прокси на локальном сервере — нужно настраивать веб-сервер.
  - Динамически загружать JSON внутрь тега `<script>` — непонятно, когда заканчивается загрузка данных.

ИТМО ВТ

JSONP

- JSON with Padding - «JSON с обёрткой».
- JSON на сервере «заворачивается» в вызов callback-функции:

```
callback125([{"symbol":"DDD",
              "price":10.610339195026,
              "change":0.053085447454327}]);
```

- Функция вызывается сразу по завершении загрузки данных.

ИТМО ВТ

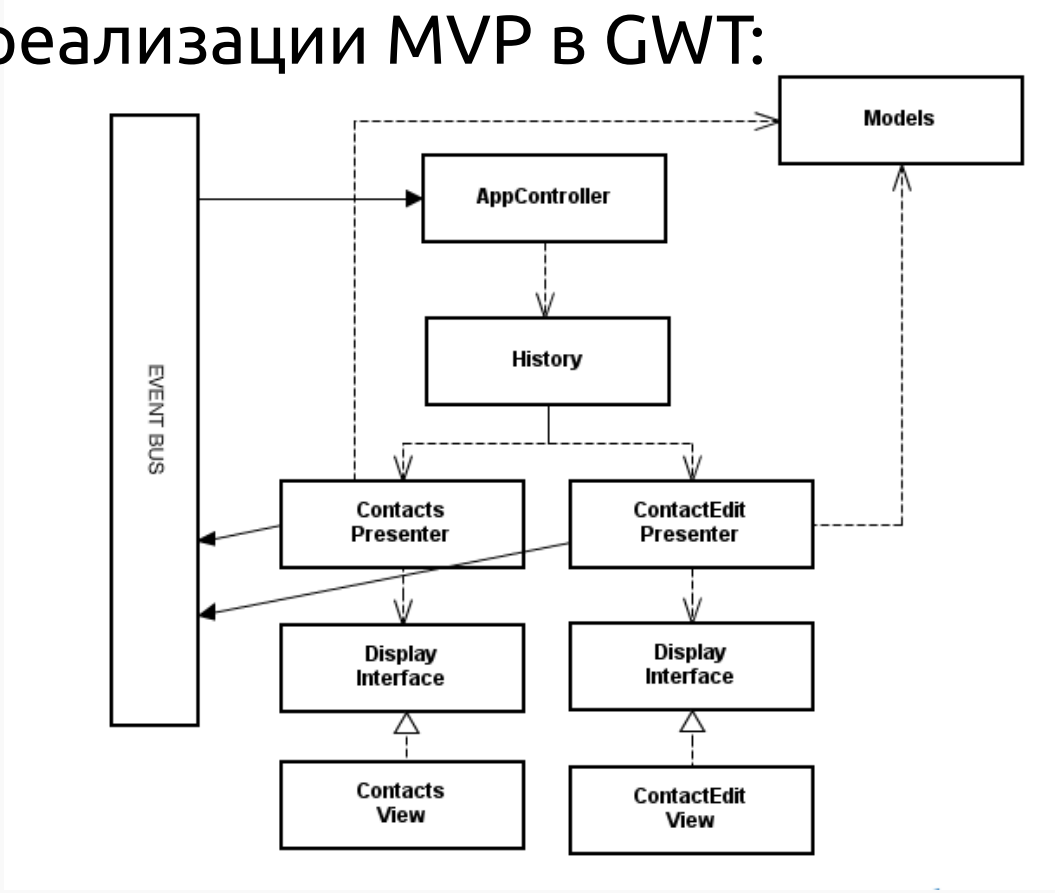
Работа JSONP в GWT

```
JsonpRequestBuilder builder =
                new JsonpRequestBuilder();

builder.requestObject(url,
  new AsyncCallback<JsArray<StockData>>() {

    public void onFailure(Throwable caught) {
      displayError("Couldn't retrieve JSON");
    }

    public void onSuccess(JsArray<StockData> data) {
        // TODO handle JSON response
    }

  });
```

ИТМО ВТ

MVP в GWT

- Model-View-Presenter — архитектурный шаблон на базе MVC.
- Пример реализации MVP в GWT:



Приложение 2. Vaadin

ИТМО ВТ

Vaadin

- Финский фреймворк на базе gwt.
- Много «богатых» компонентов, активно используется AJAX.
- В отличие от GWT, логика, в основном, исполняется на сервере.
- Более «тяжёлый» по сравнению с «обычным» gwt — больше тяжёлого JS, выше нагрузка на канал.

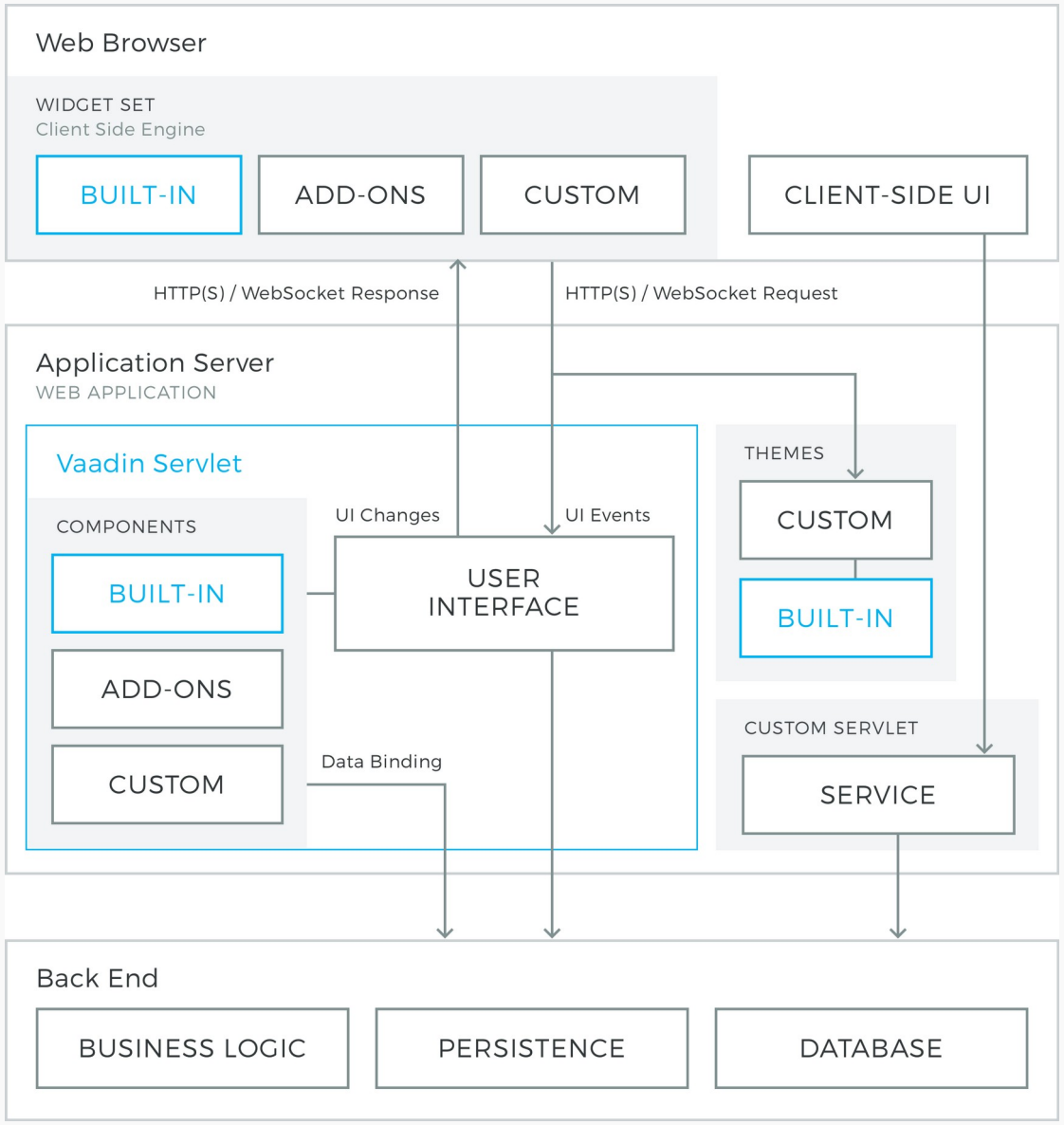
ИТМО ВТ

HelloWorld на Vaadin

```
import com.vaadin.ui.*;

public class HelloWorld
    extends com.vaadin.Application {

    public void init() {
        Window main = new Window("Hello window");
        setMainWindow(main);
        main.addComponent(
            new Label("Hello World!"));
    }
}
```

ИТМО ВТ

Архитектура Vaadin



ИТМО ВТ

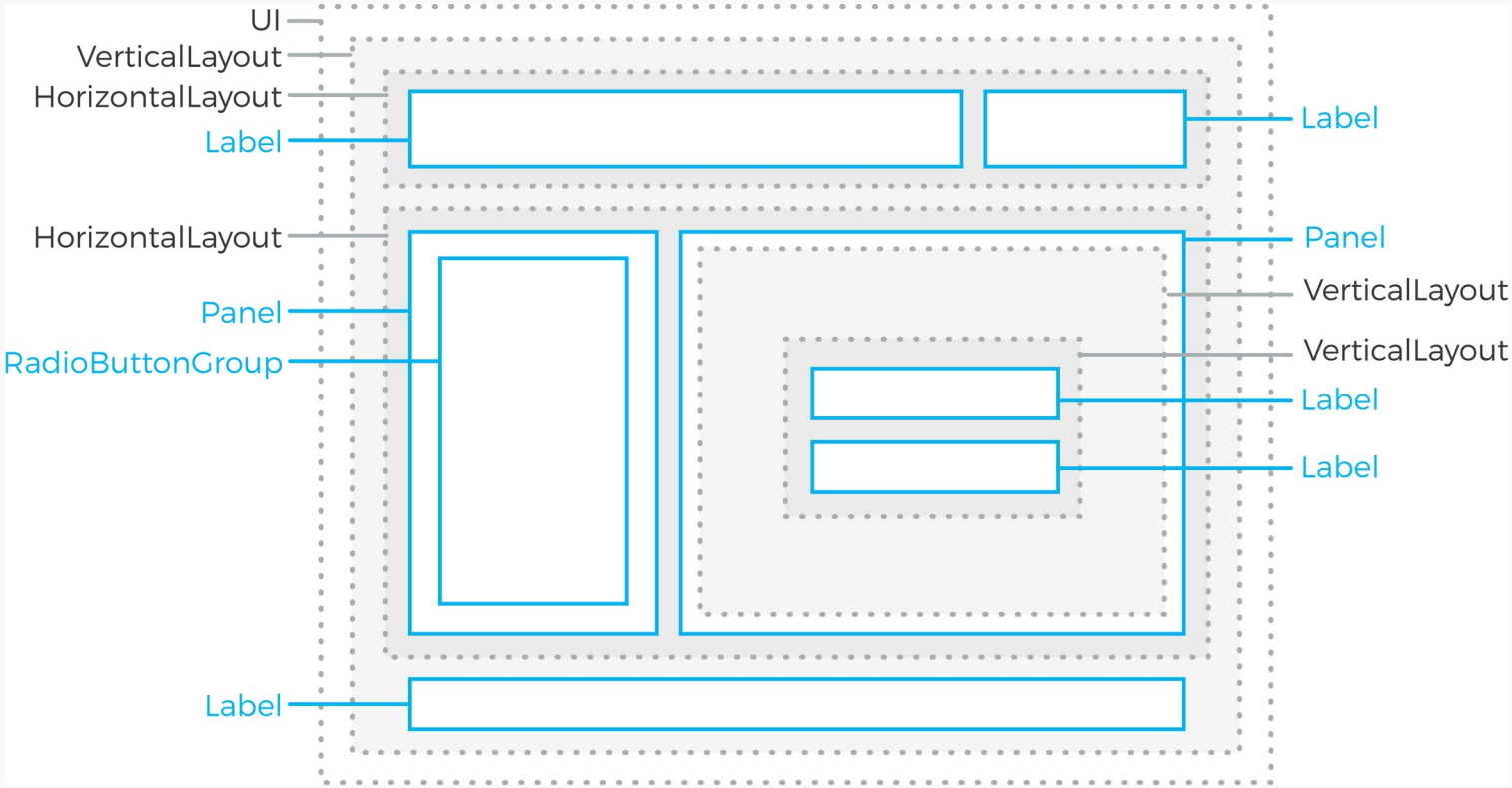
Архитектура Vaadin

- Интерфейс строится из компонентов.
- Каждый компонент состоит из виджета (а-ля GWT) и серверной составляющей.
- Компонент может иметь значение, которое автоматически (с помощью AJAX-запроса) присваивается полю на стороне сервера.
- Взаимодействие пользователя с компонентами порождает события. В отличие от GWT, события обрабатываются на сервере.
- Сервер умеет оповещать клиент об обновлении состояния модели с помощью WS.

ИТМО ВТ

Разметка страниц

- Используются шаблоны разметки (Layouts).



- Могут задаваться программно (в коде), или декларативно (в xml-шаблоне).

ИТМО ВТ

Программная конфигурация разметки

```
// Set the root layout for the UI
VerticalLayout content = new VerticalLayout();
setContent(content);

HorizontalLayout titleBar = new HorizontalLayout();
titleBar.setWidth("100%");
root.addComponent(titleBar);

Label title = new Label("The Ultimate Cat Finder");
titleBar.addComponent(title);
titleBar.setExpandRatio(title, 1.0f); // Expand

Label titleComment = new Label("for Vaadin");
titleComment.setSizeUndefined(); // Take minimum space
titleBar.addComponent(titleComment);

... build rest of the layout ...
```

ИТМО ВТ

Декларативная конфигурация разметки

```
<vaadin-vertical-layout>
    <vaadin-label>
        The Ultimate Cat Finder
    </vaadin-label>

    <vaadin-horizontal-layout>
        <vaadin-radio-button-group
            caption="Possible Places"/>
        <vaadin-panel/>

        (...)

    </vaadin-horizontal-layout>
</vaadin-vertical-layout>
```

ИТМО ВТ

Компоненты Vaadin

```
public class MyHierarchicalUI extends UI {
    @Override
    protected void init(VaadinRequest request) {
        // The root of the component hierarchy
        VerticalLayout content = new VerticalLayout();
        content.setSizeFull(); // Use entire window
        setContent(content);   // Attach to the UI

        // Add some component
        content.addComponent(new Label("<b>Hello!</b> - How are you?",
                ContentMode.HTML));

        Grid<Person> grid = new Grid<>();
        grid.setCaption("My Grid");
        grid.setItems(GridExample.generateContent());
        grid.setSizeFull();
        content.addComponent(grid);
        content.setExpandRatio(grid, 1); // Expand to fill
    }
}
```

ИТМО ВТ

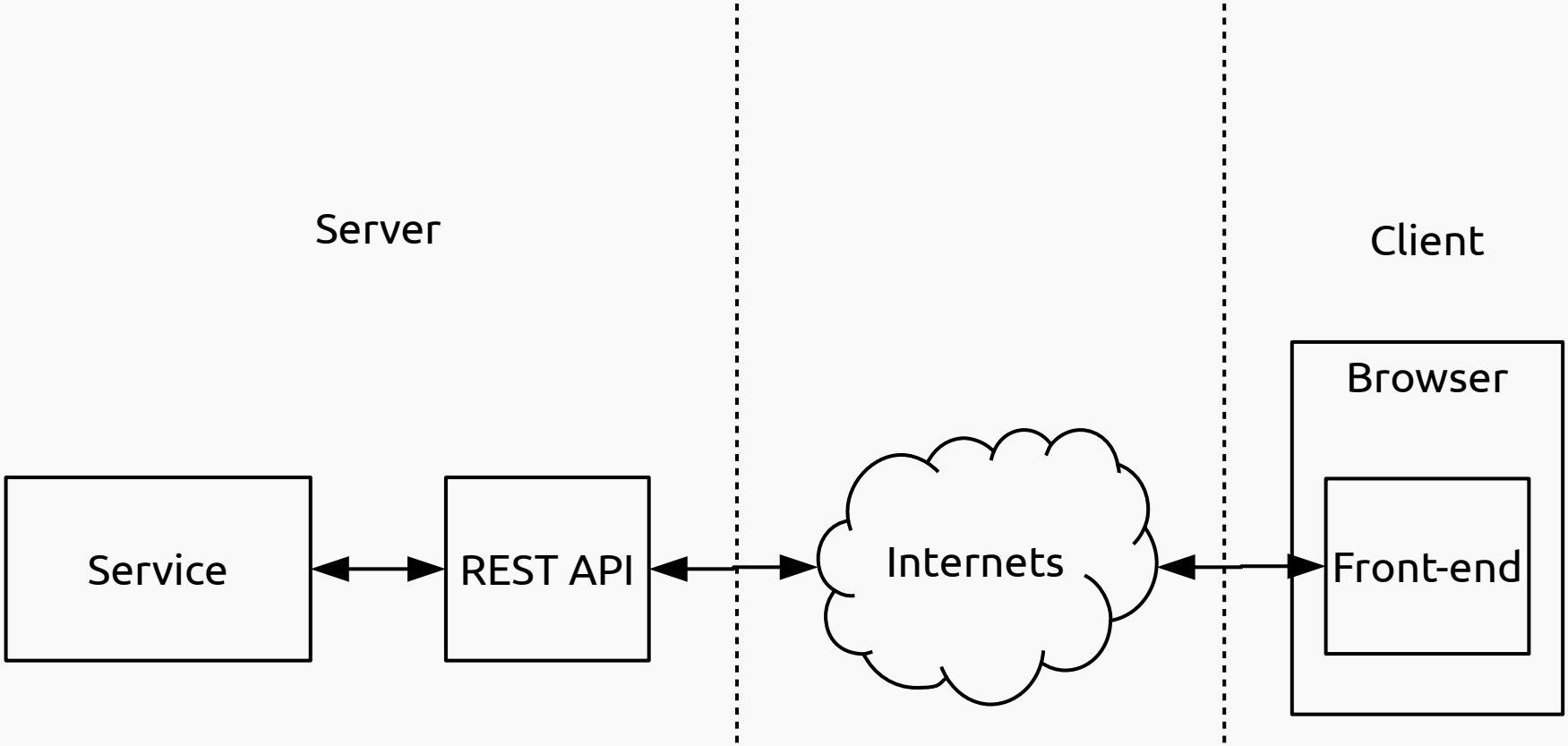
Обработка событий

```
public class Buttons extends CustomComponent {
    public Buttons() {
        setCompositionRoot(new HorizontalLayout(
            new Button("OK", this::ok),
            new Button("Cancel", this::cancel)));
    }

    private void ok(ClickEvent event) {
        event.getButton().setCaption ("OK!");
    }

    private void cancel(ClickEvent event) {
        event.getButton().setCaption ("Not OK!");
    }
}
```

20. JS Frontend Frameworks

ИТМО ВТ

Common Practice



ReactJS + навороты

См. отдельную презентацию.

ИТМО ВТ

Angular

- OpenSource-фреймворк.
- Первая версия выпущена в 2009 г.
- Предназначен для разработки SPA.
- Интерфейс строится из компонентов.
- Компоненты могут рекурсивно вкладываться друг в друга.
- Компоненты могут объединяться в библиотеки.

ИТМО ВТ

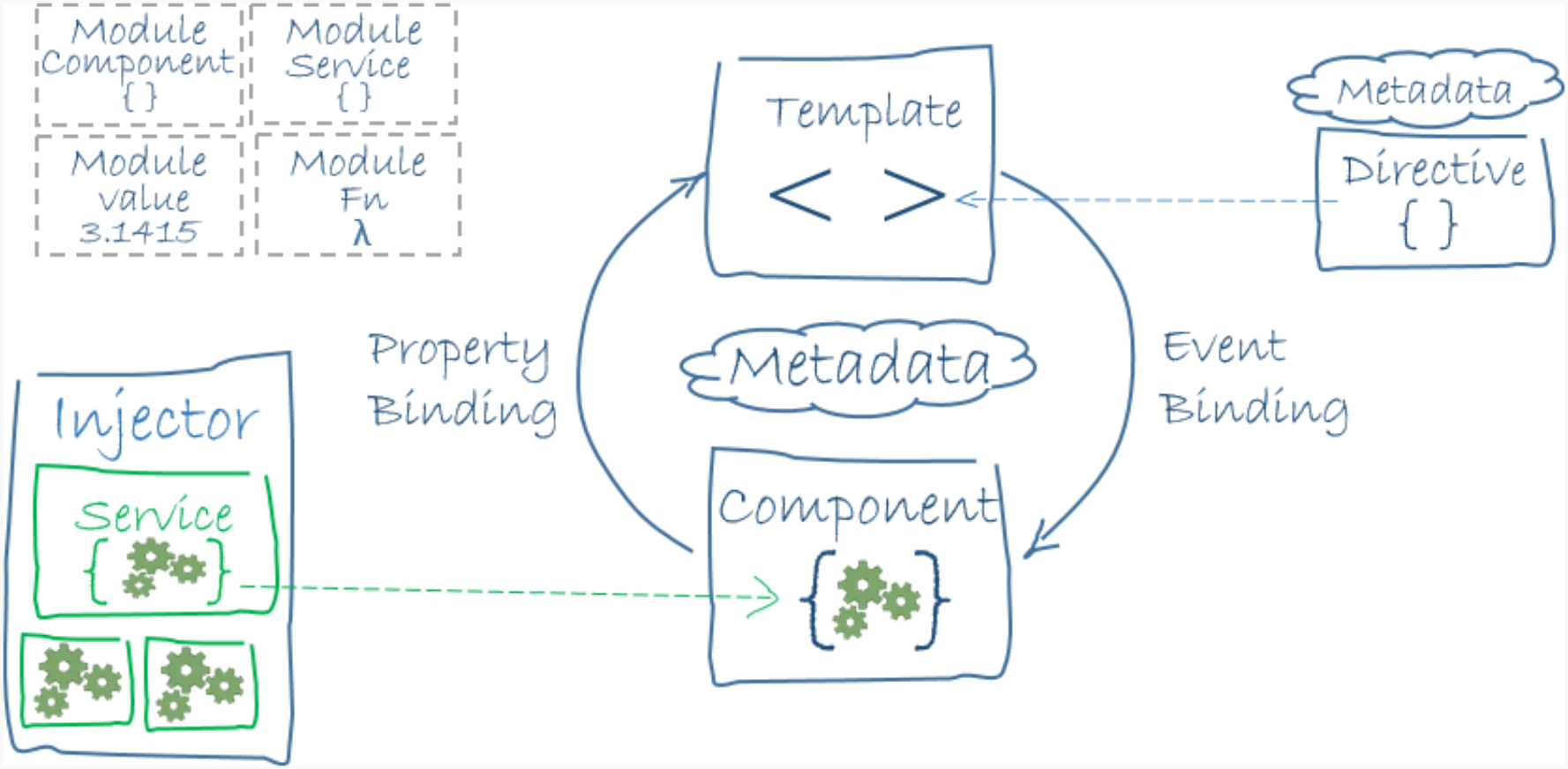
Компоненты Angular

Состоят из шаблона HTML-разметки и класса на JS / TypeScript, управляющего поведением этого блока интерфейса.

```
-----------
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>`
})

export class AppComponent { name = 'Angular'; }

-----------

// index.html
<my-app>Loading AppComponent content here...</my-app>
```

ИТМО ВТ

Архитектура Angular



ИТМО ВТ

Backbone.js

- Очередное «изобретение» MVC.
- Backbone.Model — данные + логика их обработки:

```
var Sidebar = Backbone.Model.extend({
  promptColor: function() {
    var cssColor = prompt("Пожалуйста, введите CSS-цвет:");
    this.set({color: cssColor});
  }
});
window.sidebar = new Sidebar;
sidebar.on('change:color', function(model, color) {
  $('#sidebar').css({background: color});
});

sidebar.set({color: 'white'});
sidebar.promptColor();
```

Backbone.js (продолжение)

- Backbone.View — представление:

```
var DocumentRow = Backbone.View.extend({
  tagName: "li",
  className: "document-row",
  events: {
    "click .icon":          "open",
    "click .button.edit":   "openEditDialog",
    "click .button.delete": "destroy"
  },
  initialize: function() {
    this.listenTo(this.model, "change", this.render);
  },
  render: function() {
    …
  }
});
```