Веб. Конспект первой лекции

Введение в предметную область, протокол HTTP

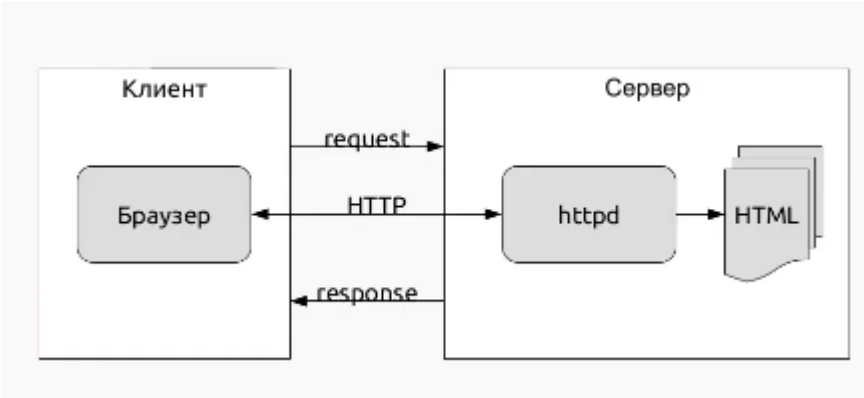
Интернет приложения

- 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

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” все равно используются одни и те же стандарты и протоколы

Сферическое интрнет-приложение в вакууме



Стандарты и протоколы сети Интернет

- Hypertext Transfer Protocol (HTTP) — предназначен для передачи гипертекста между клиентом и сервером
- Hypertext Markup Language (HTML) — язык разметки гипертекста
- Cascade StyleSheets (CSS) — язык описания внешнего вида HTML-документа
- JavaScript (JS) — язык для написания динамических сценариев, выполняемых на стороне клиента

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 идет по схеме **“запрос-ответ”**
- Stateless-протоколв (один запрос — одно соединение). Для реализации сессий используется **cookies**

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)
- URI:
  - <схема>:<идентификатор-в-зависимости-от-схемы>
- URL:
  - http://cs.ifmo.ru/spip.html
  - ../task.shtml
  - mailto:Joe.Bloggs@somedomain.com
- URN:
  - urn:isbn:5170224575
  - urn:sha1:YNCKHTQCWBTRNJIV4WNAE52SJUQCZO5C

Rest

- Representational State Transfer (передача состояние представления) — подход к архитиктуре сетевых протоколов, обеспечивающих доступа к информационным ресурсам.
- Основные концепции:
  - Данные должны передаваться в виде небольшого числа стандартных форматов (HTML, XML, JSON).
  - Сетевой протокол должен поддерживать кэширование, не должен зависеть от сетевого слоя, не должен сохрнаять информацию о состоянии между парами “запрос-ответ”.

- Антипод REST — подход, основанный на вызове удаленных процедур (Remote Procedure Call - RPC).

Структура запроса HTTP

Стартовая строка

Метод URI HTTP/Версия
GET /spip.html HTTP/1.1

Заголовки

Host: cs.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

……..……..……..

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

……..……..……..

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

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

Заголовки HTTP

- Формат:
  ключ:значение
- 4 группы:
  - **General Headers** — могут включаться в любое сообщение клиента и сервера. Пример — CacheControl.
  - **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

Примеры запросов 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...

Основы HTML

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.

Браузеры

- *Браузер* - программа, отображающая HTML-документ в его отформатированном виде.

- Популярные браузеры:
  - Google Chrome
  - Mozilla Firefox
  - Internet Explorer / Edge
  - Apple Safari
  - Opera

Структура HTML-документа

- Документ состоит из *элементов*
- Начало и конец элемента обозначаются *тегами:*
  текст
- Теги могут быть пустыми:

- Теги могут иметь *атрибуты:*
  Здесь элемент содержит атрибут href
- Элементы могут быть вложенными

  Этот текст будет полужирным,
  а этот - еще и курсивным

- 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>)

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>
```

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>

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>
```

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.

Основы CSS

Что такое CSS

- CSS - технология описания внешнего вида документа, написанного языком разметки.
- Используется для задания цветов, шрифтов и других аспектом 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).

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:
    - Стандартных стиль, используемый браузером по умолчанию для представления элементов.

Структура CSS

- Таблица стелей состит из набора **правил**.
- Каждое правило состит из набора **селекторов** и **блока определений:**

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

- Пример:

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

Приоритетеы стилей

- Если к одному элементу “подходит” сразу несколько стилей, применен будет наиболее приоритетеный
- Приоритетеы рассчитываются таким образом (от большего к меньшему):
    1. Свойство задано при помощи **!important;**
    2. Стиль прописан напрямую в теге;
    3. Наличие идентификаторов **(#td)** в селектор;
    4. Количество классов **(.class)** и псевдоклассов **(:pseudoclass)** в селекторе
    5. Количество имен тегов в селекторе
- Имеет значение относительны порядок расположения свойств — свойство, указанное позже, имеет приоритет.

Примеры 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;
}
```

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
```

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 математические операторов

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

- 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

```
//-- 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;
}
```

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;
}
```

Вложеннные 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;
}
```

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;
}
```

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;
}
```

Математика

```
.container { width: 100%; }
article[role="main"] {
 float: left;
 width: 600px / 960px * 100%;
}
aside[role="complementary"] {
 float: right;
 width: 300px / 960px * 100%;
}
```

Компиляция в CSS (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>
```

JavaScript и клиентские сценарии

- JavaScript — объекто-оринетированный 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

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 */

Структура языка

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

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

Блок не ограничивает область видимости функции:

```
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();
```

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

```
var a = 42;

function foo() {
  alert(a);
}
foo();
```

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;");
```

Объектная модель браузера (BOM)

- BOM — прослойка между ядром и DOM (API для взаимодействия с браузером).
- 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("https://cs.ifmo.ru", "contents");
```

- Возможности BOM:
  - управление фреймами (<frame></frame>);
  - поддержка задержки в исполнении кода и зацикливания с задержкой(setInterval, setTimeout)
  - системные диалоги(alert, error);
  - управление адресом открытой страницы
  - управление информацией о браузере
  - управление информацией о параметрах монитора
  - ограниченное управление историей просмотра страниц
  - поддержка работы с HTTP cookie (порции данных, которые хранятся в браузере)

Объектная модель документа (DOM)

- С помощью JavaScript можно производить следующие манипулции:
  - получаение узлов:

```
document.all(”image1”).outerHTML;
```

  - изменение узлов (добавить новый узел)
  - изменение связей между узлами (переместить картинку)

- 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?');
    };
};
```

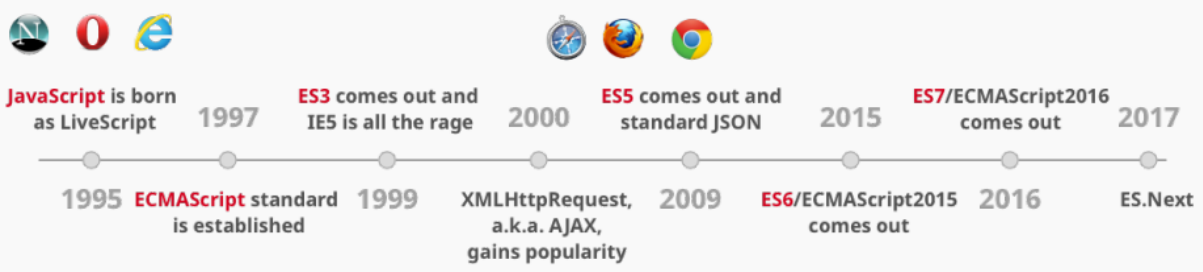
.....

```
<a href="delete.php" id="alertLink">Удалить</a>
```

- В отдельном файле:

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

ES6/ES2015+



- ES6 — новая версия языка ECMAScript, выпущенная в 2015 г.
- Добавляет в синтаксис языка множество новых возможностей
- Поддерживается всеми современными браузерами
- Для работы старых браузерах может потребоваться специальная программа — транспилер (transpiler) (Пример: Babel,

Block Scope Variables

- Два новых ключевых слова — **let** и **const** (let - область видимости в блоке)
- 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
```

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”):

```
{
 let private3 = 1;
}
// Uncaught ReferenceError
console.log(private3);(function(){
 var private2 = 1;
})();
// Uncaught ReferenceError
console.log(private2);
```

- ES6:

```
{
 let private3 = 1;
}
// Uncaught ReferenceError
console.log(private3);
```

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}.`);
```

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
```

Дестркутуризация — 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
```

Дестрктуризация 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
```

Деструктуризация и сопоставления 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
```

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
```

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();
```

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);
});
```

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}`);
```

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);
}
```

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
```

Rest-параметры

- 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);
```

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]);
```

DHTML и AJAX

DHTML

- Dynamic HTML — способ создания интерактивного веб-сайта, использующий сочетания:
  - статичного языка разметки HTML;
  - выполняемого на сторона 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>
```

Что такое AJAX

- AJAX (Asynchronous Javascript and XML) — подход к построению интереактивных 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

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

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);
 }
}

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
```

```
 }
}
```

Преимущества и недостатки AJAX

- Преимущества
  - экономия трафика (на поверхности неправда, в случае неиспользования большие запросы включают повторяющиеся данные)
  - уменьшение нагрузки на сервер (более равномерной)
  - ускорение реакции интерфейса
- Недостатки
  - отсутствие интеграции со стандартными инструментами браузера (кнопка назад в браузере)
  - динамические загружаемое содержимое недоступно поисковиками
  - старые методы учеты статистики сайтов становятся неактуальными
  - усложнение проекта
  - требуется включенный JavaScript в браузере

Протокол WebSocket

- WebSocket — протокол полнодуплескной 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 клиета
- Обмен данными ведется через отдельное TCP-соединение
- Поддерживается всеми соверменными браузерами (даже IE)
- Альтеранатива — AJAX + Long Polling

```
<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>
```

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 ее синоним **jQuery**

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>" );
```

jQuery: работы 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." );
});
```

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
    });
});
```

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));
     }
   });
```

Серверные сценарии

Веб-сайты и веб-приложения

- Веб-сайт — это набор статических файлов, HTML-страниц, графики и другиъ ресурсов
- Веб-приложение — это веб-сайт с той или иной динамической функциональностью на стороне сервера
- Веб-приложение осуществляет вызов программ на стороне севреа, к примеру:
    - Браузер отправляет на веб-сервер запрос на получение HTML-формы
    - Веб-сервер формирует HTML-форму и возвращает ее 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

Технологии для создания веб-приложений

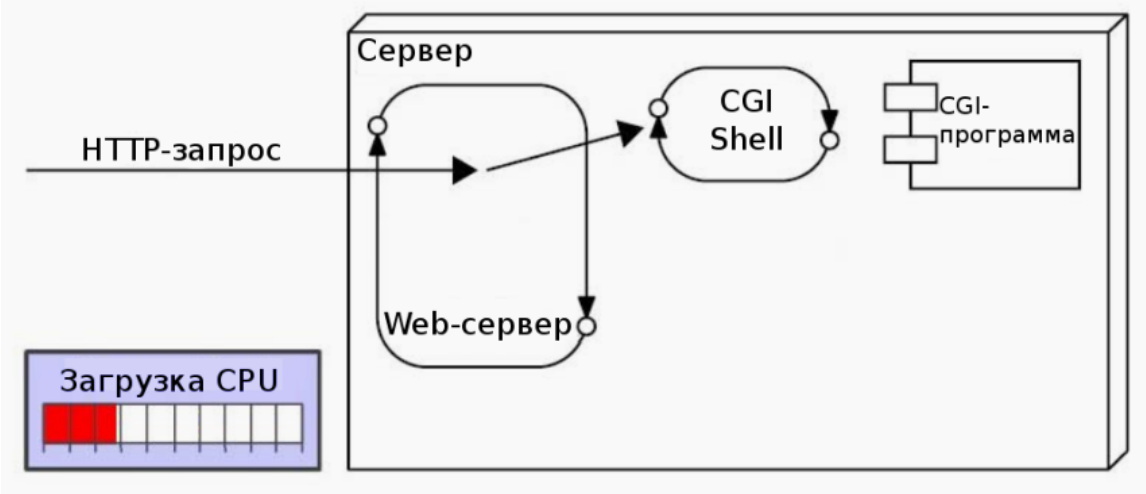
- HTML over HTTP
- Common Gateway Interface (CGI)
- FastCGI
- PHP
- Servlets
- JavaServer Pages (JSP)
- XML
- Struts
- JavaServer Faces

CGI-сценарии

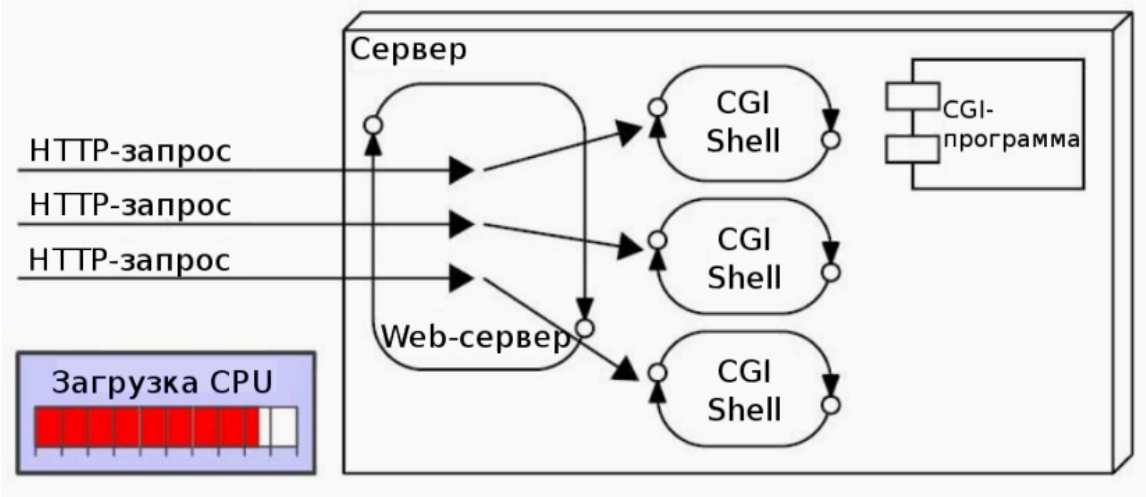
- CGI — механизм вызова пользователем программ на стороне сервера
- Данные отправляются программе посредством HTTP-запроса, формируемого веб-браузером
- То, какая именно программа будет вызывана, обычно определяется URL-запроса
- Каждый запрос обрабатывается отдельным процессом CGI-программы
- Взаимодействие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

Выполнение CGI-сценариев

- Один запрос:



- Параллельная обработка нескольких запросов:



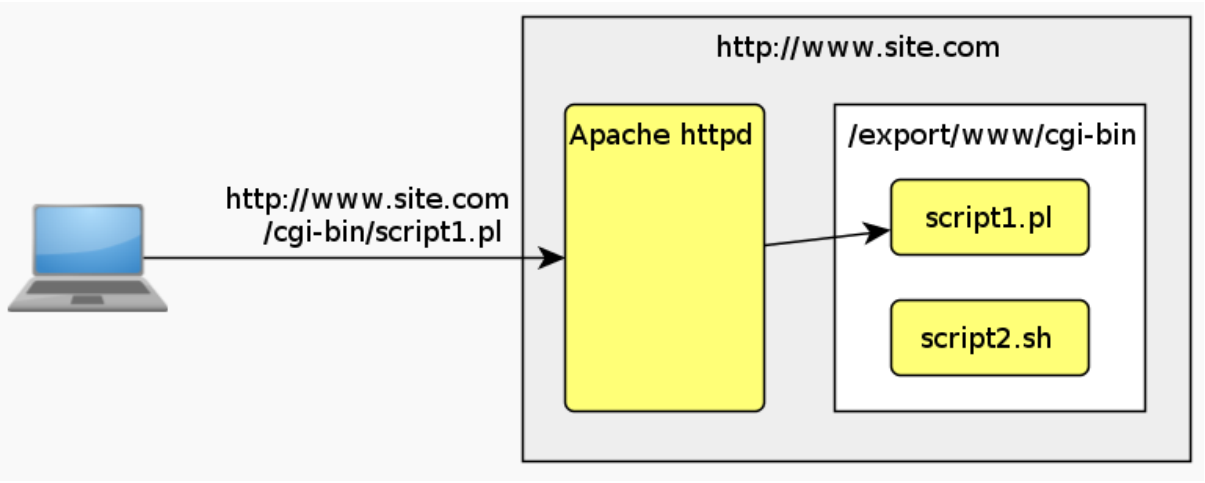
Пример реализации CGI-сценария

```
#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/”
```



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 масштабируеомсть
  - Слабое разделение уровня представления и бизнес-логики
  - Могут быть платформо-зависимыми

FastCGI

- Развитие технологии CGI.
- Все запросы могут обрабатываться одним процессом CGI-программы (фактическая реализация опрееделяется программистом)

- Веб-сервер взаимодействуеит с процессом через UNIX Domain Sockets или TCP/IP (а не через stdin и stdout)

Серверные сценарии на 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

Синтаксис PHP

- Интерпретатор выполняет код, находящийся внутри ограничителей:

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

- Имена переменных начинаются с символы “$”:

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

- Инструкции разделяются символов “;”:

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

- Весь текст вне ограничителей оставляется интерпретатором без изменений:

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

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**

Суперглобальные массивы (Superglobal Arrays)

Преопределенные массивы, имеющие глоабльную область видимости:

- $_GLOBALS — массив всех глобальных переменных
- $_SERVER — параметры, которые ОС передает серверу при его запуске
- $_ENV — переменные среды ОС
- $_GET, $_POST — параметры GET- и POST-запроса:

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

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

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 защищенными (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 “::”

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
```

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;
   /* ... */
}
?>
```

Конфигурация и варианты использования интерпретатор 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, php для Apahche)
  - С помощью CGI / FastCGI
  - Через интерфейс командной строки