# Internet технологии

ЛЕКЦИЯ №5-6

ТЕХНОЛОГИЯ AJAX

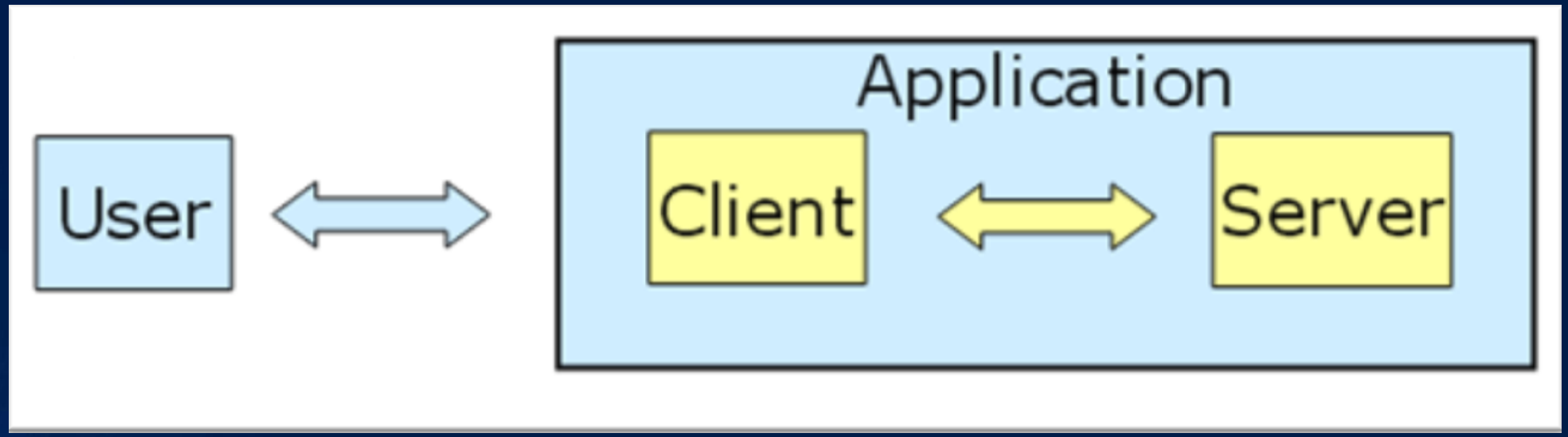
# *Содержание*

- Коммуникационные модели для построения Web-приложений.
- Технология AJAX.
- Объект XmlHttpRequest, его методы и свойства.
- jQuery.
- Формат данных XML.DOM свойства XML-документа.
- Примеры использования XML.
- Формат данных JSON. Примеры использования JSON.

Задача:

- избежать перезагрузки страницы при наступлении элементарных событий и сохранить взаимодействие с сервером.

# Организация взаимодействия



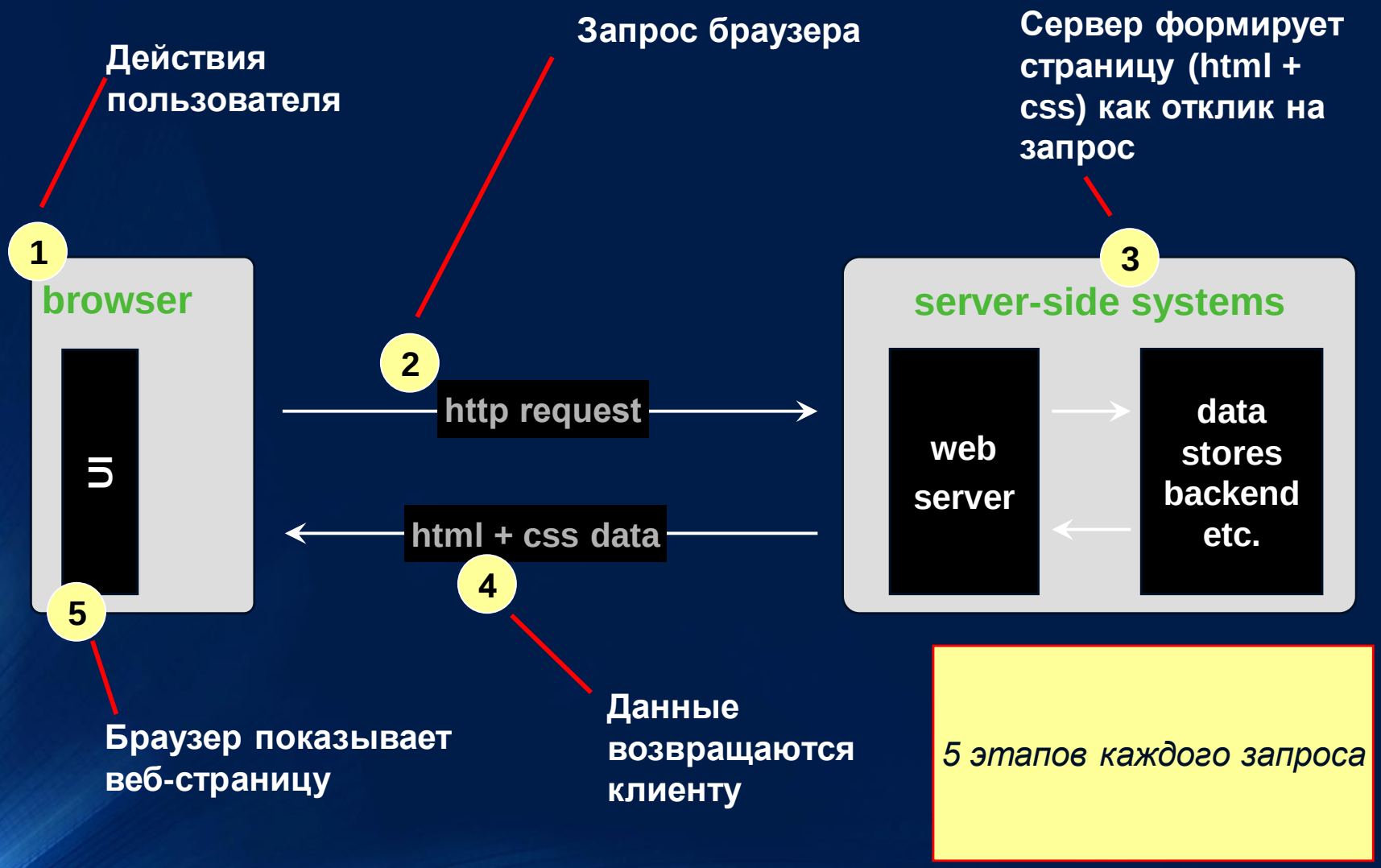

Этапы взаимодействия:

- коммуникация между пользователем и веб-приложением
- коммуникация между клиентской и серверной частями веб-приложения

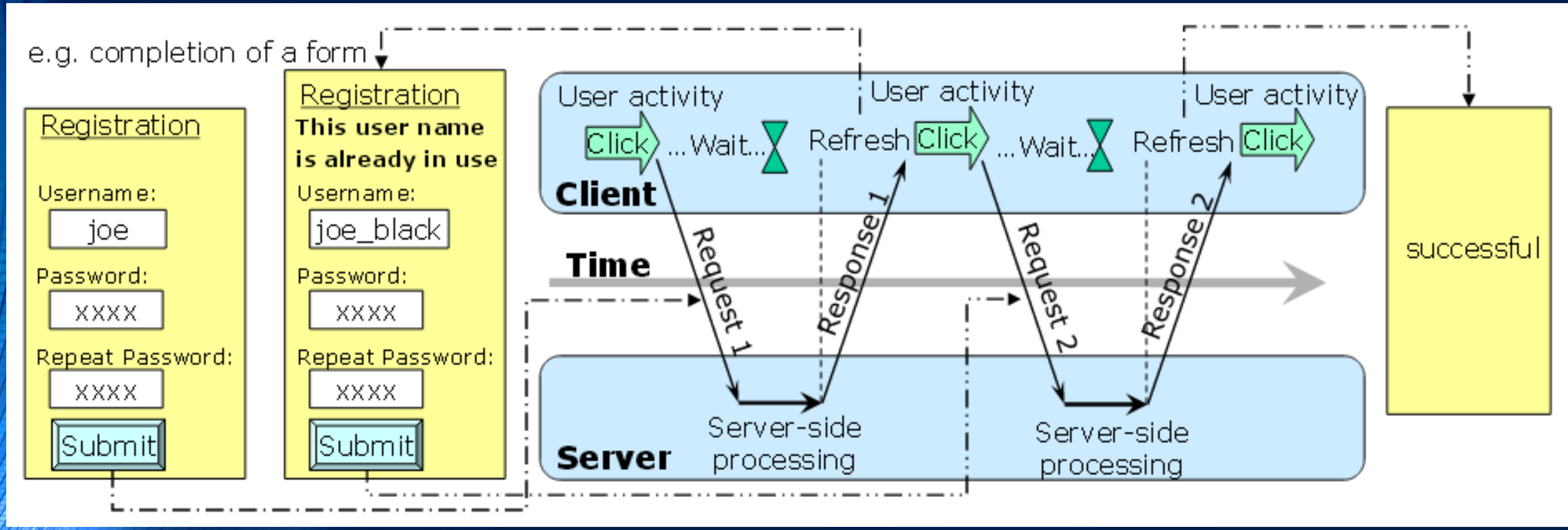
# *Модели коммуникации*

| Модель коммуникации | Способ отправки данных в соответствии с действиями пользователя |
|---|---|
| Синхронная модель –> (нажмите, подождите, загрузите) | Синхронный (активность пользователя блокируется) |
| Не блокирующая модель –> Ajax | Асинхронный (активность пользователя не блокируется) |
| Асинхронная модель –> Comet via long-polling, WebSockets | Асинхронный (активность пользователя не блокируется) |

# Модели коммуникации. Синхронная модель



# Синхронная модель. Временная диаграмма



# *Синхронная модель. Недостатки*

Преимущество – **предсказуемость**.

Недостатки классической синхронной модели:

- **Низкая производительность** из-за «нажмите, подождите, и загрузите снова».
- Чрезмерные **нагрузки на сервер** и потребление пропускной способности в связи с избыточным обновление страниц.
- Потеря контекста операции во время обновления страницы.

Результат: низкая производительность и интерактивность.

# *Асинхронная модель. История*

1996 – IFRAME (IE3, Microsoft)

1997 – LAYER (Netscape)

1998 – Microsoft Remote Scripting

1999-2000 – Outlook Web Access

2002 – **XMLHttpRequest** – это JavaScript API для передачи данных в виде сообщений между Web-браузером и Web-сервером.

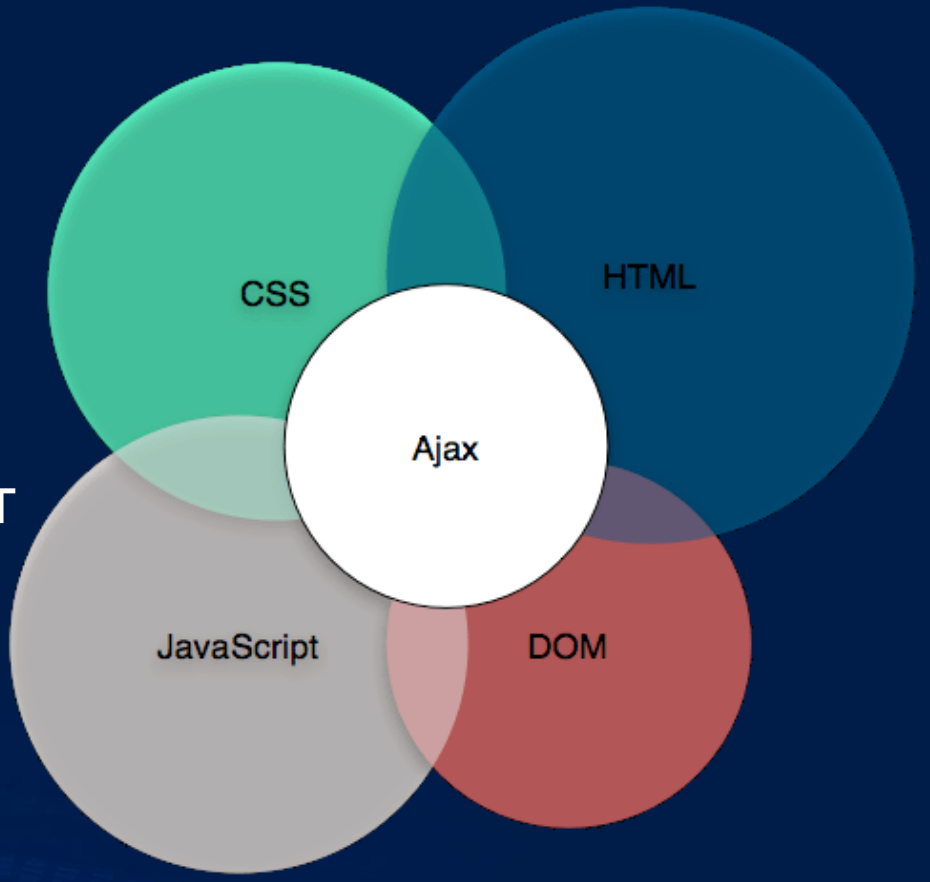
2003 – ASP.NET Callbacks (Microsoft)

2005 – AJAX

# AJAX

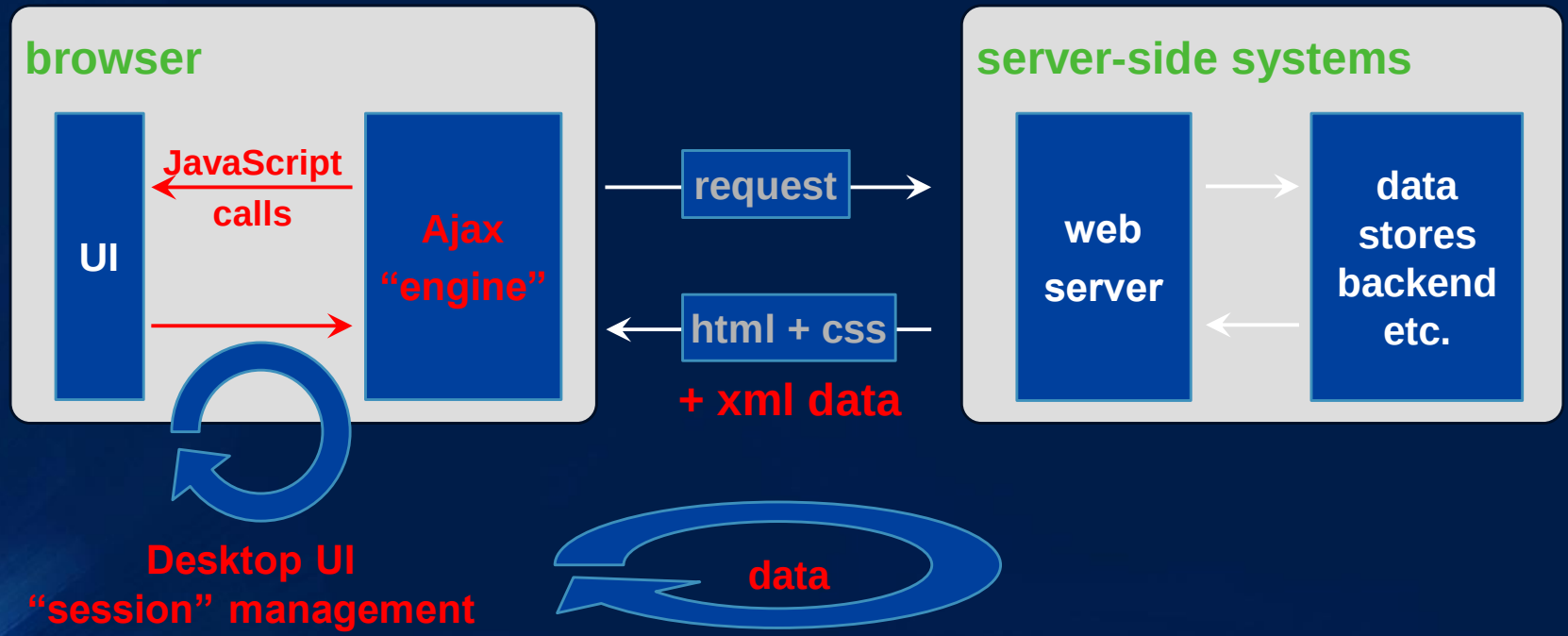
**AJAX** – Asynchronous Javascript And XML – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером, что позволяет избежать полной перезагрузки веб-страниц.

- **XHTML + CSS** – для представления и стилизации информации.
- **DOM + JavaScript** – динамическое отображение и взаимодействие.
- Объект **XMLHttpRequest** – обеспечивает асинхронный обмен данными.
- **Plain Text, HTML, XML/JSON** – форматы обмена данными.



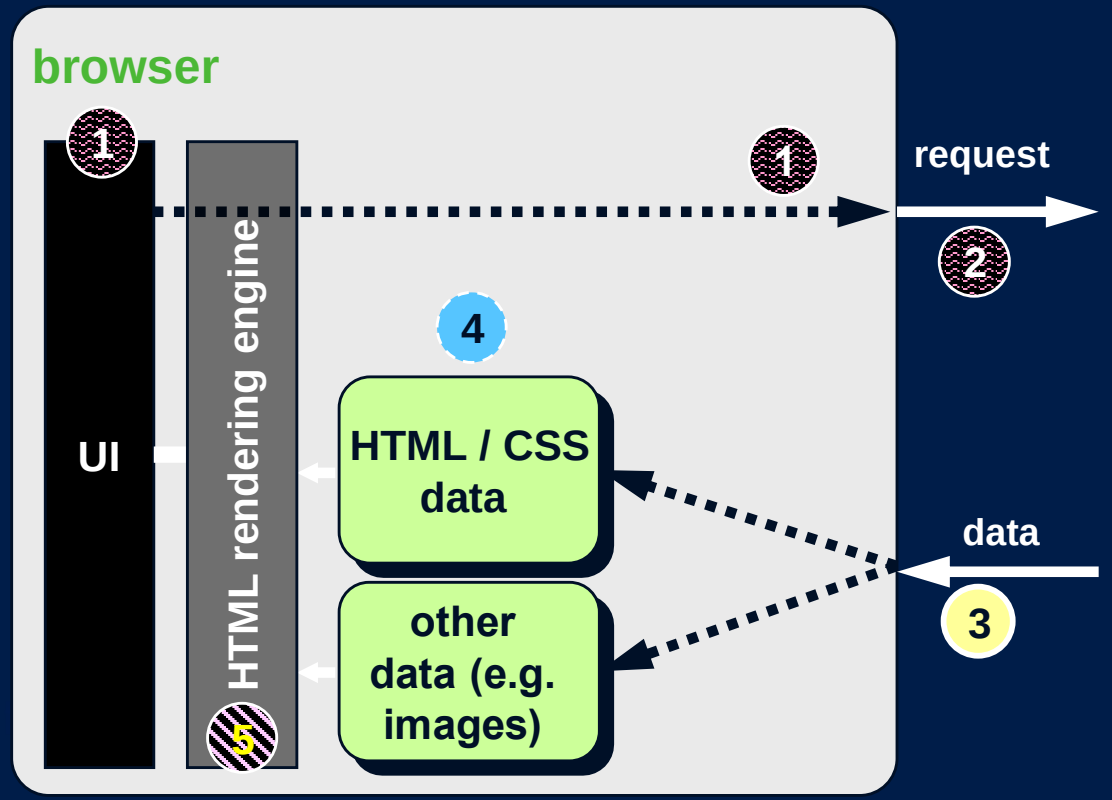
# AJAX

**Ajax** добавляет дополнительный уровень функциональности к браузеру, что позволяет управлять взаимодействием элементов управления с пользователем, упрощая передачу данных между клиентом и сервером.



# Отображение HTML-страниц в браузере

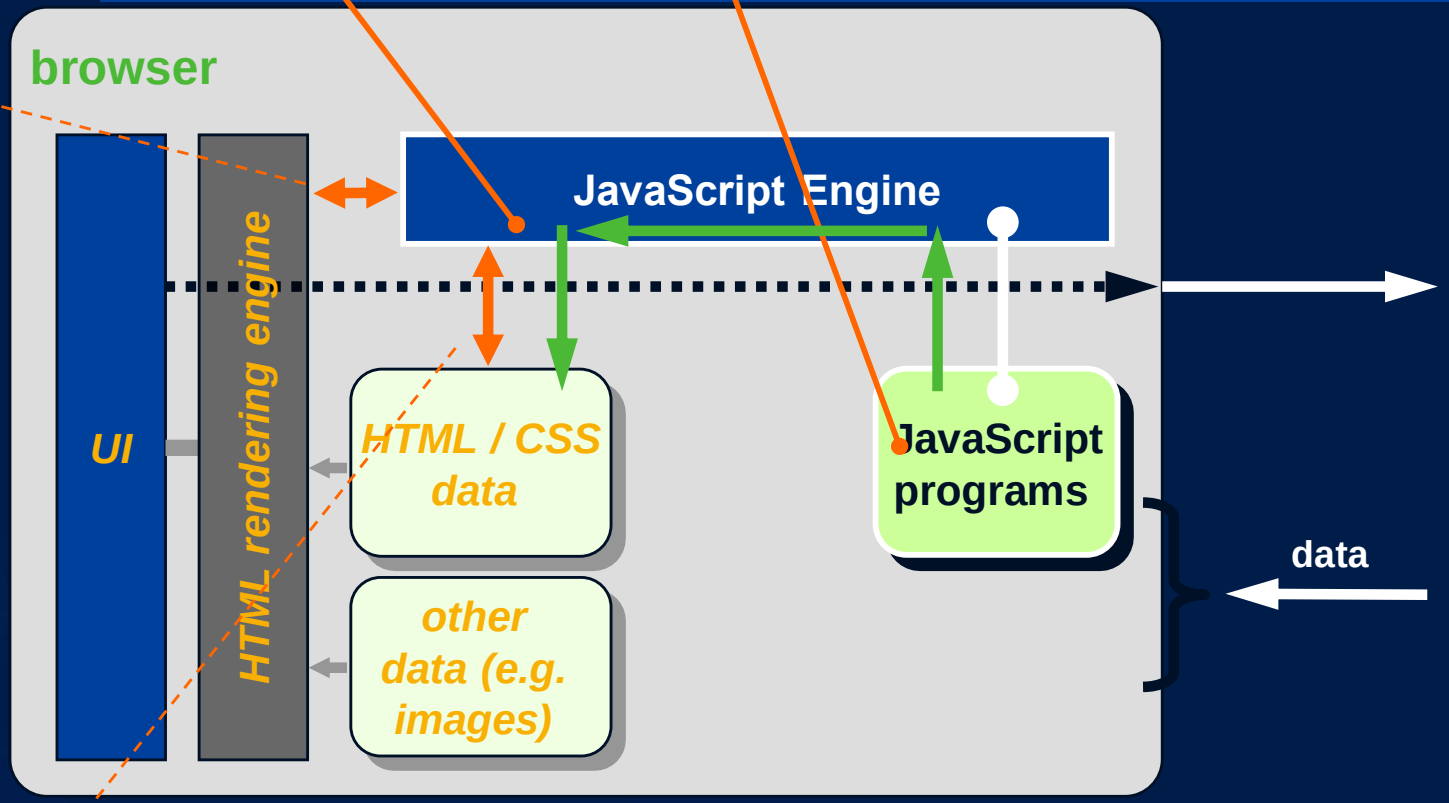
1. Пользователь выбирает ссылку
2. Браузер формирует запрос
3. Сервер присылает ответ
4. HTML and CSS поступают в механизм рендеринга, который посылает
5. дополнительные запросы на получение других данных (картинки и т.д.)



# Отображение HTML-страниц в браузере. Javascript

**Модуль Javascript** – выполняет действия над HTML, CSS и изображениями.

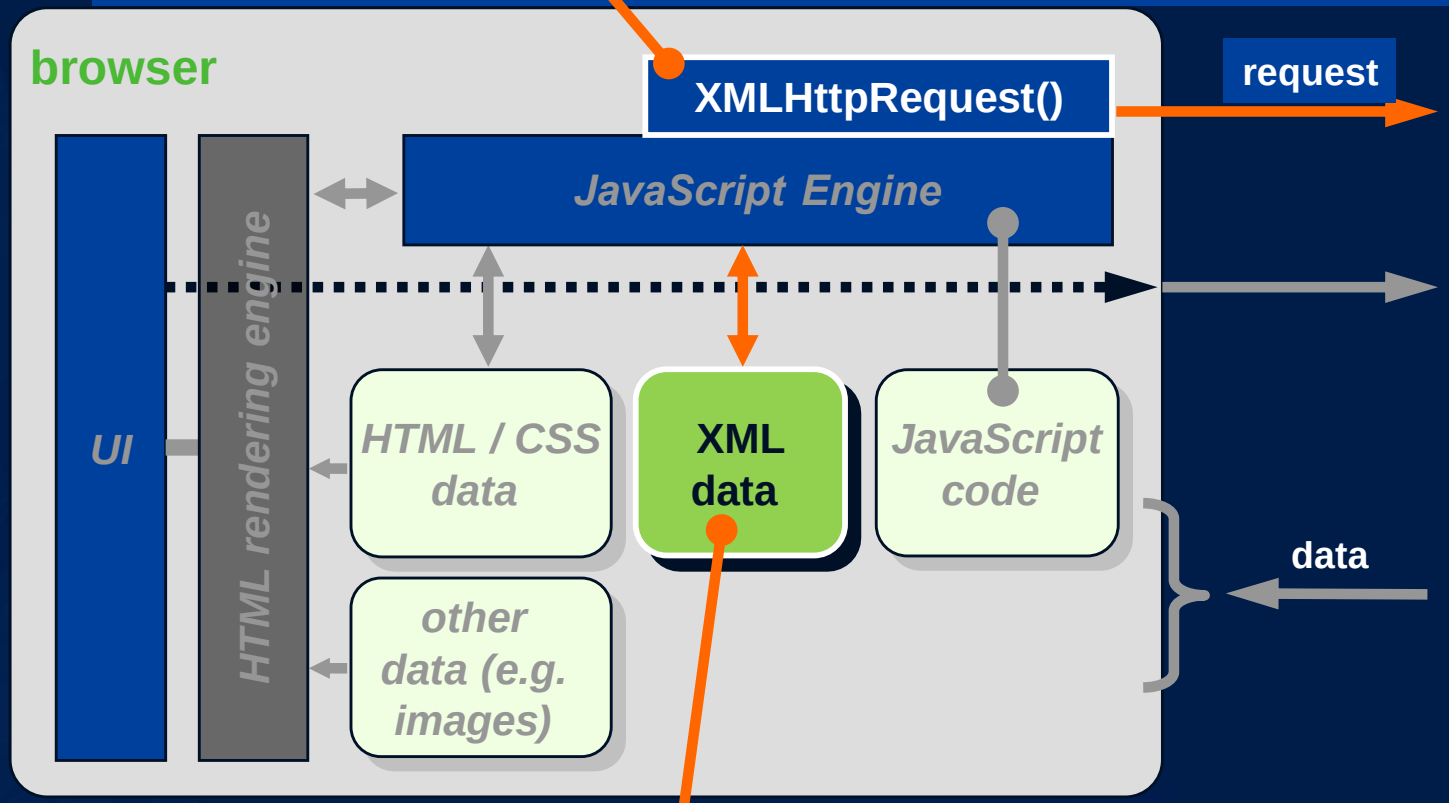
Блоки **JavaScript** реагируют на действия пользователя и исполняют соответствующие фрагменты кода




Блоки **JavaScript** могут **получать доступ и изменять данные** HTML / CSS, динамически меняя пользовательский интерфейс.

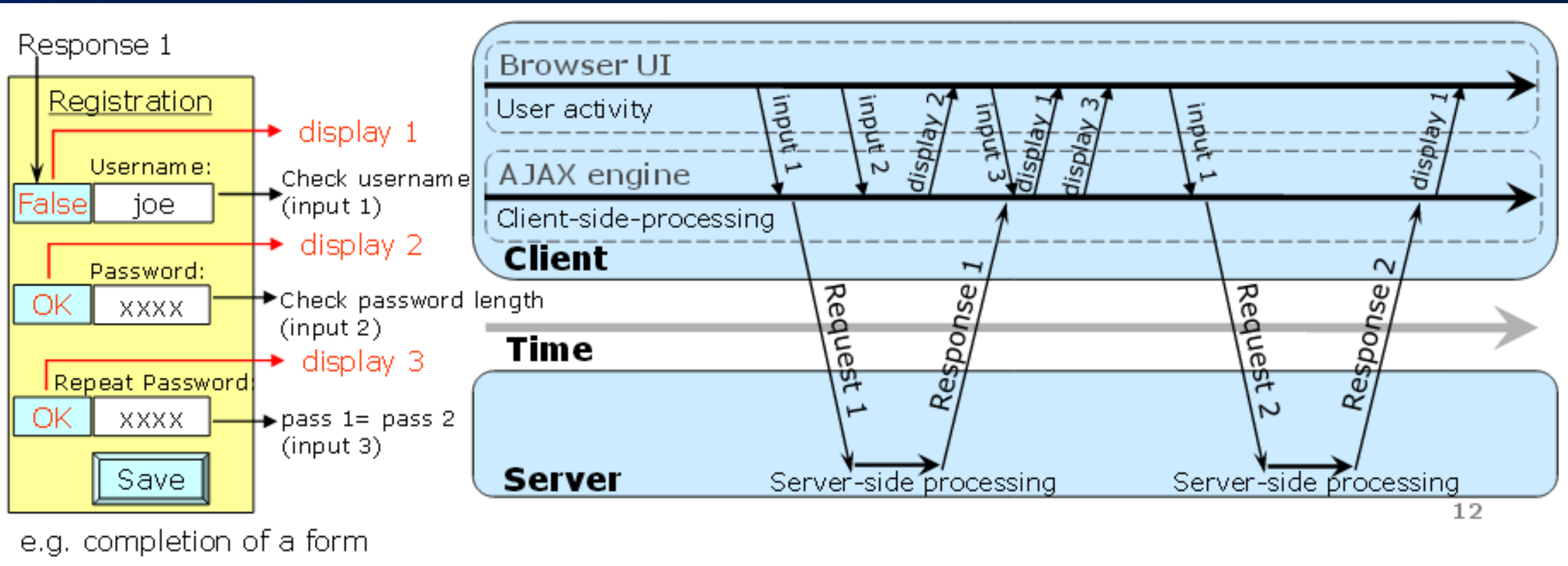
# Отображение HTML-страниц в браузере. AJAX и JS

**Новая функция JavaScript.** Использование XMLHttpRequest позволяет JavaScript блокам посылать запросы данных (изображения, XML, HTTP) вне зависимости от пользователя.



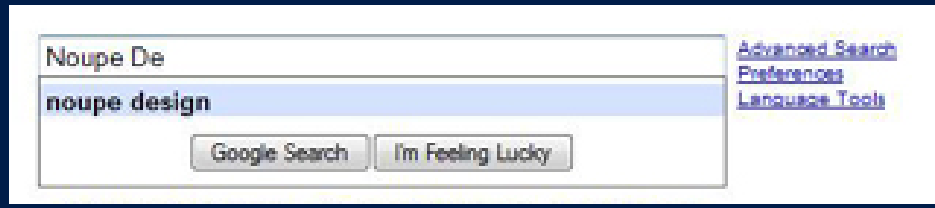

**Поддержка данных XML**. Браузер теперь может хранить XML данные и получать к ним доступ методами JavaScript.

# AJAX. Временная диаграмма



# AJAX. Сферы использования

- Форма авторизации, проверка и отправка форм (jQuery Form Plugins)
- Автозаполнение (Google Search )
- Голосование и рейтинги (Reddit)
- Небольшие элементы управления
- Обновление пользовательского контента (Twitter)
- Внешние виджеты (Google Adsense )
- Постоянная подгрузка данных с сервера (чаты, форумы)
- Использование AJAX вместе с Flash (Kongregate )

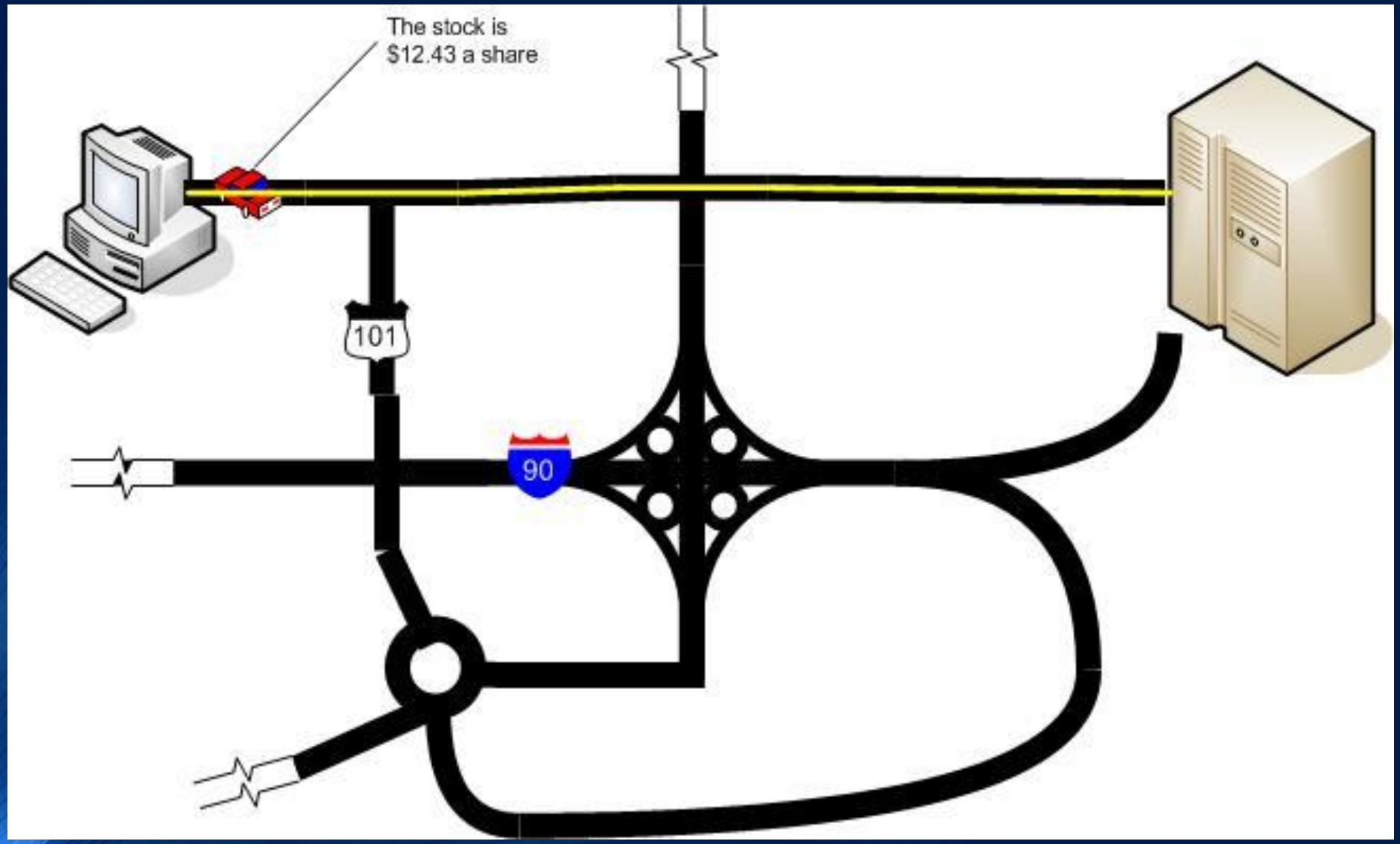

# *AJAX. Недостатки*

- **Отсутствие интеграции** со стандартными инструментами браузера. Динамически создаваемые страницы не регистрируются браузером в истории посещения страниц, поэтому не работает кнопка «Назад», предоставляющая пользователям возможность вернуться к просмотренным ранее страницам. Сложность сохранения **закладки** на желаемый материал.
- Динамически загружаемое содержимое ***недоступно поисковикам***, разработчики должны позаботиться об альтернативных способах доступа к содержимому сайта.
- Сложность **учёта статистики**.
- **Усложнение проекта**. Асинхронная модель сложнее для отладки. Усложнена обработка ошибок коммуникации (разрыв связи, и т.п.) и пользовательских ошибок (например, не хватило привилегий).
- **Требуется включенный JavaScript** в браузере.

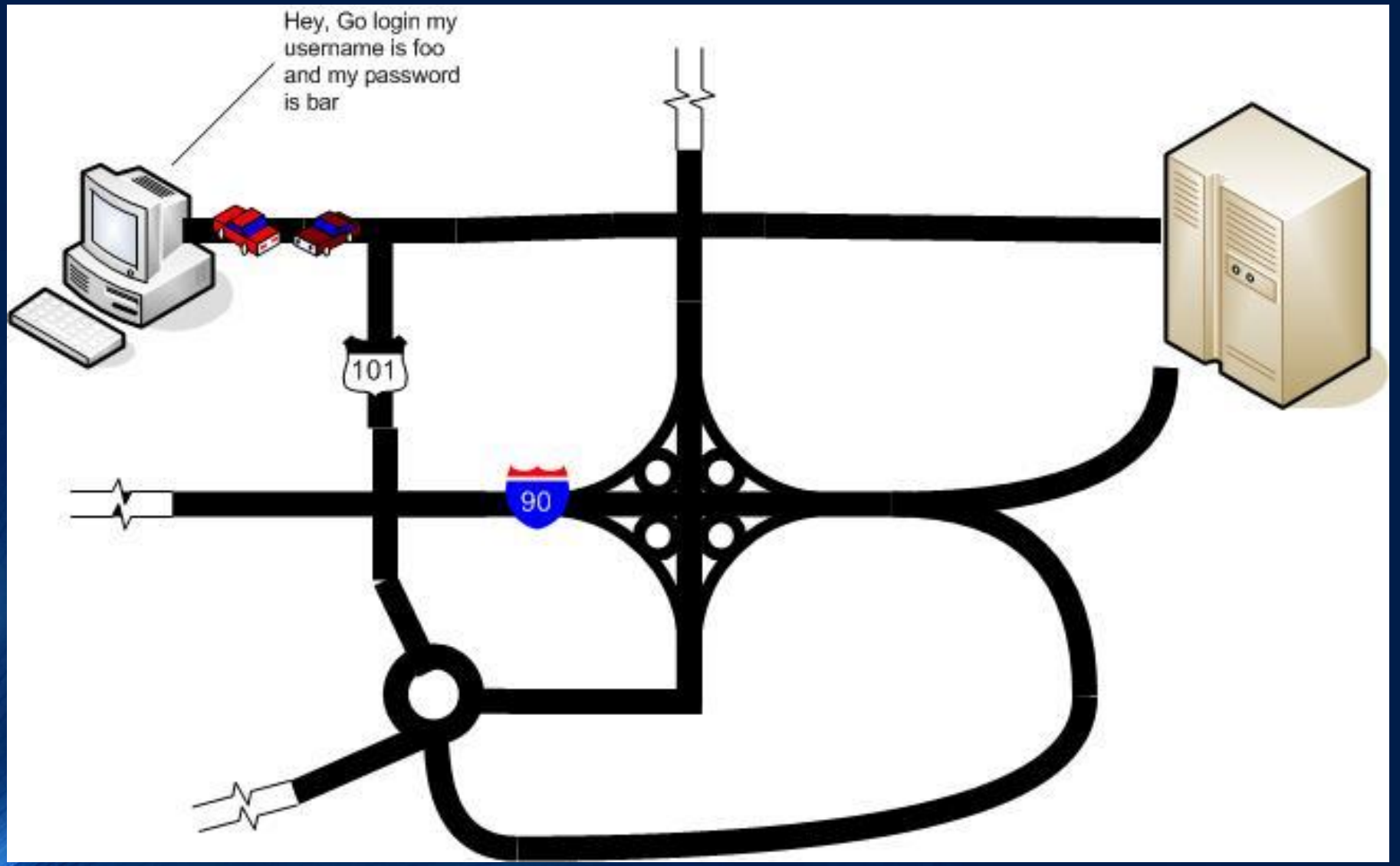
# *Особенности AJAX*

- ✓ **Race conditions**
  - Неопределенна последовательность выполнения
  - Можно делать много одновременных задач, но задача, начатая первой, может окончиться последней.
- ✓ Реакция тут же, но неизвестно, какой будет результат.
- ✓ **Интерактивность**
- ✓ **Быстрый интерфейс**

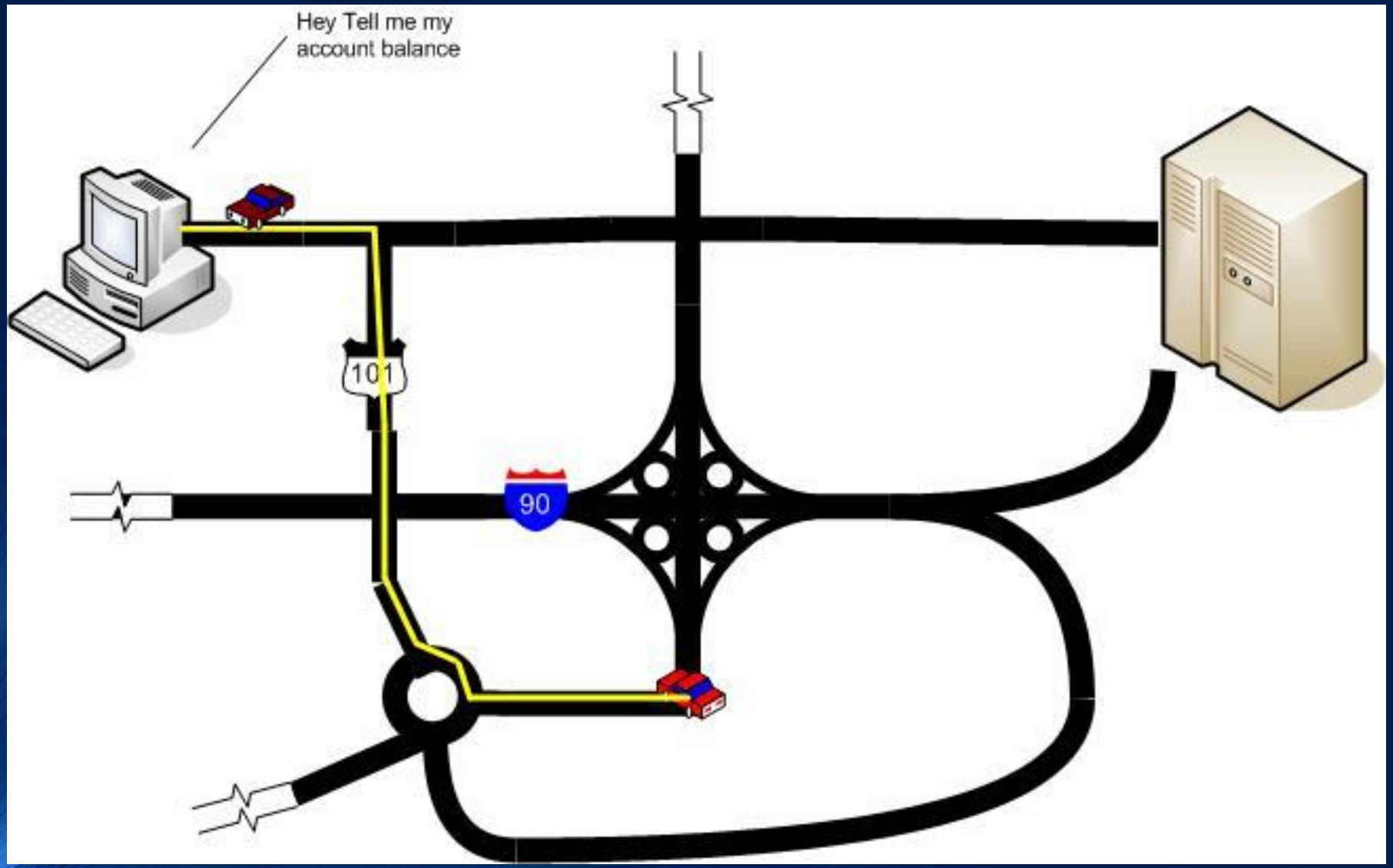
# AJAX. Race conditions



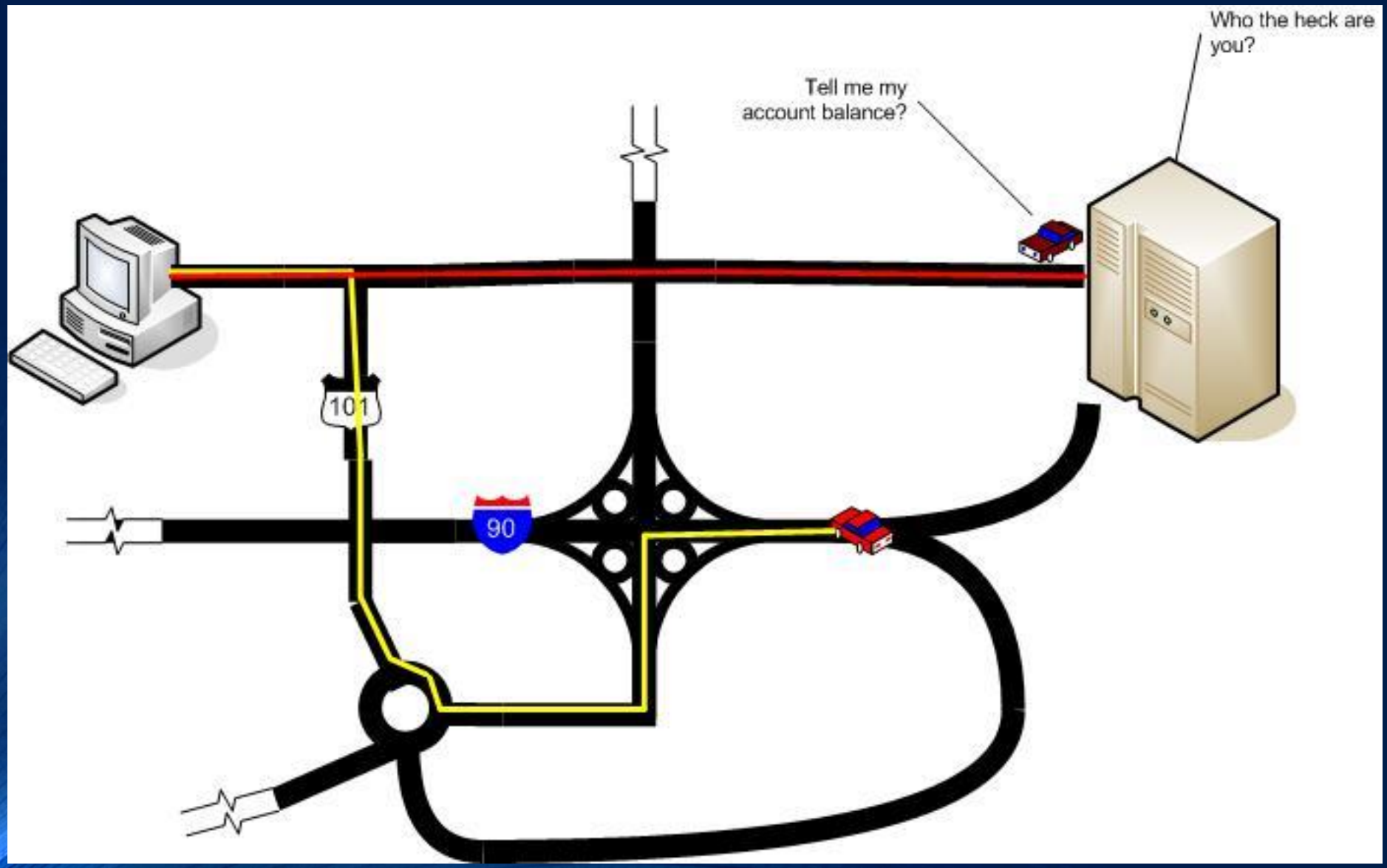
# AJAX. Race conditions



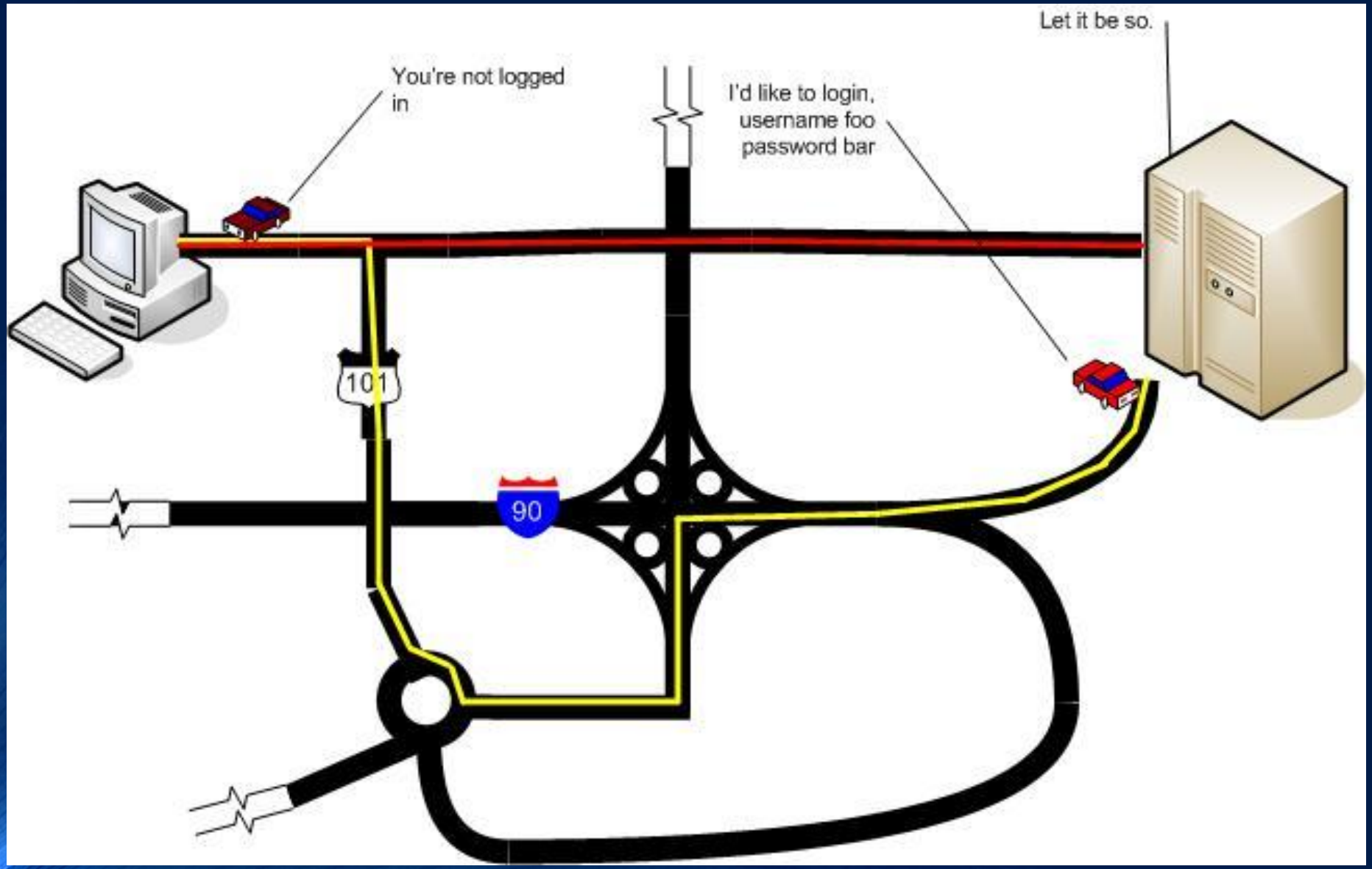
# AJAX. Race conditions



# AJAX. Race conditions



# AJAX. Race conditions



# *Этапы выполнения AJAX-запроса*

1. Функция создает объект **XmlHttpRequest**
2. Назначает обработчик ответа сервера **onreadystatechange**
3. Открывает соединение **open**
4. Отправляет запрос вызовом **send**
5. Ответ сервера принимается срабатывающей в асинхронном режиме функцией-обработчиком события **onreadystatechange**

# XmlHttpRequest. Создание объекта

```
var ajax; // глобальная переменная для хранения обработчика запросов
InitAjax();

function InitAjax() {
  try {    /* пробуем создать компонент XMLHTTP для IE старых версий */
    ajax = new ActiveXObject("Microsoft.XMLHTTP");
  } catch (e) {
    try {//XMLHTTP для IE версий >6
      ajax = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (e) {
      try {// XMLHTTP для Mozilla и остальных
        ajax = new XMLHttpRequest();
      } catch(e) {  ajax = 0;   }
    }
  }
}
```

# XmlHttpRequest. Свойства для отправки запроса

- **onreadystatechange** – ссылается на функцию-обработчик состояний запроса. *В некоторых браузерах функция имеет аргумент-событие. Не используйте его, он совершенно лишний*.
- **readyState** – номер состояния запроса от 0 до 4.

  0: **UNSENT** – объект только что был создан.
  1: **OPENED** – был успешно вызван метод open() данного объекта.
  2: **HEADERS_RECEIVED** – заголовки ответа были успешно загружены.
  3: **LOADING** – тело ответа загружается.
  4: **DONE** – запрос был выполнен, но **неизвестно** – успешно или нет (информацию о результате выполнения запроса можно получить с помощью стандартных статусов и заголовков HTTP-ответа).

# *XmlHttpRequest. Методы для отправки запроса*

- **open(method, URL, async, userName, password)** – открывает соединение.
- **send()** – отсылает запрос. Аргумент – тело запроса. Например, GET-запроса тела нет, поэтому используется send(null), а для POST-запросов тело содержит параметры запроса.
- **abort()** – вызов этого метода обрывает текущий запрос.
- **setRequestHeader(name, value)** – устанавливает заголовок name запроса со значением value. Если заголовок с таким name уже есть – он заменяется.

  Например:

```
ajax.setRequestHeader('Content-Type', 'application/x-www-form-
    urlencoded');
```

# *XmlHttpRequest. Открытие соединения*

open( method, URL )
open( method, URL, async )
open( method, URL, async, userName )
open( method, URL, async, userName, password )

- **metod** – метод передачи данных.
- **url** – запрошенный URL-адрес.
- **async** – необязательный Boolean-параметр, сигнализирующий о том является ли запрос асинхронным или нет (по умолчанию равен **True**).
- **user** – необязательный параметр, содержащий имя пользователя для аутентификации.
- **password** – необязательный параметр, содержащий пароль, используемый для аутентификации.

# XmlHttpRequest. Синхронный и асинхронный запрос

```
var ajax = getXmlHttp();
ajax.open('GET', '/xhr/test.html', false);
ajax.send(null);
if(ajax.status == 200) {
 alert(ajax.responseText);
}
```

**HTTP-статус**
100 — «Continue»,
404 — «Not Found»,
200 — «OK»,
204 — «No Content»,
302 — «Found» ….

```
var ajax = getXmlHttp();
ajax.open('GET', '/xhr/test.html', true);
ajax.onreadystatechange = function() {
 if (ajax.readyState == 4) {
   if(ajax.status == 200) {
     alert(ajax.responseText);
} }
};
ajax.send(null);
```

**Коды состояний**
0 - Unitialized
1 - Loading
2 - Loaded
3 - Interactive
4 - Complete

# XmlHttpRequest. Отправка данных в GET и POST-запросах

```
request.open("GET",url+"?a1=1&a2=2",true);
request.send ('');
```

```
request.open("POST",url, true);
request.setRequestHeader("Content-type","application/x-www-form-urlencoded");// при POST обязателен заголовок Content-Type, содержащий кодировку. Это указание для сервера – как обрабатывать (раскодировать) пришедший запрос.
request.send("param1=1&param2=2");
```

# *Кодирование данных*

Во время обычной отправки формы **браузер сам кодирует** значения полей и составляет тело GET/POST-запроса для посылки на сервер. При отправке данных через XMLHttpRequest, это нужно делать самим, в javascript-коде – формировать запрос "руками", кодируя поля функцией encodeURIComponent.

Пропускать через encodeURIComponent стоит только те переменные, в которых могут быть спецсимволы или **не** английские буквы.

Указание кодировки при отправке формы:

```
<form method="get"> // метод GET с кодировкой по умолчанию
<form method="post" enctype="application/x-www-form-urlencoded">
//enctype явно задает кодировку
<form method="post"> // метод POST с кодировкой по умолчанию
(urlencoded, как и предыдущая форма)
```

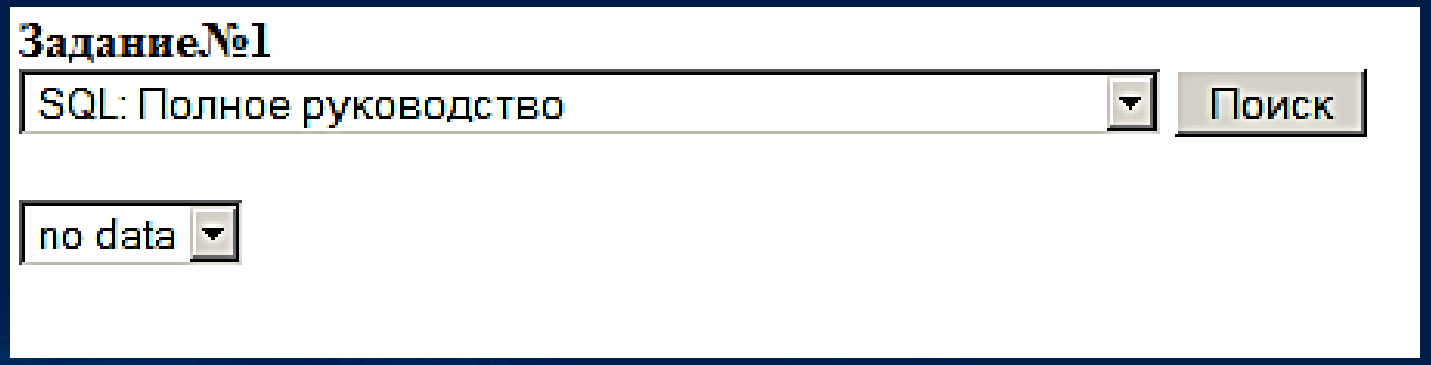
# *XMLHTTPRequest. Методы и свойства для обработки ответа*

- getAllResponseHeaders() – возвращает строку со всеми HTTP-заголовками ответа сервера.
- getResponseHeader(headerName) – возвращает значение заголовка ответа сервера с именем headerName
- **status** – стандартный HTTP-статус запроса (например, в случае успешного выполнения запроса будет возвращено значение 200).
- statusText – строка, содержащая полное описание статуса, возвращенного Web-сервером (например, 304 Not Modified).
- **responseText** – атрибут, в котором хранится текстовое представление тела запроса.
- **responseXML** – атрибут, содержащий XML-представление тела запроса – фрагмент документа с DOM и всеми соответствующими методами.
- overrideMimeType(mimeType) – позволяет указать mime-type документа, если сервер его не передал или передал неправильно.

```
ajax.overrideMimeType('text/xml');
```

# *Пример. Веб-страница*

```
<b>ЗаданиеN1</b>
<form name="form1" method="get">
  <select id="select1" name="name1">
    <option value="0">SQL: Полное руководство</option>
    <option value="1">Из истории культуры средних веков и
Возрождения</option>
    <option value="2">Ежедневный журнал</option>
  </select>
  <input type="button" name="form1submit" value="Поиск" onclick="gets2();"/>
  <select id="select2" >
    <option>no data</option>
  </select>
</form>
```



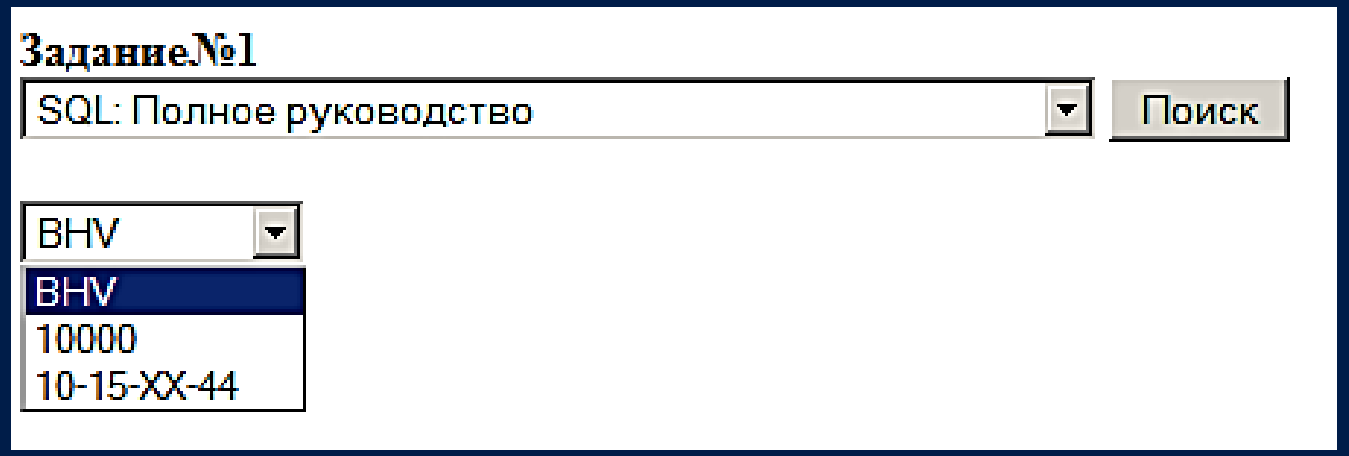
# *Пример. Отправка запроса методом GET*

```
function gets2() {
        if (!ajax) {
                alert("Ajax не инициализирован");
                return;
        }
        var s1val = document.getElementById("select1").value;
        ajax.onreadystatechange = UpdateSelect2;
        var params = 'select1=' + encodeURIComponent(s1val);
        ajax.open("GET", "get.php?"+params, true);
        ajax.send(null);
}
```

# *Пример. Серверная часть.*

```
<?php
  $id = $_GET['select1'];
  switch ($id)
  {
     case "0": echo '<option>BHV</option><option>10000</option><option>10-15-
XX-44</option>';
          break;
     case "1": echo '<option>Science</option><option>1000</option><option>44-
78-106-X </option>';
          break;
     case "2": echo '<option>BHV</option><option>150</option>';
          break;
   }
?>
```

# *Пример. Обработка ответа с помощью responseText*




```
function UpdateSelect2() {
   if (ajax.readyState == 4)  {
         if (ajax.status == 200) {
                  // если ошибок нет
                  var select = document.getElementById('select2');
                  select.innerHTML = ajax.responseText;
         }
         else alert(ajax.status + " - " + ajax.statusText);
         ajax.abort();
   }
}
```

# *jQuery*

jQuery – библиотека JavaScript, позволяющая легко получать доступ к элементу DOM, обращаться к их атрибутам и содержимому, манипулировать ими, создавать анимацию, устанавливать обработчики событий. Также библиотека jQuery предоставляет и удобный API для работы с AJAX.

Библиотека представлена в виде обычного текстового файла, имеющего расширение .js. Сейчас существует две основные ветки версий jQuery 1.x и 2.x. Их отличия заключаются лишь в том, что в версиях 2.x перестали поддерживаться браузеры IE версий 8, 7 и 6.

jQuery доступна в **сжатом** и **несжатом** варианте.

Подключение jQuery:

```
<script type="text/javascript" src="js/jquery-1.6.1.min.js"></script>
```

# *jQuery. Ajax*

jQuery имеет ряд функций, позволяющих обмениваться данными с сервером без перезагрузки страницы (технология ajax).

| | |
|---|---|
| **$.ajax()** | Производит асинхронный ajax-запрос с установленными параметрами. |
| **$.get()** | Производит запрос к серверу методом GET |
| **$.post()** | Производит запрос к серверу методом POST |
| **.load()** | Производит запрос HTML-данных у сервера и помещает их в выбранные элементы страницы |
| **$.getJSON()** | Производит запрос JSON-данных у сервера методом GET |
| **$.getScript()** | Производит запрос файла javascript методом GET, а затем выполняет код из полученного файла. |

# *jQuery.get()*

Осуществляет запрос к серверу методом GET, без перезагрузки страницы. Функция имеет несколько **необязательных** параметров.

**jQuery.get(url,[data],[callback],[dataType])**

url – url-адрес, по которому будет отправлен запрос.

data – данные, которые будут отправлены на сервер. Они должны быть представлены в объектом, в формате: {fName1:value1, fName2:value2, ...}.

**callback(data, textStatus, jqXHR)** – пользовательская функция, которая будет вызвана после ответа сервера.

data – данные, присланные с сервера.

textStatus — статус того, как был выполнен запрос.

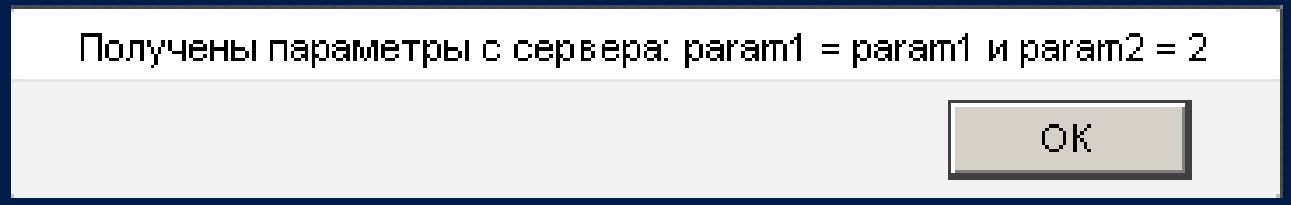
jqXHR – объект jqXHR (в версиях до jquery-1.5, вместо него использовался XMLHttpRequest)

**dataType** – ожидаемый тип данных, которые пришлет сервер в ответ на запрос.

# jQuery.get(). Пример

В этом примере на сервер будет отправлен запрос страницы http://hostname/ajaxtest.php и указаны два параметра. После получения ответа от сервера будет вызвана функция onAjaxSuccess, которая выведет на экран сообщение с данными, присланными сервером.

```
$.get( "/ajaxtest.php", { param1: "param1", param2: 2 }, onAjaxSuccess );
function onAjaxSuccess(data) {
  // Здесь мы получаем данные, отправленные сервером и выводим их на экран.
  alert(data);
}
```




```
// файл http://hostname/ajaxtest.php
<?php
   echo "Получены параметры с сервера: param1 = ".$_GET['param1']." и
param2 = ".$_GET['param2'];
?>
```

# *jQuery.post()*

Осуществляет запрос к серверу методом POST, без перезагрузки страницы. Функция имеет несколько необязательных параметров.

**jQuery.post(url,[data],[callback],[dataType])**

url – url-адрес, по которому будет отправлен запрос.

data – данные, которые будут отправлены на сервер. Они должны быть представлены в объектом, в формате: {fName1:value1, fName2:value2, ...}.

**callback(data, textStatus, jqXHR)** – пользовательская функция, которая будет вызвана после ответа сервера.

data – данные, присланные с сервера.

textStatus — статус того, как был выполнен запрос.

jqXHR – объект jqXHR (в версиях до jquery-1.5, вместо него использовался XMLHttpRequest)

**dataType** – ожидаемый тип данных, которые пришлет сервер в ответ на запрос.

# jQuery.post(). Пример

На сервер будет отправлен запрос страницы http://hostname/ajaxtest.php и указаны два параметра. После получения ответа от сервера будет вызвана функция onAjaxSuccess, которая выведет на экран сообщение с данными, присланными сервером.

```
$.post( "/ajaxtest.php", { param1: "param1", param2: 2 }, onAjaxSuccess );
function onAjaxSuccess(data) {
  // Здесь мы получаем данные, отправленные сервером и выводим их на экран.
  alert(data);
}
```

```
// файл http://hostname/ajaxtest.php
<?php
   echo "Получены параметры с сервера: param1 = ".$_ POST['param1']." и
param2 = ".$_POST['param2'];
?>
```

# *jQuery. Ajax запрос HTML-данных*

Функция **.load()** осуществляет запрос к серверу методом POST, без перезагрузки страницы. Полученные от сервера данные будут автоматически помещены **внутрь** выбранных элементов. Функция имеет несколько необязательных параметров.

**.load(url,[data],[callback],[dataType])**

url – url-адрес, по которому будет отправлен запрос.

data – данные, которые будут отправлены на сервер. Они должны быть представлены в объектом, в формате: {fName1:value1, fName2:value2, ...}.

**callback(data, textStatus, XMLHttpRequest)** – пользовательская функция, которая будет вызвана после ответа сервера.

data – данные, присланные с сервера.

textStatus — статус того, как был выполнен запрос.

XMLHttpRequest – объект XMLHttpRequest

**dataType** – ожидаемый тип данных, которые пришлет сервер в ответ на запрос.

# jQuery. Ajax запрос HTML-данных

Организуем ajax-запрос с передачей параметров, а так же обработаем завершение выполнения запроса выведя на экран соответствующее сообщение:

```
$("#result").load(
 "ajax/test.php",
 {
  param1: "param1",
  param2: 2
 },
 function(){alert("Получен ответ от сервера.")}
);
```

```
// файл ajax/test.php
<?php
   echo "Получены параметры с сервера: param1 = ".$_POST['param1']." и param2 = ".$_POST['param2'];
?>
```

Результат:

```
<div id="result">Получены параметры с сервера: param1 = param1 и param2 = 2</div>
```

# *Формат данных XML*

XML (eXtensible Markup Language) – расширяемый язык разметки.

Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

- SGML (стандарт ISO, 1986) в основном для технической документации
- XML (стандарт W3C, 1998) Упрощение и развитие SGML, широкая область применения.
- XML 1.0 (Fifth Edition) (стандарт W3C, **2008**)
- XML 1.1 (Second Edition)(стандарт W3C, **2006**)

# *Формат данных XML*

XML разрабатывался как:

- язык с простым формальным синтаксисом;
- удобный для создания и обработки документов программами;
- удобный для чтения и создания документов человеком;
- нацеленный на использование в Интернете.

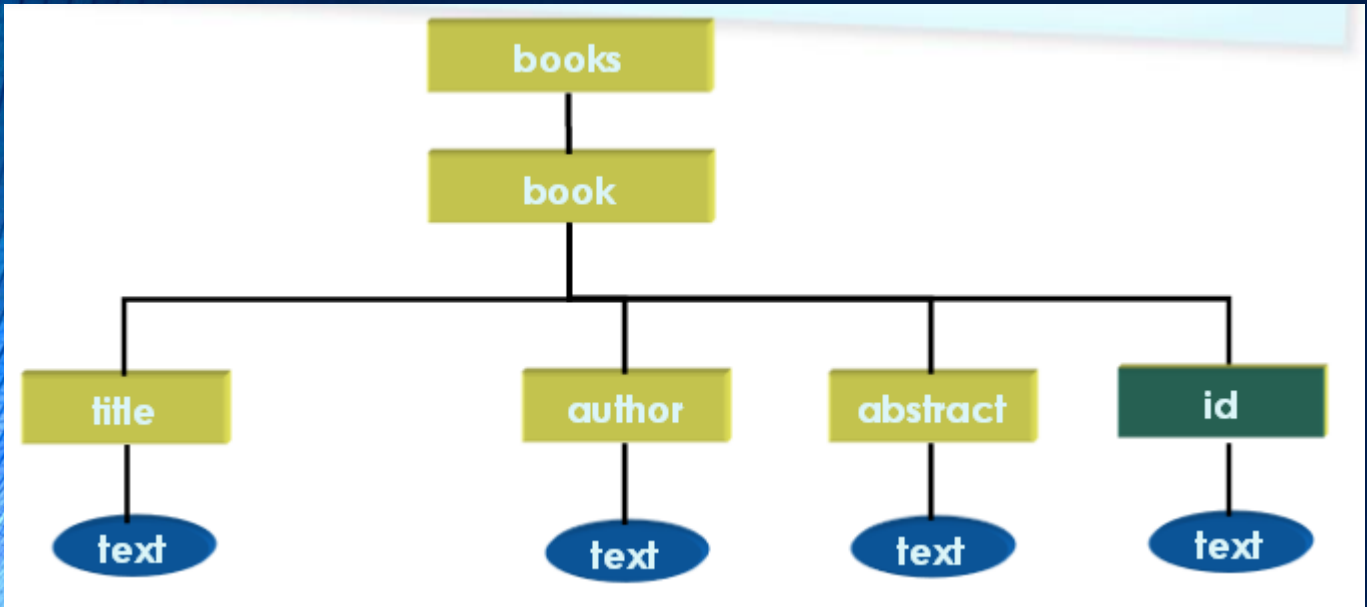
Основные возможности использования формата XML:

- XML как средство обмена данными между приложениями.
- XML как средство передачи данных в web-среде.
- Использование XML для динамической генерации HTML-документов.
- Непосредственное хранение XML-файлов в БД.

# *Правила оформления XML*

Любой XML-документ состоит из следующих частей:

- Пролог (необязательный в XML 1.1);
- Тело документа;
- Эпилог (необязательный, следует за деревом элементов).




Основные элементы:

- Пролог
- **Один корневой элемент**
- Иерархия элементов
- Атрибуты
- Текстовые элементы
- Пустые элементы

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book id="1">
    <title>Евгений Онегин</title>
    <author>А.С. Пушкин</author>
    <abstract>В книгу вошел роман
    в стихах...</abstract>
  </book>
  <!-- ... -->
</books>
```

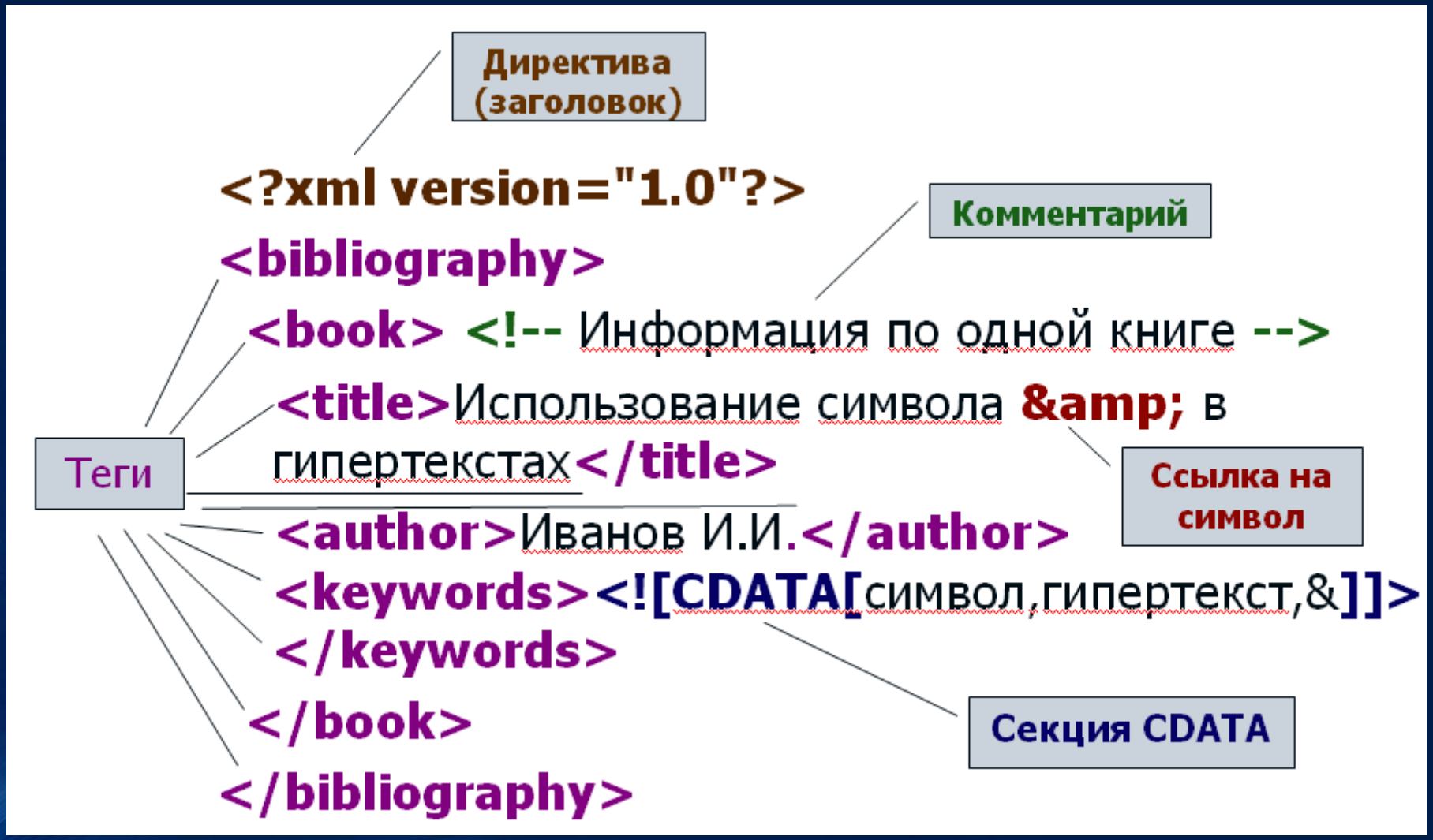
# *Правила оформления XML*

- С логической точки зрения, документ состоит из **пролога** и **корневого элемента**.
- В заголовке документа помещается **объявление XML**, в котором указывается язык разметки документа, номер его версии и дополнительная информация.
- Разметка всегда начинается символом **<** и заканчивается символом **>**. Наряду с символами < и >, специальную роль для разметки играет также символ **&**. Эти символы не могут присутствовать в **символьных данных** и в значениях атрибутов в их непосредственном виде, для их представления в этих случаях зарезервированы **< > &**
- **Комментарий** начинается последовательностью «<!--» и заканчивается последовательностью «-->», внутри не может встречаться комбинация символов «--».

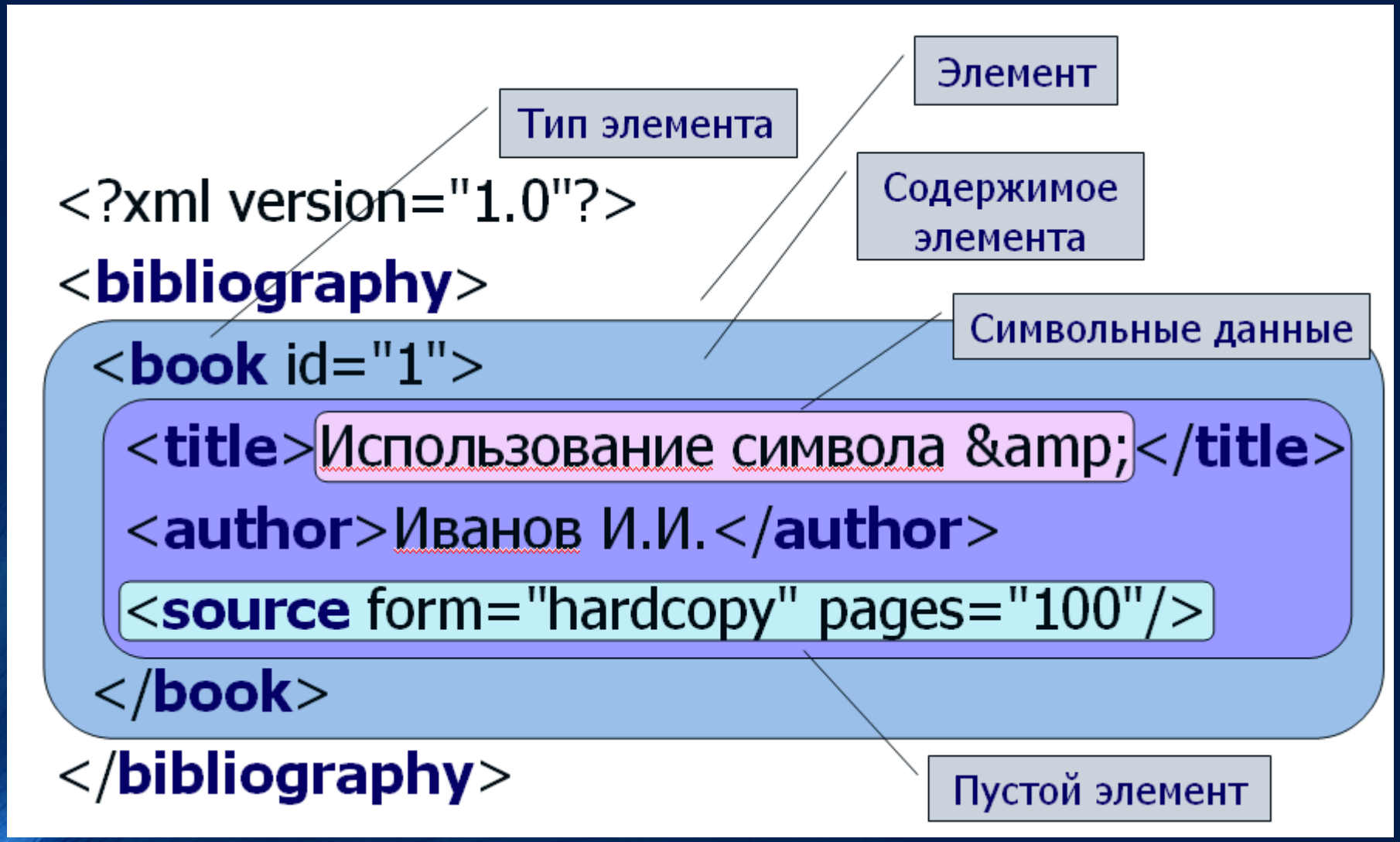
# *Правила оформления XML*

- Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь своего закрывающего "напарника", т.е., нельзя опускать **закрывающие** тэги.
- В XML учитывается **регистр** символов.
- Все значения атрибутов, используемых в определении тэгов, должны быть заключены в **кавычки**. Атрибуты могут использоваться только в начальном теге и теге пустого элемента.
- Элементы документа должны быть правильно **вложены**: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался. Элементы не могут пересекаться (<a></a>).

# Пример XML-документа



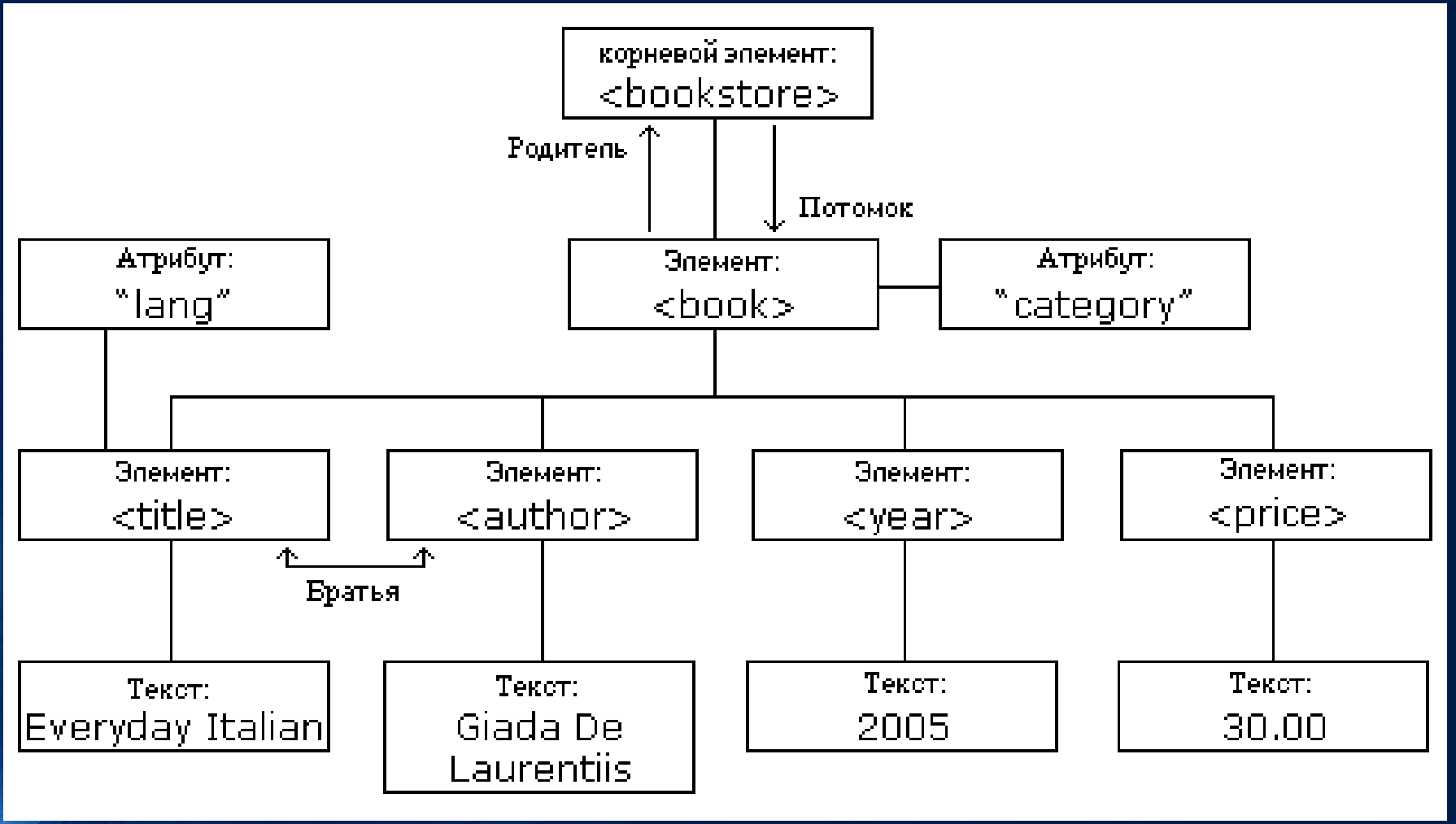
# Пример XML-документа



# *Пример XML-документа*

```
<?xml version="1.0" encoding="utf-8"?>
<recipe name="хлеб" preptime="5min" cooktime="180min">
  <title>Простой хлеб</title>
  <composition>
    <ingredient amount="3" unit="стакан">Мука</ingredient>
    <ingredient amount="0.25" unit="грамм">Дрожжи</ingredient>
    <ingredient amount="1.5" unit="стакан">Тёплая вода</ingredient>
    <ingredient amount="1" unit="чайная ложка">Соль</ingredient>
  </composition>
  <instructions>
    <step> Смешать все ингредиенты и тщательно замесить.</step>
    <step> Закрыть тканью и оставить на один час в тёплом помещении.</step>
    <!-- <step> Почитать вчерашнюю газету. </step>-->
    <step>Замесить ещё раз, положить на противень и поставить в духовку.</step>
  </instructions>
</recipe>
```

# Пример дерева XML документа



# *DOM свойства XML документа*

- **firstChild** – возвращает первый дочерний узел;
- lastChild – возвращает последний дочерний узел;
- **childNodes** – возвращает список дочерних элементов узла;
- nextSibling – возвращает узел, сразу же следующий за данным узлом;
- previousSibling – возвращает узел, идущий перед данным узлом;
- parentNode – возвращает родительский узел элемента;
- nodeName – возвращает имя узла;
- nodeType – возвращает тип узла. Целое значение для одного из 12 типов. Например, узлы-элементы имеют тип, равный 1, текстовые узлы – 3.
- **nodeValue** – устанавливает или возвращает значение узла;
- data – возвращает набор данных и содержимое комментария.

# *Пример приложения. DOM свойства XML документа*

Содержимое файла data.xml формата XML:

```
<?xml version="1.0" encoding="utf8" ?>
<user login="root" pass="">
 <name>Mark</name>
 <sname>Avdeev</sname>
</user>
```

В скрипт php поместите такой код:

```
header('Content-Type: text/xml; charset=utf8');
header("Cache-Control: no-cache, must-revalidate");
echo file_get_contents('data.xml');
```

| Переменная | Вывод переменной |
|---|---|
| ajax.responseText | MarkAvdeev |
| var  xmlDoc = ajax.responseXML; | [object XMLDocument] |
| xmlDoc.childNodes[0] | [object Element] |
| xmlDoc.childNodes[1] | undefined |
| xmlDoc.firstChild | [object Element] |
| xmlDoc.lastChild | [object Element] |

# Пример приложения. Значения DOM свойств XML документа

| var nodeUser = xmlDoc.firstChild; | |
|---|---|
| nodeUser.nodeType | 1 |
| nodeUser.nodeName | user |
| nodeUser.nodeValue | null |
| nodeUser.childNodes | [object NodeList] |
| nodeUser.childNodes.length | 5 (на самом деле 2) |
| nodeUser.childNodes[0] | [object Element] |
| nodeUser.childNodes[0].nodeType | 1 |
| nodeUser.childNodes[0].nodeName | name |
| nodeUser.childNodes[0].nodeValue | null |
| nodeUser.childNodes[0].firstChild.nodeType | 3 |
| nodeUser.childNodes[0].firstChild.nodeName | #text |
| nodeUser.childNodes[0].firstChild.nodeValue | Mark |
| nodeUser.childNodes[1].firstChild.nodeValue | Aydeev |

# *DOM методы XML документа*

- appendChild – добавляет новый дочерний узел в конец списка детей узла;
- insertBefore – вставляет новый дочерний узел перед существующим дочерним узлом;
- removeChild – удаляет дочерний узел;
- replaceChild – заменяет дочерний узел;
- **getElementsByTagName** – возвращает коллекцию узлов с указанным тегом;
- getAttribute – возвращает значение атрибута;
- hasAttributes – определяет, имеет ли элемент какие-либо атрибуты, соответствующие указанному имени;
- hasChildNodes – определяет, имеет ли элемент дочерние узлы.

# *Пример приложения. Формат данных XML*

Обработка ответа со стороны клиента:

```
if(ajax.status==200) {
  xmlDoc=ajax.responseXML;
 // var name = xmlDoc.getElementsByTagName("user")[0].getElementsByTag
Name("name")[0].firstChild.nodeValue;
  var name = xmlDoc.getElementsByTagName("name")[0].firstChild.nodeValue;
  var sname = xmlDoc.getElementsByTagName("sname")[0].firstChild.nodeValue;

  document.getElementById("printResult").innerHTML = "<b>Имя: </b>" + name +
"<br/><b>Фамилия: </b>" + sname + "<br/><b>";
 }
```

**Имя:** Mark
**Фамилия:** Avdeev

# Пример1 использования XML DOM

```
<?xml version="1.0" encoding="UTF-8"?>
 <employees>
   <lawyer money="5"/>
   <janitor name="Sue"></janitor>
   <janitor name="Bill">too poor</janitor>
 </employees>

// нулевой элемент массива длины 1
//var employeesTag = ajax.responseXML.getElementsByTagName("employees")[0];
var employeesTag = ajax.responseXML.firstChild;
// сколько денег заработал адвокат?
var lawyerTag = employeesTag.getElementsByTagName("lawyer")[0];
var salary = lawyerTag.getAttribute("money"); // "5"
// массив из 2 janitors
var janitorTags = employeesTag.getElementsByTagName("janitor");
var excuse = janitorTags[1].firstChild.nodeValue; // " too poor "
```

# Пример2 использования XML DOM

```
<?xml version="1.0" encoding="utf8" ?>
<row>
  <book name='Sue'>BOOK from XML:1</book>
  <book>BOOK from XML:2</book>
  <publisher>PUBLISHER from XML: BHV </publisher>
  <quantity>QUANTITY from XML: 10000</quantity>
</row>
```

Доступ к данным:

```
for(i=0; i<xmlDoc.getElementsByTagName("book").length; i++) //2
{
  if(xmlDoc.getElementsByTagName("book")[i].getAttribute("name") == "Sue")
    // вывод: BOOK from XML:1
    alert(xmlDoc.getElementsByTagName("book")[i].childNodes[0].nodeValue);
}
```

# Пример3 использования XML DOM

```
<?xml version="1.0" encoding="utf8" ?>
<row>
<book name='Sue'>
<subbook>BOOK from XML:1_1</subbook>
<subbook>BOOK from XML:1_2</subbook>
</book>
<book>BOOK from XML:2</book>
<publisher>PUBLISHER from XML: BHV</publisher>
<quantity comment='QUANTITY from XML' >10000</quantity>
</row>
```

Не рекомендуется смешивать в одном узле данные и их описание.

# Пример3 использования XML DOM

Доступ к данным:

```
for(i=0; i<xmlDoc.getElementsByTagName("book").length; i++) {
  if(xmlDoc.getElementsByTagName("book")[i].getAttribute("name") == "Sue") {
    alert(xmlDoc.getElementsByTagName("book")[i].getElementsByTagName
("subbook")[0].childNodes[0].nodeValue); // BOOK from XML:1_1
    alert(xmlDoc.getElementsByTagName("book")[i].getAttribute("name")); // Sue
  }
}
document.getElementById(“res").innerHTML=xmlDoc.getElementsByTagName("book")[
0].getElementsByTagName("subbook")[1].childNodes[0].nodeValue; // BOOK from
XML:1_2
document.getElementById("res").innerHTML=xmlDoc.getElementsByTagName("book")[
0].getElementsByTagName("subbook")[1].firstChild.nodeValue; // BOOK from XML:1_2
```

# *Альтернатива XML: JSON*

JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript.

XML:

```
<xml>
<contacts>
<person firstname="Joe" lastname="Smith"
phone="555-1212" />
<person firstname="Sam" lastname="Stevens"
phone="123-4567" />
</contacts>
</xml>
```

JSON:

```
{contacts:[
{"firstname":"Joe", "lastname":"Smith",
"phone":"555-1212"},
{"firstname":"Sam", "lastname":"Stevens",
"phone":"123-4567"}
]}
```

Сравнительные преимущества JSON:
- меньший объем данных (экономия трафика, плюс к скорости работы сайта)
- возможности расширения

# *Правила формирования JSON*

В качестве значений в JSON используются структуры:

- **Объект** — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
- **Массив** (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.
- Значение может быть строкой в двойных кавычках, числом, объектом, массивом, одним из литералов: true, false или null. Т.о. структуры могут быть вложены друг в друга.
- **Строка** — это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Объекты JSON отличаются от обычных JavaScript-объектов более строгими требованиями к строкам — они должны быть именно в двойных кавычках. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\».

# *Методы для работы с JSON*

На серверной стороне:

В языке PHP, начиная с версии 5.2.0, поддержка JSON включена в ядро в виде функций **json_decode**() и **json_encode**(), которые сами преобразуют типы данных JSON в соответствующие типы PHP и наоборот.

string **json_encode** ( mixed $value [, int $options = 0 ] )

Возвращает JSON-закодированную строку или FALSE в случае возникновения ошибки. Функция работает только с кодированными в UTF-8 данными.

mixed **json_decode** ( string $json [, bool $assoc = false [, int $depth = 512 [, int $options = 0 ]]] )

Принимает закодированную в JSON строку и преобразует ее в переменную PHP.

# *Методы для работы с JSON*

На клиентской стороне:

Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией eval() (не рекомендуется). Кроме того, возможна вставка JavaScript-функций.

**JSON.parse** (text [, **reviver**])
Возвращает объект или массив

**JSON.stringify** (value [, replacer] [, **space**])
Возвращает строку, содержащая текст JSON.

```
var person = eval(xhr.responseText); alert(person.firstName);
```

# *Пример работы с JSON на сервере*

```
$data = array('book' => $id, 'publisher' => 'Smit', 'quantity' => 1000);
echo json_encode($data);
```

Вывод данных:

```
{"book":"SQL","publisher":"Smit","quantity":1000}
```

```
$json = '{"a":1,"b":2,"c":3,"d":4,"e":5}';
var_dump(json_decode($json, true));
```

Вывод данных:

```
array(5) { ["a"] => int(1) ["b"] => int(2) ["c"] => int(3) ["d"] => int(4) ["e"] => int(5) }
```

# *Пример работы с JSON на клиенте*

```
var res = JSON.parse(ajax.responseText);
document.getElementById("book").innerHTML = res.book;
document.getElementById("publisher").innerHTML=res.publisher;
document.getElementById("quantity").innerHTML=res.quantity;

var jsontext = '{ "hiredate": "2008-01-01T12:00:00Z", "birthdate": "2008-12-
25T12:00:00Z" }';
var dates = JSON.parse(jsontext, dateReviver);
function dateReviver(key, value) {
/*код для преобразование формата времени с использованием регулярных
выражений*/
};
```

# *Пример приложения. Формат данных JSON*

Обработка ответа со стороны клиента:

```
if (ajax.readyState == 4) {
  if (ajax.status == 200) {
    alert(ajax.responseText);
    var res = JSON.parse(ajax.responseText);
    document.getElementById("book").innerHTML
= res.book;
   document.getElementById("publisher").inner
HTML=res.publisher;
   document.getElementById("quantity").inner
HTML=res.quantity;
  }
  else {    alert(ajax.status + " - " + ajax.statusText);
    ajax.abort();          }
}
```

В скрипт php поместите такой код:

```
header('Content-Type:
application/json');
$data = array('book' => 'book',
                    'publisher' => 'Smit',
                    'quantity' => 1000);
echo json_encode($data);
```

# Вопросы

- Каковы недостатки синхронной модели?
- Перечислите преимущества и недостатки использования технологии AJAX.
- Укажите этапы выполнения AJAX-запроса и его возможные состояния.
- Как открыть синхронное соединение, используя технологию AJAX?
- Какими объектами и методами поддерживается методология AJAX в JavaScript?
- Какие библиотеки для поддержки AJAX вам известны?
- Что такое jQuery?
- Особенности формата данных XML, правила оформления. Методы получения значений XML-элементов.
- Что такое JSON? Какие методы на стороне клиента и сервера для работы с этим форматом?