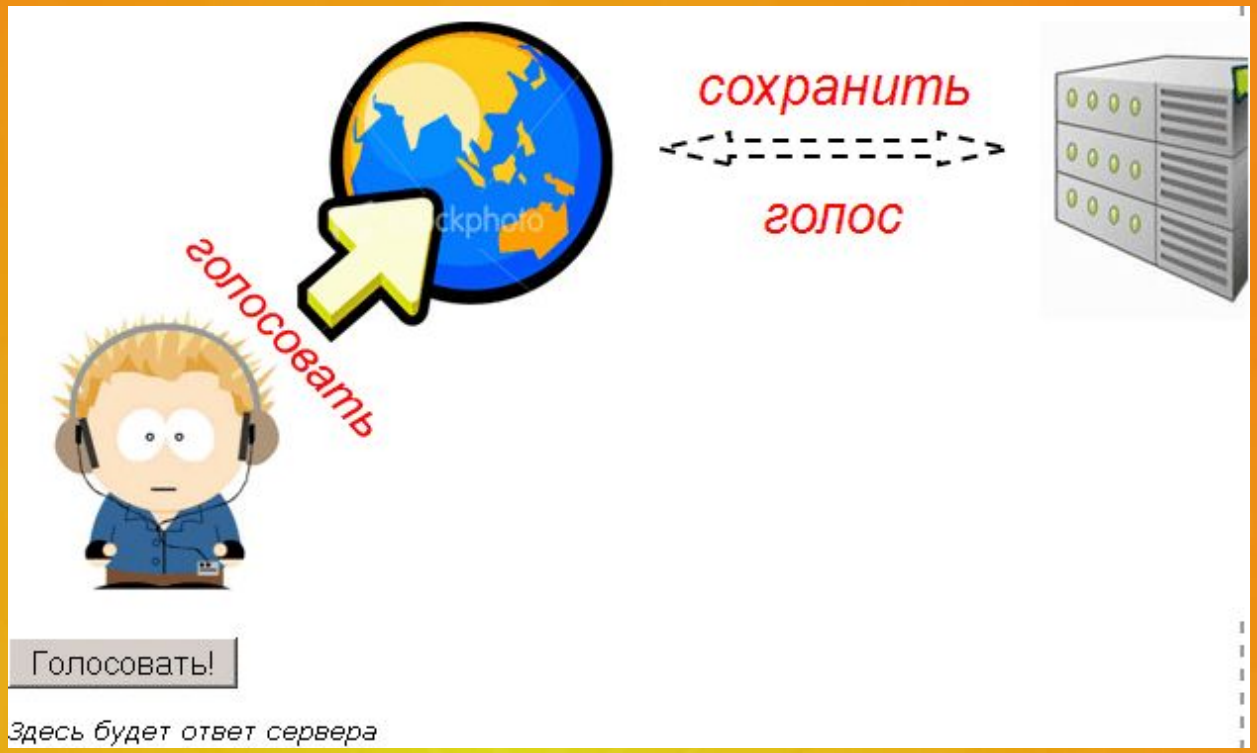
Введение в AJAX

Введение в Ajax

AJAX [*Asynchronous Javascript And Xml*] – технология для взаимодействия с сервером без перезагрузки страниц.



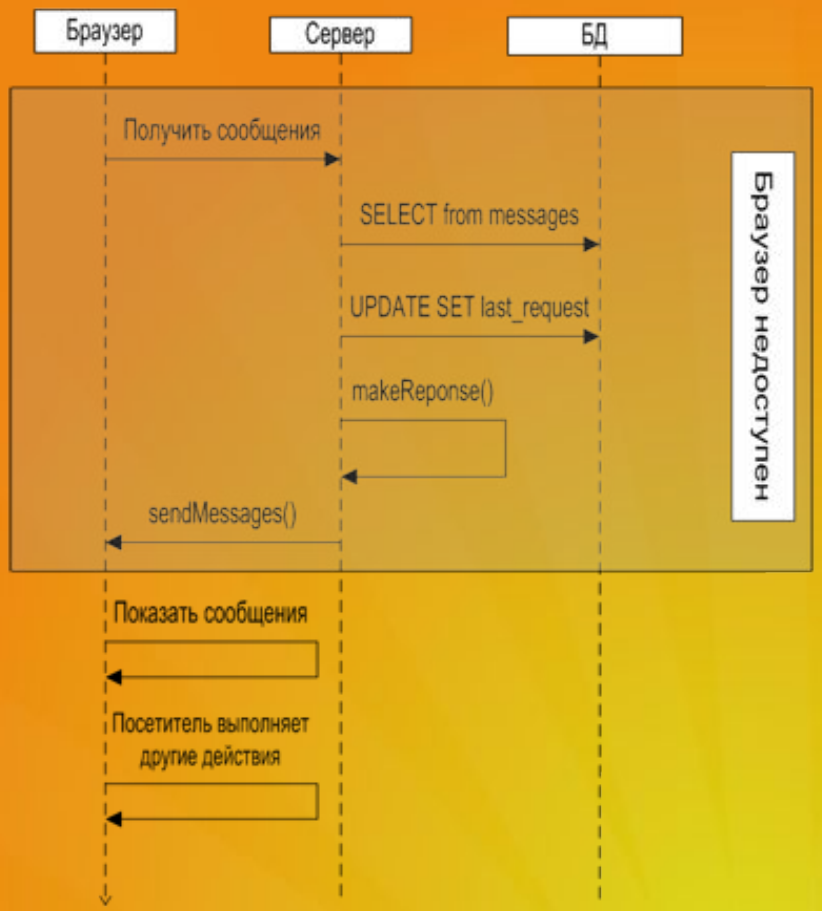
Синхронная модель обмена данными

- Все процессы выполняются последовательно, один за другим:
  - браузер отправляет запрос на сервер и ждет, пока тот совершит всю необходимую работу;
  - сервер выполняет запросы к базе данных, представляет ответ в необходимом формате и выводит его;
  - браузер, получив ответ, вызывает функцию показа.
- Сетевые задержки включены во время ожидания.
- Пока происходит синхронный обмен данными, пользователь не может совершать на странице других действий.

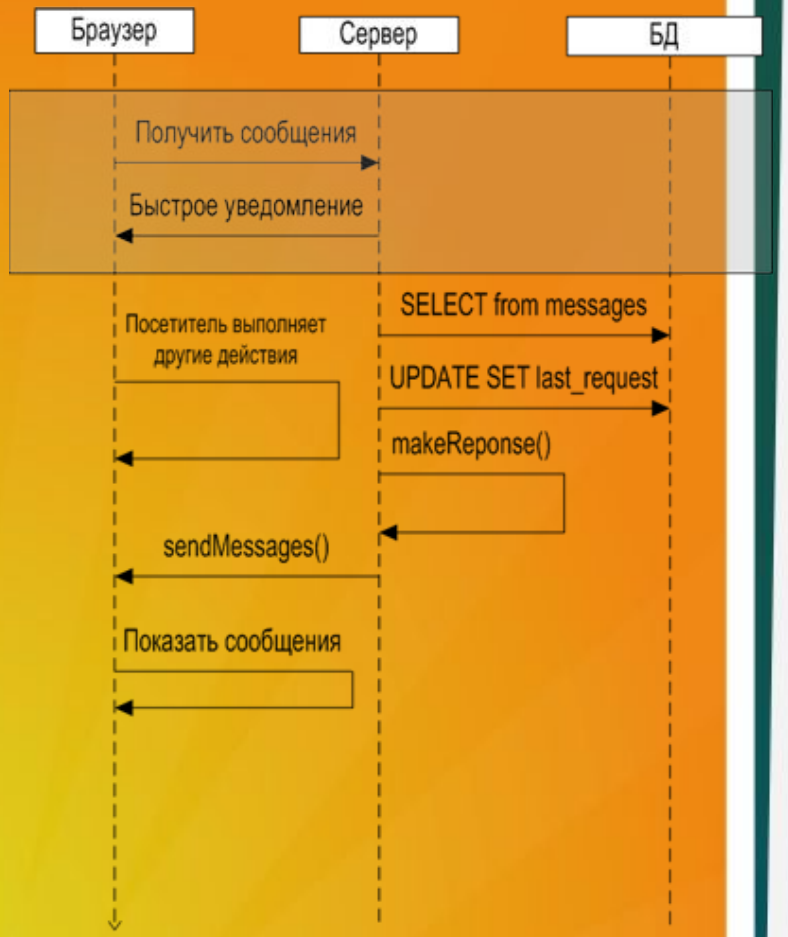
Асинхронная модель обмена данными

- Браузер отправляет запрос на сервер, сервер уведомляет браузер о том, что запрос принят на обработку, и освобождает его для дальнейшей работы.
- После пересылки сервером подготовленного ответа на браузере запускается заранее подготовленная функция показа сообщения.
- Пока ответ формируется и пересылается – браузер свободен: пользователь может добавлять комментарии, заполнять формы, посылать новые асинхронные запросы.

Синхронная модель обмена данными



Асинхронная модель обмена данными

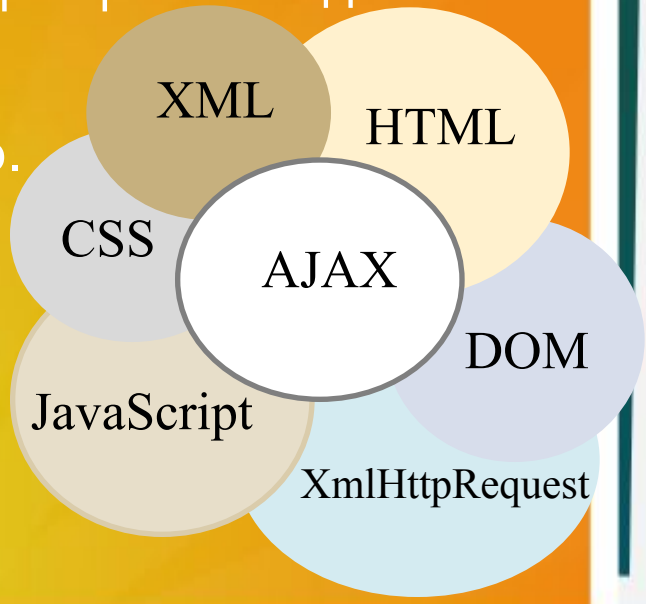


Особенности асинхронной модели обмена данными:

- Сложность в реализации и отладке:
  - заранее должно быть задано то, что сработает после;
  - недостаточные возможности браузеров.
- Неопределена последовательность выполнения запросов: поскольку можно одновременно делать несколько задач, задача, начатая первой, может окончиться последней.
- Разрыв по времени между действием и реальным результатом, приложение становится более чувствительным к ошибкам, как пользовательским (нехватка привилегий), так и ошибкам коммуникации (разрыв связи и т.п.).
- Контроль целостности.
- **Интерактивность**.
- **Быстрый интерфейс**: мгновенная реакция на действия пользователя.

AJAX – интеграция технологий

- (X)HTML, CSS для представления и стилизации информации.
- JavaScript производит операции над DOM-моделью на стороне клиента для обеспечения динамического отображения информации и интерактивности.
- XMLHttpRequest для асинхронного обмена данными с веб-сервером.
- XML и XSLT, HTML, JSON для формирования данных.
- JavaScript объединяет все перечисленное в единую технологию.



Типичное AJAX–приложение состоит их двух частей:

Применение

- Небольшие элементы управления, связанные с элементарными действиями: добавить в корзину, подписаться на рассылку и т.п.
- Динамическая подгрузка данных с сервера, например, построение оглавления в виде дерева, узлы которого подгружаются по мере раскрытия.
- Незаметные для пользователя действия, например, автосохранение результатов на сервере при редактировании статьи.
- Непрерывная подгрузка данных с сервера, например, чат или отображение биржевых котировок в реальном времени.

Примеры: Google suggest

Google - одна из первых систем, предложившая "живой" поиск (live search). Пользователь печатает поисковую фразу, а система автодополняет ее, получая список самых вероятных дополнений с сервера.

Примеры: Google suggest

Принцип реализации:

- Активируется примерно при каждом нажатии клавиши: система отслеживает время посылки последнего запроса, при "обычной" скорости печати запрос отсылается при каждом нажатии, при "программисткой" скорости – каждые несколько нажатий.
- Создается скрытый <DIV>, который показывается при начале печати.
- <DIV> заполняется ответом сервера: текущий результат подсвечивается.
- Результаты кэшируются: при нажатии на "удалить", обращения серверу не происходит.
- Для управления частотой запросов отслеживается время на осуществление запроса: при подключении по выделенной линии запросы идут чаще, чем при подключении через модем.

Примеры: Gmail

На момент появления являлся единственным открытым почтовым сервисом, использующим AJAX.

- Проверка ошибок ввода формы ДО Submit: на сервер передается имя пользователя, результат проверки возвращается на страницу.
- Быстрая загрузка: браузер обращается к серверу только за данными, а не обновляет весь интерфейс.
- Автоматическая "доставка" писем в открытую папку: на сервер периодически отправляется запрос и, если пришли новые письма, они появляются в браузере.
- Автодополнение: достаточно ввести первые буквы имени адресата, остальные дополняются автоматически.

Преимущества

- Экономия трафика при работе веб-приложением: вместо загрузки всей страницы достаточно загрузить только изменившуюся часть.
- Уменьшение нагрузки на сервер: например, на странице работы с почтой при отметке прочитанных писем серверу достаточно внести изменения в базу данных и отправить на клиент сообщение об успешном выполнении операции без необходимости повторно создавать страницу и передавать её клиенту.
- Ускорение реакции интерфейса: поскольку загружается только изменившаяся часть, пользователь видит результат своих действий быстрее.

Недостатки

- Отсутствие интеграции со стандартными инструментами браузера:
  - динамически создаваемые страницы не регистрируются браузером в истории посещения страниц;
  - не функционирует кнопка «Назад»;
  - невозможно сохранить закладки на желаемый материал.
- Динамически загружаемое содержимое недоступно поисковикам, поэтому необходимо предусматривать альтернативные способы доступа к содержимому сайта.
- Становятся неактуальными старые методы учёта статистики сайтов, т.к. многие статистические сервисы ведут учет просмотра новых страниц.
- Кроссбраузерность: разные браузеры поддерживают стандарты неодинаково.

jQuery функция $.ajax()

jQuery функция $.ajax() позволяет выполнить асинхронный AJAX запрос. AJAX (от англ. Asynchronous Javascript and XML — "асинхронный JavaScript и XML") — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в "фоновом" обмене данными браузера с веб-сервером.

jQuery синтаксис:

jQuery функция $.ajax() лежит в основе всех AJAX запросов, отправленных с использованием jQuery. Не смотря на то, что функция $.ajax() может использоваться более гибко, в большинстве случаев в этом нет необходимости. В jQuery существуют такие альтернативные методы как $.get() и .load(), которые проще в использовании.

Параметры и описание

url

Строка, содержащая URL адрес, на который отправляется запрос.

settings

Набор пар ключ/значение, которые настраивают запрос AJAX. Все параметры являются необязательными. Допускается, но не рекомендовано установить значение по умолчанию для любого параметра с использованием метода $.ajaxSetup().

accepts (по умолчанию: зависит от dataType).

Тип: PlainObject.

Набор пар ключ/значение, которые отправляется в Accept заголовка запроса. Этот заголовок сообщает серверу, какой ответ запрос будет принимать в ответ. Обратите внимание, что значение параметра, указанного в dataType (тип данных, которые мы ожидаем от сервера) сопоставляется с указанным в параметре. Кроме того, для корректной обработки ответа от сервера необходимо в параметре

Пример

```
$.ajax({
  accepts: {
    mycustomtype:
   "application/x-some-custom-type"
  },
  // указываем как обрабатывать ответ
  converters: {
    "text mycustomtype": function( result )
   {
    // возвращаем преобразованное
```

async (по умолчанию: true).

Тип: Boolean.

По умолчанию, все запросы отправляются асинхронно, если вам необходимо организовать синхронные запросы, то установите этот параметр в false. Обратите внимание, что кроссдоменные запросы и элемент, параметр dataType которого имеет значение "jsonp" не поддерживают запросы в синхронном режиме. Учтите, что используя синхронные запросы вы можете временно заблокировать браузер отключив какие-либо действия пока запрос будет активен.

beforeSend

Тип: Function(jqXHR jqXHR, PlainObject settings).

Функция обратного вызова, которая будет вызвана перед осуществлением AJAX запроса. Функция позволяет изменить объект jqXHR (в jQuery 1.4.x объект XMLHTTPRequest) до его отправки. Объект jqXHR это надстройка расширяющая объект XMLHttpRequest, объект содержит множество свойств и методов, которые позволяет получить более полную информацию об ответе сервера, а так же объект содержит Promise методы. Если функция beforeSend возвращает false, то AJAX запрос будет отменен. Начиная с версии jQuery 1.5 функция beforeSend будет вызываться независимо от типа запроса.

Cache (по умолчанию: true, для dataType "script" и "jsonp" false).

Тип: Boolean.

Если задано значение false, то это заставит запрашиваемые страницы не кэшироваться браузером. Обратите внимание, что значение false будет правильно работать только с HEAD и GET запросами.

complete.

Тип: Function(jqXHR jqXHR, String textStatus).

Функция, которая вызывается, когда запрос заканчивается (функция выполняется после AJAX событий "success" или "error"). В функцию передаются два параметра: jqXHR (в jQuery 1.4.x объект XMLHTTPRequest) и строка соответствующая статусу запроса ("success", "notmodified", "nocontent", "error", "timeout", "abort", или "parsererror"). Начиная с версии jQuery 1.5 параметр complete может принимать массив из функций, которые будут вызываться по очереди.

contents.

Тип: PlainObject.

Объект состоящий из пар строка/регулярное выражение, определяющих, как jQuery будет обрабатывать (парсить) ответ в зависимости от типа содержимого. Добавлен в версии jQuery 1.5.

contentType (по умолчанию: "application/x-www-form-urlencoded; charset=UTF-8").

Тип: Boolean, или String.

Определяет тип содержимого, которое указывается в запросе при передаче данных на сервер. С версии с jQuery 1.6 допускается указать значение false, в этом случае jQuery не передает в заголовке поле Content-Type совсем.

context

Тип: PlainObject.

При выполнении AJAX функций обратного вызова контекстом их выполнения является объект window. Параметр context позволяет настроить контекст исполнения функции таким образом, что $(this) будет ссылаться на определенный DOM элемент, или объект.

пример

```
$.ajax({
  url: "test.html", // адрес, на который будет отправлен запрос
  context: $( ".myClass" ), // новый контекст исполнения функции
  success: function(){ // если запрос успешен вызываем функцию
    $( this ).html( "Всё ок" ); // добавлем текстовое содержимое в элемент с классом .myClass
  }
});
```

converters

Значения по умолчанию:

```
{
  "* text": window.String, // любой тип в текст
  "text html": true, // текст в html
  "text json": jQuery.parseJSON, // текст в JSON
  "text xml": jQuery.parseXML // текст в XML
}
```

Тип: PlainObject.

Объект, содержащий тип данных для конвертации и способ его преобразования. Значение каждого преобразователя является функцией, которая возвращает преобразованное значение ответа. Добавлен в версии jQuery 1.5.

crossDomain (по умолчанию: false для запросов внутри того же домена, true для кроссдоменных запросов).

Тип: Boolean.

Если вы хотите сделать кроссдоменный запрос находясь на том же домене (например jsonp-запрос), то установите этот параметр в true. Это позволит, к примеру, сделать перенаправление запроса на другой домен с вашего сервера. Добавлен в версии jQuery 1.5.

data

Тип: PlainObject, или String, или Array.

Данные, которые будут отправлены на сервер. Если они не является строкой, то преобразуются в строку запроса. Для GET запросов строка будет добавлена к URL. Для того, чтобы предотвратить автоматическую обработку вы можете воспользоваться параметром processData со значением false. Если данные передаются в составе объекта, то он должен состоять из пар ключ/значение. Если значение является массивом, то jQuery сериализует несколько значений с одним и тем же ключом (в зависимости от значения параметра traditional, который позволяет задействовать традиционный тип сериализации основанный на методе $.param).

dataFilter

Тип: Function(String data, String type) => Anything.

Функция вызывается после успешного выполнения AJAX запроса и позволяет обработать "сырые" данные, полученные из ответа сервера. Возврат данных должен происходить сразу после их обработки. Функция принимает два аргумента: data - данные полученные от сервера в виде строки и type - тип этих данных (значение параметра dataType).

method (по умолчанию: "GET").

Тип: String.

Метод HTTP, используемый для запроса (например, "POST", "GET", "PUT"). Добавлен в версии jQuery 1.9.0.

success

Тип: Function(Anything data, String textStatus, jqXHR jqXHR).

Функция обратного вызова, которая вызывается если AJAX запрос выполнится успешно. Функции передаются три аргумента:

data - данные возвращенные с сервера. Данные форматируюся в соответствии с параметрами dataType, или dataFilter, если они указаны

textStatus - строка описывающая статус запроса.

jqXHR - объект jqXHR (до версии jQuery 1.4.x объект XMLHttpRequest).

type (по умолчанию: "GET").

Тип: String.

Псевдоним (алиас) для параметра method. Вы должны использовать type, если вы используете версии jQuery до 1.9.0.

url (по умолчанию: текущая страница).

Тип: String.

Строка, содержащая URL адрес, на который отправляется запрос.

Пример использования

```
<script>
$( document ).ready(function(){
  $( "button" ).click(function(){ // задаем функцию при нажатиии на элемент <button>
    $.ajax({
      method: "POST", // метод HTTP, используемый для запроса
      url: "about.php", // строка, содержащая URL адрес, на который отправляется запрос
      data: { // данные, которые будут отправлены на сервер
        name: "Denis",
        city: "Erebor"
      },
      success: [function ( msg ) { // функции обратного вызова, которые вызываются если AJAX запрос
    выполнится успешно (если несколько функций, то необходимо помещать их в массив)
        $( "p" ).text( "User saved: " + msg ); // добавляем текстовую информацию и данные возвращенные с
    сервера
      },
      function () { // вызов второй функции из массива
        console.log( "next function" );
      }],
      statusCode: {
        200: function () { // выполнить функцию если код ответа HTTP 200
          console.log( "Ok" );
        }      }    })  });});
</script><body>
<button>Клик</button>
<p></p>
</body>
```

В этом примере с использованием jQuery функции $.ajax() мы при нажатии на элемент <button> (кнопка) выполняем асинхронный AJAX запрос со следующими параметрами:

method - указываем метод HTTP POST, используемый для запроса.

url - файл, к которому мы обращаемся ("about.php"), он содержит следующий PHP код:

```
<?php
  $name = $_POST['name']; // создаем переменную name, которая содержит переданные скрипту через HTTP метод POST данные (с ключом name)
  $city = $_POST['city']; // создаем переменную name, которая содержит переданные скрипту через HTTP метод POST данные (с ключом city)
  echo $name.", ".$city; // выводим текстовое содержимое (значение выше созданных переменных, перечисленных через запятую)
?>
```

data - данные, которые будут отправлены на сервер.

success - функции обратного вызова, которые вызываются если AJAX запрос выполнится успешно. Обратите внимание, что если используется несколько функций, то необходимо помещать их в массив. Каждая функция выполняется в свою очередь.

statusCode - объект числовых кодов HTTP и функции, которые будут вызываться, когда код ответа сервера имеет соотвествующее значение (определенный код HTTP). В нашем случае код 200.