**Практическая работа №1**

**"Асинхронное взаимодействие клиентской и серверной части Web-**

**приложений с использованием технологии AJAX"**

**Цель работы:** приобрести навыки организации асинхронного взаимодействия клиентской и серверной частей приложения и передачи данных в формате json и xml.

**Теоретические сведения.**

Традиционный обмен клиента и сервера предполагает, что браузер взаимодействует с пользователем, передает от его имени запросы к серверу, который реализует как бизнес-логику Web-приложения, так и логику представления данных, формирует html-страницу и возвращает ее браузеру. Такой подход порождает ряд проблем:

- вся трудоемкость выполнения Web-приложения перенесена на серверную сторону, что при большом количестве запросов от клиентов может сильно нагружать Web-сервер, в то время как возможности современных браузеров позволяют перенести существенную часть функционала Web-приложения на клиентскую сторону;

- трафик между клиентской и серверной стороной Web-приложения становится избыточным – вместо готового html-представления с его тегами и стилями сервер может возвращать лишь данные, на основе которых клиент сам может сформировать внешний вид Web-страницы;

- каждый запрос к серверу приводит к перерисовке страницы в браузере, что выглядит не очень эстетично – во многих случаях по результатам обработки запроса достаточно изменить (перерисовать) лишь небольшой фрагмент страницы;

- стандартными Web-запросами невозможно реализовать некоторые, уже ставшие привычными функции Web-приложений, как, например, автозаполнение полей форм при вводе данных.

Решением всех этих проблем стал асинхронный обмен данными между клиентской и серверной стороной. Технология, позволяющая осуществлять подобный обмен, получила название AJAX (Asynchronous Javascript And Xml») – загрузки данных с сервера без перезагрузки. Сам термин AJAX вошел в обиход Web-разработчиков в 2005-м году, хотя инструментарий, позволявший создавать асинхронные Web-приложения, были добавлены в браузере еще в середине 90-х годов прошлого века.

AJAX не является технологией в полном смысле этого слова, а объединяет в себе целый ряд методологий:

• стандартизированное представление Web-страниц с помощью языков HTML и CSS;

• динамическое отображение и взаимодействие с пользователем с помощью DOM API и DHTML;

• обмен и обработка данных в виде XML и XSLT или JSON;

• асинхронные запросы с помощью объекта XMLHttpRequest.

Основу функциональности технологии AJAX обеспечивает объект XMLHttpRequest. Его назначение – передавать серверу HTTP- или HTTPS-запросы и обеспечивать прием и обработку ответов сервера.

Функциональность этого объекта доступна благодаря набору свойств и методов. Свойства объекта XMLHttpRequest:

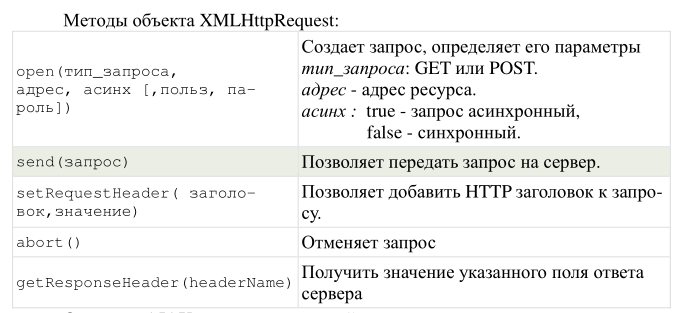
*onreadystatechange Событие изменения состояния готовности сервера.*

*readyState Позволяет узнать состояние готовности сервера.*

*responseText Хранит ответ сервера как строку символов.*

*responseXML Хранит ответ сервера как XML файл.*

*status Хранит код ответа сервера.*



Отправка AJAX-запроса клиентской частью приложения осуществляется в следующей последовательности:

1. Создается экземпляр объекта XMLHttpRequest;

2. Для этого объекта определяется функция-обработчик события onreadystatechange. Это событие наступает при каждой смене состояния объекта XMLHttpRequest. Данная функция занимает одно из центральны мест в процессе обмена, поскольку именно в ней осуществляется обработка ответа сервера;

3. Открывается соединение с сервером, которое параметризуется указанием типа запроса (GET или POST), URL серверной части, флага асинхронного режима и имени и пароля пользователя (если необходимо);

4. Запрос отправляется серверу.

В коде на языке javascript эта последовательность выглядит следующим образом:

*var xhttp;*

*//создаем объект*

*if (window.XMLHttpRequest)*

*{ xhttp=new XMLHttpRequest(); }*

*else {//IE6*

*3*

*xhttp=new ActiveXObject("Microsoft.XMLHTTP");*

*}*

*Xhttp.timeout=5000;*

*xhttp.open('GET', 'ajaxService.php', false);*

*xhttp.send();*

*//Точка останова клиентского скрипта*

Приведенный пример демонстрирует синхронный вариант AJAX-запроса (третий параметр метода open равен false), когда метод send останавливает выполнение клиентского скрипта до момента возврата ответа сервера от сервера. Ответ сервера может быть получен из свойств объекта XMLHttpRequeststatus, respnseText, responseXML.

*if(xhttp.status==200)*

*document.getElementById("some").*

*innerHTML=xhttp.responseText;*

Свойство status хранит статус завершения из заголовка HTTP-ответа сервера. После того как сервер пришлет ответ, его можно прочитать в свойстве responseText или responseXML (в зависимости от формата возвращаемых данных).

В рассмотренном выше фрагменте этот ответ напрямую заносится в тег с идентификатором «some».

При асинхронном запросе клиент не останавливает работу после направления запроса сервера. Момент прихода ответа сервера фиксируется в функции-обработчике события onreadystatechange:

*<html>*

*<body>*

*<div id="some"><h2>Изменяемый блок</h2></div>*

*<button type="button" onClick="ajaxReq()">Загрузить данные</button>*

*</body>*

*<script type="text/javascript">*

*function ajaxReq()*

*{var xhttp = new XMLHttpRequest();*

*xhttp.open('GET', 'ajaxService.php', true);*

*xhttp.send();*

*xhttp.onreadystatechange = function() {*

*if (xhttp.readyState != 4) return;*

*if (xhttp.status != 200)*

*document.getElementById("some").innerHTML=*

*xhttp.status+ ': ' + xhttp.statusText;*

*else document.getElementById("some").innerHTML=*

*xhttp.responseText;*

*}*

*}*

*</script></html>*

Событие onreadystatechange активизируется (а, соответственно, и вызывается функция-обработчик этого события) неоднократно при обработке AJAX запроса. В связи с чем вызван обработчик этого события можно узнать из свойства readyState. Это свойство может принимать значения:

UNSENT (0) Начальное состояние, объект создан

OPENED (1) Вызван метод open

HEADERS\_RECEIVED (2) Получен заголовок HTTP-ответа

LOADING (3) Загружается тело ответа (пришел очередной пакет)

DONE (4) Ответ полностью получен или зарегистрирована

ошибка

Таким образом, функция-обработчик события onreadystatechange вызывается как на стадии инициализации запроса, так и при его отправке, получении заголовка, всего ответа, возникновении ошибки при обработке запроса, что дает возможность гибко конфигурировать приложение, реагировать на различные стадии и статусы его обработки. Использование этого свойства для современных реализаций можно заменить на обработку специфичных событий. Если в ранних версиях реализации объекта XMLHttpRequest было определено лишь одно событие для контроля и обработки различных стадий обработки запроса (onread-ystatechange), то в современной спецификации таких событий определено несколько, и они стали специфичны тому или иному этапу обработки запроса:

*loadstart запрос начат.*

*progress браузер получил очередной пакет данных, можно прочитать*

*текущие полученные данные в responseText.*

*abort запрос был отменён вызовом abort().*

*error произошла ошибка.*

*load запрос был успешно (без ошибок) завершён.*

*timeout запрос был прекращён по таймауту.*

*loadend запрос был завершён (успешно или неуспешно)*

Шаблон использования объекта XMLHttpRequest в этом случае будет та-

ким:

*function ajaxReq()*

*{*

*var xhttp = new XMLHttpRequest();*

*xhttp.open('GET', 'ajaxService.php', true);*

*xhttp.setRequestHeader('Content-Type', 'application/html; char-*

*set=utf-8');*

*xhttp.timeout=10000;*

*xhttp.onloadstart = function() {…}*

*xhttp.onerror = function() {…}*

*xhttp.onload = function() {…}*

*xhttp.ontimeout = function() {…}*

*xhttp.send();*

*}*

Если с запросом необходимо передать параметры, то в случае GET-запроса

они указываются в методе open:

*xhttp.open('GET', 'ajaxService.php?param1=value1&param2=value2',true);*

Параметры POST-запроса необходимо передавать в теле запроса, поэтому они указываются в параметрах метода send:

*xhttp.open('POST', 'ajaxService.php', true);*

*xhttp.send(“param1=value1&param2=value2”);*

Обрабатывая запрос клиента, сервер формирует ответ, который может быть отправлен в различных форматах: html, json, xml. Серверный скрипт, возвращающий ответ в html-формате может выглядеть следующим образом:

*<?php*

*require\_once("db\_func.php");*

*$response = array('status'=>'no', 'message'=>'');*

*header('Content-Type: application/html; charset=utf8');*

*$db=connect\_db();*

*if($resStr!="ok")*

*print(”Нет соединения с базой данных”);*

*else*

*{*

*$res=getData($\_GET[“param1”], $\_GET[“param2”]);*

*if($res==null)*

*print("Не удалось извлечь данные");*

*else*

*{*

*print(“<h1>Данные о товаре</h1>”);*

*print(“<p>Наименование: <span>{$res[“goodName”]}</span></p>”);*

*print(“<p>Стоимость: <span>{$res[“goodPrice”]}</span></p>”);*

*}*

*}*

*?>*

В приведенном примере серверный скрипт запрашивает данные о товаре, формирует их html-представление и возвращает из клиенту, который может напрямую добавить их в некоторый элемент разметки. Однако, современные тенденции разработки Web-приложений отводят серверной части лишь реализацию части бизнес-логики, которая извлекает и обрабатывает данные, а затем возвращает их клиенту. Формирование представления на основе результатов обработки запроса ложится на клиентский скрипт. Формат html при таком подходе не совсем подходит, так как с его помощью формируется готовое представление. Для передачи непосредственно данных можно использовать форматы json или xml.

Формат JSON (JavaScript Object Notation) позволяет легко сериализовать (преобразовать в строку) и десериализовать программные объекты для их передачи в запросах и ответах в виде текстовых строк.

Например, объект:

*var subject = {*

*fio: “Иванов И.И.", age: 30,*

*address:*

*{ city: “Волжский", street:"Мира", house: 22}*

*};*

методом JSON. stringify(subject) сериализуется в строку:

*‘{"fio":“Иванов И.И.", "age":30, "address": {"city":«Волжский","street":"Мира","house":22}}’*

из которой впоследствии легко десериализуется. Причем оба преобразования (объект->строки и строка->объект) легко реализуются как средствами javascript, так и языками серверных приложений (php, perl, с#, тот же javascript).

Если серверный скрипт реализован на языке php, для сериализации можно использовать метод json\_encode:

*<?*

*$obj=getSubjectInfo(); //формируем объект*

*echo json\_encode($obj); //сериализуем его в строку*

*?>*

Тогда на клиенте в обработчике события onload свойство resnonseText будет содержать сериализованный объект, на основе значений полей которого клиент может изменять представление фрагмента страницы:

var xhttp = new XMLHttpRequest();

xhttp.open('GET', 'ajaxService.php', true);

xhttp.setRequestHeader('Content-Type', 'application/json;

charset=utf-8');

xhttp.timeout=10000;

xhttp.onload = function() {

if (xhttp.status == 200)

{ var obj=JSON.parse(xhttp.responseText);

var el=document.createElement(“div");

hEl=document.createElement(“h1”)’

hEl.innerHTML=”Информация о клиенте”;

el.appendChild(hEl);

pName=document.createElement(“p”)’

pName.innerHTML=”ФИО: ”;

spName= document.createElement(“span”);

spName.innerHTML=obj.fio;

spName.style.color=”red”;

pName.appendElement(spName);

el.appendChild(pName);

pAge=document.createElement(“p”)’

pAge.innerHTML=”возраст: ”;

spAge= document.createElement(“span”);

spAge.innerHTML=obj.age;

spAge.style.color=”blue”;

pAge.appendElement(spAge);

el.appendChild(pAge);

var userList=document.getElementById(“userList”);

userList.appendChild(el) ;

}

}

xhttp.send();

Как следует из приведенного примера, после десериализации объекта методом JSON.parse его можно использовать как обычный программный объект javascript.

JSON-формат можно использовать и для передачи данных от клиента серверу. В этом случае на стороне клиента программный объект необходимо сериализовать методом JSON.stringify, а на серверной стороне (в случае использования языка php) десериализовать методом json\_decode:

На клиенте:

*var xhttp = new XMLHttpRequest();*

*xhttp.open('POST', 'ajaxServiceJSON.php', true);*

*xhttp.setRequestHeader("ContentType",*

*"application/x-www-form-urlencoded");*

*xhttp.send("obj="+JSON.stringify(subject))*

*Серверный скрипт:*

*<?*

*if(isset($\_POST[“obj”]))*

*{*

*$obj=json\_decode($\_POST["obj"]);*

*updateInfo($obj->fio, $obj->age, $obj->address);*

*}*

*?>*

Альтернативным форматом для обмена данных при асинхронной передаче является xml. XML (eXtensible Markup Language) – расширенный язык разметки, ориентированный на хранение и передачу регулярных данных с ориентацией на сеть Интернет. С использованием XML можно структурировать описание сущностей, валидировать корректность их описания с использованием DTD-описания или XML-схем, описывать визуальное представление сущностей с использованием XSLT. Вот как может выглядеть список субъектов, зарегистрированных в некоторой информационной системе, средствами языка XML:

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

*<PersonList>*

*<Person status=”служащий”>*

*<FIO>Иванов Иван Иванович</FIO>*

*<BD>*

*<day>10</day>*

*<month>12</month>*

*<year>1990</year>*

*</BD>*

*<Address>ул.Мира, 22-181</Address>*

*</Person>*

*<Person status=”пенсионер”>…</Person>*

*<Person status=”студент”>…</Person>*

*</PersonList>*

*Инструментарий обработки данных в xml-формате поддерживается боль-*

*шинством используемых в Web-разработке языков программирования. Для*

*разбора xml-ответа на клиентский стороне можно использовать XML DOM:*

*xmlhttp=new XMLHttpRequest();*

*xmlhttp.open("GET",”some.xml",false);*

*xmlhttp.send();*

*xmlDoc=xmlhttp.responseXML;*

*document.getElementById(“fio").innerHTML=*

*xmlDoc.getElementsByTagName(“PersonList")[0].childNodes[0].nodeValue;*

Разбор XML-документа средствами DOM API может оказаться рутинной задачей. Упростить поиск, выборку, изменение данных в xml-документах можно с использованием языка XPath. XPath — это язык запросов к элементам xml или xhtml документа. Синтаксически схож с SQL, представляет собой декларативный язык запросов. XPath рассматривает документ как иерархическую модель из узлов, атрибутов и текстовых элементов.

var xhr = new XMLHttpRequest();

var xmldoc;

xhr.open("get", "personlist2.xml", true);

xhr.setRequestHeader("Content-type", "text/xml")

xhr.onreadystatechange = function(){

if (xhr.readyState == 4){

if (xhr.status >= 200 && xhr.status < 300){

xmldoc = xhr.responseXML;

var personCount = document.evaluate( 'count(//Person)', xmldoc,

null, XPathResult.ANY\_TYPE, null );

alert(“В ответе ”+PersonCount+” персоны”);

}

}

};

xhr.send(null);

XPath-запрос на клиентской стороне выполняется с использованием метода:

*document.evaluate(xpathExpression, contextNode,*

*namespaceResolver, resultType, result);*

где

xpathExpression – xpath выражение

contextNode – узел документа, относительно которого вычисляется выражение

namespaceResolver – функция для координации пространств имен выражения и документа

resultType – тип результата вычисления выражения

result - ссылка на уже существующий XPathResult

Перебор всех персон в вышеприведенном xml-файле с использованием xpath не вызывает трудностей:

str="";

var iterator = document.evaluate('//Person', xmldoc, null,

XPathResult.UNORDERED\_NODE\_ITERATOR\_TYPE, null);

var pNode = iterator.iterateNext();

while (pNode) {

str+="Персона:" +pNode.children[0].textContent+", Ид="+

pNode.attributes["status"].textContent+"<br>";

pNode = iterator.iterateNext();

}

d=document.getElementById("d");

d.innerHTML=str;

Программная реализация AJAX-обмена на клиентской стороне предполагает написание стереотипного, повторяющегося для различных запросов кода: кроссбраузерное создание объекта XMLHttpRequest, его параметризация, объявление функций-обработчиков событий для данного объекта. Упростить код и сосредоточиться только на разработки кода логики приложения можно, используя объекты классов FrontEnd-фреймворков - таких, как jQuery, Prototype, VanillaJS и др. Например, в библиотеке JQuery определен метод ajax, инкапсулирующий AJAX-обмен на клиентской стороне:

*jQuery.ajax( url [, settings] )*

или

*jQuery.ajax(settings )*

где url - URL адрес, на который будет отправлен Ajax запрос;

settings - набор параметров вида ключ / значение, которые настраивают запрос Ajax

Параметр settings позволяет всесторонне параметризовать ajax-обмен. Объект, передаваемый в качестве этого параметра, поддерживает широкий набор полей.

Шаблон ajax-запроса, сформированного средствами jQuery, выглядит следующим образом:

*$.ajax({*

*url: '/ajax/service.php',*

*dataType : "json",*

*data: {param1:val1,*

*param2;val2 },*

*success: function (data, textStatus) {*

*/\* ... \*/},*

*error:function(){*

*/\*…\*/ }*

*)};*

Метод ajax является «швейцарским ножом» асинхронных запросов библиотеки jQuery: он очень универсален, имеет множество настроек и применима для любых типов запросов. Если учесть, что чаще всего необходимо подавать обычные get- или post-запросы, то более удобными и компактными инструментами могут стать методы post() и get(). Например, метод get() имеет прототип:

*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 форме объекта, 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

Пример вызова метода get:

*$.get( "test.php", {“param1”: “value1”, “param2”:”value2”},*

*function( data ) {*

*alert( "Получены данные: " + data );*

*});*

*Фактически, вызов метода get() приводит к вызову метода ajax с парамет-*

*рами:*

*$.ajax({*

*url: url,*

*type: "GET",*

*data: data,*

*success: callback,*

*dataType: dataType*

*});*

**Порядок выполнения практической работы**

1. Изучите исходные тексты примеров серверных скриптов в настоящих указаниях.

2. Проанализируйте задание на работу.

3. Разработайте протокол обмена данными между клиентской и серверной частями Web-приложения (типы клиентских запросов, их параметры, формат и содержание ответа серверной части).

4. Напишите и отладьте скрипты, обеспечивающие асинхронный обмен данными между клиентской и серверной частями Web-приложения в соответствии с заданием к практической работе.

5. Составьте отчёт по практической работе

**Содержание отчёта**

Отчёт по практической работе должен содержать следующие сведения:

- название и цель работы

- вариант задания

- исходные тексты скриптов с комментариями

Контрольные вопросы

1. В чем преимущество асинхронных запросов к серверу? Каков механизм

их формирования и обработки?

2. Какой инструментарий формирования и обработки AJAX-запросов вы

знаете?

3. Какие форматы данных используются при асинхронном обмене данных

между клиентом и сервером?

4. Каким атакам подвержен асинхронный обмен данными между клиентом

и сервером?

5. Какие события сопровождают асинхронный обмен между клиентом и

сервером?

6. Обоснуйте свой выбор инструментария и протокола обмена при реализации асинхронного обмена данными между клиентом и сервером в рамках выполнения практической работы.

**Задание к практической работе**

Реализовать возможность асинхронного обмена данных клиентской и серверной частей с использованием технологии AJAX. Реализовать вывод информации об объектах Web-ресурса в краткой форме и асинхронную подгрузку полной информации об объектах при его выборе. Данные с сервера должны передаваться в формате json или xml. Для зарегистрированных пользователей реализовать возможность редактирования и удаления информации об объектах (желательно эти команды также подавать на сервер асинхронно).

**Список литературы и источников информации.**

1. В. Дронов. JavaScript и AJAX в Web-дизайне. СПб.: БХВ Петербург, 2012.

– 736 стр.

2. Д. Резиг, Р. Фергюсон, Д. Пакстон. JavaScript для профессионалов. М.:

Вильямс, 2015. – 240с

3. Современный учебник JavaScipt. https://learn.javascript.ru/

4. А. Фримен. JQuery для профессионалов. М.: Вильямс, 2015. – 1040с.

5. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. Н.

Прохоренок, В. Дронов. СПб.: БХВ Петербург, 2015. – 768 стр.