Министерство образования и науки Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Карельский Государственный Педагогический Университет

Студент 551 группы
И. В. Ермолин

Использование технологии AJAX в разработке web-приложений

Дипломная работа

Кафедра информатики
Научный руководитель:
старший преподаватель
А. С. Кюршунов

Петрозаводск
2007

Оглавление

Введение

Всемирная сеть изменялся на протяжении всей истории своего существования. Из набора проводов между компьютерами нескольких университетов США, она переросла в систему, объединяющую огромное количество разнообразных устройств. Выйдя за рамки военного эксперимента, сеть стала инструментом доступным простому человеку. С момента первого модемного соединения в 1990 году прошло лишь полтора десятилетия, однако, в настоящее время Интернет уже доступен через спутниковые связи, радиосигнал, кабельное телевидение, сотовые соединения, специальные оптико-волоконные линии, электропровода и даже через трубы водопровода.

С изменениями физической структуры сети, ростом её популярности, изменились и подходы к её использованию. Электронный ресурс сегодня далеко не простое хранилище текстовых документов, это настоящее приложение, чутко реагирующее на действия пользователя.

Всемирная сеть стала неотъемлемой частью жизни в развитых и развивающихся странах. Большинство организаций используют представительские функции Интернета, донося информацию о себе потенциальным клиентам. Многие из них идут дальше и переводят в сеть механизмы управления. Помимо этого существует ряд коммерческих проектов, основная деятельность которых связанна с оказанием исключительно информационных услуг в сети Интернет.

Серьёзное влияние развитие сетей оказывает и на процессы, происходящие в образовательной среде. Сегодня делаются огромные ставки на дистанционное обучение. Исходя из того, что профессиональные знания стареют очень быстро, необходимо их постоянное совершенствование. Дистанционную форму обучения дает возможность

создания систем массового непрерывного самообучения, всеобщего обмена информацией, независимо от временных и пространственных поясов. Кроме того, системы дистанционного образования дают равные возможности всем людям независимо от социального положения в любых районах страны и за рубежом реализовать права на образование и получение информации.

Помимо систем дистанционного образования, разрабатываются и внедряются системы, позволяющие дополнить и расширить взаимодействие между участниками традиционного учебного процесса.

Целью дипломной работы стало рассмотрение возможностей применения модели AJAX (одной из передовых моделей, применяемых в разработке динамичных web-интерфейсов) для создания прототипа системы поддержки учебного процесс. При этом решить задачи:

- изучить технологии, используемые в модели AJAX;
- разработать проект системы поддержки учебного процесса;
- выбрать технические средства для построения системы;
- реализовать прототип системы поддержки учебного процесса.

Дипломная работа состоит из 3-х глав. В первой главе «Модель AJAX» описаны основные принципы и технологии модели. Во второй главе «Выбор технических средств» рассмотрен критерии выбора и сам выбор технологий реализации системы. В последней главе «Реализация системы поддержки элементов учебного процесса» описываются технические моменты практической части дипломной работы.

1. Модель AJAX

В данной главе рассмотрены основные принципы и технологии, используемые при разработке web-приложений с применением модели AJAX.

Общая схема работы стандартного web-приложения выглядит следующим образом. Пользователи заполняют поля форм на HTML-странице и нажимают кнопку «Подтвердить». Затем форма полностью передается на сервер, который запускает необходимый сценарий, и возвращает результаты его работы в виде новой HTML-страницы. Это может документ с новой формой, либо документ подтверждения, либо документ с вариантами, зависящими от введенных в оригинальную форму данных. Но пока сценарий или программа на сервере не обработает и не вернёт новую форму, пользователи должны ждать. Их экраны очистятся, и будут перерисовываться по мере поступления новых данных от сервера. Вот где проявляется низкая интерактивность — пользователи не получают немедленной обратной реакции и определенно чувствуют себя не так, как при работе с настольными приложениями. Сократить разрыв в интерактивности сложившийся между обычным настольным приложением и web-приложением и призвана технология AJAX.

AJAX (Asynchronous JavaScript and XML) базируется на двух основных принципах:

- использование технологии динамического обращения к серверу «на лету», без перезагрузки всей страницы полностью;
- использование DHTML для динамического изменения содержания страницы.

По существу AJAX помещает JavaScript между вашей web-формой и сервером. Когда пользователи заполняют формы, данные передаются в JavaScript-код, который эти данные собирает и в фоновом режиме передает на сервер; в это время пользователь даже не замечает, что происходит какая-то работа с сервером, он может продолжать вводить данные, прокручивать страницу и работать с приложением.

Ядром и главной особенностью модели AJAX является динамическое общение с сервером. Для осуществления этого чаще всего прибегают к использованию объекта XMLHttpRequest.

1.1. Объект XMLHttpRequest

Данный объект пока не стандартизирован в W3C, поэтому разные браузеры поддерживают его немножко по-разному. Например, в Microsoft Internet Explorer это ActiveX объект, в Firefox, Safari, Opera это объект языка JavaScript. Однако такая ситуация не должна нас сильно настораживать. Даже такие стандарты как HTML, JavaScript, CCS, которые консорциум W3C уже давно стандартизировал, до сих пор обрабатываются в разных браузерах по-своему. К примеру: почти полную и правильную поддержку стандарта CSS реализуют лишь последние версии Opera, остальные популярные браузеры поддерживают стандарт несколько свояка, что не всегда заметно в стандартных страницах, но выдаётся на синтетических тестах. Поэтому при разработке любого web-приложения следует ориентироваться на возможности популярных браузеров, занимающих 99% рынка.

После инициализации объекта XMLHttpRequest, мы можем с помощью него отправить запрос к любому http-серверу в сети Интернет. Как только сервер вернёт результат, наш объект изменит свойства и вызовет заранее описанный метод обработки результата.

Основные методы. Перечислим основные методы объекта XMLHttpRequest, которые поддерживаются всеми популярными браузерами:

- abort(): обрывает текущий запрос.
- getAllResponseHeaders(): помимо самого документа сервер передаёт дополнительные сведения о нём, которые хранятся в заголовках ответа. Получить все заголовки в виде строки позволяет данный метод.
- getResponseHeader(header): возвращает в виде строки заголовок, название которого указанно в параметре header.
- open(method, URL[, asyncFlag, userName, userPass]): присваивает запросу параметры:
    - method — метод соединения (GET, POST, PUT);
    - URL — адрес документа;
    - asyncFlag — продолжать ли работу сценария во время выполнения запроса (true, false)?
    - userName, userPass — имя пользователя и пароль, если это необходимо.
- send(content): отправка запроса.
- setRequestHeader(header, value): установить заголовок header запроса в значение value.

Основные свойства. Перейдём к рассмотрению основных свойств объекта:

- onreadystatechange: событие возникающее при смене статуса объекта.
- readyState: значение статуса (integer), может принимать следующие значения: 0 — не инициализированный запрос, 1 — идёт загрузка ответа, 2 — ответ загружен, 3 — обработка ответа, 4 — запрос выполнен.

- responseText: возвращённые сервером данные в виде строки.
- responseXML: возвращённые сервером данные в виде DOM.
- status: возвращённый сервером стандартный HTTP-код статуса (404 — документ не найден, 200 — всё прошло гладко).
- statusText: возвращённое сервером текстовое сообщение статуса.

Пример использования. Для начала взглянем на простейший пример страницы использующей объект XMLHttpRequest. Потребуем, чтобы наша страница получила данные из файла «`01_data.txt`» расположенного на сервере, и отобразила их во всплывающем окне.

Файл «01_doc.html»

```
01. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02. <html>
03.
04. <head>
05.   <link rel="stylesheet" type="text/css" href="../style/style.css" />
06.   <title>Пример использования XMLHttpRequest</title>
07.
08.   <script type="text/javascript">
09.     var my_request;
10.
11.     function my_request_create() {
12.       if (window.ActiveXObject) {
13.         my_request = new ActiveXObject("Microsoft.XMLHTTP");
14.       } else if (window.XMLHttpRequest) {
15.         my_request = new XMLHttpRequest();
16.       }
17.     }
18.
19.     function my_request_response() {
20.       if(my_request.readyState == 4) {
21.         if(my_request.status == 200) {
22.           alert("Сервер сообщает: " + my_request.responseText);
23.         }
24.       }
25.     }
26.
27.     function my_request_start() {
28.       my_request_create();
29.       my_request.onreadystatechange = my_request_response;
30.       my_request.open("GET", "01_data.txt", true);
31.       my_request.send(null);
32.     }
33.   </script>
34. </head>
35.
36. <body>
37.   <h1>Динамическое обращение к серверу</h1>
38.   <form action="#">
39.     <input type="button" value="Обратиться к серверу" onclick="my_request_start();" />
40.   </form>
```

```
41. </body>
42. </html>
```

HTML-документ включает в себя две основные части: HTML-код, описывающий расположение формы с кнопкой, и JavaScript-код, выполняющий работу с объектом XMLHttpRequest.

Все действия описаны в трёх функциях. При нажатии на кнопку «Обратиться к серверу» будет вызвана функция my_request_start(), которая осуществляет обращение к серверу. Она состоит из четырёх частей:

Сначала создаётся объект XMLHttpRequest, производится это в функции my_request_create(), которую следует рассмотреть более подробно. В ней глобальная переменная my_request объявляется либо как ActiveX объект (для браузера Internet Explorer), либо как объект XMLHttpRequest (для браузеров, не поддерживающих ActiveX), но, не смотря на различия в инициализации экземпляра класса, дальнейшая работа с объектами будет происходить одинаково во всех браузерах.

После выполнения инициализации, мы устанавливаем функцию, которую необходимо вызвать при возникновении каких-либо событий в объекте (строка 29). Пока нас интересует только событие, оповещающее о получении ответа от сервера.

Назначив функцию обработки событий, мы открываем запрос и связываем его с относительным адресом URL «01_data.txt» (можно использовать и абсолютную адресацию). Затем мы передаём серверу наш запрос, прикрепив к нему пустые данные (строка 31).

Но обратившись к документу мы не получим мгновенного ответа. Сервер должен обработать наш запрос и выдать получившийся результат, что естественно занимает некоторое время. Но как только ответ от сервера

будет получен, интерпретатор JavaScript оповестит об этом объект XMLHttpRequest и вызовет указанную ранее функцию обработки событий.

1.2. Объектная модель документа

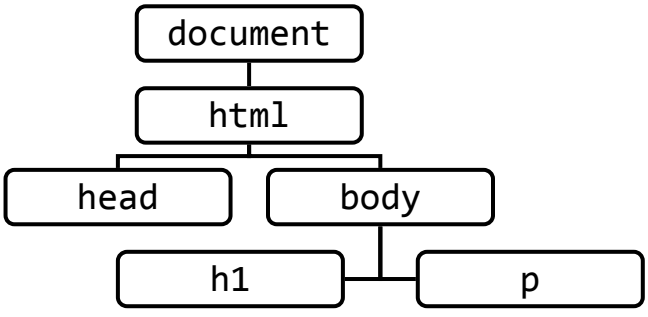
Мы должны иметь возможность изменять документ, иначе любое общение с сервером становится бесполезным.

Для начала необходимо познакомиться с понятием DOM (объектная модель документа). Не углубляясь в подробности, браузер представляет XML-документ (и HTML, как частный случай) в виде специального дерева, состоящего из узлов.

HTML-документ

```
01. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
02. <html>
03.   <head> <title>Название страницы</title> </head>
04.   <body>
05.     <h1 style="color:#FF0000;">Заголовок</h1>
06.     <p id="p1">Текст</p>
07.   </body>
08. </html>
```

Данный HTML-документ может быть представлен, например, как дерево:



Каждый узел содержит свойства, отражающие его структуру и методы позволяющие перемещаться по соседним узлам дерева документа. Корневым узлом является объект document. У него есть дочерние узлы

(child nodes), но отсутствуют родительский узел (parent node) и узлы одного с ним уровня.

Перемещение по дереву документа можно осуществлять несколькими способами. Относительное перемещение:

- element.firstChild: первый потомок узла;
- element.childNodes: массив потомков узла;
- element.lastChild: последний потомок узла;
- element.previousSibling: соседний узел;
- element.nextSibling: соседний узел;
- element.parentNode: родитель узла.

Абсолютное перемещение, полезно в динамически меняющемся дереве узлов и доступно с помощью методов:

- document.getElementsByTagName("p"): возвращает массив элементов, у которых имя тега равно p;
- document.getElementById("p1"): возвращает указатель на элемент, у которого атрибут id="p1".

Получив указатель на элемент объектной модели HTML-документа, можно изменить его свойства и содержимое, меняя тем самым оформление и наполнение документа в целом.

Однако, на данный момент всё ещё приходится сталкиваться с определёнными трудностями, связанными с различной поддержкой разными браузерами стандартов DOM.

1.3. Возможные схемы обмена данными

Мы уже использовали обращение к статическим текстовым данным. Но в больших проектах разбор приходящих строк реализовать гораздо сложнее. Для облегчения данной задачи существуют стандартные схемы общения сервера и клиента.

Первая схема основывается на объектной модели XML-документов. При её использовании, сервер возвращает результаты в виде XML-документа, который на стороне клиента преобразуется в объект DOM (свойство `responseXML` объекта XMLHttpRequest), обладающий широким спектром методов по извлечению из себя необходимых данных.

Возвращаемый сервером текст

```
01. <?xml version="1.0" encoding="UTF-8"?>
02. <data>
03.   <record>
04.     <author>Author 1</author>
05.     <album>Album 1</album>
06.     <track>Track 1</track>
07.     <track>Track 2</track>
08.     <track>Track 3</track>
09.   </record>
10. </data>
```

Функция обработки входящих данных

```
01. function my_request_response() {
02.   if(my_request.readyState == 4) {
03.     if(my_request.status == 200) {
04.       var my_data = my_request.responseXML;
05.       var t1 = my_data.getElementsByTagName("track");
06.
07.       var t1_length = t1.length;
08.       var t1_text = "";
09.       for (i=0; i<t1_length; i++) {
10.         t1_text = t1_text + "\n" + t1[i].firstChild.data;
11.       };
12.
13.       alert(t1_text);
14.     };
15.   };
16. };
```

Функция обработки ответа извлекает из объекта XMLHttpRequest DOM-структуру (строка 04); получает массив элементов этой структуры обозначенные тэгом <track> (строка 05). Затем, достаёт из элементов

массива текстовые данные (строки 07-11) и выводит их во всплывающем окне (строка 13).

Вторая схема основана на том, что вместо текста, который ещё необходимо обработать, сервер может вернуть текст, который уже является готовым к выполнению JavaScript-кодом. И если мы полностью доверяем данным сервера, то мы можем выполнить полученный код с помощью функции eval().

Возвращаемый сервером текст

```
01. alert("text");
```

Функция обработки входящих данных

```
01. function my_request_response() {
02.   if(my_request.readyState == 4) {
03.     if(my_request.status == 200) {
04.       eval(my_request.responseText);
05.     };
06.   };
07. };
```

В данном примере функция выполняет пришедший от сервера JavaScript-код (строка 04).

Первая схема передачи данных достаточно универсальна и легка в отладке, вторая же позволяет создавать web-приложения, код которых достаточно велик и не может быть полностью передан клиенту единовременно, вместо этого код поступает частями по мере необходимости.

2. Выбор технических средств

В главе рассматриваются технические средства, используемые при создании системы электронной поддержки элементов учебного процесса, а также причины их выбора. Хочется отметить, что отправными точками при выборе технических средств, стали две предпосылки: первое — личное предпочтение к языку программирования java, второе — желание разобраться в незнакомых ранее технологиях.

2.1. HTTP-сервер Tomcat

В связи с ориентацией на язык программирования Java, выбор HTTP-сервера стоял между Apache Tomcat и Resin и был сделан в пользу первого, как более распространённого.

Контейнер сервлетов занял место основного HTTP-сервера, был настроен на 80-ый порт (для протокола HTTP) и на 443 порт (для протокола HTTP с шифрованием данных). Использовалась версия 6.10, поддерживающая спецификации Java Servlet 2.5 и Java Server Pages 2.1.

2.2. Система компиляции Ant

На первых этапах разработки наблюдались сложности связанные с компиляцией исходных кодов проекта (в основном при работе с директивой classpath). Для автоматизации процесса и компиляции приложения, была использована программа Ant. В пользу системы можно отметить использование её, как основы в некоторых средах разработки, например, NetBeans.

2.3. Сервер баз данных MySQL

Использование сервера баз данных нельзя оправдать его резкой необходимостью в проекте. Важными предпосылками к использованию такого решения стали. Наращивание личного опыта по использованию баз данных в java-ориентированных web-приложениях. Возможность, без серьезного изменения кода проекта, разделить уровень данных и уровень логики. Помимо этого, реализовать выборку из базы иногда оказывается легче, чем писать собственную систему хранения данных на основе файлов.

Выбор был остановлен на MySQL, в основном из-за его свободности и распространённости данной системы управления базами данных.

Так же, потребовалось настроить связь контейнера сервлетов с базой данных MySQL, связь была настроена путём размещения в директории libs сервера Tomcat jar-библиотеки mysql-connector-java-5.0.5-bin.jar, которая содержит драйвер com.mysql.jdbc.Driver.

2.4. JavaScript-библиотека jQuery

Как уже говорилось, разработчики web-обозревателей не всегда строго соответствуют стандартам W3C. Избежать оглядки на модели браузеров при написании JavaScript-кода позволяют специальные JS-библиотеки, предоставляющие функционал, работающий во всех популярных системах и достаточный для полного управления страницей.

Здесь наблюдаются две тенденции. С одной стороны разрабатываются библиотеки, предоставляющие только базовый набор функций для перемещения и изменения дерева объектов HTML-страницы, для работы со свойствами элементов, для манипуляции событиями и асинхронными запросами к серверу. Дополнительные функции приобретаются по мере

развития библиотек в основном в виде отдельных JavaScript-приложений сторонних разработчиков. Среди таких библиотек можно выделить:

- prototype;
- jQuery;
- mootools.

С другой же стороны стоят библиотеки «всё в одном», представляющие из себя модульные системы и включающие огромный набор дополнительных интерфейсов. Интерфейсов для работы с табличными данными, с непосредственным форматированным текстовой информации и так далее. Наиболее популярные:

- Dojo;
- Yahoo User Interface (YUI);
- Atlas.

Хочется отметить, что зачастую библиотеки снабжены скудной документацией, на этом фоне с положительной стороны можно выделить библиотеку YUI. Однако, в проекте мне не требуется постоянное использования специфических элементов управления, исходя из этого я выбрал библиотеку jQuery, как компактную (21 Кб) библиотеку, обладающую богатым набором базовых функций, поддерживающую браузеры Internet Explorer 5.5+, Firefox 1.0+, Safari 1.3+, Opera 8.5+.

Селектор $(elem) является ядром системы и возвращает объект класса jQuery, связанный с элементами, подходящими под описание elem, с которым затем можно производить различные манипуляции. Например:

- $(‘.my_c’): получение объекта jQuery, связанного со всеми элементами страницы, у которых установлен атрибут class=«my_c»;

- $('#my_i'): получение объекта jQuery, связанного со всеми элементами страницы, у которых установлен атрибут id=«my_i»;
- $(body): получение объекта jQuery, связанного с элементом body HTML-страницы.

Основные методы jQuery для работы с DOM:

- Навигация:
    - $(elem).children(): получение массива дочерних объектов;
    - $(elem).parent(): получение родительского объекта.
- Манипуляция:
    - $(elem).empty(): удаление содержимого подходящих элементов;
    - $(elem).append(content): добавление контента content подходящим элементам.
- Работа с атрибутами:
    - $(elem).attr(key, value): установка значения атрибута с именем key и значением value для всех подходящих элементов;
    - $(elem).attr(key): получение значения атрибута с именем key от первого подходящих элементов;

Основные методы jQuery для работы с CSS:

- $(elem).css(key, value): установка свойства стилей key в значением value, если используется численное значение, то оно автоматически конвертируется в значение в пикселях;
- $(elem).css(key): получение значения свойства стилей key от первого подходящего элемента.

Основные методы jQuery для работы с событиями:

- $(elem).bind(ev, fn): установка функции fn для обработки события ev, происходящего во всех подходящих элементах;
- $(elem).keypress(fn): установка функции fn для обработки нажатий клавиш в подходящих элементах.

Основные эффекты:

- $(elem).show(speed, fn): показывает каждый из подходящих элементов, используя анимацию путём увеличения высоты, вызывает функцию fn по завершению анимации. Параметр speed указывает либо одну из определенных скоростей («slow», «normal» или «fast»), либо время работы анимации в миллисекундах;
- $(elem).hide(speed, fn): аналогично скрывает подходящие элементы путём уменьшения высоты.
- $(elem).fadeIn(speed, fn): аналогично показывает подходящие элементы, путём уменьшения прозрачности.
- $(elem).fadeOut(speed, fn): аналогично скрывает подходящие элементы путём увеличения прозрачности.

Основная функция осуществления запроса к http-серверу:

- $.ajax(prop): осуществление асинхронного запроса с параметрами prop. Пример, обращение к адресу «http://localhost/», передача параметров x=1 и y=2, вывод сообщения после получения ответа:

```
Пример осуществления асинхронного запроса к серверу
```

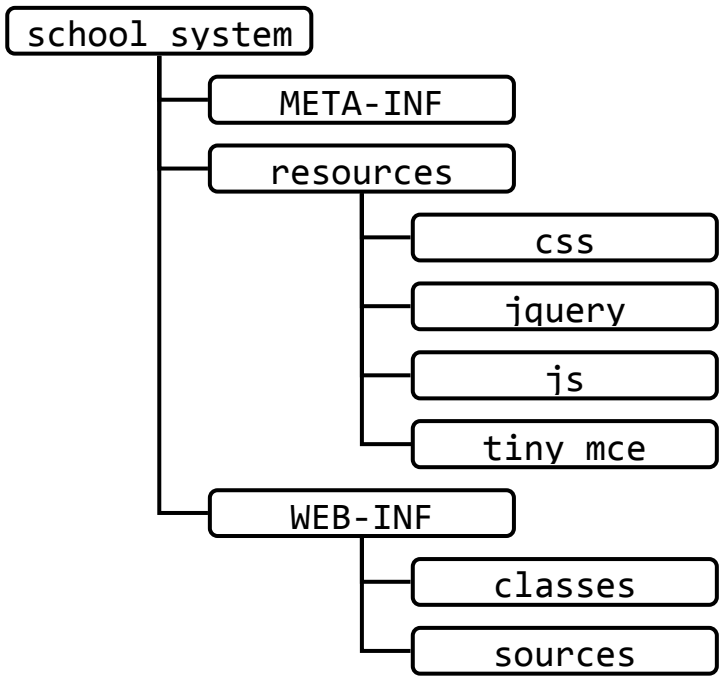
```
$.ajax({
  type: "POST",
  url: "http://localhost/",
  data: {
    x: "1",
    y: "2"
  },
  success: function(msg){
    alert("Data Saved: " + msg);
  }
});
```

3. Реализация системы поддержки элементов учебного процесса

В главе описывается реализация системы поддержки элементов учебного процесса school_system. Сначала рассматривается структура приложения; далее работа клиентской части приложения; затем реализация серверной части в целом и её модулей в частности.

3.1. Структура приложения

Структура каталогов проекта school_system:



Рассмотрим файлы, влияющие на настройку контейнера сервлетов, для работы с системой school_system.

Во-первых, в каталоге META-INF содержится информация, о ресурсах доступных всем классам проекта. В данном случае описан только один ресурс — хранили данных jdbc/public_db. Оно связано по средствам драйвера com.mysql.jdbc.Driver с базой данных school_system, которая

расположена на сервере MySQL по адресу 127.0.0.1:3306. Доступ к базе данных можно получить под именем пользователя school_system с паролем school_system.

context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>
  <Resource name="jdbc/public_db"
            auth="Container"
            type="javax.sql.DataSource"
            username="school_system"
            password="school_system"
            driverClassName="com.mysql.jdbc.Driver"
            url="jdbc:mysql://127.0.0.1:3306/school_system?autoReconnect=true"
            maxActive="100"
            maxIdle="30"
            maxWait="5000">
  </Resource>
</Context>
```

Драйвер com.mysql.jdbc.Driver в виде jar-архива должен быть предварительно расположен в каталоге libs сервера tomcat.

Во-вторых, в каталоге WEB-INF хранится файл настройки контейнера сервлетов — web.xml. Он подключает к системе только что описанное хранилище данных:

Из файла «context.xml»

```
...
 <!-- resources -->
 <resource-ref>
  <res-ref-name>jdbc/public_db</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
 </resource-ref>
...
```

Затем настраивает фильтр, перехватывающий все запросы к сервлетам и устанавливающий в них кодировку UTF-8, для корректной работы с русскими символами:

Из файла «context.xml»

```
...
<!-- filters -->

 <filter>
  <filter-name>filter_charset</filter-name>
  <filter-class>ru.ptz.box.school_system.filters.filter_charset</filter-class>
  <init-param>
   <param-name>request_encoding</param-name>
   <param-value>UTF-8</param-value>
  </init-param>
 </filter>

<filter-mapping>
  <filter-name>filter_charset</filter-name>
  <url-pattern>/*</url-pattern>
 </filter-mapping>
...
```

Далее сервлеты привязываются с определёнными URL-адресами:

Из файла «context.xml»

```
...
<!-- servlets -->

<servlet>
  <servlet-name>servlet_main</servlet-name>
  <servlet-class>ru.ptz.box.school_system.servlets.servlet_main</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>servlet_main</servlet-name>
  <url-pattern>/servlets/servlet_main/</url-pattern>
 </servlet-mapping>
...
```

В-третьих, в том же каталоге `WEB-INF` расположена система компиляции проекта, основанная на использовании программы `Ant`. Файл `compil.bat` запускает программу `Ant`, передавая ей имя `build.xml` файла в качестве параметра.

build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="main" default="compil" basedir=".">

  <property name="java_lib" value="C:\Java\java_lib\" />
```

```
  <path id="classpath">
    <fileset dir="${java_lib}">
      <include name="*.jar" />
    </fileset>
  </path>

  <target name="compil">
    <javac srcdir="sources" destdir="classes" encoding="UTF-8">
      <compilerarg value="-Xlint"/>
      <classpath refid="classpath" />
    </javac>
  </target>

</project>
```

Сначала генерируется строка, содержащая пути к классам, которые могут понадобиться для компиляции, затем все исходные файлы в каталоге sources компилируются и размещаются в каталоге classes.

Рассмотрев настройку контейнера сервлетов, перейдём к описанию основной системы.

Условно систему можно разделить на две части. Клиентская часть — index.html и подключаемые к ней файлы из каталога resources; серверная часть — классы генерирующие ответы на запросы клиентской части.

3.2. Клиентская часть

Файл index.html образует систему панелей:

<table>
<tr><td>USER (заголовок)</td><td>Рабочая область (заголовок)</td></tr>
<tr><td>USER (информационная часть)
USER (тело панели)</td><td rowspan="4">Рабочая область</td></tr>
<tr><td>MSG (заголовок)</td></tr>
<tr><td>MSG (информационная часть)
MSG (тело панели)</td></tr>
</table>

В левой области расположены панелями навигации, служащими для доступа либо к части системы по работе с данными пользователя, либо к части, которая работает с сообщениями пользователей (сервлеты servlets.servlet_pnl2_u, и servlets.servlet_pnl2_m соответственно).

Изначально, навигационные панели находятся в свёрнутом состоянии. Не отображается тело панели, но видны заголовок и информационная часть. Необходимо отметить, что в информационной части отображается информация необходимая всегда (например, количество новых сообщений в информационной области панели MSG). Тело же панели содержит информацию, требующуюся только при работе с выбранной подсистемой.

При нажатии на панель USER происходит стандартный запрос к сервлету servlets.servlet_pnl2_u с командой show в качестве параметра, после чего сервлет возвращает JS-код, который выполняясь, будет оформляться рабочую область и скрытое до этого тело панели USER.

Аналогичным образом действует панель MSG, только общается она с сервлетом servlets.servlet_pnl2_m.

Процедура отправки стандартного запроса состоит из двух частей. Первый этап запрос на обновление рабочей области и тела панели навигации вызвавшей этот запрос.

Но на этом процедура не заканчивается, после того, как начнут поступать первые байты ответа на стандартный запрос, клиентская часть сгенерирует дополнительный запрос к сервлету servlets.servlet_main который вышлет, JS-код для необходимого изменения в информационных частях обоих панелей. Таким образом, информационные части панелей навигации поддерживаются в актуальном состоянии.

Итак, лицом клиентской части является файл index.html. Первым делом к нему подключаются таблицы стилей и библиотеки JavaScript из директории resources.

Из файла index.html

```
...
<!-- Подключение таблиц стилей -->
<link rel="stylesheet" type="text/css" href="resources/css/reset.css" />
<link rel="stylesheet" type="text/css" href="resources/css/main.css" />

<!-- Подключение сторонних библиотек -->
<script type="text/javascript" src="resources/jquery/jquery-1.1.2.pack.js">
</script>
<script type="text/javascript" src="resources/tiny_mce/tiny_mce.js">
</script>

<!-- Подключение собственного кода -->
<script type="text/javascript" src="resources/js/main.js">
</script>
...
```

Подключается библиотека jQuery, WYSWYG-редактор TinyMCE и собственный JavaScript-код, выполняющий базовые операции. Структура html-файла:

Из файла index.html

```
...
<!-- Основная часть -->
<div id="my_body">
  <!-- Область навигации -->
  <div id="pnls" >
    <!-- Панель "пользователи" -->
    <div class="pnls_pnl" id="pnl_u">
      <div class="pnls_pnl0" id="pnl0_u">Пользователь</div>
      <div class="pnls_pnl1" id="pnl1_u">
        <h1>Гостевая запись</h1>
        <p>Гость</p>
      </div>
      <div class="pnls_pnl2" id="pnl2_u">body users</div>
    </div>

    <!-- Панель "сообщения" -->
    <div class="pnls_pnl" id="pnl_m">
      <div class="pnls_pnl0" id="pnl0_m">Сообщения</div>
      <div class="pnls_pnl1" id="pnl1_m">info messages</div>
      <div class="pnls_pnl2" id="pnl2_m">body messages</div>
    </div>
  </div>
```

```
  <!-- Рабочая область -->
  <div id="info">
    <div class="pnls_pnl0" id="info_head">Загрузка…</div>
    <div id="info_body"></div>
  </div>
</div>

<!-- Дополнительная часть -->
<div id="dialog_background" onclick="f_mdl_close();"></div>
<div id="dialog_main">
  <div id="dialog_head">Диалоговое окно</div>
  <div id="dialog_body"></div>
</div>
...
```

Основная часть документа доступна пользователю постоянно и, как уже говорилось, разделена на две части: область навигации (левая), рабочая область (правая). В области навигации расположены две панели: «USER» и «MSG», при нажатии на которые происходит переход к работе с той или иной подсистемой программы.

Дополнительная часть документа содержит диалоговое окно, которое появляется только в определённых ситуациях и по умолчанию скрыто. Видно, что документ не наполнен каким-либо содержательным смыслом, наполнение документа будет происходить по мере обращения к серверу.

Базовые JS-функции системы, могут быть сгруппированы следующим образом:

- функции отправки запросов к серверу и приёма от него ответов;
- функции обработки перемещений между панелями левой области страницы (переключение между модулем обработки данных о пользователе и модулем обработки сообщений);
- функции проявления и скрытия области диалогового окна;
- функции инициализации.

Рассмотрим функции запросов к серверу и приёма от него ответов. Стандартный запрос к серверу начинается с вызова функции:

Из файла main.js: функция f_request(p_addr, p_param)

```
...
function f_request(p_addr, p_param) {
  $.ajax({
    type: "POST",
    url: p_addr,
    data: p_param,
    success: f_success_01
  });
};
...
```

Параметр p_addr — адрес, к которому необходимо обратиться и p_param — объект содержащий параметры для передачи серверу. Полученные параметры перенаправляются в стандартную функцию библиотеки jQuery, которая сориентируется в клиентском браузере, осуществит корректный запрос по указанному адресу и вызовет f_success_01 в случае успешного приёма ответа от сервера, передав ей полученное сообщение в качестве параметра.

Функция f_success_01 первым делом отсылает дополнительный запрос к серверу, а затем, выполняет код ответа на стандартный запрос:

Из файла main.js: функция f_success_01(msg)

```
...
var f_success_01 = function(msg) {
  $.ajax({
    type: "POST",
    url: "/school_system/servlets/servlet_main/",
    success: f_success_02
  });
  if(!(msg == null)) {
    eval(msg);
  };
};
```

В конце концов, поступает ответ на дополнительный запрос, выполнением кода которого занимается функция f_success_02:

Из файла main.js: функция f_success_02(msg)

```
var f_success_02 = function(msg) {
```

```
  if(!(msg == null)) {
    eval(msg);
  };
};
...
```

Опустив функции, описать которые можно, как осуществляющие открытие / закрытие панелей навигации и диалогового окна, выделим основные моменты в инициализации.

Из файла main.js: инициализация

```
...
v_pnl_u = $("#pnl_u");
v_pnl0_u = $("#pnl0_u");
v_pnl1_u = $("#pnl1_u");
v_pnl2_u = $("#pnl2_u");
v_pnl_m = $("#pnl_m");
v_pnl0_m = $("#pnl0_m");
v_pnl1_m = $("#pnl1_m");
v_pnl2_m = $("#pnl2_m");
v_info = $("#info");
v_info_head = $("#info_head");
v_info_body = $("#info_body");
v_dialog_head = $("#dialog_head");
v_dialog_body = $("#dialog_body");

f_pnl_click(v_pnl_u);

$(".pnls_pnl").bind("click", f_pnl_click);
...
```

Здесь, ряд глобальных переменных связывается с элементами документа index.html, имитируется нажатие на панель «пользователи», тем самым формируется запрос для обновления рабочей (правой) области страницы. Последняя строка сообщает, что функцию f_pnl_click необходимо вызывать при каждом нажатии на навигационные панели (элементы документа, принадлежащие классу «pnls_pnl»).

3.3. Структура серверной части системы

Перед непосредственной работой с запросом клиента, он попадает на обработку специальному фильтру. При инициализации фильтр получает от

контейнера сервлетов параметр, определяющий кодировку, которую необходимо устанавливать на входящие запросы:

Из файла filter_charset.java

```
...
enc = config.getInitParameter("request_encoding");
...
```

Далее при обнаружении запросов, заголовок которых начинается со строки application/x-www-form-urlencoded, фильтр назначает им нужную кодировку. И передаёт далее по цепочке фильтров:

Из файла filter_charset.java

```
...
String temp_s = rq.getContentType();
if (temp_s != null) {
  if (temp_s.startsWith("application/x-www-form-urlencoded ")) {
    rq.setCharacterEncoding(enc);
  };
};
next.doFilter(rq, rs);
...
```

Основную серверную часть проекта можно рассматривать как трёхуровневую систему. Во-первых, уровень данных — таблицы в базе данных MySQL:

- таблица box_user:
    - id: INTEGER AUTO INC — идентификатор пользователя;
    - name_short: VARCHAR(32) — псевдоним (логин);
    - name_full: TEXT — фамилия имя отчество;
    - group: VARCHAR(5) — группа (admin, teach, child);
    - group_addon: VARCHAR(32) — примечание (10б, Математика, Литература и т.д.);
    - year: DATE — дата рождения;
    - password: VARCHAR(32) — пароль;

    - uid: VARCHAR(32) — для сохранения сеанса между посещениями системы;
    - last_msgread: DATETIME — время последнего чтения сообщений (для отображения новых);
    - last_registre: DATETIME — время регистрации (для отсечения сообщений, присланных до регистрации);
- таблица box_messages:
    - id: INTEGER AUTO INC — идентификатор сообщения;
    - date: DATETIME — дата отправки сообщения;
    - text: TEXT — сам текст сообщения;
    - from_id: INTEGER — идентификатор пользователя от которого поступило сообщение;
    - to_name_short: VARCHAR(32) — сообщение для пользователя с данным логином;
    - to_group_addon: VARCHAR(32) — сообщение для пользователей с данным примечанием (например 10б, 11а, Математика и т.д.).

Во-вторых, классы, работающие с этими данными непосредственно:

- users.box_user;
- users.box_user_record;
- messages.box_messages.

В-третьих, сервлеты:

- servlets.servlet_main;
- servlets.servlet_pnl2_u;
- servlets.servlet_pnl2_m.

Сервлеты на основе работы выше перечисленных классов генерируют JavaScript-код для выполнения на стороне клиента. Все они построены на основе одного каркаса:

Сначала выясняется, от имени, какого пользователя выполняется запрос и создание на основе этой информации объекта класса users.box_user. При простом перемещении по страницам, объект может быть извлечён из сессии пользователя.

Каркас: выявление, от имени кого выполняется запрос

```
...
HttpSession sess = req.getSession();
box_user user = (box_user) sess.getAttribute("user");
...
```

В противном случае, не исключено, что пользователь ранее общался с системой и сохранил в ней своё посещение. Для выявления такой ситуации необходимо сопоставить информацию из cookie пользователя с псевдонимом uid с определенным параметром box_user.uid в базе данных.

Каркас: выявление, от имени кого выполняется запрос

```
...
if (user == null) {
  user = new box_user();
  Cookie[] cooks = req.getCookies();
  if (cooks != null) {
    for (int i = 0; i < cooks.length; i++) {
      if (cooks[i].getName().equals("uid")) {
        user.uid_load(cooks[i].getValue());
      };
    };
  };
};
...
```

Далее из запроса извлекаются имя команды, которую необходимо выполнить и дополнительные параметры для неё.

Каркас: извлечение параметров запроса

```
...
String cmd = req.getParameter("cmd");
if (cmd == null) {
  cmd = "";
};
String p_id = req.getParameter("id ");
if (p_id == null) {
  p_id = "";
} else {
  p_id = URLDecoder.decode(p_id, "UTF-8");
};
...
```

Затем, выполняется обработка пришедшей команды. И на её основе составляется строка для отправки. В последнюю очередь, произошедшие в объекте user изменения записываются в сессию, устанавливается заголовок ответа и печатается только что составленная строка (содержащая JavaScript-код для выполнения на стороне клиента):

Каркас: завершение работы сервлета

```
...
sess.setAttribute("user", user);
res.setContentType("text/javascript; charset=UTF-8");
PrintWriter out = res.getWriter();
out.println(result.toString());
out.close();
...
```

Условно, каждый сервлет можно обозначить как отдельный модуль системы. Рассмотрим каждый модуль в отдельности.

4.3. Модуль работы с пользователями

4.4. Модуль работы с сообщениями

4.5. Обобщающий информационный модуль

Заключение

В дипломной работе были рассмотрены технические основы одной из моделей, используемых при разработке web-приложений — AJAX. Цель работы (создание прототипа системы поддержки элементов учебного процесса) была достигнута. Хочется отметить трудности, возникающие при использовании подобного подхода.

Во-первых, использование AJAX, как основы для web-приложения, пока не целесообразно. Это связанно с отсутствием JS-библиотек совмещающих в себе и набор всех элементов интерфейса, и достойную документацию, и чёткую модульную систему. Из библиотек рассмотренных мною, при подготовке дипломной работы, могу выделить только чётко структурированную и хорошо документированную систему YUI, которую, правда, пока нельзя назвать полнофункциональной. С другой стороны использовать небольшие библиотеки (такие, как jQuery), для разработки отдельных элементов web-приложений, можно и нужно уже сейчас.

Второй трудностью становится выбор системы обмена данными между клиентом и сервером. Здесь важно соблюсти баланс кодом, который передаётся на сторону клиента единовременно при первом входе и кодом, который подгружается «на лету». Одна крайность приводит к долгой начальной загрузке системы, вторая к трудностям в отладке и плохой читаемости кода на стороне сервера.

Третьей трудностью, с которой мне пришлось столкнуться, стало малое количество серьёзной литературы, описывающей не отдельные элементы использования технологии AJAX, а рассматривающей стратегию применения технологии в больших проектах.

С другой стороны модель AJAX обладает и сильными сторонами, заметными уже сейчас. Можно отметить возможность снижения трафика между клиентом и сервером за счёт отсутствия передачи оформления данных для каждой страницы сайта, а передаче его один раз при первом визите.

Несомненно, расширение стандартного web-интерфейса за счёт AJAX-элементов управления повышает интерактивность приложений. При этом не требуется загрузки дополнительных компонентов (таких как flash player или виртуальная машина Java), необходим лишь браузер (с поддержкой JavaScript).

Уже сейчас AJAX постепенно проникает в повседневные web-приложения (форумы, чаты, блоги и др.). А если заглянуть чуть дальше и проанализировать такие проекты, как Gmail, Google Maps, то можно сделать менее очевидные выводы. Ведь уже реализованы прототипы приложений заменяющих word и excel, таким образом, браузер может стать, чуть ли не единственной необходимой программой.

Список литературы

1. Крейн, Д. Ajax в действии / Д. Крейн, Д. Джеймс, Э. Паскарелло // Вильямс, 2006.
2. Дари, К. AJAX и PHP. Разработка динамических web-приложений / К. Дари, Б. Бринзаре, Ф. Черчез-Тоза, М. Бусика // Символ-Плюс, 2006.
3. Тарасов В. А. Учебное пособие по спецкурсу Java / В. А. Тарасов, В. В. Тарасов, А. С. Кюршунов.
4. Руководство по Web-сервисам. [Электронный ресурс]/ Электрон. ст. — Режим доступа к ст.: http://ru.sun.com/pdf/j2ee/WST.pdf
5. IBM developerWorks: SOA и web-сервисы: техническая библиотека [Электронный ресурс] / Электрон. ст. — Режим доступа к ст.: http://www.ibm.com/developerworks/ru/views/webservices/libraryview.jsp
6. Флэнаган, Д. JavaScript. Подробное руководство (4-е издание) / Д. Флэнаган // Символ-Плюс, 2004.
7. Гудман Д. JavaScript и DHTML. Сборник рецептов. Для профессионалов / Д. Гудман // Питер, 2004.
8. jQuery API Browser [Электронный ресурс] / Электрон. ст. — Режим доступа к ст.: http://jquery.bassistance.de/api-browser/