Министерство образования Российской Федерации
Карельский государственный педагогический университет

Сервлеты

Дипломная работа

Работу выполнил:
студент 551 гр. физико-
математического факультета
Капустин А.Н.

Научный руководитель:
ассистент кафедры информатики
Кюршунов А.С.

Петрозаводск
2003г.

Содержание:

Введение

В настоящее время все большую популярность приобретают системы дистанционного обучения (ДО). Современная система ДО основана на использовании сетевых технологий и включает в себя обязательное тестирование слушателей в качестве контроля за их учебной деятельностью. В этом случае необходимо, чтобы все вычисления и обработка результатов проводилась на серверной стороне, недоступной пользователю. Это можно осуществить используя Java технологию, а именно сервлеты.

Сервлеты (servlets) - это маленькие программы, которые выполняются на серверной стороне Web-соединения. Точно так же, как апплеты динамически расширяют функциональные возможности Web-браузера, сервлеты динамически расширяют функциональные возможности Web-сервера.

Сейчас, сервлеты, возникшие сравнительно недавно, находят все большее применение в Интернете, так как являются платформенно-независимыми. Сервлеты исполняются на серверах приложений в многозвенных прикладных системах. Тенденция написания сетевых программ состоит в том, чтобы побольше функций возложить на сервер и поменьше оставить клиентской части. Это позволяет использовать клиентскую часть программы на самых старых и маломощных компьютерах.

Рассмотрим применение сервлетов на примере создания системы тестирования, использование сервлетов в данном случае актуально так, как они обладают всеми преимуществами Java программ (безопасность, переносимость, устойчивость и т.п.). При создании систем тестирования важно выделить два режима доступа:

a) Режим пользователя. В данном случае предусмотрено прохождение теста и получение за это определенного результата (оценки).

b) Режим администратора. В этом случае предоставляется возможность редактировать тесты и результаты.

Использование сервлетов позволяет достигнуть такого разграничения прав пользователей. Также немаловажной особенностью использования сервлетов состоит в том, что они могут вызываться и работать вместе с апплетами, а апплеты в свою очередь имеют

мощные средства для работы с графикой, что позволяет создавать тестируемые программы с более дружественным интерфейсом для пользователей.

Цель данной дипломной работы показать принципы и подходы использования сервлетов на серверной стороне соединения, что достигнуто на примере создания системы тестирования, в которой есть разделение прав пользователей.

Содержание дипломной работы следующее.

В первой главе рассмотрена технология создания сервлетов, их преимущества по сравнению с CGI-интерфейсом, описан жизненный цикл сервлета, а также дана характеристика инструментального набора Java Servlet Development Kit.

Во втором пункте подробно рассмотрена система тестирования, приведен листинг сервлетов и пояснения к ним, описаны ключевые принципы работы с программой.

1. Обзор сетевых Java технологий

В данной главе рассматриваются язык программирования Java, его основные особенности и характеристики; сервлеты, их назначение и преимущества по сравнению с CGI-интерфейсом.

1.1 Язык программирования Java

Java – это простой, объектно-ориентированный, распределенный, интерпретирующий, безопасный, архитектурно-нейтральный, переносимый, высокопроизводительный, многопоточный и производимый язык.

Компилятор Java читает исходные файлы и превращает их в байт-код (byte-code). Байт-код представляет собой промежуточную стадию между исходным кодом и машинным кодом, как можно более близкую к машинному коду. Но близкую не настолько, чтобы стать платформо-зависимой. Если точнее, то байт-код является машинным кодом, но не для какой-нибудь физически существующей машины, а для Java Virtual Machine – виртуальной машины, чье поведение в точности определено Sun Microsystems. Спецификации Java Virtual Machine (JVM) описывают поведение, ожидаемое от любой физической машины, которая выполняет любой заданный байт-код. Подчинение спецификациям JVM – вот что обеспечивает переносимость программ Java.

Развитие Internet и World Wide Web заставляет совершенно по-новому рассматривать процессы разработки и распределения программного обеспечения. Для того, чтобы выжить в мире электронного бизнеса и распространения данных, язык Java должен быть

- безопасным,
- высокопроизводительным,
- надежным.

Работа на различных платформах гетерогенных сетей отбрасывает традиционную схему распределения ПО, версий ПО, модификации ПО, объединения ПО и т.д. Для решения проблем гетерогенных сред язык должен быть

1.нейтральным к архитектуре,

2.переносимым,

3.динамически подстраиваемым.

Разработчики Java с самого начала хорошо понимали, что язык, предназначенный для решения проблем гетерогенных сред, также должен быть

- простым - его должны с легкостью использовать все разработчики
- ясным - разработчики должны без больших усилий выучить Java
- объектно-ориентированным - он использует все преимущества современных методологий разработки ПО и подходит для написания распределенных клиент-серверных приложений
- многопоточным - для обеспечения высокой производительности приложений, выполняющих одновременно много действий (например, в мультимедийных системах)
- интерпретируемым - для переносимости и большей динамичности

Необходимо более подробно рассмотреть перечисленные характеристики Java.

Простота

Простота языка входит в ключевые характеристики Java: разработчик не должен длительное время изучать язык, прежде чем он сможет на нем программировать. Фундаментальные концепции языка Java быстро схватываются и программисты с самого начала могут вести продуктивную работу. Добавилось автоматическое освобождение памяти за счет неиспользуемых объектов, упрощающая процесс программирования, но несколько усложняющая систему в целом.

Объектно-ориентированность

Язык Java с самого начала проектировался как объектно-ориентированный. Задачам распределенных систем клиент-сервер отвечает объектно-ориентированная парадигма: использование концепций инкапсуляции, наследования и полиморфизма. Java предоставляет ясную и действенную объектно-ориентированную платформу разработки.

Программисты на Java могут использовать стандартные библиотеки объектов, обеспечивающие работу с устройствами ввода/вывода, сетевые функции, методы создания графических пользовательских интерфейсов. Функциональность объектов этих библиотек может быть расширена.

Надежность

Платформа Java разработана для создания высоконадежного прикладного программного обеспечения. Большое внимание уделено проверке программ на этапе компиляции, за которой следует второй уровень - динамическая проверка (на этапе выполнения). Виртуальная машина Java использует модель управления динамической памятью исключающую возникновение ошибок при работе с динамически выделенной памятью.

Безопасность

Java разработана для оперирования в распределенных средах, это означает, что на первом плане должны стоять вопросы безопасности. Средства безопасности, встроенные в язык, и система исполнения Java позволяют создавать приложения, на которые невозможно "напасть" извне. В сетевых средах приложения, написанные на Java, защищены от вторжения неавторизованного кода, пытающегося внедрить вирус или разрушить файловую систему.

Независимость от архитектуры

Java разработан для поддержки приложений, внедряемых в гетерогенные сетевые среды. В подобных средах приложения должны исполняться на различных аппаратных архитектурах, под управлением различных операционных систем и во взаимодействии с интерфейсами различных языков программирования. Для обеспечения платформо-независимости программ компилятор Java генерирует байт-код - архитектурно-нейтральный промежуточный формат программы, создаваемый для эффективной передачи кода на различные аппаратные и программные платформы. При выполнении программы байт-код интерпретируется исполняющей машиной Java. Один и тот же Java-байткод будет исполняться на любой платформе.

Переносимость

Архитектурная независимость - лишь составная часть переносимости. Форматы типов данных и операции над ними четко определены. Тем самым, программы остаются неизменными на любой платформе - не существует несовместимости типов данных на аппаратных и программных архитектурах.

Архитектурная независимость и переносимость программного обеспечения Java обеспечивается виртуальной машиной Java (Java Virtual Mashine - JVM) - абстрактной машиной, для которой компилятор Java генерирует код. Специальные реализации JVM для конкретных аппаратных и программных платформ предоставляют уже конкретную виртуальную машину. JVM базируется на стандарте интерфейса переносимых операционных систем (POSIX).

Многопоточность

Большинству современных сетевых приложений обычно необходимо осуществлять несколько действий одновременно. В Java реализован механизм поддержки легковесных процессов-потоков (нитей). Многопоточность Java предоставляет средства создания приложений с множеством одновременно активных потоков.

Динамичность

Язык Java был разработан специально для подстройки под изменяющееся окружение. В то время как компилятор Java на этапе компиляции и статических проверок не допускает никаких отклонений, процесс сборки и выполнения сугубо динамический. Классы связываются только тогда, когда в этом есть необходимость. Новые программные модули могут подключаться из любых источников, в том числе, поставляться по сети. В результате возможно создание интерактивных служб, безболезненно модифицируемых, обслуживающих большое количество клиентов и обеспечивающих развитие электронного бизнеса через Internet.

Заключение

Если описанные выше характеристики рассматривать по отдельности, то их можно найти во многих программных платформах. Радикальное новшество заключается в способе,

предлагаемом Java и системой выполнения, который сочетает в себе все характеристики для предоставления гибкой и мощной системы программирования.

Разработка приложений на Java приводит к получению программного обеспечения, которое:

- переносимо на разные архитектуры, операционные системы и графические пользовательские интерфейсы
- безопасно
- высокопроизводительно

Благодаря Java работа по разработке программного обеспечения значительно упрощается, все старания направлены на достижение конечной цели: вовремя получить передовой продукт, опирающийся на солидную основу Java [6] .

1.2 Java сервлеты

Чтобы понимать преимущества сервлетов, нужно иметь основные представления о том, как кооперируются Web-браузеры и серверы, чтобы предоставить их содержимое пользователю. Рассмотрим запрос статической Web-страницы. Пользователь вводит унифицированный локатор ресурса (URL, uniform resource locator) в браузер. Браузер генерирует HTTP-запрос к соответствующему Web-серверу. Web-сервер устанавливает соответствие этого запроса с определенным файлом. Данный файл возвращается браузеру в HTTP-ответе. HTTP-заголовок в этом ответе указывает тип своего содержимого в формате MIME (Multipurpose Internet Mail Extension, многоцелевое расширение почты Internet). Например, обычный ASCII-текст имеет MIME-тип text/plain. Исходный HTML-текст Web-страницы имеет MIME-тип text/html.

Теперь рассмотрим динамическое содержание. Предположим, что сетевая тестирующая программа использует базу данных для хранения информации относительно ее тестов, результатов тестирования и т.д. Требуется сделать эту информацию доступной для пользователей через Web-страницы. Чтобы отражать самую последнюю информацию в базе данных, содержание этих Web-страниц должно быть сгенерировано динамически.

В ранних версиях Web-сервер мог динамически конструировать страницу, создавая отдельный процесс для обработки каждого запроса клиента. Чтобы получать необходимую информацию, процесс мог открывать соединения к одной или нескольким базам данных. Он общался с Web-сервером через интерфейс, известный как Common Gateway Interface (CGI). CGI позволял отдельному процессу читать данные из HTTP-запроса и записывать данные в HTTP-ответ. Для построения CGI-программ использовался ряд различных языков, включая C, C++ и Perl.

Однако у CGI-программ отмечены серьезные проблемы с эффективностью. Создание отдельного процесса для каждого запроса клиента требует больших затрат ресурсов памяти и процессора. Больших затрат требует также необходимость открывать и закрывать соединения базы данных для каждого запроса клиента. Кроме того, программы CGI не являлись платформно - независимыми. Поэтому были разработаны другие методы общения, включая сервлеты.

Сервлеты обеспечивает несколько преимуществ по сравнению с CGI-интерфейсом:

1. Повышена эффективность. При использовании традиционного CGI для каждого HTTP-запроса запускается свой процесс. Если CGI-программа сама по себе является короткой, то накладные расходы по запуску процесса несопоставимо велики по сравнению со временем ее выполнения. При использовании сервлетов Java Virtual Machine (виртуальная Java-машина) остается в режиме выполнения и обрабатывает каждый запрос, используя упрощенный Java-поток, а не сложный процесс операционной системы. К тому же если в традиционном CGI к одной и той же программе одновременно поступает N запросов, код CGI-программы загружается в память N раз. При использовании сервлетов одновременно может существовать N потоков, но в памяти будет находится единственная копия класса сервлета. И, наконец, когда CGI-программа заканчивает обработку запроса, программа завершает выполнение. Это усложняет кэширование вычислений, поддержание открытыми соединений с базой данных и выполнение других оптимизированных действий, которые основаны на сохраняемости (постоянстве существования) данных. Сервлеты же остаются в памяти даже после выполнения запроса, что дает возможность произвольным образом сохранять сложные данные между запросами.

2. Удобство. Сервлеты обладают расширенной инфраструктурой для осуществления автоматического синтаксического анализа и декодирования данных HTML-форм, чтения и задания HTTP-заголовков, обработки cookies, отслеживания сеансов и многих других подобных высокоуровневых утилит.

3. Мощность. Сервлеты обладают некоторыми возможностями, которые сложно или невозможно представить в обычном CGI-программировании. Они могут общаться непосредственно с Web-сервером, в то время как обычные CGI-программы не могут этого делать, по крайне мере без использования особого, зависящего от сервера, API. А возможность общения с сервером упрощает, например, преобразование относительных адресов URL в правильные, полные имена путей. Кроме того, несколько сервлетов могут совместно использовать данные, упрощая реализацию организации пула соединений с базой данных и другие методы оптимизации совместного использования ресурсов. Также сервлеты могут сохранять информацию между запросами, упрощая реализацию таких технологий, как отслеживание сеансов и кэширование предыдущих вычислений.

4. Переносимость. Сервлеты не зависят от платформы так как они написаны на Java. Несколько Web-серверов от таких поставщиков, как Sun, Netscape и Microsoft, предлагают Servlet API. Программы, разработанные для этого API, могут быть перемещены в любую из указанных сред без перетрансляции.

5. Безопасность. Менеджер безопасности Java (Java Security Manager) на сервере поддерживает набор ограничений для защиты ресурсов на машине сервера. Одной из причин уязвимости традиционных CGI-программ является то, что они часто выполняются в оболочках операционных систем общего назначения. Поэтому программист должен быть очень внимателен и отфильтровывать символы, которые особым образом трактуются оболочкой, например, обратные кавычки и точки с запятой. Это гораздо сложнее, чем кажется с первого взгляда, и слабые стороны программ, возникающие по этой причине, постоянно обнаруживаются в широко используемых библиотеках CGI. Вторым источником проблем является то, что некоторые CGI-программы обрабатываются языками, которые не проверяют автоматически границы массивов или строк. Поэтому, если программисты забывают проверять выход за границы массивов, они открывают системы для преднамеренных или случайных атак переполнения буфера. У сервлетов не существует подобных проблем. Даже если для активизации программы, находящийся в локальной

операционной системе, сервлет выполняет удаленный системный вызов, он не использует для этого оболочку. И конечно, проверка границ массива, как и другие функции защиты памяти, является важнейшей частью языка программирования Java.

6. Сервлетам доступны полные функциональные возможности библиотек классов Java. Они могут связываться с апплетами, базами данных, или другими программным обеспечением через сокеты и RMI-механизмы.

7. Стоимость. В настоящее время есть бесплатные или очень недорогие Web-серверы, удобные для “персонального” использования или создания Web-сайтов небольшого объема. Однако за исключением сервера Apache, который является бесплатным, большинство Web-серверов коммерческого уровня являются относительно дорогими. Тем не менее, если у вас имеется Web-сервер, то независимо от его цены добавление поддержки сервлетов не обойдется вам дорого. Это отличает сервлеты от многих альтернатив CGI, которые требуют значительных начальных инвестиций на приобретение пакетов.

1.3 Программный интерфейс сервлета

Для того чтобы Web-сервер мог выполнять сервлеты, в его состав должны входить JVM и средства связи с сервлетами. Набор инструментов разработки Java-сервлетов (JSDK, Java Servlet Development Kit) содержит основные интерфейсы и классы, описывающие сервлеты, собраны в пакет javax.servlet. Расширения этих интерфейсов и классов, использующие конкретные особенности протокола HTTP, собраны в пакет javax.servlet.http. Все эти пакеты входят в состав J2SDK Enterprise Edition. Вместе они представляют Servlet API.

Пакет **javax.servlet** содержит ряд интерфейсов и классов, устанавливающих обрамление, в котором работают сервлеты. Наиболее значительный из них - это Servlet. Все сервлеты должны реализовывать указанный интерфейс или расширять класс, который его реализует.

Интерфейс	**Описание**
Servlet	Объявляет методы цикла жизни сервлета
ServletRequest	Используется для чтения данных из запроса клиента
ServletResponse	Используется для записи данных в ответ клиента

Класс	Описание
GenericServlet	Реализует интерфейсы Servlet и ServletConfig
ServletException	Указывает, что произошла ошибка сервлета

Интерфейс Servlet

Все сервлеты должны реализовывать интерфейс Servlet. Он объявляет методы *init(), service()* и *destroy(),* которые вызываются сервером в течении цикла жизни сервлета. Существует также метод, который позволяет сервлету получать любые параметры инициализации. Методы, определенные в Servlet, показаны ниже.

Метод	Описание
void destroy()	Вызывается, когда сервлет выгружается
void init (ServletConfig sc) throws ServletException	Вызывается, когда сервлет инициализируется. sc – определяет параметры его инициализации. Если сервлет не может быть инициализирован, выбрасывается исключение UnavailableException
void service (ServletRequest req, ServletResponse res) throws ServletException, IOException	Вызывается, чтобы обработать запрос клиента. Запрос от клиента может читаться из req. Ответ клиенты может быть записан в res. Если случаются проблемы сервлета или ввода\вывода, генерируется исключение ServletException или IOException

Центральное место в жизненном цикле сервлета занимает три метода: init(), service() и destoy(). Они реализуются каждым сервлетом и вызываются в определенные моменты сервером. Чтобы понять, когда эти методы вызываются, рассмотрим следующий типичный сценарий пользователя.

1. Предположим, что пользователь вводит в Web-браузер унифицированный адрес ресурса URL. Затем браузер генерирует HTTP-запрос для этого URL-адреса и посылает его соответствующему серверу.

2. Web-сервер получает этот HTTP-запрос и направляет его к специальному сервлету, который динамически извлекается и загружается в адресное пространство сервера.

3. Сервер вызывает метод init() сервлета (метод вызывается, только тогда когда сервлет предварительно загружен в память). Параметры инициализации можно переслать сервлету так, чтобы он мог сам себя конфигурировать.

4. Сервер обращается к методу service() сервлета, который вызывается для обработки HTTP-запроса. Сервлет может читать данные, которые были переданы в HTTP-запросе, а также формировать HTTP-ответ для клиента.

5. Сервлет остается в адресном пространстве сервера и может обрабатывать любые другие HTTP-запросы, полученные от клиентов. Метод service() вызывается для каждого HTTP-запроса.

6. Наконец сервер может решить выгрузить сервлет из своей памяти. Алгоритмы, с помощью которых это намерение выполняется, специфичны для каждого сервера. Сервер вызывает метод destroy(), чтобы освободить любые ресурсы, такие как дескрипторы файлов, которые распределены для сервлета. Важные данные могут быть сохранены в постоянном хранилище. Затем может быть выполнена сборка мусора в памяти, распределенной для сервлета и его объектов.

Класс GenericServlet

Класс GenericServlet обеспечивает реализацию основных методов жизненного цикла сервлета. Разработчики сервлетов обычно работают с его подклассами. GenericServlet реализует интерфейсы ServletConfig и Servlet. Кроме того, имеется метод для добавления строки в файл регистрации сервера. Сигнатура этого метода:

void (String s)

Здесь s – строка, которая будет добавлена к файлу регистрации.

Пакет **javax.servlet.http** содержит несколько интерфейсов и классов, которые обычно используются разработчиками сервлетов. Его функциональные возможности облегчают построение сервлетов, которые работают с HTTP-запросами и ответами.

Интерфейс	Описание
HttpServletRequest	Разрешает сервлетам читать данные из HTTP-запроса
HttpServletResponse	Разрешает сервлетам записывать данные в HTTP-ответ

Класс	Описание
HttpServlet	Обеспечивает методы для обработки запросов и ответов HTTP

Класс HttpServlet

Класс HttpServlet расширяет GenericServlet. Он обычно используется при разработке сервлетов, которые принимают и обрабатывают HTTP-запросы. Методы класса описаны ниже.

Метод	Описание
void doGet (HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Выполняет GET-запрос HTTP
void doPost (HttpServletRequest req, HttpServletResponse res) throws IOException, ServletException	Выполняет POST-запрос HTTP

Создание и компиляция исходного кода сервлета.

Простой пример сервлета (файл с именем Helloservlet.java):

```
import java.io.*;

import javax.servlet.*;

public class Helloservlet extends GenericServlet {

public void service (ServletRequest req, ServletResponse res) throws ServletException, IOException {
```

```
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        pw.println("Hello!");
        pw.close();
    }
}
```

Эта программа импортирует пакет javax.servlet, который содержит классы и интерфейсы, требуемые для построения сервлетов. Затем, программа определяет Helloservlet как подкласс GenericServlet. Класс GenericServlet обеспечивает функциональные возможности, которые облегчают обработку запросов и ответов.

Внутри Helloservlet переопределяется метод service() (который унаследован из GenericServlet). Этот метод обрабатывает запросы клиента. Первый аргумент этого метода – это объект класса ServletRequest. Он дает возможность сервлету читать данные, которые получаются через запрос клиента. Второй аргумент – объект класса ServletResponse. Он позволяет сформировать ответ для клиента.

Вызов setContentType() устанавливает MIME-тип HTTP-ответа. В этом вызове указан MIME-тип text/html, который заставляет браузер интерпретировать содержимое как исходный HTML-код.

Далее, метод getWriter() заполняет объект PrintWriter. Все записанное в этот поток посылается клиенту как часть HTTP-ответа. Затем используется println(), чтобы записать некоторый простой исходный код HTML как HTTP-ответ.

2. Разработка системы тестирования с помощью сервлетов

В данной главе рассматривается разработка системы тестирования, написанная на языке Java с использованием сервлетов и апплетов.

Для работы системы тестирования, необходима локальная сеть. Среди компьютеров необходимо выделить сервер, на этом компьютере будет располагаться система тестирования, а также файлы тестов и оценок. Остальные компьютеры будут выступать в роли рабочих станций, на них можно подключаться к серверу и работать с тестирующей программой.

При создании этой программы предусмотрено два возможных взаимодействия. Можно зайти в программу как ученик и выполнить один из предлагаемых тестов, причем фамилия ученика и его оценка по выполнению задания будут сохранены на сервере в файле оценок, который будет недоступен для данного пользователя. И можно зайти в программу как учитель, у этого пользователя есть права для редактирования, просмотра и изменения файлов тестов и оценок. Для того, чтобы зайти в программу как учитель, необходимо знать пароль доступа. Пароль может быть изменен, но это можно сделать только на сервере.

Приведен листинг основных сервлетов и их методов, структура взаимодействия клиентской и серверной части. Также описаны ключевые принципы работы с программой, ее использования, изменения тестирующих заданий и файла отчета.

2.1 Реализация приложения, его структура

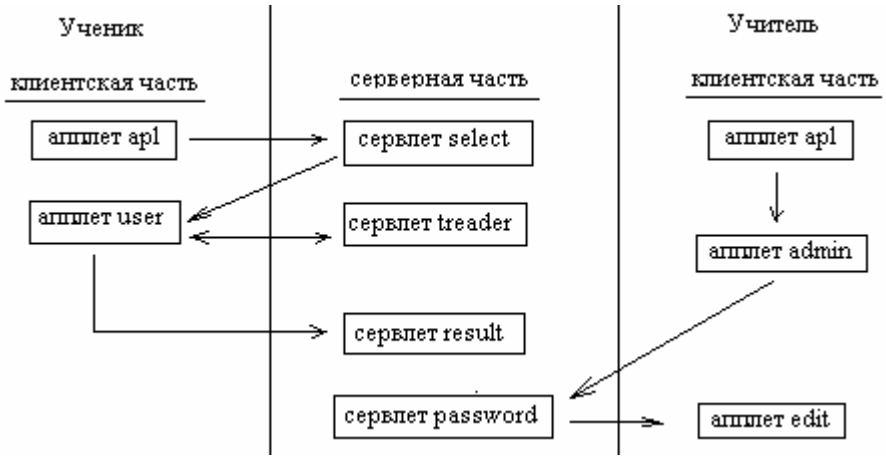

Рис. 1

Принцип работы системы тестирования следующий . В начале в окне браузера загружается апплет apl.class, где можно выбрать (нажав на соответствующую кнопку) либо начало тестирования, не забыв в этом случае ввести свою фамилию и выбрать интересующий тест, либо можно нажать на кнопку ADMIN, тем самым заходя в программу как учитель.

Листинг апплета apl.class

```
public class apl extends Applet implements ActionListener{
…
public void init() { … //инициализация апплета }
private void sendData() { //метод отправка информации сервлету select
try {
     String someData = "familia=" + URLEncoder.encode(familia) +
"&" + "file="+ URLEncoder.encode(namefiles[i]);
     URL page = getCodeBase();
     URL servUrl = new
URL(page.getProtocol(),page.getHost(),page.getPort(),"/servlet/sel
ect" + "?" + someData); // отправка фамилии ученика и названия теста
сервлету select
     getAppletContext().showDocument(servUrl, "_self"); // передача
управления сервлету
        } catch(MalformedURLException e)
{System.err.println(e.getMessage());}
}

public void actionPerformed(ActionEvent ae) { //метод обработки
событий на нажатие кнопок
     String str = ae.getActionCommand();
     if (str.equals("ADMIN")) goAdmin(); //апплет реагирует на нажатие
кнопки ADMIN
          familia=fam.getText();
          sendData();
   }

private void goAdmin() { // загрузка html-страницы admin
try {
        URL page = getCodeBase();
        URL servUrl = new
URL(page.getProtocol(),page.getHost(),page.getPort(),"/TEST/admin.
htm");
        getAppletContext().showDocument(servUrl, "_self");
        } catch(MalformedURLException e) {}
   }}
```

Метод **sendData**() отправляет сервлету select данные такие как фамилия учащегося и название теста и полностью передает ему управление. Данный сервлет генерирует HTML-страницу со встроенным в него апплетом user. Апплет user связывается с сервлетом treader

для получения текста выбранного теста, проводит тестирование и затем вызывает сервлет result для записи результата в файл res.txt.

Метод **actionPerformed**() реагирует на события, которые произвел пользователь, а именно при нажатии кнопки ADMIN, запускается метод goAdmin(), если же нажата была кнопка TEST, запускается метод sendData().

Метод **goAdmin**() открывает HTML-страницу admin.htm, куда встроен апплет admin.class. Данный апплет (предназначен для …) предложит ввести пароль доступа, после чего отправит его обрабатывать сервлету pasw.class и если пароль верен будет запущен апплет edit.class. Данный апплет предоставляет пользователю (учителю) возможность редактирования и просмотра тестовых заданий и файла отчета.

Листинг сервлета select

```
public class select extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response)
                    throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String familia = request.getParameter("familia");
String fl = request.getParameter("file");
// сервлет получает параметры от апплета apl(фамилия ученика и название файла)
out.println("<HTML>"); // формирование http-ответа
out.println("<BODY>");
out.println("<applet codebase='../TEST/' code='User.class' width =
400 height = 200 >");
// загрузка апплета user в html странице ответа
out.println("<param name=familia value="+familia+">");
out.println("<param name=file value="+fl+">");
// передача параметров апплету user
out.println("</applet>");
out.println("</BODY>");
out.println("</HTML>");}
```

Листинг апплета admin

```
public class admin extends Applet implements ActionListener{
…
public void init() { …// метод инициализации апплета }
public void actionPerformed(ActionEvent ae) { // метод обработки
событий на нажатие кнопки
     password(); }
```

```
private void password(){ // метод отправки информации сервлету password
  try {
  String someData = "pasw=" + URLEncoder.encode(pasw.getText());
  URL page = getCodeBase();
  URL dataURL = new URL (page.getProtocol(), page.getHost(),
page.getPort(), "/servlet/password" + "?" + someData);
 getAppletContext().showDocument(dataURL, "_self");
 } catch(MalformedURLException e) {}
 }}
```

Метод **password**() предназначен для передачи дальнейшего управления работой сервлету password и предоставить ему пароль для проверки.

Листинг апплета user

```
public class User extends Applet implements ActionListener{
public void init() { …//метод инициализация апплета };
private void getTest(){ // метод вызова сервлета treader для чтения теста
try {
URL page = getCodeBase();
URL dataURL = new URL(page.getProtocol(), page.getHost(),
page.getPort(), "/servlet/treader ? testFile = "+namefile); //
передача сервлету имени файла теста
URLConnection con = dataURL.openConnection();
con.setUseCaches(false);
ObjectInputStream in = new ObjectInputStream(con.getInputStream()
) ;
// получение от сервлета treader данных по тесту (вопросы, варианты ответов и
т.п.)
tema = (String) in.readObject();
Questions = (String[]) in.readObject();
Answers = (String[]) in.readObject();
Keys = (String[]) in.readObject();
in.close();}
      catch (Exception e) {}
   }
private void test(){ … // метод вывода новых вопросов и вариантов ответов
на экран
}
public void actionPerformed (ActionEvent ae){ … //метод обработки
событий   }
public void result(){// метод подведения результатов
… ;
sendData();}
private void sendData() { // метод отправки данных (оценка за тест,
фамилия ученика и т.д.) сервлету result
try {
   String someData =
   "familia=" + URLEncoder.encode(familia) + "&" +
   "num=" + URLEncoder.encode(new Integer(num).toString()) + "&" +
```

```
     "mark=" + URLEncoder.encode(mark) + "&" +
     "right=" + URLEncoder.encode(new Integer(right).toString());
     URL page = getCodeBase();
     URL servUrl = new URL (page. getProtocol() ,page.getHost()
     ,page.getPort() ,"/servlet/results" + "?" + someData);
     getAppletContext().showDocument(servUrl, "_self");}
catch(MalformedURLException e) {};
}}
```

Листинг сервлета treader

```
public class treader extends HttpServlet {
…
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
response.setContentType("application/x-java-serialized-object");
testFileName = request.getParameter("testFile");//получение имени
тестового файла
OpenTestFile("../tests/"+testFileName);
//передача тестовых данных апплету user, на стороне клиента
ObjectOutputStream out = new ObjectOutputStream (response.
getOutputStream() );
out.writeObject(tema);
out.writeObject(Questions);
out.writeObject(Answers);
out.writeObject(Keys);
out.flush();
}
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws IOException, ServletException{ doGet(request,
response);}
private void OpenTestFile(String Name) {//метод открытия тестового
файла
 …
 BufferedReader in = new BufferedReader(new FileReader(Name));
 }}
```

Листинг сервлета result

```
public class results extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String familia = request.getParameter("familia"); // получение
параметров от апплета user
String num = request.getParameter("num");
String mark = request.getParameter("mark");
String right = request.getParameter("right");
```

```
out.println("<HTML>" + "<HEAD><TITLE>" + Rus.dec("Отчет")
+"</TITLE></HEAD>\n" + "<meta http-equiv=\"Content-Type\" content
=\ "text/html; charset=windows-1251\">" + "<BODY BGCOLOR=\
"#FDF5E6\ " >\n " + "<H1 ALIGN=CENTER>" + Rus.dec("Отчет") +
"</H1>\n" + Rus.dec("Фамилия: ") + familia + "\n" +
"<H2>"+Rus.dec("Результаты выполнения теста:")+":</H2>\n" +
"<P>"+Rus.dec("Всего заданно")+" " + num +Rus.dec(" вопросов ") +
".\n" + "<BR>"+Rus.dec("Ваша оценка") + " - <I>" + mark +
"</I>.\n" + "<BR>"+Rus.dec("Правильных ответов ")+" - " + right +
".\n" + "</BODY></HTML>"); // формирование http-ответа
try {
File file = new File("../tests/result/res.txt"); // сохранение
результатов в файл res.txt
PrintWriter out_f = new PrintWriter(new BufferedWriter(new
FileWriter(file.toString(),true)));
out_f.println("Дата: " + (new Date().toGMTString()));
out_f.println(" Фамилия: " + Rus.ASCIItoUTF(familia));
out_f.println(" Результат (" + num +" вопросов" + "):");
out_f.println(" Оценка - " + Rus.ASCIItoUTF(mark));
out_f.println(" Правильных ответов - " + right);
out_f.println("-----------------------------------------------
--");
out_f.close();}
catch(IOException e) {};}
public void doPost(HttpServletRequest request, HttpServletResponse
response)throws IOException, ServletException { doGet(request,
response); }
}
```

2.2 Описание работы системы тестирования

Для работы системы тестирования понадобиться установить Web-сервер, например можно использовать Jakarta Tomcat. Далее необходимо поместить папку Test с апплетами apl, admin, user и edit в каталог <instaldir>\webapps\ROOT. Сервлеты нужно поместить в каталог <instaldir>\webapps\ROOT\WEB-INF\classes. А папку tests в которой находятся тесты, файл отчета и пароля нужно разместить в корневом каталоге Web-сервера Tomcat.

В окне браузера необходимо ввести IP адрес машины на которой установлен Web-сервер Jakarta Tomcat и путь до стартовой страницы. На рисунке 1, как вы видите, в окне браузера введен следующий адрес: http://localhost:8080/TEST/index.htm, вместо IP адреса машины, написано *localhost* так, как в данном случае Web-сервер Jakarta Tomcat установлен на том же компьютере, на котором и запускается тестирующая программа.

Пусть мы заходим как ученик. Для этого необходимо ввести свою фамилию и выбрать интересующий нас тест (например по геометрии, как показано на рисунке 2). Далее необходимо нажать кнопку TEST.

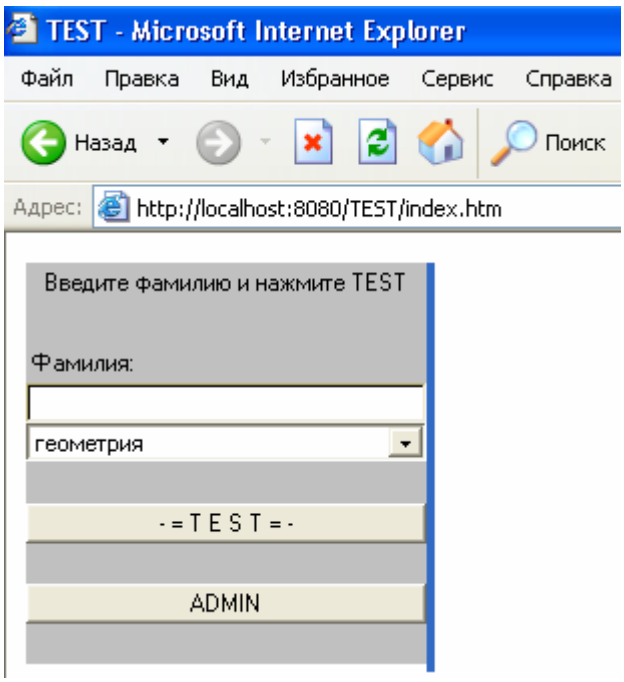

Рис. 2

После выполнения выше перечисленных действий мы увидим в окне браузера тест с выборочными вариантами ответов.

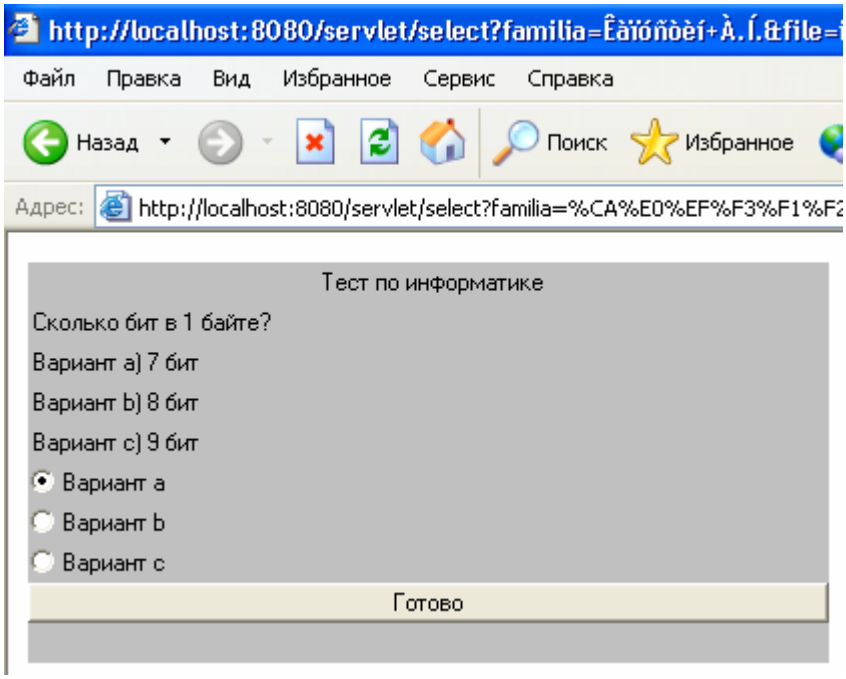

Рис. 3

Нам будет предложен вопрос и три варианта ответа на него, выбрав вариант ответа, необходимо нажать на кнопку Готово, чтобы перейти к следующему вопросу. По окончанию прохождения теста будет выведен отчет, включающий в себя фамилию ученика и его оценку, одновременно с этим результат будет сохранен на сервере.

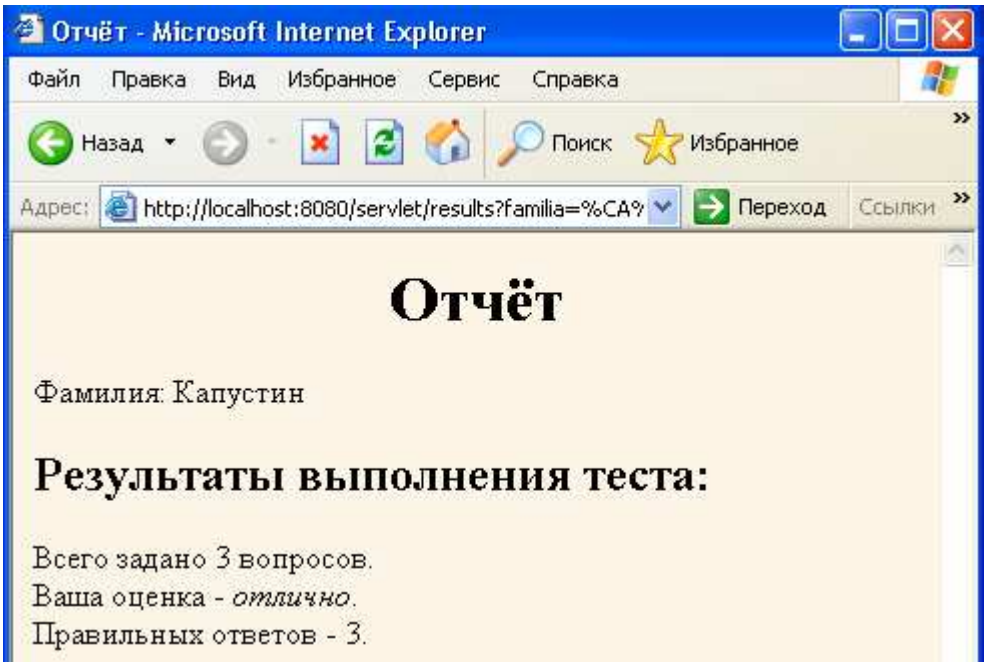

Рис. 4

В случае когда нам нужно зайти в программу как учитель, то необходимо нажать на кнопку ADMIN. После этого нам необходимо будет ввести пароль доступа и нажать на кнопку ACCEPT.

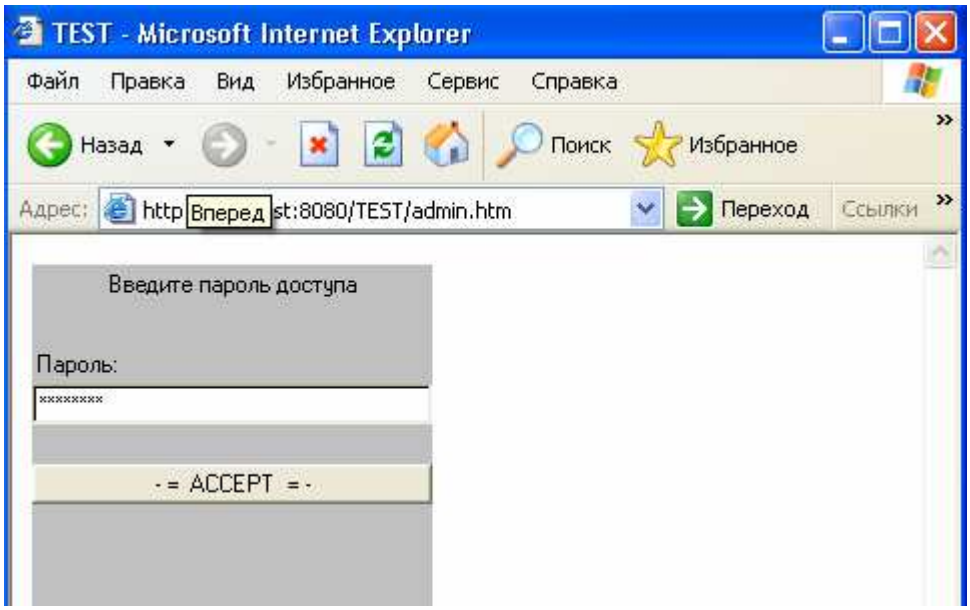

Рис. 5

После чего, если пароль набран верно, пользователь сможет редактировать и просматривать тесты и файл отчёта тестирования.

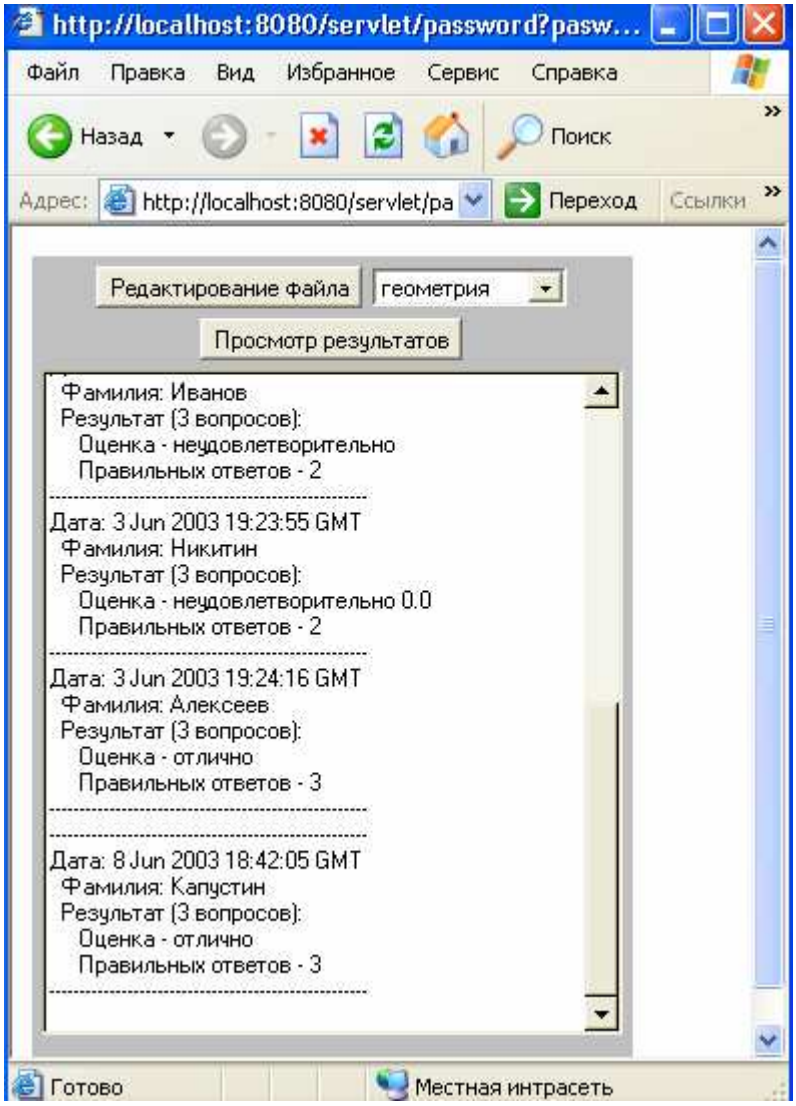

Рис. 6

3. Выводы

Глобальная информатизация общества приводит к тому, что потребность в информации, растет с каждым новым пользователем сети. При этом задачей специалистов в области информационных технологий обеспечить пользователей полной и достоверной информацией путем простого и удобного для пользователей доступа к накопленным массивам данных.

Одним из наиболее общих приложений сервлетов является обращение к базе данных и динамическое создание ответов HTTP из найденной информации.

Цель дипломной работы изучить работу сервлетов на серверной стороне соединения, на примере создания системы тестирования. При разработке системы тестирования, использование сервлетов позволило создать два режима работы (ученик - учитель), тем самым разделив права пользователей, подведение результатов производится на серверной стороне, с последующей записью результатов в файл отчета на сервере.

Данная система тестирования может использоваться в различных учебных заведениях, при подведении итогов по какому-либо предмету, где есть локальная сеть, возможно также установить ее в Интернете.

Литература

1. Ноутон П., Шилдт Г. Java™ 2: Пер. с англ. – СПб.: БХВ-Петербург, 2001.
2. Холл М. Сервлеты и JavaServer Pages: Библиотека программиста – СПб.: Питер, 2001.
3. http://ru.sun.com/win/java/start/intro/properties.html
4. http://ois.mesi.ru/html-docs/tutorial/servlets/index.html
5. http://www.sun.ru/java/start/intro/history.html
6. http://www.java.sun.com