ТЕМА 7
Создание веб-приложений на основе сервлетов

Цель лабораторной работы

Научиться создавать веб-приложения на основе сервлетов для платформы Java Enterprise Edition (EE).

1 Разработка веб-приложений на основе Servlet API

Спецификация Java Servlet API предоставляет стандартный и платформенно-независимый каркас для взаимодействия между web-контейнером и сервлетами, работающими внутри него. Этот каркас состоит из набора классов и интерфейсов Java, принадлежащих пакету javax.servlet, которые в сумме и представляют Servlet API.

Web-контейнер отвечает за управление жизненным циклом сервлетов, отображение обращений к URL в вызовы определённых сервлетов, проверку прав доступа при обращении к сервлетам.

Жизненный цикл сервлета состоит из следующих этапов (рисунок 1.1):

- не загружен (Unloaded),
- загружен (Loaded),
- проинициализирован (Initialized),
- в обработке запросов (servicing),
- разрушен (Destroyed).

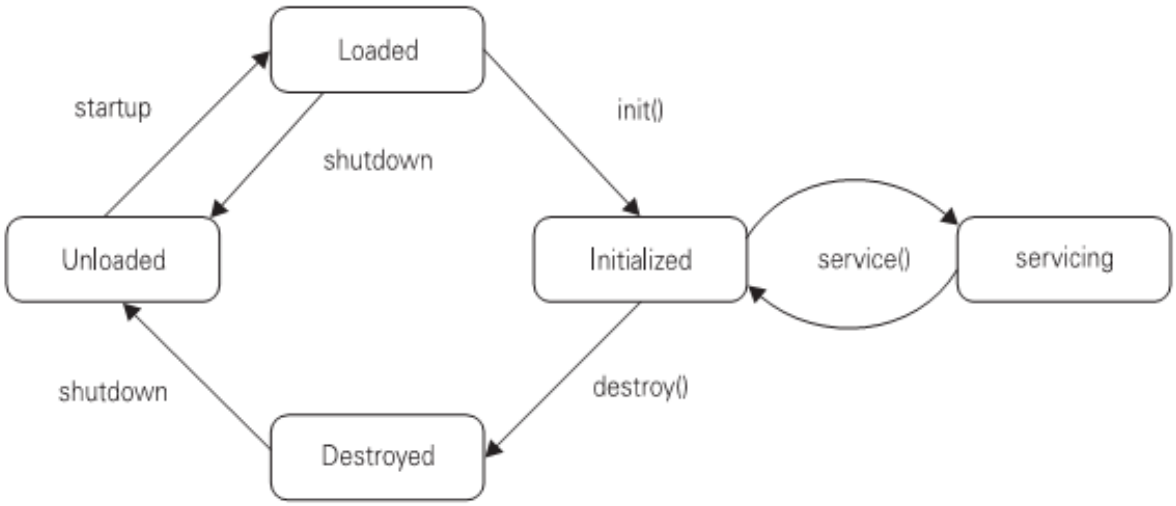


Рисунок 1.1 – Жизненный цикл сервлета

Существует два основных способа передачи данных от пользователя сервлету – с использованием методов GET и POST протокола HTTP.

Характеристика	GET	POST
Тип запрашиваемого ресурса	Статический или динамический	Динамический
Тип передаваемых данных	Текст	Текст или двоичные данные
Объём данных	Хотя HTTP не ограничивает длину URL, некоторые браузеры и серверы могут видеть только 255 символов.	Ограничен только настройками web-сервера (заданными администратором)
Видимость	Данные являются частью URL и видны пользователю в строке адреса.	Данные не являются частью URL и отправляются в теле HTTP-запроса, т.е. не видны.
Кэширование	Данные могут кэшироваться браузером в истории посещений.	Данные не кэшируются браузером в истории посещений.

2 Краткая справка по необходимым программным компонентам

Основным в Servlet API является интерфейс `Servlet`, и любой сервлет прямо или косвенно должен его реализовывать. Он включает пять методов:

Название метода	Описание
`init()`	Вызывается для уведомления сервлета о необходимости проинициализироваться и быть готовым к работе. В качестве параметра передаётся экземпляр класса `ServletConfig`.
`service()`	Вызывается для каждого поступившего от пользователя запроса, позволяя сервлету отреагировать на него.
`destroy()`	Метод вызывается для уведомления сервлета о необходимости освободить занимаемые ресурсы и приготовиться к выгрузке из контейнера.
`getServletConfig()`	Возвращает информацию о сервлете, такую как параметр для метода `init()`.
`getServletInfo()`	Должен возвращать информацию о сервлете – авторе, версии, авторских правах.

Для разработки сервлетов, предназначенных для обработки HTTP-запросов, в рамках Servlet API представлен абстрактный класс `HttpServlet`. В данном классе определяется перегруженная версия метода service() для работы с HTTP-запросами и HTTP-ответами, а также определяется ряд методов, соответствующих методам протокола HTTP: `doGet()`, `doPost()`, `doDelete()`, `doOptions()`, `doPut()`, `doTrace()`.

Интерфейс `HttpServletRequest` является средством доступа к параметрам запроса пользователя. Среди наиболее важных методов класса можно отметить:

- `getParameter()`, `getParameterValues()`, `getParameterNames()` – доступ к именам и значениям переменных, включенных в запрос;
- `getHeader()`, `getHeaders()`, `getHeaderNames()` – доступ к HTTP-заголовкам запроса;
- `getCookies()` – доступ к cookies, отправленным в запросе;
- `getSession()` – доступ к сессии.

Методы `HttpServletRequest` для доступа к переменным запроса:

Метод	Описание
`String getParameter (String parameterName)`	Метод возвращает только одно значение, связанное с указанной переменной.
`String[] getParameterValues (String parameterName)`	Метод возвращает все значения, связанные с указанной переменной.
`Enumeration getParameterNames()`	Метод возвращает множество имён полученных переменных.

Методы `HttpServletRequest` для доступа к заголовкам HTTP-запроса:

Метод	Описание
`String getHeader (String headerName)`	Метод возвращает только одно значение, связанное с указанным заголовком.
`String[] getHeaderValues (String headerName)`	Метод возвращает все значения, связанные с указанным заголовком.
`Enumeration getParameterNames()`	Метод возвращает множество имён полученных HTTP-заголовков.

Методы `HttpServletRequest` для доступа к данным сессии:

Метод	Описание
`HttpSession getSession(boolean create)`	Возвращает экземпляр текущей сессии, ассоциированной с запросом. В случае, если сессия ещё не начата, а флаг create = true, создаёт новую сессию.
`HttpSession getSession()`	Вызов данного метода является эквивалентом getSession(true);
`void setAttribute(String name, Object value)`	Добавляет объект value в сессию под именем name.
`Object getAttribute(String name)`	Извлекает из сессии объект, ассоциированный с именем name, или null, если с таким именем не связано объектов.

Интерфейс `HttpServletResponse` является средством отправки пользователю ответа, сконструированного на основе запроса. Среди наиболее важных методов ответа являются:

- `setHeader()`, `addHeader()` – установка HTTP-заголовков ответа;
- `containsHeader()` – проверка, был ли уже установлен такой HTTP-заголовок;
- `getWriter()` – получить экземпляр текстового выходного потока для записи ответа;
- `getOutputStream()` – получить экземпляр двоичного выходного потока для записи ответа.

3 Пример веб-приложения

Задание: составить программу интернет-чата на основе протокола HTTP. При входе в чат пользователю предлагается выбрать себе имя. Если такое имя в чате отсутствует, то пользователь попадает на список сообщений чата. Если выбранное имя уже занято, то посетителю предлагается выбрать другое имя. Если во время общения пользователь нажимает на ссылку «Выйти из чата», то занятое им имя освобождается и становится доступным немедленно. Если пользователь закрывает окно браузера, то его имя недоступно для выбора другими пользователями ещё в течение 60 минут. Если за эти 60 минут пользователь вернулся в чат, то без повторного выбора имени он автоматически попадает на список сообщений.

Внешний вид чата представлен на рисунках 3.1 – 3.2.

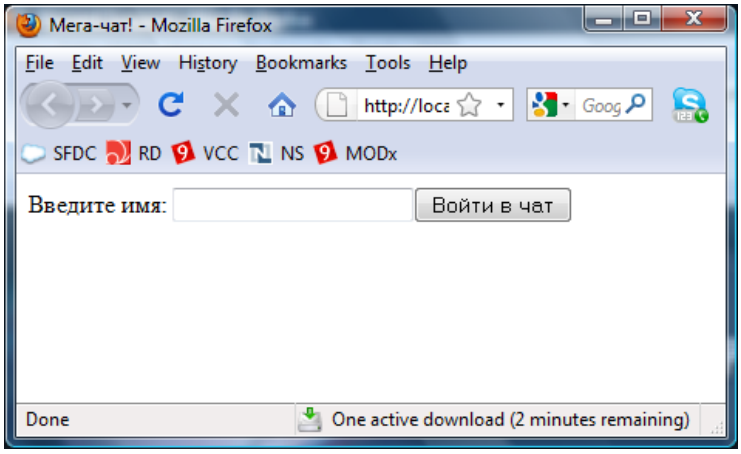


Рисунок 3.1 – Внешний вид чата при вводе имени пользователя

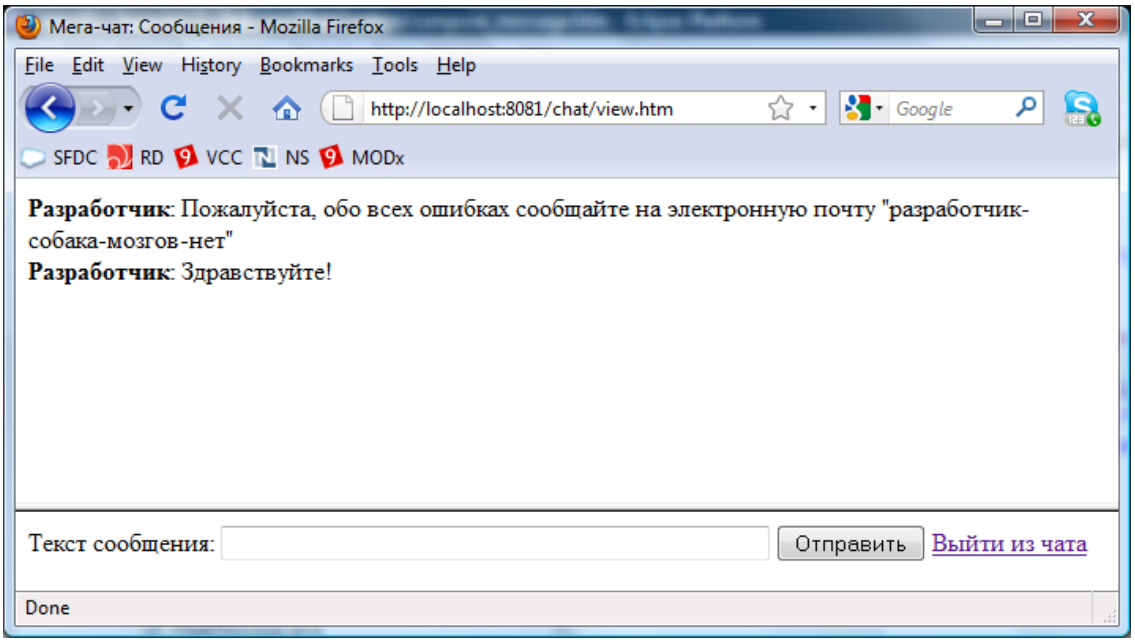


Рисунок 3.2 – Внешний вид чата в процессе общения

3.1 Структура веб-приложения

Структура данного приложения включает сервлеты, обеспечивающие функциональность чата, а также ряд вспомогательных классов.

Класс `ChatUser` (пользователь чата) описывает одного пользователя чата и имеет поля: имя пользователя; идентификатор сессии пользователя; последнее время взаимодействия пользователя с сервером (измеряется в миллисекундах, прошедших с 1 января 1970 года).

Класс `ChatMessage` (сообщение чата) описывает одно сообщение чата и имеет поля: текст сообщения; автор сообщения (ссылка на экземпляр класса `ChatUser`); временная метка, когда было добавлено сообщение.

Сервлет `LoginServlet` является точкой входа в чат и обрабатывает поступающие от новых пользователей запросы выбора имени. Сервлет должен иметь доступ к карте пользователей чата, которую он может изменять.

Сервлет `LogoutServlet` реализует функциональность завершения общения и выхода пользователя из чата. Сервлет должен иметь доступ к карте пользователей чата, которую он может изменять.

Сервлет `MessageListServlet` отображает список всех сообщений чата. Сервлет должен иметь доступ к карте пользователей и списку сообщений чата.

Сервлет `NewMessageServlet` добавляет в список сообщений чата новое сообщение. Сервлет должен иметь доступ к карте пользователей и списку сообщений чата.

Изначально, базовым классом (родителем) для всех сервлетов является класс `HttpServlet`. Однако заметим, что все сервлеты должны совместно пользоваться некоторыми общими структурами данных: карой пользователей и списком сообщений. Так как эти структуры данных являются разделяемыми, то и размещены они должны быть не внутри конкретного сервлета, а в некоторой совместно используемой области. В роли такой области в веб-приложениях на языке Java используется класс `ServletContext`, представляющий веб-контейнер, в котором функционируют все сервлеты веб-приложения. Любой из сервлетов во время своей инициализации (в методе `init`) должен извлечь из контекста сервлета ссылки на структуры данных карты пользователей и списка сообщений, и сохранить их в своих внутренних полях. Для того, чтобы избежать дублирования кода, ссылки на требуемые структуры данных и необходимую функциональность их получения из контекста сервлета целесообразно вынести в отдельный класс `ChatServlet`, являющийся предком для всех сервлетов нашего чата.

Диаграмма классов приложения, на которой видны выделенные классы и сервлеты, приведена на рисунке 3.3:

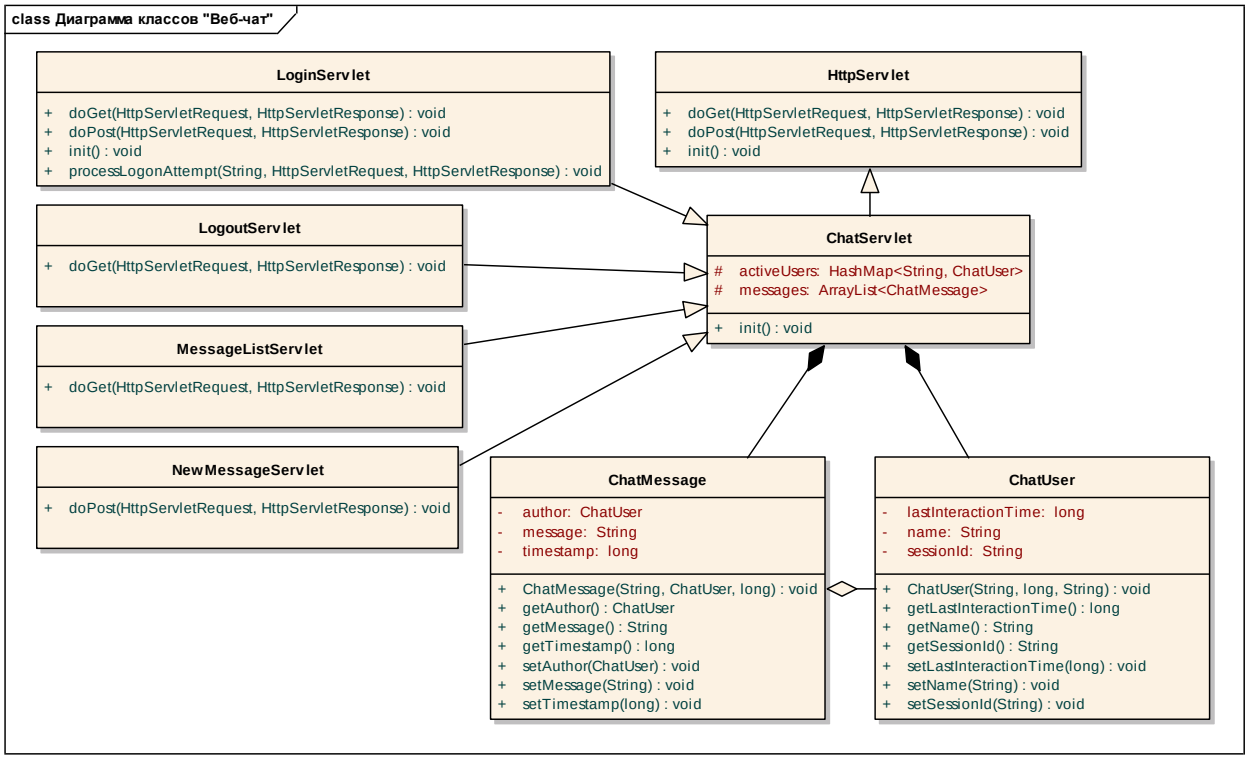


Рисунок 3.3 – Диаграмма классов веб-приложения

3.2 Подготовительный этап

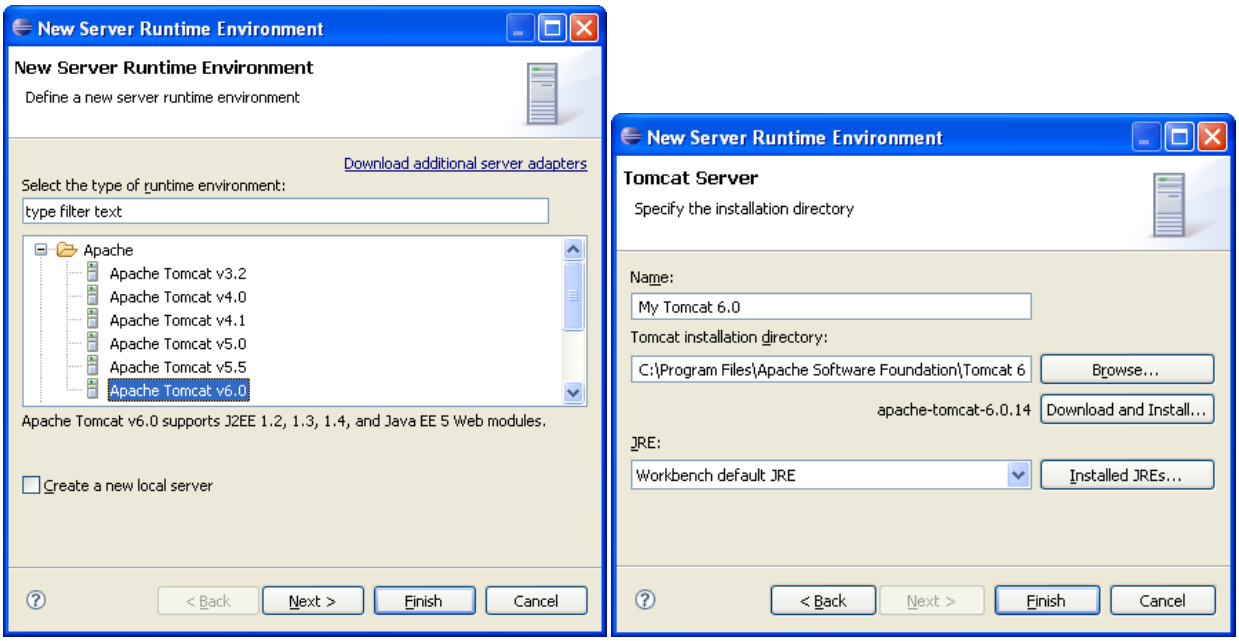
Для создания нового проекта веб-приложения в Eclipse необходимо активировать пункт меню «*File* → *New* → *Project*», и из списка возможных типов проектов в разделе «*Web*» следует выбрать «*Dynamic Web Project*».

В поле «*Project Name*» указывается название проекта, например «*Задание 7 – Веб-чат*».

Далее необходимо указать, в каком рабочем окружении будет выполняться проект (поле «*Target Runtime*»), и какая будет использоваться конфигурация (поле «*Configuration*»). Если ранее рабочее окружение уже было создано, то следует его выбрать из выпадающего списка, в противном случае его необходимо настроить. Для этого следует нажать кнопку «*New*», в появившемся диалоговом окне (рисунок 3.4, а) в разделе «*Apache*» выбрать «*Apache Tomcat v.6.0*», нажать «Next», в следующем диалоговом окне (рисунок 3.4, б) указать имя создаваемого окружения, нажав «*Browse*» задать папку, в которой был установлен Apache Tomcat (скорее всего имя папки будет *C:\Program Files\Apache Software Foundation\Tomcat 6.0*), нажать «*Finish*».

Для внесения изменений в конфигурацию рабочего окружения следует нажать кнопку «*Modify*» рядом с выпадающим списком «*Configuration*»). В появившемся диалоговом окне (рисунок 3.5) будет показано, поддержка

каких технологий в данный момент включена для настраиваемого рабочего окружения. На данном этапе нам нужна поддержка только самого языка Java и спецификации Servlet API, поэтому флажок напротив JavaScript Toolkit следует сбросить, после чего нажать кнопку «*OK*».



а) – выбор сервера приложений, б) указание расположения сервера

Рисунок 3.4 – Создание нового рабочего окружения

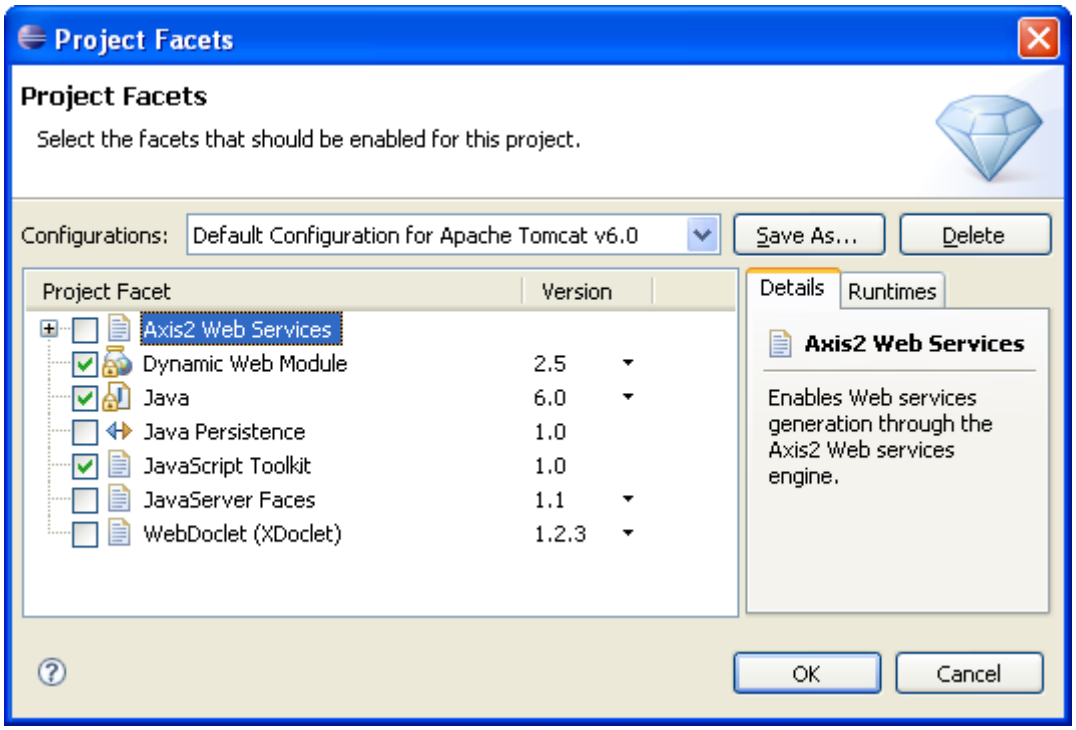


Рисунок 3.5 – Настройка конфигурации рабочего окружения

В следующем диалоговом окне (рисунок 3.6) следует указать имя контекста разрабатываемого приложения, после чего нажать «*Finish*».

Изменять значения полей «*Content Directory*» и «*Java Source Directory*» не рекомендуется.

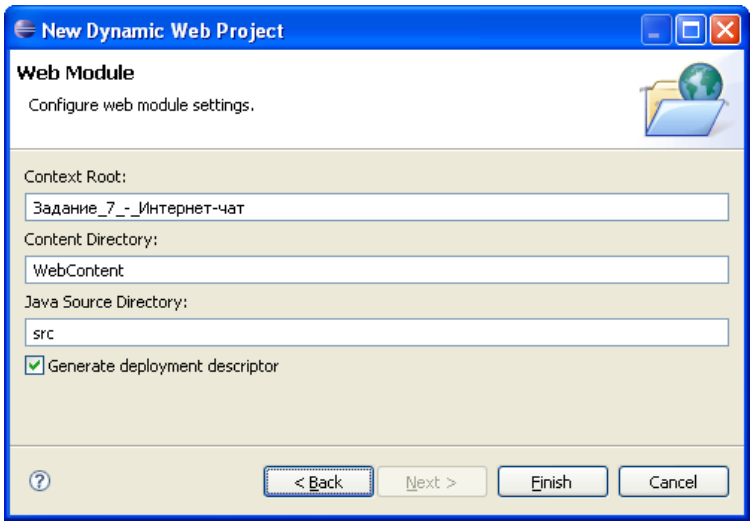


Рисунок 3.6 – Задание параметров модуля веб-приложения

Краткие сведения о контексте приложения

Будем полагать, что разрабатываемое приложение будет размещено на некотором хосте *hostname* (это может быть и www.mysuperchat.by, и localhost:8080). Тогда все создаваемые сервлеты и HTML-страницы будут доступны по адресу http://<hostname>/<contextname>/<filename>, где <hostname> – фактическое имя хоста, <contextname> – имя контекста, <filename> – имя сервлета или HTML-файла. Так как в URL допустимы только цифры и буквы латинского алфавита, то обязательно следует заменить предлагаемое по умолчанию имя контекста (полученное на основе имени проекта, например *Задание_7_-_Интернет-чат*) на допустимое (например *mychat*).

Сгенерированный проект будет иметь структуру, схожую с представленой на рисунке 3.7. В дальнейшем в папку «*Java Resources: src*» будут помещаться исходные коды классов и сервлетов проекта. В папке «*Libraries*» размещены библиотеки, используемые средой Eclipse при работе с проектом – на начальном этапе это библиотеки контейнера Tomcat, стандартные библиотеки Java Runtime Environment и библиотеки со спефицикацией Servlet API. В папку «*build*» будет помещаться откомпилированный байт-код классов и сервлетов. Папка «*WebContent*» предназначена для размещения HTML-файлов, графических файлов, JSP-страниц (о них в следующей лабораторной работе). Кроме того, внутри папки «*WebContent*» находится две специальных папки «*META-INF*» и «*WEB-INF*», удалять которые нельзя. Внутри «*WEB-INF*» размещается дескриптор веб-приложения (файл *web.xml*), описывающий его структуру, а также

библиотеки, используемые во время функционирования внутри контейнера Tomcat.

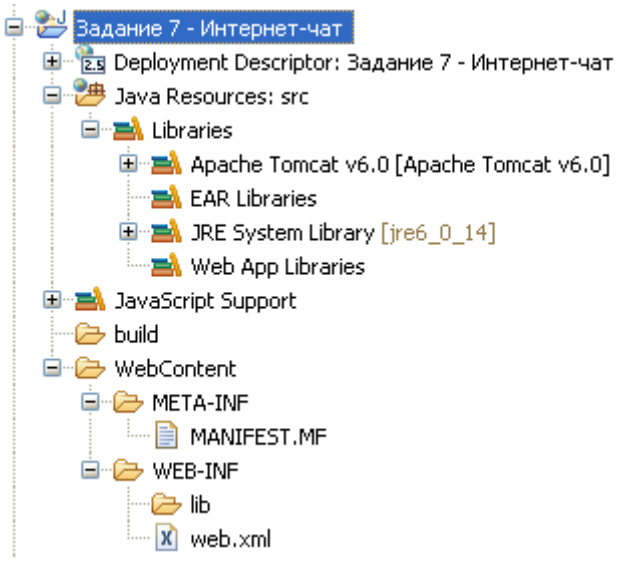


Рисунок 3.7 – Сгенерированная структура веб-приложения

Замечание: различия между папками «*Libraries*» и «*WEB-INF/lib*» заключаются в том, что библиотеки, находящиеся в первой из них, используются при работе с проектом в среде Eclipse, а библиотеки, находящиеся во второй – используются веб-приложением при функционировании внутри контейнера сервлетов (Apache Tomcat).

3.3 Реализация вспомогательных классов

Для работы с такими сущностями, как пользователь чата и сообщение чата, создадим представляющие их классы `ChatUser` и `ChatMessage`. В целях более чёткого структурирования кода, поместим оба этих класса в пакет `entity`. С учётом принятых в лабораторном практикуме правил именования пакетов, полное имя пакета будет, например, `bsu.rfe.java.group7.lab7.Ivanov.varB4.entity`.

Класс `ChatUser` имеет следующие поля данных экземпляров класса:

```
private String name;
private long lastInteractionTime;
private String sessionId;
```

Класс `ChatMessage` имеет следующие поля данных экземпляров класса:

```
private String message;
private ChatUser author;
private long timestamp;
```

В остальном классы создаются обычным способом (см. п.3.4, лабораторная работа №1), после чего с помощью средств Eclipse по

генерации кода для всех внутренних полей создаются пары «геттер-сеттер», а также создаётся конструктор для задания значений полей при конструировании экземпляров класса.

3.4 Реализация базового сервлета для веб-чата

Как уже обсуждалось ранее, всем сервлетам (являющимся независимыми классами с точки зрения Java) необходимо иметь совместный доступ к таким структурам данных, как список пользователей и список сообщений. В роли места для их хранения следует применять контекст сервлета (`ServletContext`), и при инициализации (в методе `init`) каждый сервлет должен получить из контекста ссылки на необходимые структуры данных и сохранить их во внутренних полях данных.

Замечание: Внутри контейнера сервлетов постоянно находится только один экземпляр сервлета. Он создаётся и инициализируется (в методе `init`) после получения первого HTTP-запроса (или заранее, если были сделаны необходимые настройки). Все последующие HTTP-запросы обрабатываются этим же экземпляром, но в различных потоках выполнения, что означает возможность совместной работы со значениями полей данных экземпляров класса.

Для хранения всех сервлетов приложения создадим пакет `servlet`, например `bsu.rfe.java.group7.lab7.Ivanov.varB4.servlet`.

Чтобы создать новый сервлет щёлкните правой кнопкой мыши на имени пакета, и выберите «*New* → *Servlet*». В появившемся диалоговом окне (рисунок 3.8) задайте имя сервлета, например *ChatServlet*, после чего нажмите «*Next*».

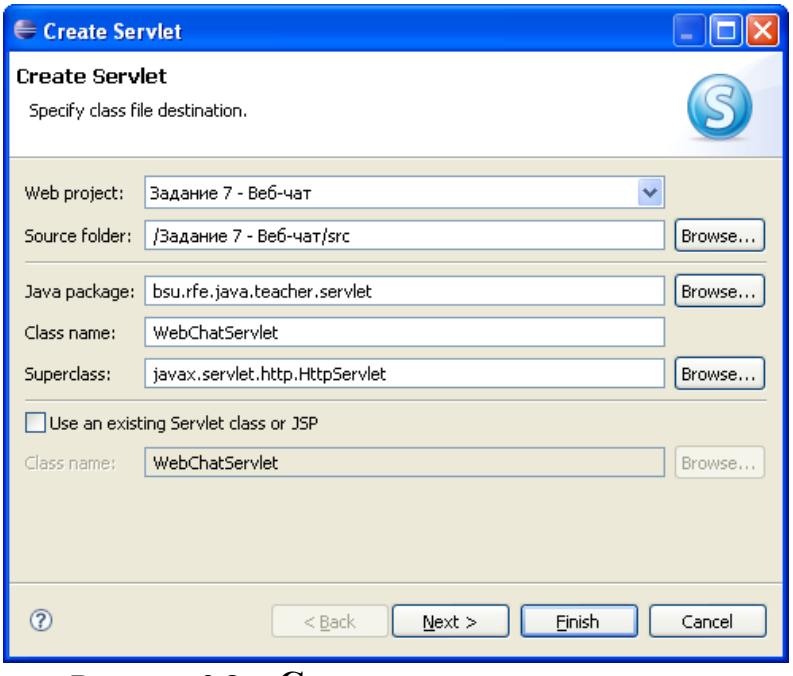


Рисунок 3.8 – Создание нового сервлета

В следующем диалоговом окне (рисунок 3.9) укажите имя, под которым сервлет будет фигурировать в дескрипторе веб-приложения (поле «*Name*»), при необходимости задайте параметры инициализации (нажав кнопку «*Add*» в блоке «*Initialization Parameters*»), а также укажите, запросы к каким URL будут передаваться сервлету на обработку (в блоке «*URL Mappings*»), после чего нажмите «*Next*». Например, если приложение размещено на локальной машине (*localhost:8080*), в качестве имени контекста (рисунок 3.6) выбрано *mychat*, а в «*URL Mappings*» указано */this-is-my-servlet*, то все запросы к URL http://localhost:8080/mychat/this-is-my-servlet будут передаваться на обработку сервлету.

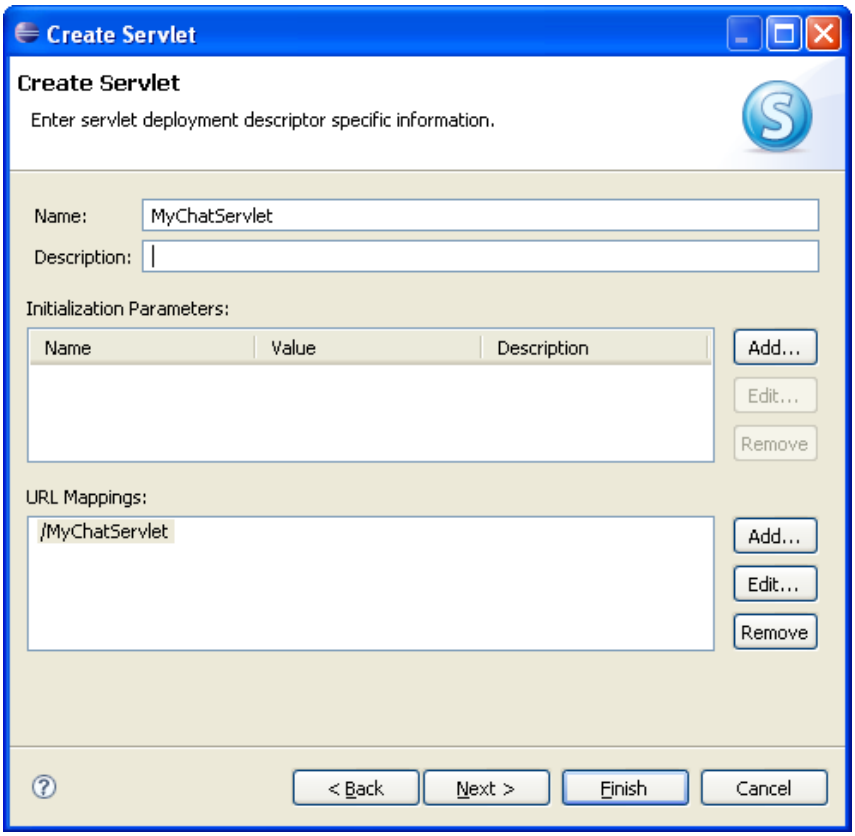


Рисунок 3.9 – Задание имени, параметров и привязки сервлета

Замечание: Если в своей работе сервлет зависит от каких-либо настраиваемых параметров (например, максимальное число пользователей в чате, после достижения которого новым пользователям отказывается во входе, или временной интервал, за который хранятся сообщения), то задание этих параметров в исходном коде сервлета является плохим решением, так как их изменение потребует внесения корректив в исходный код сервлета, перекомпиляции и переразмещения на веб-сервере всего веб-приложения. В подобном случае такие изменяемые параметры выносят из кода сервлета в дескриптор приложения в виде пар «ключ-значение». Во время работы сервлет имеет доступ к параметрам инициализации через объект `ServletConfig` (конфигурация сервлета), например:

```
// Прочитать из конфигурации сервлета значение параметра SESSION_TIMEOUT
String value = getServletConfig().getInitParameter("SESSION_TIMEOUT");
```

Замечание: Значение привязки (URL Mapping) может быть не только точным значением, но и шаблоном. Например: */this-is-my-servlet** – всё, что начинается с *this-is-my-servlet*. Кроме того, для сервлета можно задать как несколько значений привязки, так и ни одного (в этом случае сервлет не будет доступен пользователям). Отметим, однако, что мастер добавления сервлетов среды Eclipse не позволяет создать сервлет без привязки, поэтому мы создадим наш сервлет с привязкой, а затем удалим её редактированием дескриптора веб-приложения */WEB-INF/web.xml*.

В следующем диалоговом окне (рисунок 3.10) укажите, какие дополнительные интерфейсы реализует сервлет и какие его методы будут переопределяться, после чего нажмите «*Finish*». В создаваемом нами базовом сервлете не будет методов-обработчиков, а интересовать нас будет только метод `init` (см. рисунок 3.10).

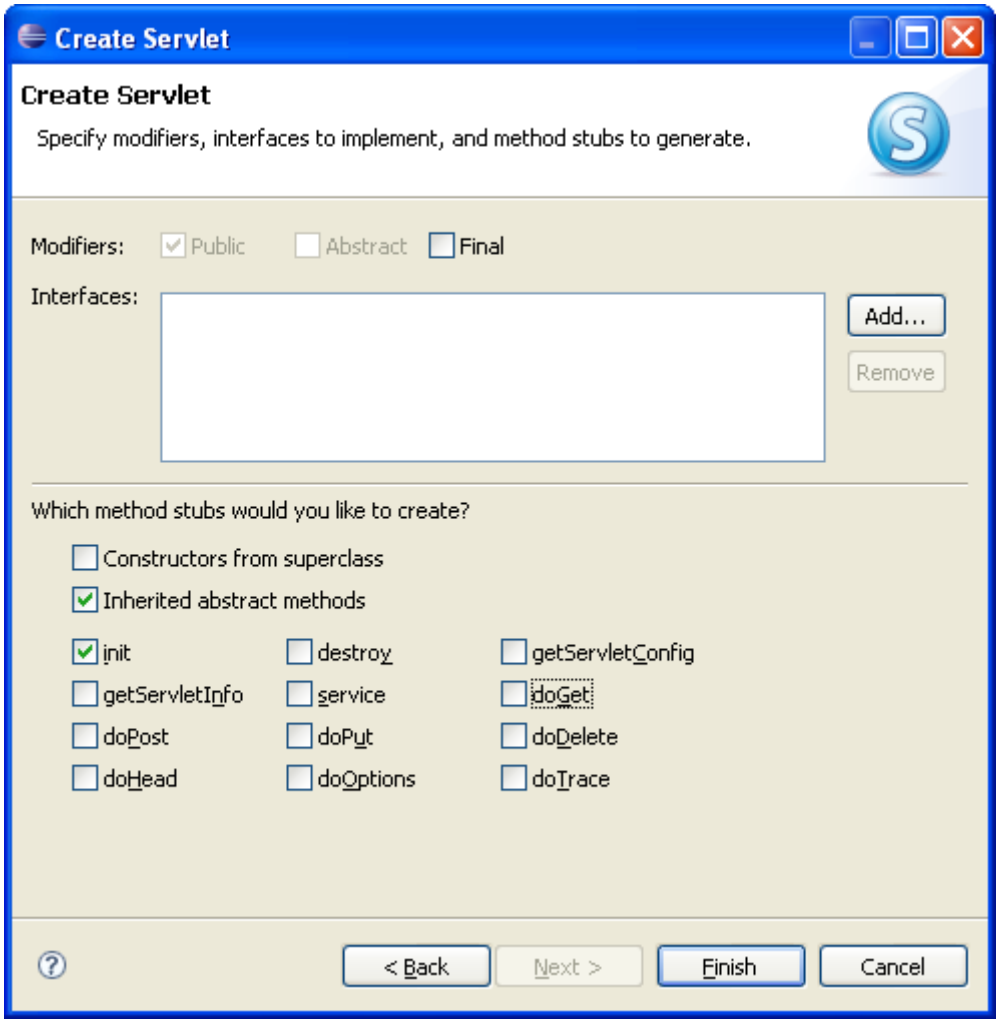


Рисунок 3.10 – Задание переопределяемых методов нового сервлета

Внутренними полями данных экземпляра класса создаваемого сервлета являются карта активных пользователей чата (в качестве ключа используется имя пользователя) и список сообщений чата:

```
// Карта текущих пользователей
protected HashMap<String, ChatUser> activeUsers;
// Список сообщений чата
protected ArrayList<ChatMessage> messages;
```

В методе инициализации сервлета `init()` из контекста сервлета извлекаются ссылки на общие структуры данных, и если оказывается, что они равны `null` (ещё не инициализированы), то выполняется их конструирование и помещение в контекст сервлета. Таким образом, загружающийся первым сервлет выполняет инициализацию общих структур данных, а сервлеты, загружающиеся позже, извлекают их из совместного контекста. С учётом сказанного, метод `init()` имеет следующий вид:

```
public void init() throws ServletException {
	// Вызвать унаследованную от HttpServlet версию init()
	super.init();
	// Извлечь из контекста карту текущих пользователей и список сообщений
	activeUsers = (HashMap<String, ChatUser>)
			getServletContext().getAttribute("activeUsers");
	messages = (ArrayList<ChatMessage>)
			getServletContext().getAttribute("messages");
	// Если карта пользователей ещё не определена ...
	if (activeUsers==null) {
		// Создать новую карту
		activeUsers = new HashMap<String, ChatUser>();
		// Поместить её в контекст сервлета,
		// чтобы другие сервлеты могли до неё добраться
		getServletContext().setAttribute("activeUsers", activeUsers);
	}
	// Если список сообщений ещё не определён ...
	if (messages==null) {
		// Создать новый список
		messages = new ArrayList<ChatMessage>(100);
		// Поместить его в контекст сервлета,
		// чтобы другие сервлеты могли до него добраться
		getServletContext().setAttribute("messages", messages);
	}
}
```

Последним шагом является удаление привязки сервлета из дескриптора веб-приложения *WEB-INF/web.xml*, чтобы этот сервлет не был доступен пользователям напрямую. Для этого дважды щёлкните мышью на файле *WEB-INF/web.xml*, в основном окне XML-редактора распахните узел *web-app* (рисунок 3.11), и последовательно удалите узлы *servlet* и *servlet-mapping*. Для этого необходимо щёлкнуть на узле правой кнопкой мыши и в контекстном меню выбрать «*Remove*».

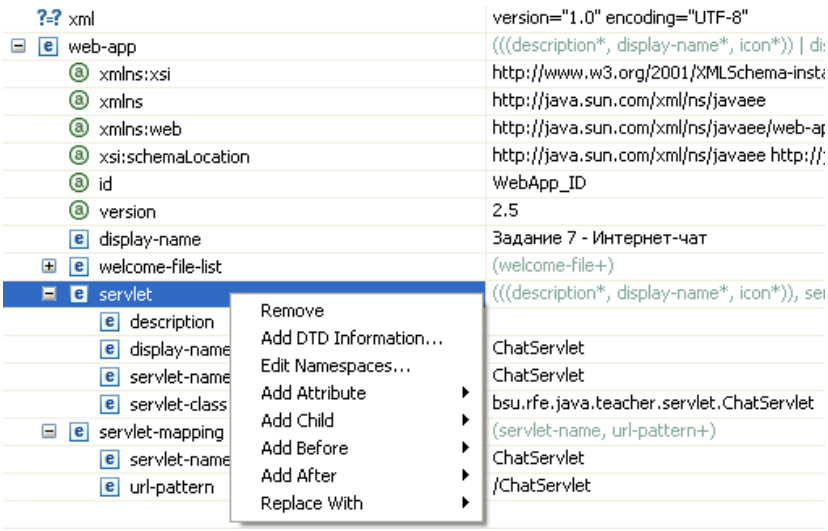


Рисунок 3.11 – Дескриптор веб-приложения, открытый в XML-редакторе

3.5 Реализация сервлета входа в чат LoginServlet

Сервлет `LoginServlet` предназначен для обработки запросов пользователей, желающих присоединиться к общению в чате. Исходя из заявленных требований к функциональности чата, данный сервлет будет обрабатывать как запросы по методу GET (пользователь набирает URL чата в строке адреса в браузере), так и по методу POST (пользователь указал своё имя в форме и нажал кнопку «*Войти в чат*»). Так как последовательность действий в обоих случаях во многом одинакова, то общую логику обработки запроса пользователя на вход в чат целесообразно выделить в отдельный метод `processLogonAttempt()`. Кроме того, если пользователь – претендент на имя – желает войти в чат с именем, которое уже занято другим – владельцем имени – (возник конфликт именам), то сервлет будет оценивать время, прошедшее с момента последней активности владельца имени. Как уже обсуждалось ранее, разумным является вынесение значения константы, определяющей максимальную допустимую длительность бездействия, из исходного кода сервлета в его конфигурационные настройки в дескрипторе веб-приложения. Следовательно, метод инициализации `init()` должен быть переопределён для загрузки настроек.

3.5.1 Создание каркаса сервлета

Создание класса сервлета выполняется по шагам, описанным в разделе 3.4 с некоторыми отличиями: 1) базовым классом для сервлета является не `HttpServlet`, а созданный в разделе 3.4 базовый сервлет `ChatServlet`; 2) в диалоговом окне задания параметров сервлета (рисунок 3.9) необходимо определить параметр с именем *SESSION_TIMEOUT* и значением, например, 3600 (т.е. 60 секунд в минуте на 60 минут = 1 час) и указать привязку к имени

/login.do ; 3) в диалоговом окне задания переопределяемых параметров (рисунок 3.10) поставить флажки на «*Inherited Abstract Methods*», «*init*», «*doGet*», «*doPost*». Сгенерированный мастером код сервлета будет иметь следующий вид:

```
package bsu.rfe.java.teacher.servlet;

import java.io.IOException;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginServlet extends ChatServlet {
	private static final long serialVersionUID = 1L;

	public LoginServlet() {
		// TODO Auto-generated constructor stub
	}

	public void init(ServletConfig config) throws ServletException {
		// TODO Auto-generated method stub
	}

	protected void doGet(HttpServletRequest request,
	  HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

	protected void doPost(HttpServletRequest request,
	  HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Поле данных экземпляров класса `sessionTimeout` определяет максимальную длительность промежутка бездействия:

```
// Длительность бездействия, в секундах
private int sessionTimeout = 600;
```

3.5.2 Реализация метода init()

По умолчанию длительность бездействия равна 600 секундам, но может быть переопределена с помощью конфигурационных настроек, что выполняется в методе инициализации `init()`:

```
public void init() throws ServletException {
	super.init();
	// Прочитать из конфигурации сервлета значение параметра SESSION_TIMEOUT
	String value = getServletConfig().getInitParameter("SESSION_TIMEOUT");
	// Если он задан, переопределить длительность сессии по умолчанию
	if (value!=null) {
		sessionTimeout = Integer.parseInt(value);
	}
}
```

3.5.3 Обработка GET-запросов методом doGet()

Получение сервлетом GET-запроса означает, что пользователь попал на страницу входа в чат либо набрав её адрес в строке браузера, либо воспользовавшись закладкой, либо перемещаясь по истории посещённых страниц. При обработке поступившего GET-запроса в методе `doGet()` сервлет выполняет следующие действия:

1) Проверяет, сохранено ли в сессии имя пользователя или сведения об ошибке. Если присутствует, то пользователь ранее уже выбрал имя, показ формы ввода имени не требуется, пользователь может переходить непосредственно к обмену сообщениями.

```
// Проверить, есть ли в сессии имя пользователя?
String name = (String)request.getSession().getAttribute("name");
// Извлечь из сессии сведения о предыдущей ошибке (возможной)
String errorMessage = (String)request.getSession().getAttribute("error");
```

2) Если в сессии имя отсутствует, то сервлет анализирует список cookies в поисках сохранённого идентификатора сессии. Так делается потому, что сессия уничтожается при закрытии всех окон браузера. При открытии нового окна будет создана новая сессия с отличным идентификатором, т.е. сохранённое в сессии имя будет утеряно. Для того, чтобы преодолевать перезагрузки браузера, при входе пользователя в чат идентификатор его сессии запоминается не только на сервере (в списке активных пользователей), но и в cookie (установленном на компьютере клиента и способном существовать длительное время), а при повторном входе изучается список установленных cookies.

```
// Если в сессии имени нет, то попытаться восстановить его через cookie
String previousSessionId = null;
if (name==null) {
      // Найти cookie с именем sessionId
      for (Cookie aCookie: request.getCookies()) {
            if (aCookie.getName().equals("sessionId")) {
                  // Запомнить значение этого cookie - это старый
                  // идентификатор сессии
                  previousSessionId = aCookie.getValue();
                  break;
            }
      }
      if (previousSessionId!=null) {
            // Мы нашли session cookie
            // Найти пользователя, у которого sessionId = найденному
            for (ChatUser aUser: activeUsers.values()) {
                  if (aUser.getSessionId().equals(previousSessionId)) {
                        // Мы нашли такого, т.е. восстановили имя
                        name = aUser.getName();
                        aUser.setSessionId(request.getSession().getId());
                  }
            }
      }
}
```

3) Если в сессии найдено имя пользователя, либо в cookie найден идентификатор сессии, то сервлет пытается обработать запрос на вход в чат. В случае успеха пользователь переадресуется к списку сообщений чата.

```
// Если в сессии имеется не пустое имя пользователя, то...
if (name!=null && !"".equals(name)) {
      errorMessage = processLogonAttempt(name, request, response);
}
```

4) В случае, если в сессии нет сохранённого имени пользователя, в cookies не сохранён идентификатора сессии, или попытка обработки запроса на вход завершилась неудачно, сервлет отображает форму для ввода имени пользователя. Форма отсылает введенное имя POST-методом этому же сервлету, но так как используется метод POST, то для обработки поступившего запроса будет вызван уже другой метод.

```
// Пользователю необходимо ввести имя. Показать соответствующую форму
// Задать кодировку HTTP-ответа
response.setCharacterEncoding("utf8");
// Получить поток вывода для HTTP-ответа
PrintWriter pw = response.getWriter();
pw.println("<html><head><title>Мега-чат!</title><meta http-equiv='Content-
Type' content='text/html; charset=utf-8'/></head>");
// Если возникла ошибка - сообщить о ней
if (errorMessage!=null) {
      pw.println("<p><font color='red'>" + errorMessage + "</font></p>");
}
// Вывести форму
pw.println("<form action='/chat/' method='post'>Введите имя: <input
type='text' name='name' value=''><input type='submit' value='Войти в чат'>");
pw.println("</form></body></html>");
// Сбросить сообщение об ошибке в сессии
request.getSession().setAttribute("error", null);
```

3.5.4 Обработка POST-запросов методом doPost()

Обработка сервлетом POST-запроса является более простой и прямолинейной:

1) Из HTTP-запроса извлечь выбранное имя пользователя;

```
// Задать кодировку HTTP-запроса - очень важно!
// Иначе вместо символов будет абракадабра
request.setCharacterEncoding("UTF-8");
// Извлечь из HTTP-запроса значение параметра 'name'
String name = (String)request.getParameter("name");
// Полагаем, что изначально ошибок нет
String errorMessage = null;
```

2) Если имя не пустое, то попытаться обработать запрос пользователя на вход в чат;

```
if (name==null || "".equals(name)) {
      // Пустое имя недопустимо - сообщить об ошибке
      errorMessage = "Имя пользователя не может быть пустым!";
} else {
      // Если ия не пустое, то попытаться обработать запрос
```

```
        errorMessage = processLogonAttempt(name, request, response);
}
```

3) Если было выбрано пустое имя, либо запрос не удалось успешно обработать (такое имя уже существует), то в сессии сохраняется сообщение об ошибке, после чего пользователь методом GET переадресуется на стартовую страницу чата с формой для ввода имени (т.е. запрос передаётся на обработку этому же сервлету, но методу `doGet`).

```
if (errorMessage!=null) {
        // Сбросить имя пользователя в сессии
        request.getSession().setAttribute("name", null);
        // Сохранить в сессии сообщение об ошибке
        request.getSession().setAttribute("error", errorMessage);
        // Переадресовать обратно на исходную страницу с формой
        response.sendRedirect(response.encodeRedirectURL("/chat/"));
}
```

3.5.5 Обработка запросов на вход в чат

Непосредственная обработка запросов пользователей на вход в чат выполняется в методе `processLogonAttempt()`, вызываемом как из `doGet()`, так и из `doPost()`. При этом выполняются следующие действия:

1) Проверяется, существует ли в чате пользователь с таким именем. Если такой пользователь отсутствует, то новый пользователь добавляется в список активных.

```
// Определить идентификатор Java-сессии пользователя
String sessionId = request.getSession().getId();
// Извлечь из списка объект пользователя, связанный с указанным именем
ChatUser aUser = activeUsers.get(name);
if (aUser==null) {
        // Если оно свободно, то добавить нового пользователя в список активных
        aUser = new ChatUser(name, Calendar.getInstance().getTimeInMillis(),
                                sessionId);
        // Так как одновременно выполняются запросы от множества пользователей
        // то необходима синхронизация на ресурсе
        synchronized (activeUsers) {
                activeUsers.put(aUser.getName(), aUser);
        }
}
```

2) Если идентификатор текущей сессии пользователя совпадает с идентификатором сессии пользователя, вошедшего под выбранным именем (т.е. на имя претендует его владелец), либо владелец имени бездействует дольше допустимого, то имя нового пользователя сохраняется в сессии, а идентификатор сессии – в cookie (как защита от потери сессии из-за закрытия всех окон браузера).

```
if (aUser.getSessionId().equals(sessionId) ||
        aUser.getLastInteractionTime()<
                (Calendar.getInstance().getTimeInMillis()-sessionTimeout*1000)) {
        // Если указанное имя принадлежит текущему пользователю,
        // либо оно принадлежало кому-то другому, но сессия истекла,
        // то одобрить запрос пользователя на это имя
```

```
        // Обновить имя пользователя в сессии
        request.getSession().setAttribute("name", name);
        // Обновить последнее время взаимодействия пользователя с сервером
        aUser.setLastInteractionTime(Calendar.getInstance().getTimeInMillis());
        // Обновить идентификатор сессии пользователя в cookies
        Cookie sessionIdCookie = new Cookie("sessionId", sessionId);
        // Установить срок годности cookie 1 год
        sessionIdCookie.setMaxAge(60*60*24*365);
        // Добавить cookie в HTTP-ответ
        response.addCookie(sessionIdCookie);
        // Перейти на сервлет со списком сообщений и формой ввода сообщений
        response.sendRedirect(response.encodeRedirectURL("/chat/view.htm"));
        // Вернуть null, т.е. сообщений об ошибках нет
        return null;
}
```

3) Если идентификаторы сессий не совпали (т.е. владелец имени и претендент на имя – разные личности) и максимальный срок бездействия пользователя не достигнут, то возвращается сообщение об ошибке и регистрация пользователя в чате не выполняется.

```
// Сохранённое в сессии имя уже закреплено за кем-то другим.
// Извиниться, отказать и попросить ввести другое имя
return "Извините, но имя <strong>" + name + "</strong> уже кем-то занято.
Пожалуйста выберите другое имя!";
```

3.6 Реализация сервлета выхода из чата LogoutServlet

Сервлет `LogoutServlet` предназначен для обработки запросов пользователей, желающих покинуть чат, и получает только GET-запросы.

3.6.1 Создание каркаса сервлета

Создание класса сервлета выполняется по шагам, описанным в разделе 3.4 с некоторыми отличиями: 1) базовым классом для сервлета является не `HttpServlet`, а созданный в разделе 3.4 базовый сервлет `ChatServlet`; 2) в диалоговом окне задания параметров сервлета (рисунок 3.9) необходимо указать привязку к имени */logout.do* ; 3) в диалоговом окне задания переопределяемых параметров (рисунок 3.10) поставить флажки на «*Inherited Abstract Methods*», «*doGet*».

3.6.2 Обработка GET-запросов методом doGet()

При обработке запросов сервлет исходит из того, что в сессии сохранено имя текущего пользователя. По этому имени из карты активных пользователей извлекается объект, описывающий пользователя с таким именем, а затем проверяется совпадение его идентификатора сессии с идентификатором сессии пользователя, пытающегося покинуть чат. Другими словами, выполняется проверка, тот ли пользователь пытается выйти из чата, что и входил в него.

```
protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
      String name = (String) request.getSession().getAttribute("name");
      // Если в сессии имеется имя пользователя...
```

```
        if (name!=null) {
              // Получить объект, описывающий пользователя с таким именем
              ChatUser aUser = activeUsers.get(name);
              // Если идентификатор сессии пользователя, вошедшего под этим
              // именем, совпадает с идентификатором сессии пользователя,
              // пытающегося выйти из чата (т.е. выходит тот же, кто и входил)
              if (aUser.getSessionId().equals((String)
                                          request.getSession().getId())) {
                    // Удалить пользователя из списка активных
                    // Т.к. запросы обрабатываются одновременно - синхронизация
                    synchronized (activeUsers) {
                          activeUsers.remove(name);
                    }
                    // Сбросить имя пользователя в сессии
                    request.getSession().setAttribute("name", null);
                    // Сбросить ID сессии в cookie
                    response.addCookie(new Cookie("sessionId", null));
                    // Перенаправить на главную страницу
                    response.sendRedirect(response.encodeRedirectURL("/chat/"));
              } else {
                    // Пользователь пытается аннулировать чужую сессию –
                    // не делать ничего
                    response.sendRedirect(
                          response.encodeRedirectURL("/chat/view.htm"));
              }
        } else {
              // Перенаправить пользователя на главное окно чата
              response.sendRedirect(
                    response.encodeRedirectURL("/chat/view.htm"));
        }
}
```

3.7 Реализация сервлета печати всех сообщений MessageListServlet

Данный сервлет предназначен для отображения списка всех сообщений в чате в обратном порядке, т.е. чтобы последние сообщения выводились первыми. Запросы к сервлету поступают только по методу GET.

3.7.1 Создание каркаса сервлета

Создание класса сервлета выполняется по шагам, описанным в разделе 3.4 с некоторыми отличиями: 1) базовым классом для сервлета является не `HttpServlet`, а созданный в разделе 3.4 базовый сервлет `ChatServlet`; 2) в диалоговом окне задания параметров сервлета (рисунок 3.9) необходимо указать привязку к имени */messages.do* ; 3) в диалоговом окне задания переопределяемых параметров (рисунок 3.10) поставить флажки на «*Inherited Abstract Methods*», «*doGet*».

3.7.2 Обработка GET-запросов методом doGet()

Обработка GET-запросов данным сервлетом сводится к записи в поток вывода, ассоциированный с HTTP-ответом, списка всех сообщений чата.

```
protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException, IOException {
      // Установить кодировку HTTP-ответа UTF-8
      response.setCharacterEncoding("utf8");
      // Получить доступ к потоку вывода HTTP-ответа
```

```
        PrintWriter pw = response.getWriter();
        // Записть в поток HTML-разметку страницы
        pw.println("<html><head><meta http-equiv='Content-Type'
content='text/html; charset=utf-8'/><meta http-equiv='refresh'
content='10'></head>");
        pw.println("<body>");
        // В обратном порядке записать HTML-разметку для каждого сообщения
        for (int i=messages.size()-1; i>=0; i--) {
              ChatMessage aMessage = messages.get(i);
              pw.println("<div><strong>" + aMessage.getAuthor().getName() +
"</strong>: " + aMessage.getMessage() + "</div>");
        }
        pw.println("</body></html>");
}
```

3.8 Реализация сервлета печати всех сообщений NewMessageServlet

Данный сервлет предназначен для добавления присланного пользователем сообщения в список сообщений чата. Запросы к сервлету поступают только по методу POST. После обработки запроса пользователь перенаправляется на обычную HTML-страницу, содержащую форму для ввода текста сообщения.

3.8.1 Создание каркаса сервлета

Создание класса сервлета выполняется по шагам, описанным в разделе 3.4 с некоторыми отличиями: 1) базовым классом для сервлета является не `HttpServlet`, а созданный в разделе 3.4 базовый сервлет `ChatServlet`; 2) в диалоговом окне задания параметров сервлета (рисунок 3.9) необходимо указать привязку к имени */send_message.do* ; 3) в диалоговом окне задания переопределяемых параметров (рисунок 3.10) поставить флажки на «*Inherited Abstract Methods*», «*doPost*».

3.8.2 Обработка POST-запросов методом doPost()

Обработка POST-запросов данным сервлетом сводится к добавлению в совместно используемый список сообщений чата нового элемента. Внимание необходимо обратить на два момента: 1) необходимо в явном виде задать кодировку, в которой будут интерпретироваться данные HTTP-запроса (в нашем случае – UTF-8); 2) т.к. ряд запросов может обрабатываться параллельно, то необходима синхронизация на ресурсе.

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        // По умолчанию используется кодировка ISO-8859. Так как мы передаём
        // данные в кодировке UTF-8, то необходимо установить соответствующую
        // кодировку HTTP-запроса
        request.setCharacterEncoding("UTF-8");
        // Извлечь из HTTP-запроса параметр 'message'
        String message = (String)request.getParameter("message");
        // Если сообщение не пустое, то
        if (message!=null && !"".equals(message)) {
              // По имени из сессии получить ссылку на объект ChatUser
              ChatUser author = activeUsers.get((String)
```

```
                        request.getSession().getAttribute("name"));
                synchronized (messages) {
                        // Добавить в список сообщений новое
                        messages.add(new ChatMessage(message, author,
                                Calendar.getInstance().getTimeInMillis()));
                }
        }
        // Перенаправить пользователя на страницу с формой нового сообщения
        response.sendRedirect("/chat/compose_message.htm");
}
```

3.9 Создание статических HTML-страниц

Помимо сервлетов, разрабатываемый чат будет использовать два статических HTML-документа: страницу, содержащую разметку главного окна чата; страницу с формой отправки сообщений.

3.9.1 Добавление документа с разметкой главного окна чата

Для добавления документа щёлкните правой кнопкой мыши на папке *WebContent* и в контекстном меню выбрать HTML. В диалоговом окне указания положения файла (рисунок 3.12) указать имя файла (например, *view.htm*), а в окне выбора HTML-шаблона документа (рисунок 3.13) выбрать «*New HTML File (4.01 frameset)*». Данный шаблон предполагает использование в документе нескольких фреймов, т.е. независимых областей. Это нам нужно потому, что требуется необходимо одновременно отображать как список сообщений, так и форму отправки, и они должны перегружаться независимо.

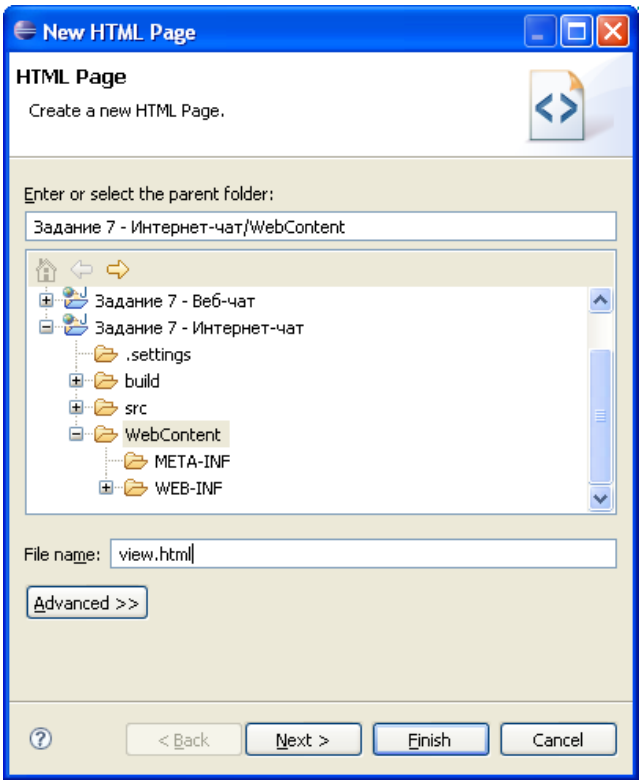


Рисунок 3.12 – Указание места расположения документа

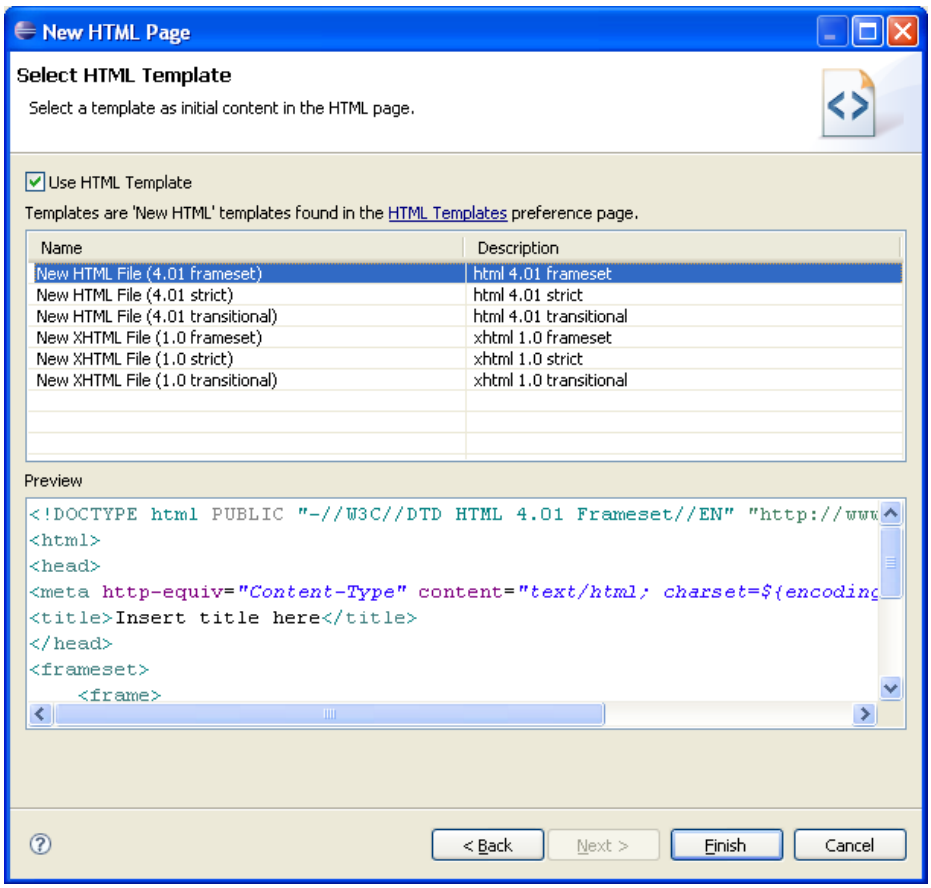


Рисунок 3.13 – Выбор используемого шаблона HTML-документа

Сгенерированная мастером HTML-разметка документа будет иметь вид схожий с представленной ниже:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<frameset>
    <frame>
    <frame>
    <noframes>
    <body>
    <p>This page uses frames. The current browser you are using does not
support frames.</p>
    </body>
    </noframes>
</frameset>
</html>
```

Для достижения требуемого результата отредактируем её до следующего вида:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Мега-чат: Сообщения</title>
</head>
<frameset rows="*,60">
    <frame name="messages" src="/chat/messages.do">
    <frame name="message" src="/chat/compose_message.htm">
    <noframes>
    <body>
    <p>Для работы этого чата необходима поддержка фреймов в Вашем
браузере.</p>
    </body>
    </noframes>
</frameset>
</html>
```

В теге `<title>` записывается заглавие документа (видимое в заголовке окна браузера). Атрибут `rows="*, 60"` в теге `<frameset>` означает, что всё пространство окна необходимо разбить на две строки: высота второй – строго 60 точек, а первая – занимает всё доступное пространство. Запись `<frameset cols="250,*,250">` означала бы: разбить всё пространство на три колонки, ширина первой – 250 точек, третьей – 250 точек, вторая занимает всё оставшееся пространство. Атрибут `src="…"` тега `<frame>` задаёт URL документа, который будет использоваться как содержание данного фрейма.

Замечание: Теги `<frameset>` могут вкладываться друг в друга, что необходимо для более сложного разбиения пространства окна. Например, следующая разметка делит окно на две части (нижнюю высотой 60 точек, верхнюю – занимающую всё оставшееся пространство), а верхнюю, кроме того, разбивает на две колонки (правую шириной 250 точек, левую – занимающую всё оставшееся пространство):

```
<frameset rows="*,60">
      <frameset cols="*,250">
          <frame name="messages" src="/chat/messages.do">
          <frame name="users" src="/chat/users.do">
      </frameset>
    <frame name="message" src="/chat/compose_message.htm">
</frameset>
```

3.9.2 Добавление документа с формой отправки сообщения

Документ с формой отправки сообщенися создаётся аналогично документу с разметкой главного окна чата с некоторыми отличиями: 1) в качестве имени файла указывается, например, compose_message.htm; 2) в качестве HTML-шаблона выбирается «*New HTML File (4.01 transitional)*».

Данный документ содержит HTML-форму с полем ввода, кнопкой отправки, а также гипессылку для выхода из чата.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
  </head>
<body>
  <form action="/chat/send_message.do" method="post">
      Текст сообщения:
      <input type="text" name="message" style="width: 50%">
      <input type="submit" value="Отправить">
      <a href="/chat/logout.do" target="_top">Выйти из чата</a>
  </form>
</body>
</html>
```

Внимание! Обратите внимание на URL-адреса, указанные в атрибуте `action` формы и гиперссылке – они соответствуют привязкам сервлетов `NewMessageServlet` и `LogoutServlet`, записанным с учётом имени контекста веб-приложения. Кроме того, атрибут `target="_top"` тега гиперссылки указывает, что содержание документа, на который ведёт гиперссылка, должно открываться не в текущем фрейме (нижней строке), а в окне верхнего уровня (основном окне браузера)

3.10 Подготовка к запуску, запуск и отладка приложения

Последним приготовлением перед запуском является задание страницы «по умолчанию», т.е. страницы, к которой будет направляться запрос пользователя в том случае, если точное имя сервлета или запрашиваемого ресурса не указано, например, если пользователь введёт адрес: *localhost:8080/chat/*. Как видно из URL, в этом случае присутствует имя хоста (*localhost:8080*), имя контекста приложения (*chat*), но отсутствует имя сервлета или HTML-документа. В таком случае запрос направляется документу по умолчанию. Для определения документа следует открыть для редактирования дескриптор веб-приложения */WEB-INF/web.xml* (дважды щёлкнув на его имени), распахнуть узлы *web-app* → *welcome-file-list* (рисунок 3.13), установить значение первого элемента *welcome-file* равным *login.do* (это привязка сервлета `LoginServlet`) и удалить другие элементы *welcome-file*.

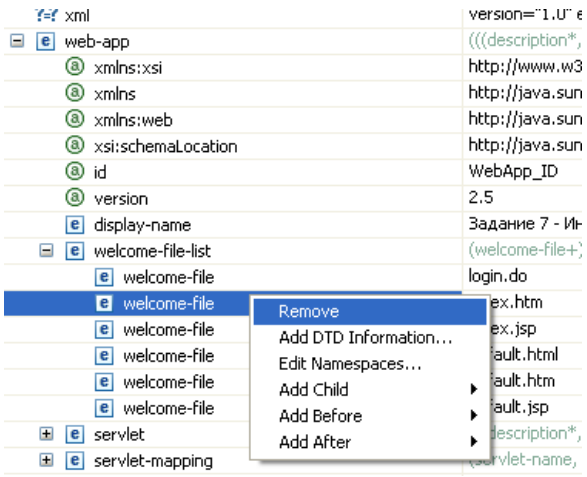
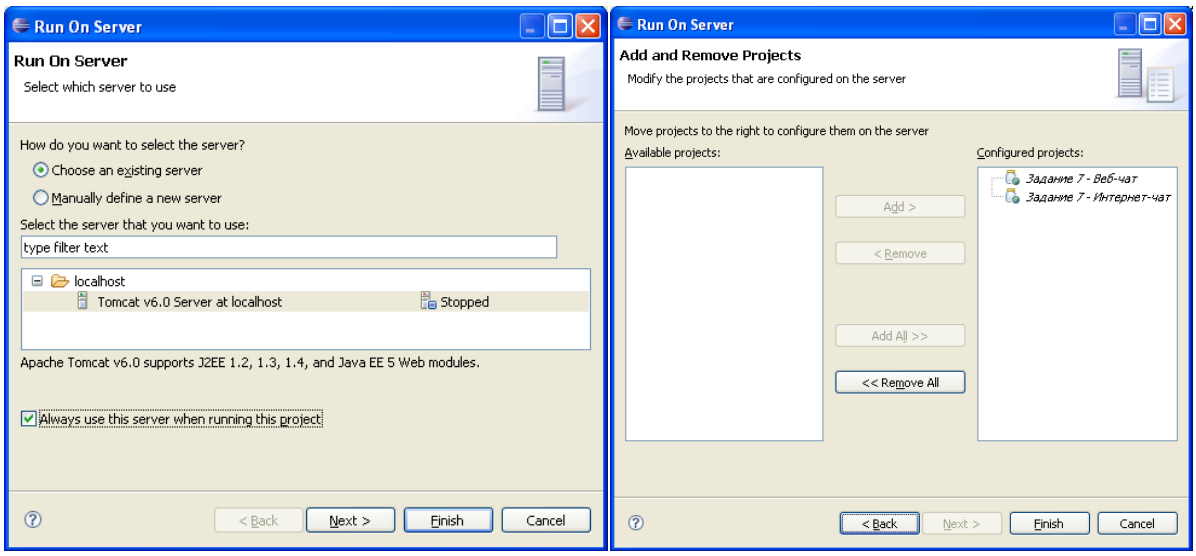


Рисунок 3.13 – Настройка страницы «по умолчанию»

Для запуска приложения следует щёлкнуть мышью на названии проекта, активировать пункт меню «*Run → Run As → Run on Server*» (рисунок 3.14, а), согласиться с использованием существующего сервера (радио-кнопка «*Choose an existing server*»), согласиться на постоянное использование данного сервера при каждом запуске проекта (флажок «*Always use this server...*»), в следующем диалоговом окне (рисунок 3.14, б) убедиться, что запускаемый проект находится в списке справа (при необходимости – выделить его в списке слева и нажать кнопку «*Add*») и нажать «*Finish*».



а) шаг 1 (слева); б) шаг 2 (справа)

Рисунок 3.14 – Настройка запуска на сервере

В результате будет выполнено динамическое развёртывание веб-приложения в контейнере Apache Tomcat, в основном окне редактора Eclipse откроется вкладка браузера, в которой будет отображено содержание документа «по умолчанию».

Замечание: Среда Eclipse и модуль интеграции с Tomcat, входящий в Eclipse, синхронизируют изменения, производимые пользователем в файлах, с копией веб-приложения, развернутого в контейнере. Тем не менее, в ряде случаев синхронизация «не лету» не может быть проведена. В этой ситуации следует перезапустить сервер самостоятельно, с переразмещением на нём приложения. Для этого во вкладке «*Servers*» правой кнопкой мыши щёлкнуть на имени сервера и в контекстном меню выбрать «*Restart*» (рисунок 3.15).

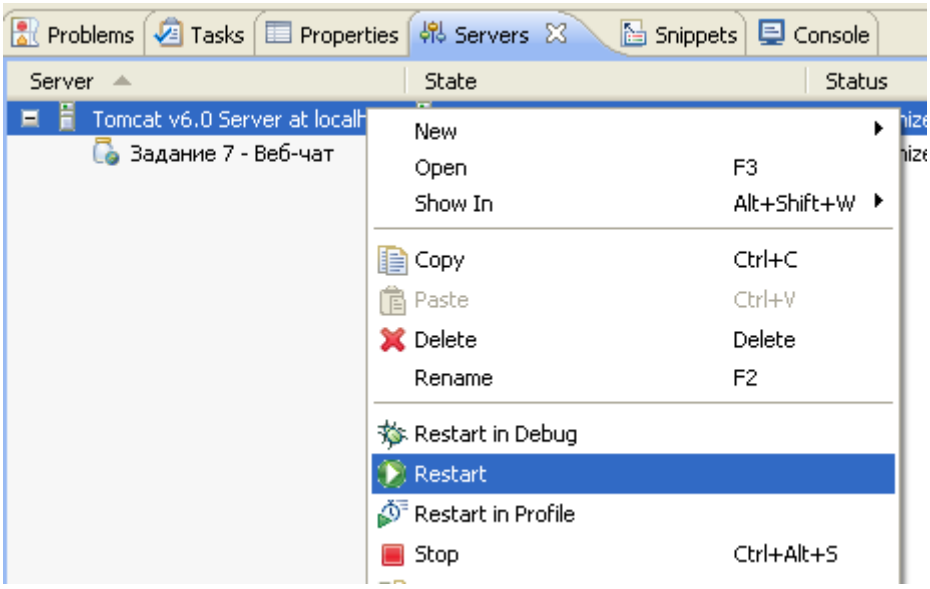


Рисунок 3.15 – Перезапуск сервера Tomcat

Для отладки веб-приложения необходимо расставить в исходном коде сервлетов точки останова (breakpoints), после чего с помощью команды меню «*Run → Debug As → Debug on Server*» запустить процесс отладки. При работе с веб-приложением в браузере в точках останова выполнение сервлетов будет приостанавливаться, а перспектива Eclipse переключаться в перспективу отладки (*Debug Perspective*), в которой отладка кода выполняется обычными средствами (F5 – выполнить строку кода с погружением внутрь функций, F6 – выполнить строку кода без погружния внутрь функций, F7 – выполнять код до возврата из текущего метода, F8 – возобновить выполнение кода).

Замечание: Отладка HTML-страниц невозможна.

4 Задания

4.1 Вариант А

Модифицировать приложение (по вариантам).

№ п/п	Цель модификаций
1	При входе пользователя в чат автоматически добавляется сообщение «Пользователь <ИмяПользователя> пришёл в чат»
2	Добавить в чат поддержку смайликов, т.е. заменять последовательности символов :) ;)
3	Отображать в чате не все сообщения, а только N последних (N задаётся через конфигурационный параметр).
4	Ограничить число пользователей в чате числом N (последующие запросы на вход в чат не обрабатывать, пользователю показывать соответствующее сообщение).
5	Подсчитывать суммарное число сообщений, отправленных каждым пользователем, и показывать это число рядом с именем пользователя в скобках (для каждого сообщения).
6	При выходе пользователя из чата автоматически добавляется сообщение «Пользователь <ИмяПользователя> покинул чат»
7	Отображать в чате сообщения только за последние X секунд (X задаётся через конфигурационный параметр).
8	Изменить вывод сообщений таким образом, чтобы если несколько последовательных фраз принадлежит одному пользователю, то его имя выводилось только в первой, а в последующих текст печатался сразу без имени.

4.2 Вариант В

Модифицировать приложение (по вариантам).

№ п/п	Цель модификаций
1	Реализовать в чате цензуру. Множество запрещённых слов, разделённых запятыми, передаётся сервлету через строку-конфигурационный параметр. Все вхождения запрещённых слов в текст сообщения заменяются на строку *бип*.
2	Добавить вывод даты и времени рядом с каждым сообщением. Для этого разобраться с классом Calendar.
3	Реализовать фильтр, вырезающий из всех сообщений HTML-теги
4	Добавить для модератора возможность выбрасывания из чата пользователей. Модератором считается первый пользователь чата. При выбрасывании пользователей их IP-адрес блокируется на X секунд.
5	Добавить для модератора возможность предупреждения пользователей. Модератором считается первый пользователь чата. При предупреждении пользователей возможность отправки ими сообщений блокируется на X секунд.
6	Добавить возможность выбора типа сообщения – шёпот (маленькими буквами курсивом), обычный, крик (большими жирными красными буквами). Выбирать тип сообщения при его отправке.
7	Добавить для пользователя возможность самостоятельно выбрать цвет букв в его сообщениях. Его выбор следует сохранять в cookies и восстанавливать при следующем визите в чат.
8	Защитить чат от несанкционированного просмотра – чтобы посторонние (не

	зарегистрировавшиеся под именем) не могли просматривать сообщения чата (сейчас это можно сделать обратившись напрямую к странице view.htm).
9	Реализовать функцию «Анти-спам» - если пользователь отправил более чем X сообщений за Y секунд, то возможность отправки им сообщений блокируется на Z секунд. X, Y, Z задавать через параметры сервлета.

4.3 Вариант C

Модифицировать приложение (по вариантам).

№ п/п	Цель модификаций
1	Если пользователь не отправляет сообщений больше N минут (задаётся через дескриптор веб-приложения), то чат (от его имени) рассказывает какую-нибудь шутку или анекдот. Подсказка – воспользоваться отдельным потоком.
2	Добавить в чат отдельный фрейм со списком пользователей. При щелчке на имя пользователя следующее отправляемое сообщение будет отправлено только указанному пользователю, а другие его не увидят (приватное сообщение).
3	Добавить возможность адресатных сообщений (от пользователя А – пользователю Б) и функцию автоответчика (т.е. на каждое полученное сообщение автоматически отсылается ответ «Спасибо большое, я обязательно подумаю об этом!»
4	Добавить функцию напоминаний для пользователя. Чат должен посылать пользователю приватные сообщения когда срок его бездействия приближается к предельному: «Внимание, до истечения бронирования вашего имени остаётся 5 (3, 1) минут. Если за это время вы не отправите ни одного сообщения, то ваше имя будет отдано первому претенденту на него».
5	Добавить возможность хранения карты пользователей и списка сообщений в файлах, чтобы при разрушении последнего сервлета (вызван метод destroy для последнего сервлета) они выгружались в файлы. При инициализации сервлетов выполнять обратное действие.
6	Добавить возможность создания в чате комнат и переходов между ними.

Приложение 1. Исходный код приложения

Класс пользователя чата ChatUser

```
package bsu.rfe.java.teacher.entity;

public class ChatUser {
	// Имя пользователя
	private String name;
	// Последнее время взаимодействия с сервером в количестве микросекунд,
	// прошедших с 1 января 1970 года
	private long lastInteractionTime;
	// Идентификатор Java-сессии пользователя
	private String sessionId;

	public ChatUser(String name,long lastInteractionTime,String sessionId) {
		super();
		this.name = name;
		this.lastInteractionTime = lastInteractionTime;
		this.sessionId = sessionId;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public long getLastInteractionTime() {
		return lastInteractionTime;
	}

	public void setLastInteractionTime(long lastInteractionTime) {
		this.lastInteractionTime = lastInteractionTime;
	}

	public String getSessionId() {
		return sessionId;
	}

	public void setSessionId(String sessionId) {
		this.sessionId = sessionId;
	}
}
```

Класс сообщения чата ChatMessage

```
package bsu.rfe.java.teacher.entity;

public class ChatMessage {
	// Текст сообщения
	private String message;
	// Автор сообщения
	private ChatUser author;
	// Временная метка сообщения (в микросекундах)
	private long timestamp;
```

```
	public ChatMessage(String message, ChatUser author, long timestamp) {
		super();
		this.message = message;
		this.author = author;
		this.timestamp = timestamp;
	}

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}

	public ChatUser getAuthor() {
		return author;
	}

	public void setAuthor(ChatUser author) {
		this.author = author;
	}

	public long getTimestamp() {
		return timestamp;
	}

	public void setTimestamp(long timestamp) {
		this.timestamp = timestamp;
	}
}
```

Базовый сервлет ChatServlet

```
package bsu.rfe.java.teacher.servlet;

import java.util.ArrayList;
import java.util.HashMap;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import bsu.rfe.java.teacher.entity.ChatMessage;
import bsu.rfe.java.teacher.entity.ChatUser;

public class ChatServlet extends HttpServlet {

	private static final long serialVersionUID = 1L;

	// Карта текущих пользователей
	protected HashMap<String, ChatUser> activeUsers;
	// Список сообщений чата
	protected ArrayList<ChatMessage> messages;

	@SuppressWarnings("unchecked")
	public void init() throws ServletException {
		// Вызвать унаследованную от HttpServlet версию init()
		super.init();
		// Извлечь из контекста карту пользователей и список сообщений
		activeUsers = (HashMap<String, ChatUser>)
				getServletContext().getAttribute("activeUsers");
		messages = (ArrayList<ChatMessage>)
				getServletContext().getAttribute("messages");
		// Если карта пользователей не определена ...
```

```
            if (activeUsers==null) {
                  // Создать новую карту
                  activeUsers = new HashMap<String, ChatUser>();
                  // Поместить её в контекст сервлета,
                  // чтобы другие сервлеты могли до него добраться
                  getServletContext().setAttribute("activeUsers",
activeUsers);
            }
            // Если список сообщений не определён ...
            if (messages==null) {
                  // Создать новый список
                  messages = new ArrayList<ChatMessage>(100);
                  // Поместить его в контекст сервлета,
                  // чтобы другие сервлеты могли до него добрать
                  getServletContext().setAttribute("messages", messages);
            }
      }
}
```

Сервлет входа в чат LoginServlet

```
package bsu.rfe.java.teacher.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Calendar;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import bsu.rfe.java.teacher.entity.ChatUser;

public class LoginServlet extends ChatServlet {

      private static final long serialVersionUID = 1L;

      // Длительность сессии, в секундах
      private int sessionTimeout = 10*60;

      public void init() throws ServletException {
            super.init();
            // Прочитать из конфигурации значение параметра SESSION_TIMEOUT
            String value =
getServletConfig().getInitParameter("SESSION_TIMEOUT");
            // Если он задан, переопределить длительность сессии по умолчанию
            if (value!=null) {
                  sessionTimeout = Integer.parseInt(value);
            }
      }
    // Метод будет вызван при обращении к сервлету HTTP-методом GET
    // т.е. когда пользователь просто открывает адрес в браузере
      protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
            // Проверить, есть ли уже в сессии заданное имя пользователя?
            String name = (String)request.getSession().getAttribute("name");
            // Извлечь из сессии сведения о предыдущей ошибке (возможной)
            String errorMessage =
(String)request.getSession().getAttribute("error");
            // Идентификатор предыдущей сессии изначально пуст
            String previousSessionId = null;
            // Если в сессии имя не сохранено, то попытаться
            // восстановить имя через cookie
            if (name==null) {
```

```
                // Найти cookie с именем sessionId
                for (Cookie aCookie: request.getCookies()) {
                      if (aCookie.getName().equals("sessionId")) {
                            // Запомнить значение этого cookie -
                            // это старый идентификатор сессии
                            previousSessionId = aCookie.getValue();
                            break;
                      }
                }
                if (previousSessionId!=null) {
                      // Мы нашли session cookie
                      // Попытаться найти пользователя с таким sessionId
                      for (ChatUser aUser: activeUsers.values()) {
                            if
(aUser.getSessionId().equals(previousSessionId)) {
                                  // Мы нашли такого, т.е. восстановили имя
                                  name = aUser.getName();
                            User.setSessionId(request.getSession().getId());
                            }
                      }
                }
          }
          // Если в сессии имеется не пустое имя пользователя, то...
          if (name!=null && !"".equals(name)) {
                errorMessage = processLogonAttempt(name, request, response);
          }
          // Пользователю необходимо ввести имя. Показать форму
          // Задать кодировку HTTP-ответа
          response.setCharacterEncoding("utf8");
          // Получить поток вывода для HTTP-ответа
          PrintWriter pw = response.getWriter();
          pw.println("<html><head><title>Мега-чат!</title><meta http-
equiv='Content-Type' content='text/html; charset=utf-8'/></head>");
          // Если возникла ошибка - сообщить о ней
          if (errorMessage!=null) {
                pw.println("<p><font color='red'>" + errorMessage +
"</font></p>");
          }
          // Вывести форму
          pw.println("<form action='/chat/' method='post'>Введите имя:
<input type='text' name='name' value=''><input type='submit' value='Войти в
чат'>");
          pw.println("</form></body></html>");
          // Сбросить сообщение об ошибке в сессии
          request.getSession().setAttribute("error", null);
     }

     // Метод будет вызван при обращении к сервлету HTTP-методом POST
     // т.е. когда пользователь отправляет сервлету данные
     protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
          // Задать кодировку HTTP-запроса - очень важно!
          // Иначе вместо символов будет абракадабра
          request.setCharacterEncoding("UTF-8");
          // Извлечь из HTTP-запроса значение параметра 'name'
          String name = (String)request.getParameter("name");
          // Полагаем, что изначально ошибок нет
          String errorMessage = null;
          if (name==null || "".equals(name)) {
                // Пустое имя недопустимо - сообщить об ошибке
                errorMessage = "Имя пользователя не может быть пустым!";
          } else {
                // Если ия не пустое, то попытаться обработать запрос
                errorMessage = processLogonAttempt(name, request, response);
```

```
            }
            if (errorMessage!=null) {
                  // Сбросить имя пользователя в сессии
                  request.getSession().setAttribute("name", null);
                  // Сохранить в сессии сообщение об ошибке
                  request.getSession().setAttribute("error", errorMessage);
                  // Переадресовать обратно на исходную страницу с формой
                  response.sendRedirect(response.encodeRedirectURL("/chat/"));
            }
      }

      // Возвращает текстовое описание возникшей ошибки или null
      String processLogonAttempt(String name, HttpServletRequest request,
HttpServletResponse response) throws IOException {
            // Определить идентификатор Java-сессии пользователя
            String sessionId = request.getSession().getId();
            // Извлечь из списка объект, связанный с этим именем
            ChatUser aUser = activeUsers.get(name);
            if (aUser==null) {
                  // Если оно свободно, то добавить
                  // нового пользователя в список активных
                  aUser = new ChatUser(name,
Calendar.getInstance().getTimeInMillis(), sessionId);
                  // Так как одновременно выполняются запросы
                  // от множества пользователей
                  // то необходима синхронизация на ресурсе
                  synchronized (activeUsers) {
                        activeUsers.put(aUser.getName(), aUser);

                  }
            }
            if (aUser.getSessionId().equals(sessionId) ||
aUser.getLastInteractionTime()<(Calendar.getInstance().getTimeInMillis()-
sessionTimeout*1000)) {
                  // Если указанное имя принадлежит текущему пользователю,
                  // либо оно принадлежало кому-то другому, но сессия истекла,
                  // то одобрить запрос пользователя на это имя

                  // Обновить имя пользователя в сессии
                  request.getSession().setAttribute("name", name);
                  // Обновить время взаимодействия пользователя с сервером

      aUser.setLastInteractionTime(Calendar.getInstance().getTimeInMillis());
                  // Обновить идентификатор сессии пользователя в cookies
                  Cookie sessionIdCookie = new Cookie("sessionId", sessionId);
                  // Установить срок годности cookie 1 год
                  sessionIdCookie.setMaxAge(60*60*24*365);
                  // Добавить cookie в HTTP-ответ
                  response.addCookie(sessionIdCookie);
                  // Перейти к главному окну чата
      response.sendRedirect(response.encodeRedirectURL("/chat/view.htm"));
                  // Вернуть null, т.е. сообщений об ошибках нет
                  return null;
            } else {
                  // Сохранённое в сессии имя уже закреплено за кем-то другим.
                  // Извиниться, отказать и попросить ввести другое имя
                  return "Извините, но имя <strong>" + name + "</strong> уже
кем-то занято. Пожалуйста выберите другое имя!";
            }
      }

}
```

Сервлет выхода из чата LogoutServlet

```
package bsu.rfe.java.teacher.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import bsu.rfe.java.teacher.entity.ChatUser;

public class LogoutServlet extends ChatServlet {

	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		String name = (String) request.getSession().getAttribute("name");
		// Если в сессии имеется имя пользователя...
		if (name!=null) {
			// Получить объект, описывающий пользователя с таким именем
			ChatUser aUser = activeUsers.get(name);
			// Если идентификатор сессии пользователя, вошедшего под
			// этим именем, совпадает с идентификатором сессии
			// пользователя, пытающегося выйти из чата
			// (т.е. выходит тот же, кто и входил)
			if (aUser.getSessionId().equals((String)
request.getSession().getId())) {
				// Удалить пользователя из списка активных
				// Т.к. запросы обрабатываются одновременно,
				// нужна синхронизация
				synchronized (activeUsers) {
					activeUsers.remove(name);
				}
				// Сбросить имя пользователя в сессии
				request.getSession().setAttribute("name", null);
				// Сбросить ID сессии в cookie
				response.addCookie(new Cookie("sessionId", null));
				// Перенаправить на главную страницу
			response.sendRedirect(response.encodeRedirectURL("/chat/"));
			} else {
				// Пользователь пытается аннулировать чужую сессию –
				// не делать ничего
	response.sendRedirect(response.encodeRedirectURL("/chat/view.htm"));
			}
		} else {
			// Перенаправить пользователя на главное окно чата
	response.sendRedirect(response.encodeRedirectURL("/chat/view.htm"));
		}
	}

}
```

Сервлет списка сообщений MessageListServlet

```
package bsu.rfe.java.teacher.servlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import bsu.rfe.java.teacher.entity.ChatMessage;
```

```
public class MessageListServlet extends ChatServlet {
	private static final long serialVersionUID = 1L;

	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		// Установить кодировку HTTP-ответа UTF-8
		response.setCharacterEncoding("utf8");
		// Получить доступ к потоку вывода HTTP-ответа
		PrintWriter pw = response.getWriter();
		// Запись в поток HTML-разметку страницы
		pw.println("<html><head><meta http-equiv='Content-Type'
content='text/html; charset=utf-8'/><meta http-equiv='refresh'
content='10'></head>");
		pw.println("<body>");
		// В обратном порядке записать в поток HTML-разметку для каждого
сообщения
		for (int i=messages.size()-1; i>=0; i--) {
			ChatMessage aMessage = messages.get(i);
			pw.println("<div><strong>" + aMessage.getAuthor().getName()
+ "</strong>: " + aMessage.getMessage() + "</div>");
		}
		pw.println("</body></html>");
	}
}
```

Сервлет добавления нового сообщения NewMessageServlet

```
package bsu.rfe.java.teacher.servlet;

import java.io.IOException;
import java.util.Calendar;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import bsu.rfe.java.teacher.entity.ChatMessage;
import bsu.rfe.java.teacher.entity.ChatUser;

public class NewMessageServlet extends ChatServlet {
	private static final long serialVersionUID = 1L;

	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
		// По умолчанию используется кодировка ISO-8859. Так как мы
		// передаём данные в кодировке UTF-8
		// то необходимо установить соответствующую кодировку HTTP-запроса
		request.setCharacterEncoding("UTF-8");
		// Извлечь из HTTP-запроса параметр 'message'
		String message = (String)request.getParameter("message");
		// Если сообщение не пустое, то
		if (message!=null && !"".equals(message)) {
			// По имени из сессии получить ссылку на объект ChatUser
			ChatUser author = activeUsers.get((String)
request.getSession().getAttribute("name"));
			synchronized (messages) {
				// Добавить в список сообщений новое
				messages.add(new ChatMessage(message, author,
Calendar.getInstance().getTimeInMillis()));
			}
		}
		// Перенаправить пользователя на страницу с формой сообщения
		response.sendRedirect("/chat/compose_message.htm");
	}

}
```

Страница с общим видом чата view.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Мега-чат: Сообщения</title>
</head>
<frameset rows="*,60">
    <frame name="messages" src="/chat/messages.do">
    <frame name="message" src="/chat/compose_message.htm">
    <noframes>
    <body>
    <p>Для работы этого чата необходима поддержка фреймов в Вашем
браузере.</p>
    </body>
    </noframes>
</frameset>
</html>
```

Страница с формой ввода сообщения compose_message.htm

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<form action="/chat/send_message.do" method="post">
      Текст сообщения:
      <input type="text" name="message" style="width: 50%">
      <input type="submit" value="Отправить">
      <a href="/chat/logout.do" target="_top">Выйти из чата</a>
</form>
</body>
</html>
```