Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

В.Г. Резник

РАСПРЕДЕЛЕННЫЕ СЕРВИС-ОРИЕНТИРОВАННЫЕ СИСТЕМЫ
(ЛАБОРАТОРНЫЕ РАБОТЫ)

Учебно-методическое пособие
для студентов технических направлений подготовки и специальностей

Томск
2020

УДК 004.75
ББК 30.2-5-05
Р 344

Рецензенты:

Бойченко И.В., программист АО «ИнфоТеКС», кандидат техн. наук

Ефремов В.А., ведущий инженер ООО «КС Групп», кандидат техн. наук

Резник, Виталий Григорьевич

Р 344 Распределенные сервис-ориентированные системы (лабораторные работы) : учеб. метод пособие / В.Г. Резник. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2020. – 63 с.

Представлено описание лабораторных работ по дисциплине «Распределенные сервис-ориентированные системы».

Для студентов направления подготовки магистратуры: 09.04.01 «Информатика и вычислительная техника», направленность (профиль) программы - «Программное обеспечение вычислительных машин, систем и компьютерных сетей».

Одобрено на заседании каф. АСУ, протокол №_5_ от _22.04.2021_

УДК 004.75
ББК 30.2-5-05

Введение

Данное пособие содержит учебно-методический материал для выполнения лабораторных работ в пределах дисциплины «Распределенные сервис-ориентированные системы» (РСОС), изучаемых на уровне магистратуры по направлению подготовки 09.04.01 «Информатика и вычислительная техника».

Предметная область лабораторных работ охватывает теоретический материал, изложенный в учебном пособии [1]. Сами работы выполняются в среде учебного программного комплекса [2] и предполагают наличие у студента начальных формирующих знаний, полученных им при изучении следующих дисциплин уровня бакалавриата: «Распределенные вычислительные системы» [3], «Базы данных», «Объектно-ориентированное программирование», «Операционные системы», «Сети и телекоммуникации».

Целью проведения лабораторных работ является получение базовых практических навыков использования технологий РСОС, которые выражаются получением компетенций, представленных в таблице В.1.1.

Таблица В.1.1 — Целевые компетенции изучаемой дисциплины

Код	**Содержание**
УК-2	Способен управлять проектом на всех этапах его жизненного цикла.
УК-4	Способен применять современные коммуникативные технологии, в том числе на иностранном(ых) языке(ах), для академического и профессионального взаимодействия.

Основной задачей выполняемых работ является освоение технологий РСОС, реализованных на платформе языка Java и изложенных в теоретическом материале учебного пособия [1], разделенного на шесть тематических частей:

- **Тема 1**. Предметная область и терминология РСОС.
- **Тема 2**. Компонент JSF контейнера Web.
- **Тема 3**. Современные средства доступа к данным.
- **Тема 4.** Обработка XML и JSON.
- **Тема 5**. Web-службы SOAP.
- **Тема 6**. Web-службы в стиле REST.

Плановый объем выполняемых работ составляет 36 академических часов, что соответствует девяти лабораторным работам (по четыре академических часа каждая), описанных в отдельных главах данного пособия и распределенных по темам учебного пособия [1] согласно таблице В.1.2.

Таблица В.1.2 — Соответствие лабораторных работ темам пособия [1]

№ работы	№ темы	Название работы
1	1	Дистрибутив ОС УПК АСУ
2	2	Использование компонента JavaServer Faces
3	2	Области действия технологии JSF
4	3	Современные способы доступа к данным
5	4	Представление информации с помощью XML
6	4	Представление информации с помощью JSON
7	5	Классические средства описания Web-сервисов
8	5	Классические средства реализации Web-сервисов
9	6	Web-службы в стиле REST

Плановая последовательность выполнения лабораторных работ осуществляется в порядке их нумерации и проводится следующим образом:

- на первом занятии студент получает от преподавателя индивидуальную рабочую область пользователя ***upk*** учебной среды ОС УПК АСУ [2], представленную файлом ***rsos-home.ext4fs.gz*** и которую он размещает на личном ***flashUSB***, согласно требованиям лабораторной работы №1;
- в процессе выполнения лабораторной работы, студент, в произвольной форме, заполняет соответствующий раздел единого индивидуального отчета, размещенного на рабочем столе пользователя ***upk*** в файле ***Отчет.doc***;
- по окончанию учебного времени лабораторной работы, каждый студент сохраняет на личном ***flashUSB*** дубликат файла ***Отчет.doc*** и архивирует свою рабочую область.

Студент несет полную ответственность за сохранность индивидуального отчета, который он должен предоставлять преподавателю для контроля, по первому его требованию.

Индивидуальный отчет студента является документом, с помощью которого преподаватель проводит текущий контроль успеваемости студента.

1 Работа 1. Тестирование ПО рабочей области студента

Лабораторная работа №1 посвящена индивидуальной настройке и тестированию рабочей области студента, которая входит составной частью в программный учебный комплекс кафедры АСУ [2].

Данная работа предназначена для закрепления теоретического материала главы 1 «*Предметная область и терминология РСОС*» учебного пособия [1], посвященного следующим вопросам:

- 1.1 Этапы развития распределенных систем.
- 1.2 Становление систем с сервис-ориентированной архитектурой.
- 1.3 Современные парадигмы сервис-ориентированных архитектур.
- 1.4 Программная платформа Java Enterprise Edition.
- 1.5 Инструментальные средства реализации РСОС.

Учебная цель данной лабораторной работы — проверка состава, наличия, доступности и работоспособности учебного материала и специального программного обеспечения, необходимого для выполнения пяти последующих лабораторных заданий.

В плане технологий, которые используются всеми практически значимыми РСОС, являются инструменты *СУБД*, управляющие хранилищами информации, интегрированные среды разработки (IDE) программного обеспечения, применяемые на всех стадиях создания и модификации РСОС, сервера приложений, обеспечивающие среду функционирования РСОС.

Платформа Java предоставляет большое разнообразие имеющихся инструментальных средств. Начальная часть данного лабораторного курса базируется на ПО СУБД Derby, IDE Eclipse EE и сервере приложений TomEE.

Выполнение заданий текущей лабораторной работы проводится в четыре этапа, изложенных в соответствующих подразделах данного пособия, и начинается с обязательного анализа дистрибутива ОС УПК АСУ.

Далее предполагается, что студент запустил ОС УПК АСУ, как это описано в учебно-методическом пособии [2], подключил выданную ему преподавателем рабочую область и вошел в систему от имени пользователя ***upk***.

1.1 Дистрибутив ОС УПК АСУ

Успешное выполнение лабораторных заданий во многом зависит от хорошего знания студентом структуры дистрибутива ОС УПК АСУ. Следующие пункты данного подраздела содержат всю необходимую информацию.

1.1.1 Структура ПО для проведения лабораторных работ

Полный комплект дистрибутива ОС УПК АСУ размещается в отдельном каталоге ***asu64upk*** корня файловой системы ОС MS Windows, известном как диск ***C:***. Такое размещение дистрибутива подробно описано в источнике [2, подраздел 1.1] и полностью соответствует архитектуре учебного ПО, установленного в учебных классах кафедры АСУ или на личных компьютерах студентов.

Студент должен убедиться в наличии на диске ***C:*** каталога ***asu64upk***, а также — структуры его файловой согласно данным таблицы 1.1.

Таблица 1.1 — Назначение компонент дистрибутива ОС УПК АСУ

Компонента	**Назначение компоненты**
boot	Каталог размещения ядра ОС и временной файловой системы.
opt	Каталог размещения дополнительного ПО, используемого в лабораторных работах.
themes	Каталог размещения рабочей области студента после подключения его личного архива.
upkasu	Каталог размещения ПО ОС УПК АСУ.
upk_asu.pdf	Файл учебно-методического пособия, который следует использоваться вместо источника [2].

Студент должен убедиться в наличии каталога ***asu64upk*** в корне файловой системы личного ***flashUSB*** и выполнить следующие действия:

- взять у преподавателя файл ***rsos-home.ext4fs.gz*** и поместить его на личный flashUSB в каталог ***/asu64upk/themes***;
- проверить загрузку ОС УПК АСУ с личного flashUSB;
- войти в систему пользователем ***asu*** и выполнить подключение файла ***rsos-home.ext4fs.gz*** к домашнему каталогу пользователя ***upk***;
- войти в систему пользователем ***upk*** и выполнить проверки, описанные в следующем пункте данной лабораторной работы.

Положительное завершение всех выполненных действий считается конеч-

ным результатом работ по данному пункту методического пособия.

1.1.2 Рабочий стол и инструменты рабочей области

На рисунке 1.1 показано изображение рабочего стола пользователя ***upk***, выполняющего лабораторные работы по дисциплине «*Распределенные сервис-ориентированные системы*» (РСОС). По общей традиции методики преподавания, на рабочем столе пользователя ***upk*** находятся значки:

- ***Учебный материал*** — ссылка на каталог с файлами учебных материалов по дисциплине (см. далее пункт 1.1.3);
- ***upk_asu.pdf*** — ссылка на обновленный вариант документа [2];
- ***Отчет.doc*** — ссылка на шаблон личного отчета студента;
- ***EclipseEE*** — значек запуска среды разработки Eclipse EE.

Рисунок 1.1 — Изображение рабочего стола пользователя upk

1.1.3 Состав учебного материала изучаемой дисциплины

Рабочая область пользователя ***upk*** содержит набор файлов учебного материала, размещенного в каталоге ***$HOME/Документы***. На рисунке 1.2 показан обязательный список учебных материалов по изучаемой дисциплине.

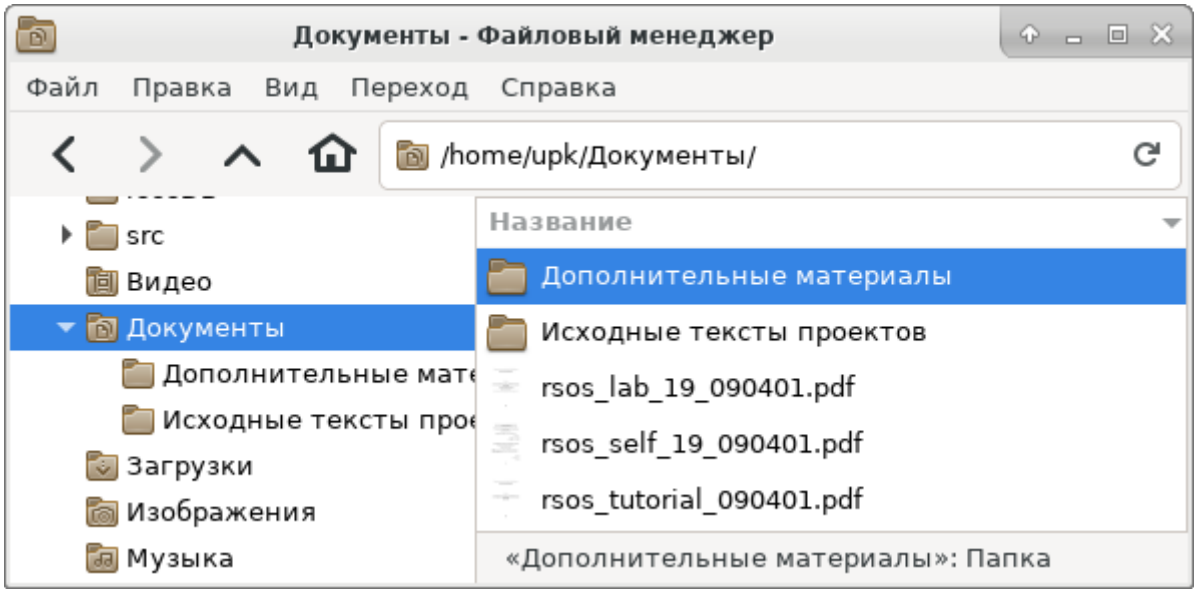


Рисунок 1.2 — Список учебных материалов по изучаемой дисциплине

К обязательным учебно-методическим пособиям относятся следующие файловые источники, составляющие УМО данной дисциплины:

– ***rsos_lab_090401.pdf*** — учебно-методическое пособие по выполнению лабораторных работ, которое сейчас читает студент;

– ***rsos_self_090401.pdf*** — учебно-методическое пособие по выполнению самостоятельной и индивидуальной работе студента, содержащий перечень вопросов по всем главам изучаемой дисциплины;

– ***rsos_tutorial_090401.pdf*** — основное учебное пособие по изучаемой дисциплине, обозначаемое в данном пособии как источник [1].

Каталог ***~/Документы*** содержит две директории, в которых расположены вспомогательные материалы:

– ***Дополнительные материалы*** — учебная литература, выбранная по усмотрению преподавателя;

– ***Исходные тексты проектов***, где расположены исходные тексты всех примеров учебного пособия [1] и описаний лабораторных работ.

1.2 ПО СУБД Apache Derby

Проект Apache Derby, появившийся в 1997 году, является примером СУБД полностью написанным на языке Java. Его характеризуют достаточно малые размеры, высокое быстродействие и возможность работы как в сетевом варианте, так и во встроенном режиме (режиме ***embedded***).

> Студенты, изучавшие бакалаврский курс «Распределенные вычислительные системы», уже владеют основами работы с этой СУБД. Следует повторить учебный материал подраздела 2.6 учебного пособия [3] или воспользоваться его аналогом в виде файла ***rvs_tutorial.pdf***, специально размещенного в отдельном каталоге ***~/Документы/Дополнительные материалы рабочей области студента***.

В данном подразделе рассматривается вариант выбора приемлемой версии дистрибутива Apache Derby, который согласован с используемой версией платформы языка Java. Также проводится минимальное тестирование полученного ПО в соответствии с ограничениями ОС УПК АСУ и задачами его дальнейшего использования.

На момент написания данного пособия, в текущей реализации ОС УПК АСУ использована версия 1.8 OpenJDK, что хорошо видно из сообщения команды ***java -version***, показанной на рисунке 1.3.

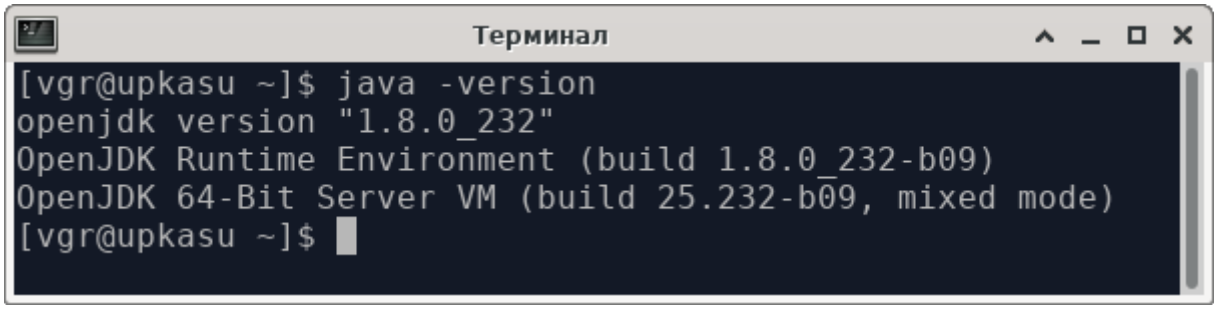


Рисунок 1.3 — Проверка версии платформы языка Java

> Обычно, разработчики сложных систем стараются использовать последние версии дистрибутивов программного обеспечения, поскольку они лучше согласуются с текущими и будущими версиями аппаратных средств ВМ и системных средств операционных систем.

1.2.1 Дистрибутивы Apache Derby

Поддерживаемые версии дистрибутивов СУБД Apache Derby размещены на его официальном сайте [4]. Перейдя на страницу Download, показанную на рисунке 1.4, мы увидим, что нашей версии платформы Java соответствует последняя версия 10.14.2.0 искомой СУБД.

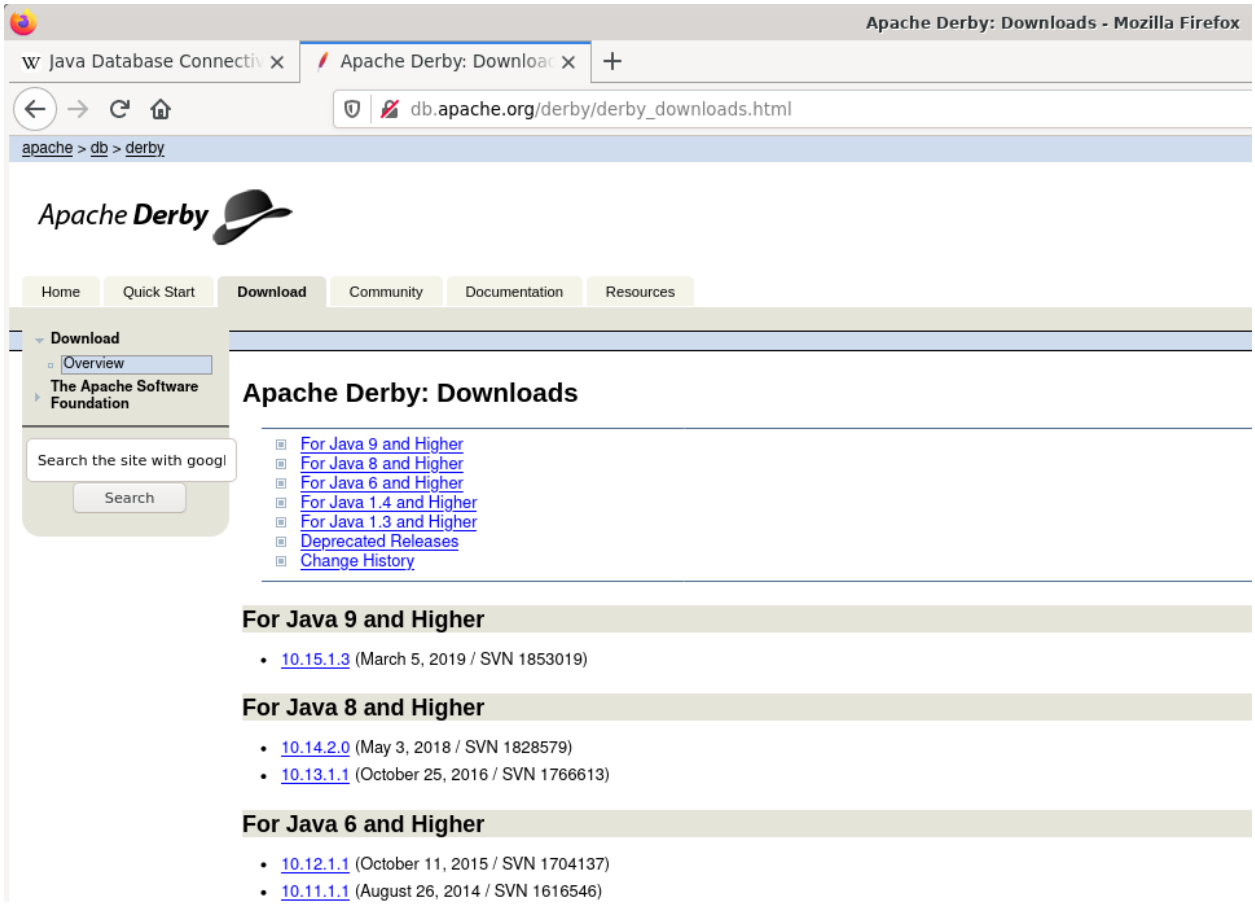


Рисунок 1.4 — Выбор дистрибутива СУБД на сайте Apache Derby

Перейдя по ссылке выбранного дистрибутива, скачиваем файл с именем ***db-derby-10.14.2.0-bin.tar.gz***, размещаем его в нужной директории инсталляции и распаковываем командой:

```
sudo tar -xvfz db-derby-10.14.2.0-bin.tar.gz
```

В результае указанных действий образуется каталог с соответствующим именем ***db-derby-10.14.2.0-bin***, который рассматривается как корневой каталог

СУБД и полный путь к нему должен содержаться в актуальной на момент запуска переменной среды ***DERBY_HOME***.

С целью оптимизации размера ПО и особенностей использования ОС УПК АСУ выполнены следующие преобразования распакованного дистрибутива СУБД Apache Derby:

- из каталога ***db-derby-10.14.2.0-bin*** удалена излишняя документация и примеры;
- содержимое каталога ***db-derby-10.14.2.0-bin*** преобразовано в файл с именем ***db-derby-10.14.sfs*** и структурой файловой системы ***squashfs***;
- файл ***db-derby-10.14.sfs*** размещен в каталоге ***opt*** (см. назначение и содержание таблицы 1.1).

1.2.2 Тестирование работы ПО СУБД Apache Derby

Тестирование установленного в ОС УПК АСУ дистрибутива СУБД Apache Derby проведем с целью достижения следующих целевых условий:

- дистрибутив должен быть доступен из каталога ***/opt/derby***;
- сетевой вариант запуска СУБД должен обеспечивать работу с базами данных, расположенными в каталоге ***$HOME/rsosDB***.

Для достижения поставленной цели, в рабочей области пользователя ***upk*** создан каталог ***$HOME/rsosDB***, в котором, как показано на рисунке 1.5, размещены необходимые сценарии для работы с этой СУБД.

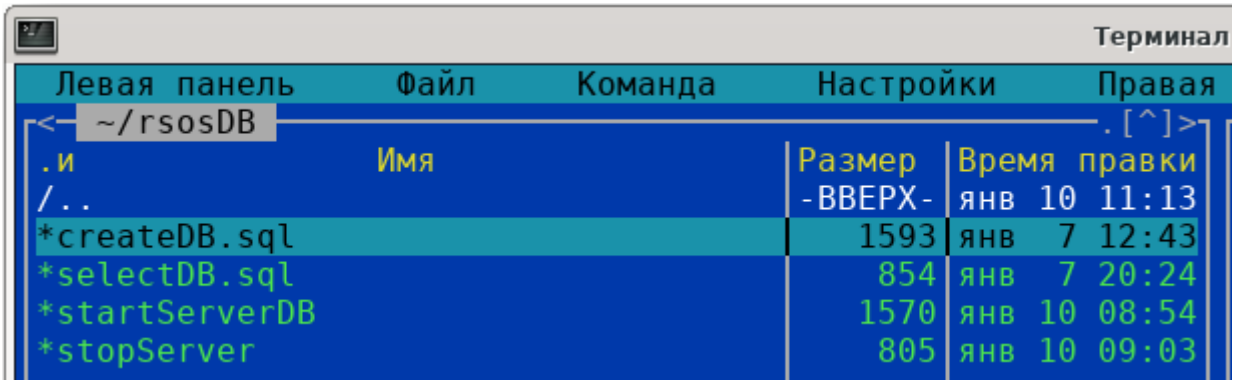


Рисунок 1.5 — Каталог размещения баз данных сетевого варианта запуска СУБД Apache Derby

Использование специальных сценариев обусловлена необходимостью монтирования дистрибутива СУБД к каталогу ***/opt/derby***, а также с особенностями его последующего сетевого варианта запуска.

Важное обстоятельство! Сетевой вариант работы СУБД Apache Derby обслуживает базы данных того каталога, из которого он был запущен.

Дополнительно. Необходимо настроить значения переменных среды ОС, инициируемых в файле ***$HOME/.bashrc***. Как это выглядит наглядно показано на рисунке 1.6.

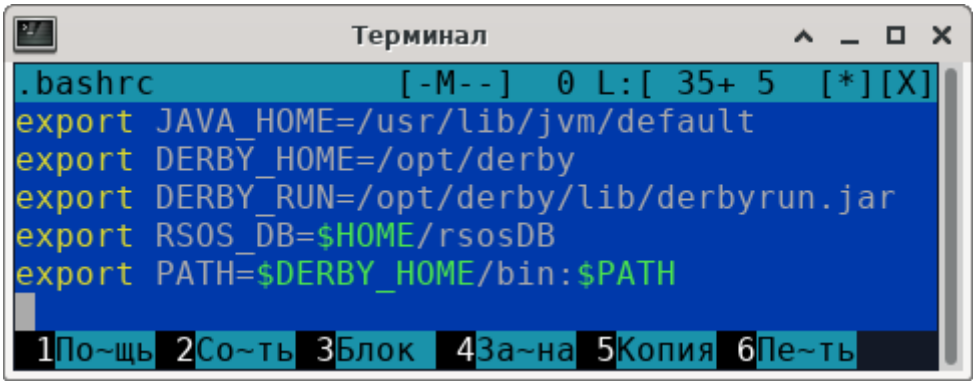


Рисунок 1.6 — Настройки переменных среды в файле .bashrc

Само тестирование проводится после перезапуска терминала в виде последовательности шагов, предполагающих выполнение следующих действий.

Шаг 1. Проверить наличие пустого каталога ***/opt/derby*** и файла с полным именем ***/run/basefs/asu64upk/opt/db-derby-10.14.sfs***.

Шаг 2. Из каталога ***$HOME/rsosDB*** запустить на выполнение сценарий ***startServerDB***, содержимое которого показано на рисунке 1.7. Этот сценарий проведет монтирование файла ***/run/basefs/asu64upk/opt/db-derby-10.14.sfs*** к каталогу ***/opt/derby***, если это не было сделано ранее, запустит по умолчанию сетевой вариант сервера на локальной ВМ (по адресу ***localhost***, порт ***1527***), выведет информацию о параметрах запуска и предложит проверить результат запуска.

Правильным результатом этого шага является факт монтирования ПО СУБД к каталогу ***/opt/derby*** и явное сообщение о запуске сервера.

Шаг 3. Из каталога ***$HOME/rsosDB*** запустить на выполнение сценарий ***createDB.sql*** командой:

```
ij createDB.sql
```

Правильным результатом шага 3 является размещение в используемом каталоге ***$HOME/rsosDB*** новой директории ***exempleDB***, что означает правильное создание тестовой базы данных.

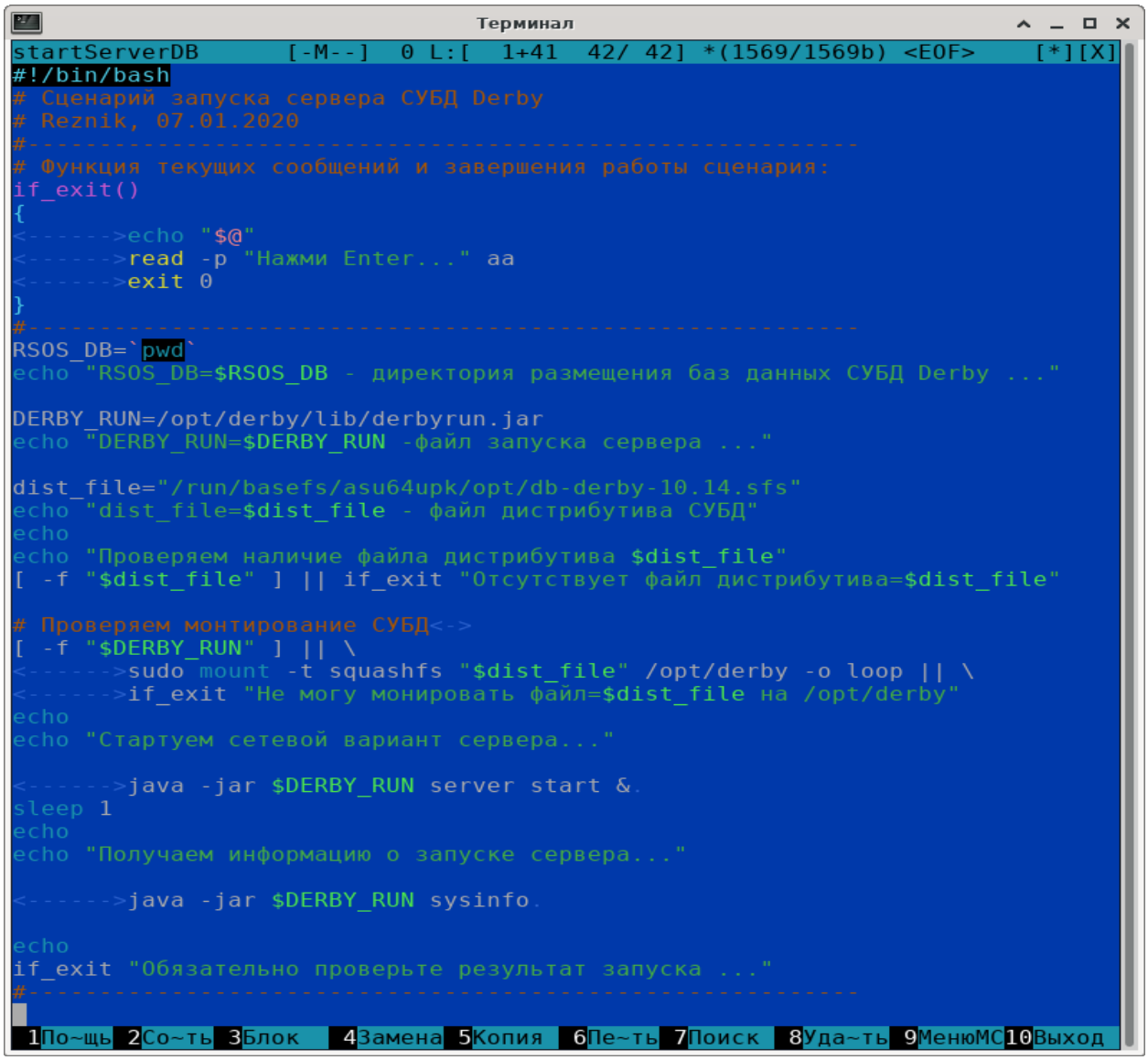


Рисунок 1.7 — Содержимое сценария startServerDB

Шаг 4. Из каталога ***$HOME/rsosDB*** запустить на выполнение сценарий ***selectDB*** командой:

```
ij selectDB.sql
```

Правильным результатом шага 4 является вывод на терминал содержимого таблицы, созданной на предыдущем шаге.

Шаг 5. Из каталога ***$HOME/rsosDB*** запустить на выполнение сценарий ***stopServer***, содержимое которого показано на рисунке 1.8. Этот сценарий проведет останов сервера СУБД Apache Derby, тем самым завершив последовательность команд тестирования.

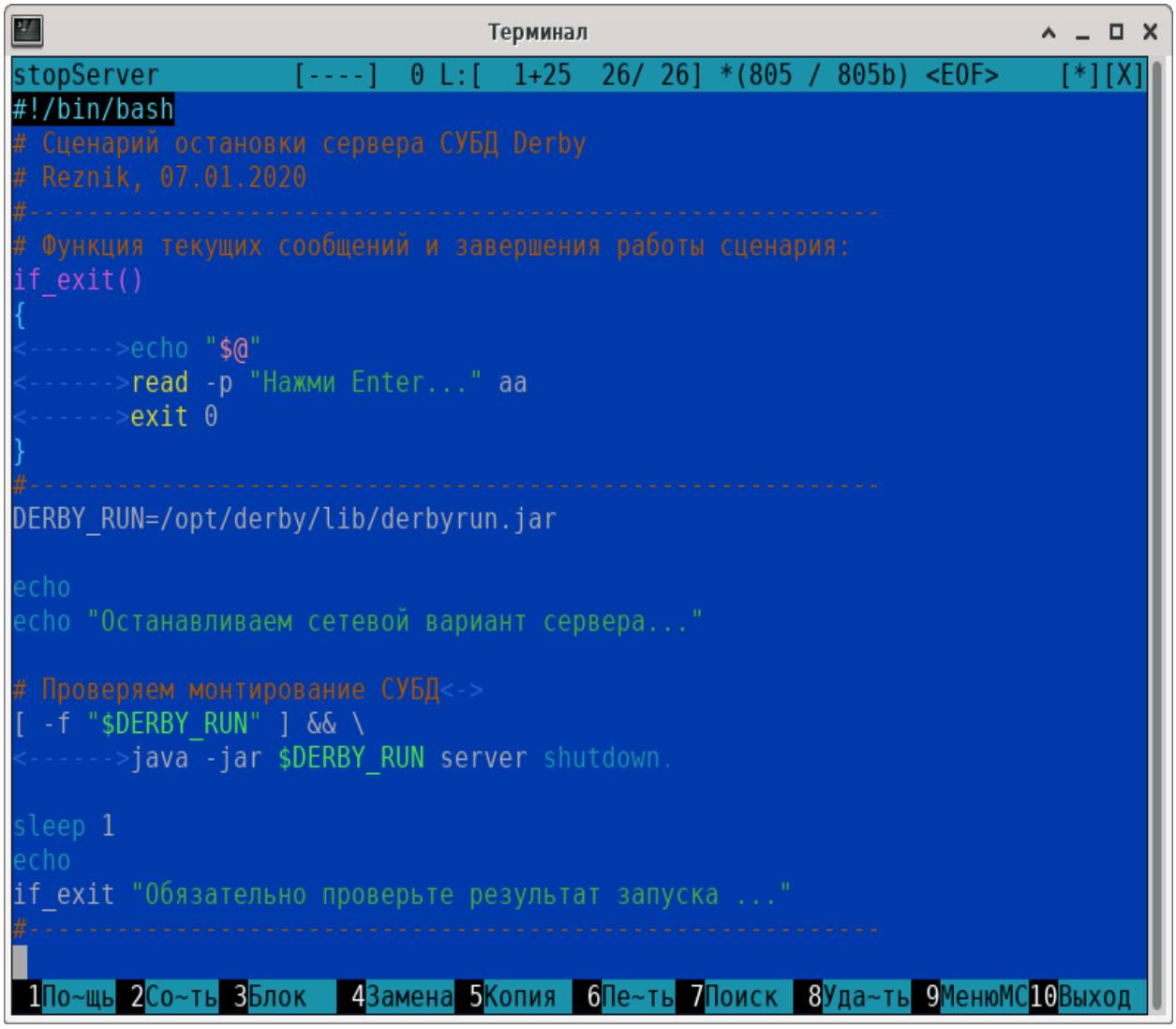


Рисунок 1.8 — Содержимое сценария stopServer

1.3 ПО сервера приложений TomEE

Сервер приложений Apache TomEE, официально появившийся в 2012 году, рассматривается как объединение проектов Apache Tomcat и платформы Java EE. Он представляет открытый вариант реализации технологии Web-сервисов, и выбран в качестве объекта изучения в данной дисциплине.

Студенты, изучавшие бакалаврский курс «Распределенные вычислительные системы», уже имеют знания по основам работы с системой Apache Tomcat. Следует повторить учебный материал подраздела 4.3 учебного пособия [3] или воспользоваться его аналогом в виде файла ***rvs_tutorial.pdf***, размещенного в каталоге ***~/Документы/Дополнительные материалы*** рабочей области студента.

В данном подразделе рассматривается вариант выбора приемлемой версии дистрибутива ***Apache Tomee***, который согласован с используемой версией платформы языка Java. Тестирование сервера ограничено только требованиями его запуска и остановки в среде ОС УПК АСУ. Дальнейшее тестирование завершается в следующей главе, где оно проводится совместно с ПО дистрибутива Eclipse EE.

Напоминание. На момент написания данного пособия, в текущей реализации ОС УПК АСУ использована версия 1.8 OpenJDK, что хорошо видно из сообщения команды ***java -version***, показанного ранее на рисунке 1.3.

1.3.1 Дистрибутивы ПО TomEE

Для доступа к дистрибутивам сервера следует перейти на официальный сайт Apache TomEE [5], а затем открыть вкладку ***Download***.

Страница с доступными дистрибутивами, отражающая только варианты версий 8.0.3, показана на рисунке 1.9. Ниже по странице доступны и другие версии, но мы выберем функционально наиболее полный вариант дистрибутива (***TomEE plume***) размером ***65 МБ***, доступный в виде сжатого файла ***apache-tomee-8.0.3-plume.tar.gz***.

Далее поступаем, как и ранее, с дистрибутивом Apache Derby. Распаковываем файл ***apache-tomee-8.0.3-plume.tar.gz*** командой:

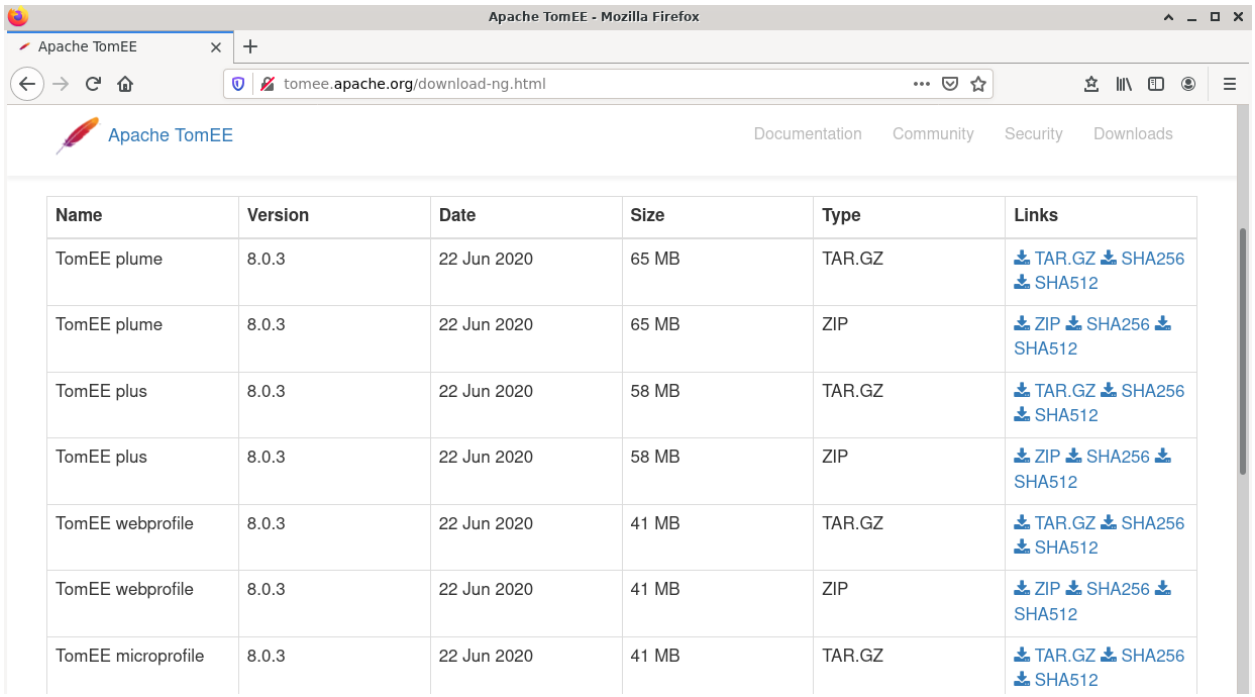
```
sudo tar -xvfz apache-tomee1-8.0.3-plume.tar.gz
```

Создаем новый файл дистрибутива Apache TomEE командой:

```
mksquashfs ./apache-tomee-8.0.3-plume \
    apache-tomee-plume-803.sfs
```

Помещаем файл ***apache-tomee-plume-803.sfs*** в каталог ***opt*** (см. назначение и содержание таблицы 1.1);

Создаем каталоги ***/opt/tomee*** и ***$HOME/tomee***, а затем переходим к пункту тестирования установленного ПО.



Name	Version	Date	Size	Type	Links
TomEE plume	8.0.3	22 Jun 2020	65 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE plume	8.0.3	22 Jun 2020	65 MB	ZIP	ZIP SHA256 SHA512
TomEE plus	8.0.3	22 Jun 2020	58 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE plus	8.0.3	22 Jun 2020	58 MB	ZIP	ZIP SHA256 SHA512
TomEE webprofile	8.0.3	22 Jun 2020	41 MB	TAR.GZ	TAR.GZ SHA256 SHA512
TomEE webprofile	8.0.3	22 Jun 2020	41 MB	ZIP	ZIP SHA256 SHA512
TomEE microprofile	8.0.3	22 Jun 2020	41 MB	TAR.GZ	TAR.GZ SHA256 SHA512

Рисунок 1.9 — Страница с дистрибутивами Apache TomEE

Запомните, для пользователя ***upk*** файл ***apache-tomee-plume-803.sfs*** видим как ***/run/basefs/asu64upk/opt/apache-tomee-plume-803.sfs***.

1.3.2 Настройка и запуск сервера Apache TomEE

В основе базовой технологии сервера Apache TomEE лежит контейнерная технология сервера Apache Tomcat. Фактически, сервер приложений TomEE и запускается как сервер Tomcat, активизируя прослушивание по умолчанию: порт ***8080*** и адрес ***localhost***.

> Полная документация по всем аспектам использования сервера Apache TomEE находится по адресу: http://tomee.apache.org/tomee-8.0/docs/.
>
> В корне дистрибутива находится файл ***RUNNING.txt***, который описывает все аспекты конфигурирования и запуска сервера. Им и воспользуемся при описании данного пункта.

При запуске сервера Apache TomEE (пока в режиме Tomcat) используются следующие основные переменные среды ОС:

1) **JAVA_HOME** — каталог дистрибутива JDK;
2) **JAVA_JRE** — каталог дистрибутива JRE;
3) **CATALINA_HOME** — каталог дистрибутива TomEE, являющийся обязательным параметром при запуске сервера (в нашем случае — это каталог ***/opt/tomee***, куда должна монтироваться файловая система файла ***/run/basefs/asu64upk/opt/apache-tomee-plume-803.sfs***);
4) **CATALINA_BASE** — каталог для «персональных» настроек дистрибутива, не являющийся обязательным параметром при запуске сервера (в нашем случае — это каталог ***$HOME/tomee***).

Добавим в файл ***$HOME/.bashrc*** переменные среды сервера, как это показано на рисунке 1.10.

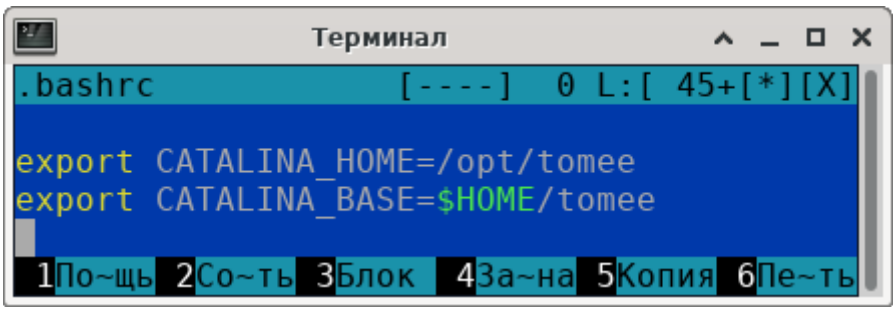


Рисунок 1.10 — Настройка переменных среды сервера Apache TomEE

Добавим в каталог ***$HOME/.bashrc*** сценарий ***mountTomee***, обеспечивающий монтирование дистрибутива сервера, как это показано на рисунке 1.11.

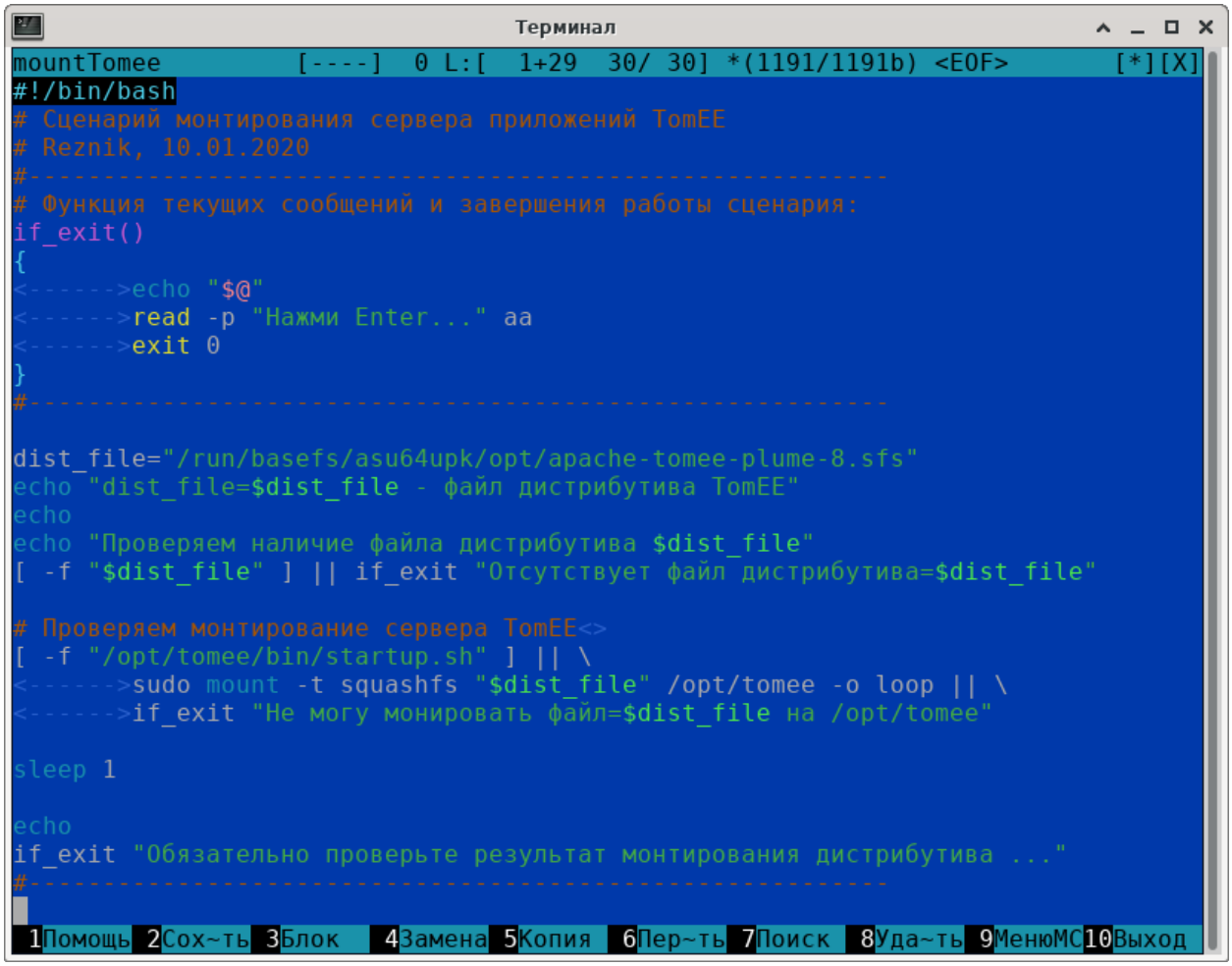


Рисунок 1.11 — Содержимое сценария ***mountTomee***, который монтирует дистрибутив сервера Apache Tomee в каталог ***/opt/tomee***

После запуска сценария ***mountTomee***, переходим в каталог ***/opt/tomee*** и убеждаемся, что дистрибутив сервера Apache Tomee подключен к среде ОС УПК АСУ, а на следующем шаге выполняем «персональную» настройку дистрибутива. Для этого, копируем из каталога ***/opt/tomee*** в каталог ***$HOME/tomee*** содержимое файлов, показанных на рисунке 1.12.

В каталоге ***$HOME/tomee*** следует создать пустую директорию ***lib***, которая нужна для возможных расширений ПО дистрибутива.

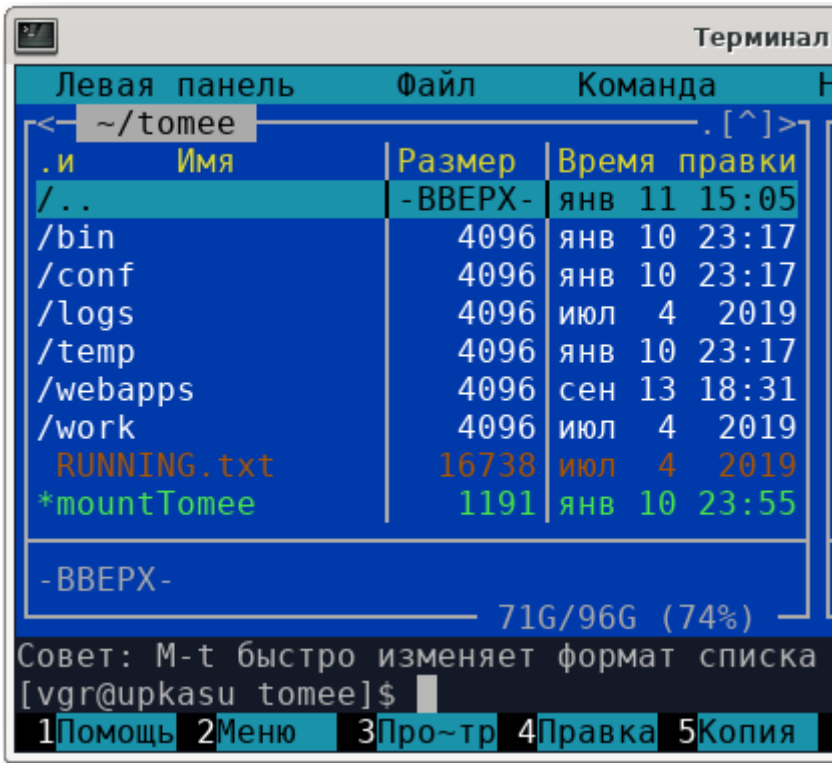


Рисунок 1.12 — Содержимое каталога «персональных» настроек дистрибутива сервера Apache TomEE

На данном этапе — все готово для запуска сервера. Если из каталога ***$HOME/tomee*** выполнить команду:

```
./bin/startup.sh
```

то сервер запуститься, как показано на рисунке 1.12.

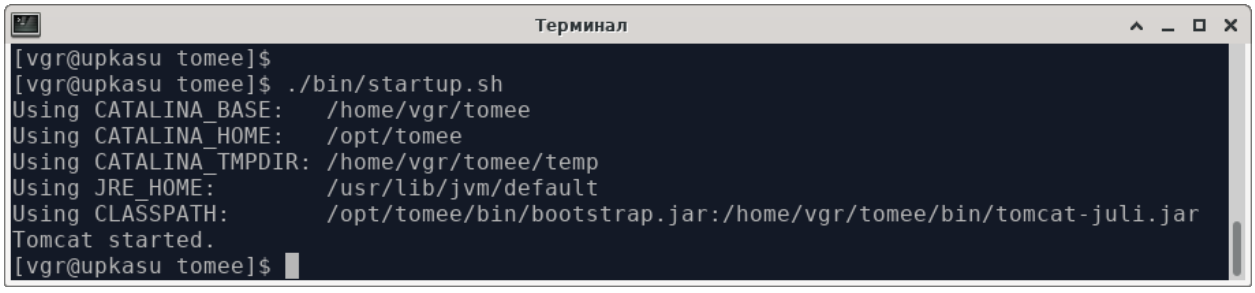


Рисунок 1.13 — Пример запуска сервера Apache TomEE

Хорошо видно, что при старте сервер показывает основные системные

переменные среды, с которыми он запущен.

Если теперь браузером подключитьсяя к адресу http://localhost:8080/, то появится базовая страница сервера Apache TomEE, показанная на рисунке 1.14.

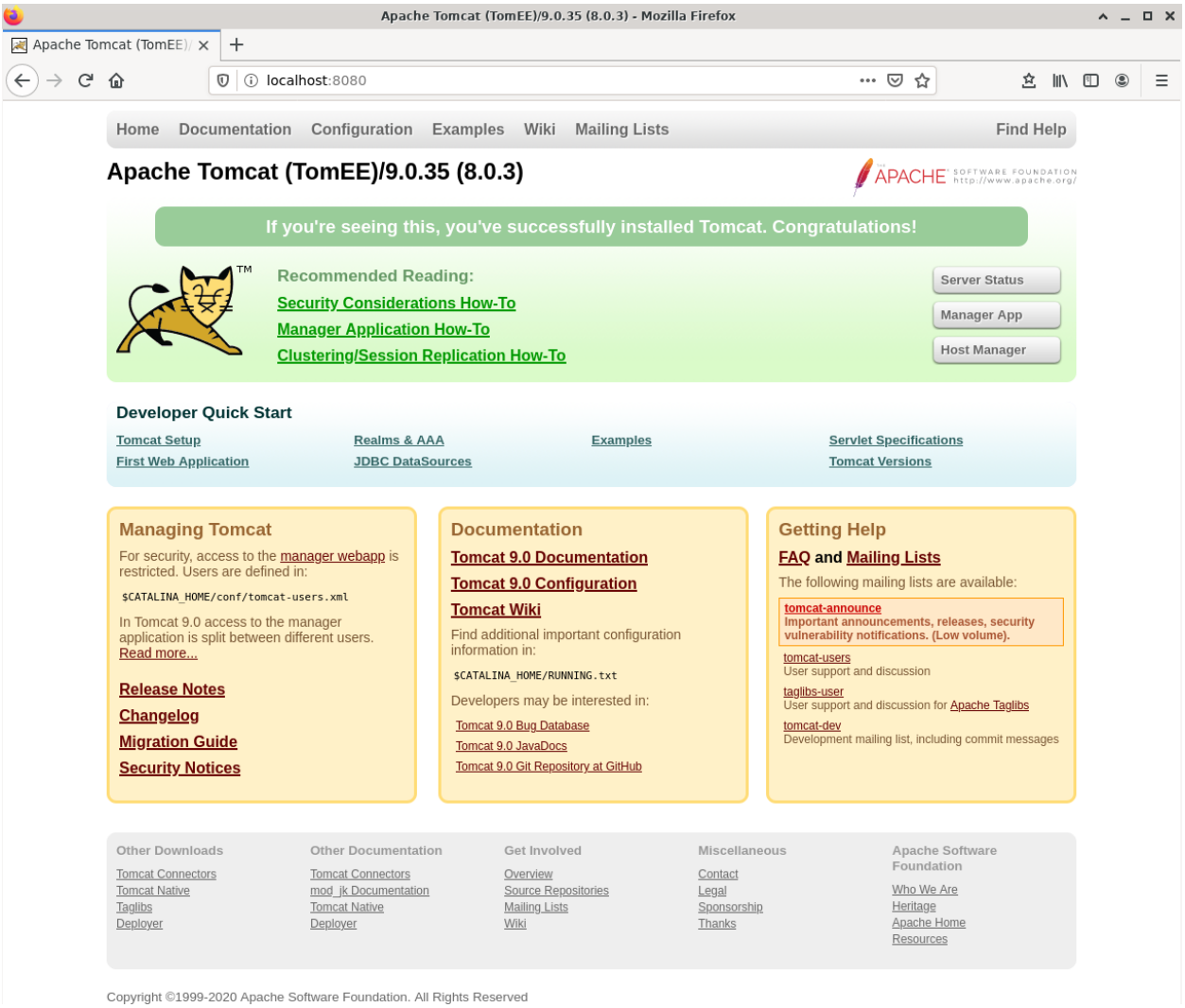


Рисунок 1.14 — Соединение с запущенным сервером Apache TomEE

Данная страница показывает, что сервер приложений Apache TomEE 8.0.3 основан на сервере Apache Tomcat версии 9.0.35. Зеленым цветом выделены ссылки на общие положения по безопасности и дополнительной конфигурации сервера. Завершение работы сервера можно выполнить командой:

```
./bin/shutdown.sh
```

В следующем подразделе проводятся дополнительные тесты сервера.

1.4 ПО IDE Eclipse EE

Появившись в 2001 году, интегрированная среда разработки (IDE) Eclipse была предназначена для разработки программного обеспечения на языке Java для корпорации IBM. С тех пор эта среда претерпела множество изменений и сейчас все разработки (проекты) этой IDE координируются некоммерческой организацией Eclipse Foudation. В частности, в среде ОС УПК АСУ инсталлирован проект ***Eclipse CDT***, предназначенный для программирования на платформах языков C/C++, что необходимо для проведения занятий по бакалаврскому курсу «*Операционные системы*».

Непосредственно для изучаемой дисциплины необходим проект IDE, известный как ***Eclipse EE*** (*Eclipse Enerprise Edition*), который поддерживает платформу языка ***Java EE*** (*J2EE*) и обеспечивает разработку систем РСОС. Этот дистрибутив IDE необходимо установить опционально (также как Apache Derby и Apache TomEE), чтобы он не конфликтовал с дистрибутивом Eclipse CDT.

1.4.1 Дистрибутивы Eclipse EE

Если зайти на официальный сайт Eclipse Foundation [6], то можно потеряться среди различных предложений по инсталляции проектов, что затрудняет нужный выбор. Поэтому лучше сразу перейти по нужному нам адресу https://www.genuitec.com/eclipse-packages/get/?pack=eclipse-jee&platform=linux64 и скачать предложенный файл (на середину 2020 года — ***eclipse-jee-2020-06-R-linux-gtk-x86_64.tar.gz***).

После преобразований, аналогичных описанным в предыдущих двух пунктах, полученный дистрибутив Eclipse EE станет доступен пользователю ***upk*** как файл ***/run/basefs/asu64upk/opt/eclipse-jee-2020.sfs***.

1.4.2 Тестирование ПО Eclipse EE

Перед запуском среды разработки этот файл преобразованного дистрибутива должен быть смонтирован в каталог ***/opt/eclipseEE***. Для этой цели, в каталог ***$HOME/bin*** помещен сценарий ***mountEclipseEE***, содержимое которого показано на рисунке 1.15. Этот сценарий выполняет все необходимые действия по монированию и запуску среды разработки.

После запуска сценария ***mountEclipseEE*** и монтирования дистрибутива, следует провести редактирование значка запуска среды разработки, как это показано на рисунке 1.16. После указанных действий, Eclipse EE запускается, как показано на рисунке 1.17.

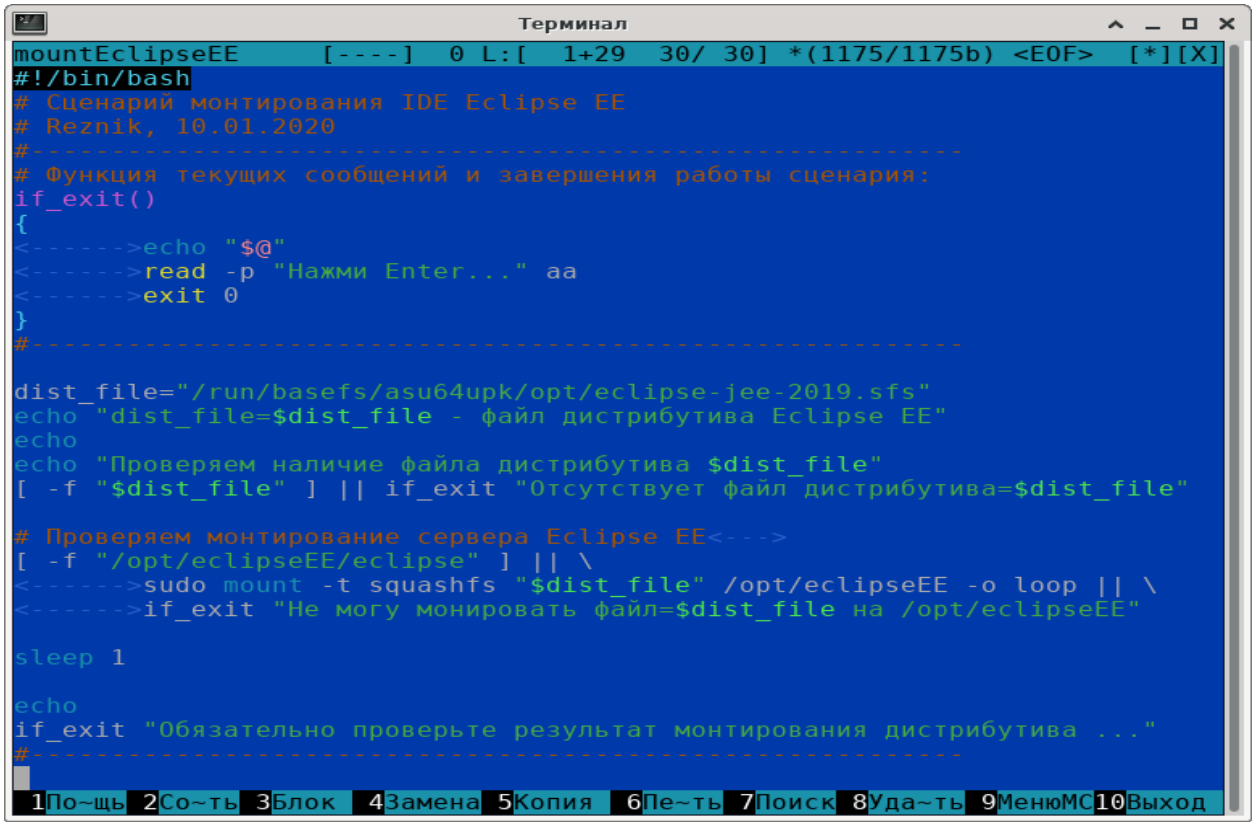


Рисунок 1.15 — Содержимое сценария ***mountEclipseEE***, монтирующий дистрибутив IDE Eclipse EE в каталог ***/opt/eclipseEE***

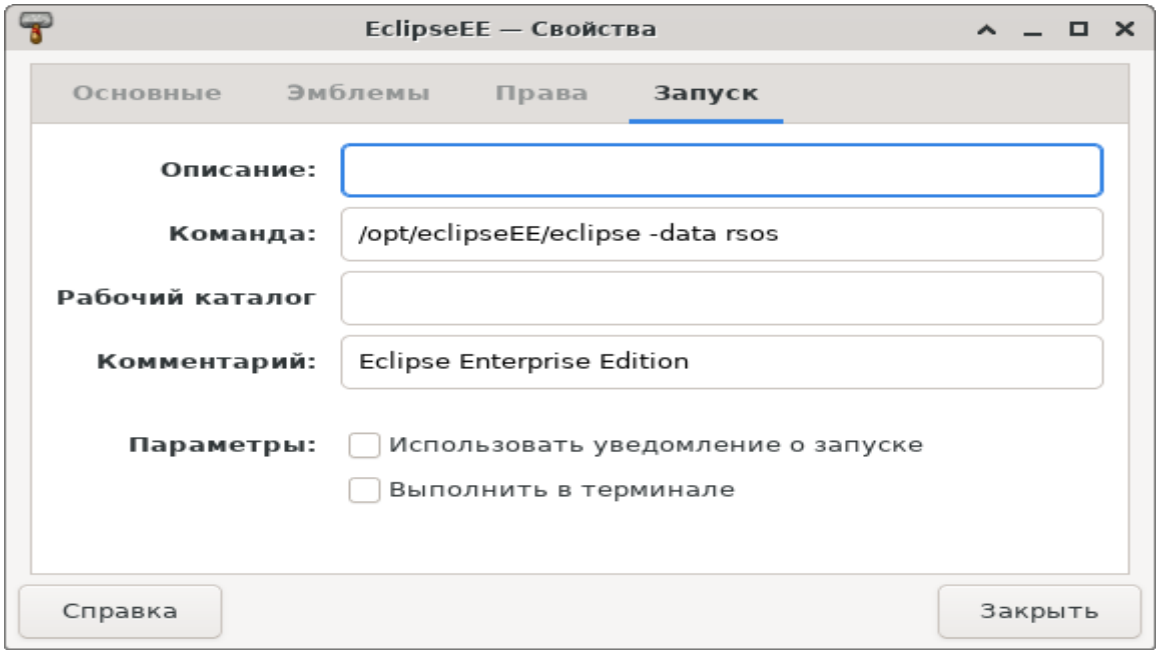


Рисунок 1.16 — Редактирование свойств значка EclipseEE

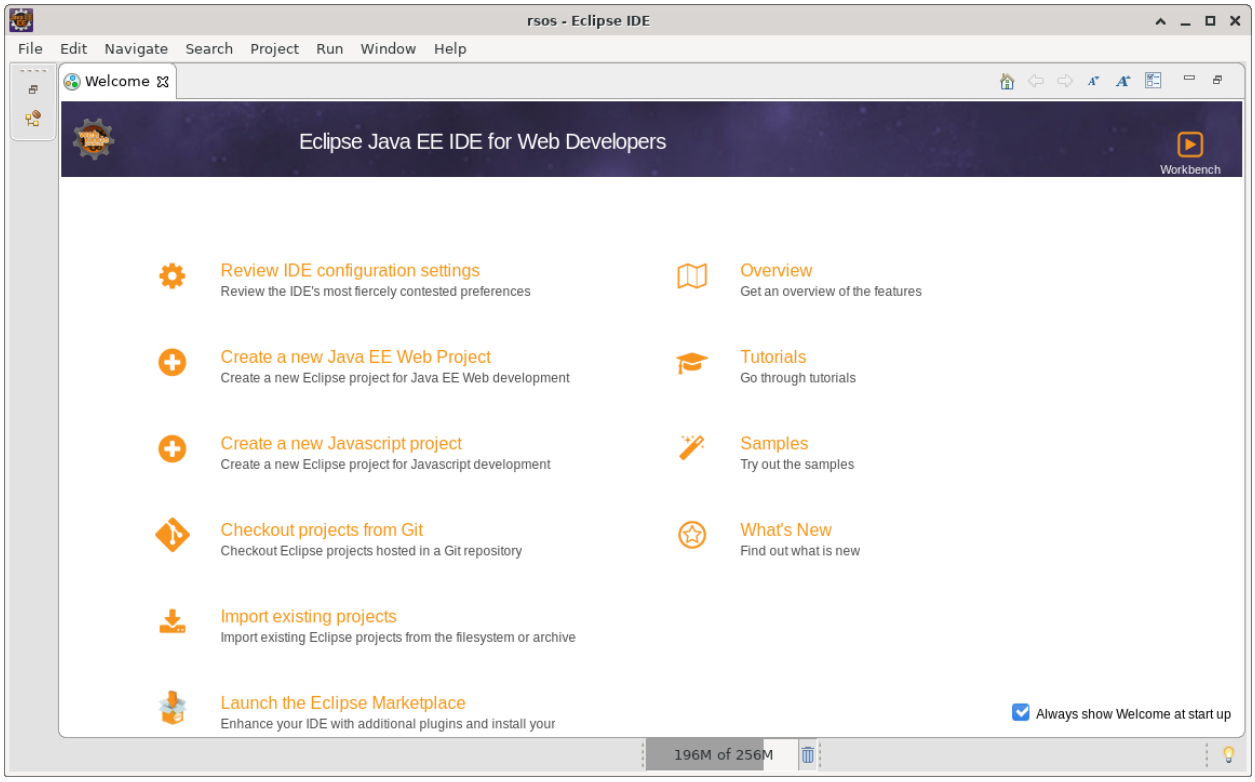


Рисунок 1.17 — Результат запуска IDE Eclipse EE

В изучаемой дисциплине Eclipse EE, как инструментальная среда разработки, предназначена для создания приложений, которые будут размещаться на сервере приложений Apache TomEE. Для этой цели необходимо провести тестирование взаимодействия этих систем, запустив простейший пример, который студент легко может создать.

В качестве примера рассмотрим проект с именем ***test***, содержащий сервлет с именем ***TestTomee***, который выполняет следующие функции:

1) принимет запросы программы-клиента (браузера) с помощью методов ***doGet()*** и ***doPost()***;
2) отвечает программе клиенту HTML-страницей, созданной JSP-сервлетом с именем ***test.jsp***.

Перед созданием проекта следует убедиться, что ПО сервера приложений подмонтировано к системе ОС, но сам сервер не запущен.

Другими словами, порты 8005, 8009, 8080 и 8443 ОС УПК АСУ должны быть свободными.

Создадим тестовый проект, выбрав на рисунке 1.17 ссылку «***Create a new Java EE Web Project***». Появиться стандартное окно создания проекта типа

«***Dynamic Web Project***», показанное на рисунке 1.18.

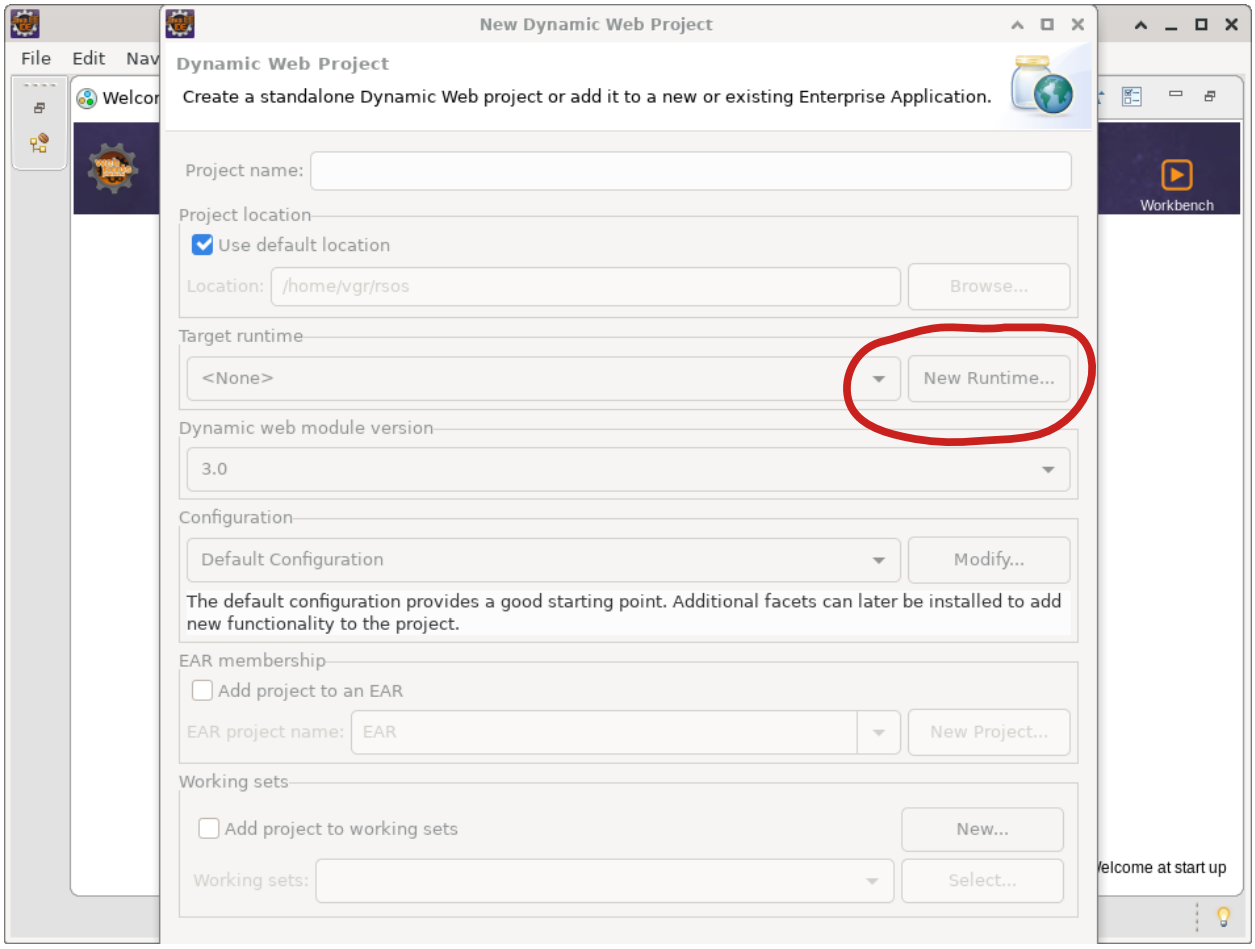


Рисунок 1.18 — Окно создания нового пректа типа Dynamic Web Project

В этом окне необходимо: ввести в поле «***Project name:***» имя проекта ***test***, а затем — нажать кнопку «***New Runtime...***».

В результате появится окно выбора версии сервера, который необходимо отметить, как показано на рисунке 1.19. Нажав кнопку «***Next >***», мы перейдем к окну установки переменных среды выбранного сервера, которые необходимо установить, как показано на рисунке 1.20.

В этом окне, для сервера Apache Tomcat создаются переменные среды ***JRE_HOME*** и ***CATALINA_HOME***. Что касается переменной среды «персо-нальных» настроек ***CATALINA_BASE***, то она будет указывать на каталог создаваемого проекта ***test***.

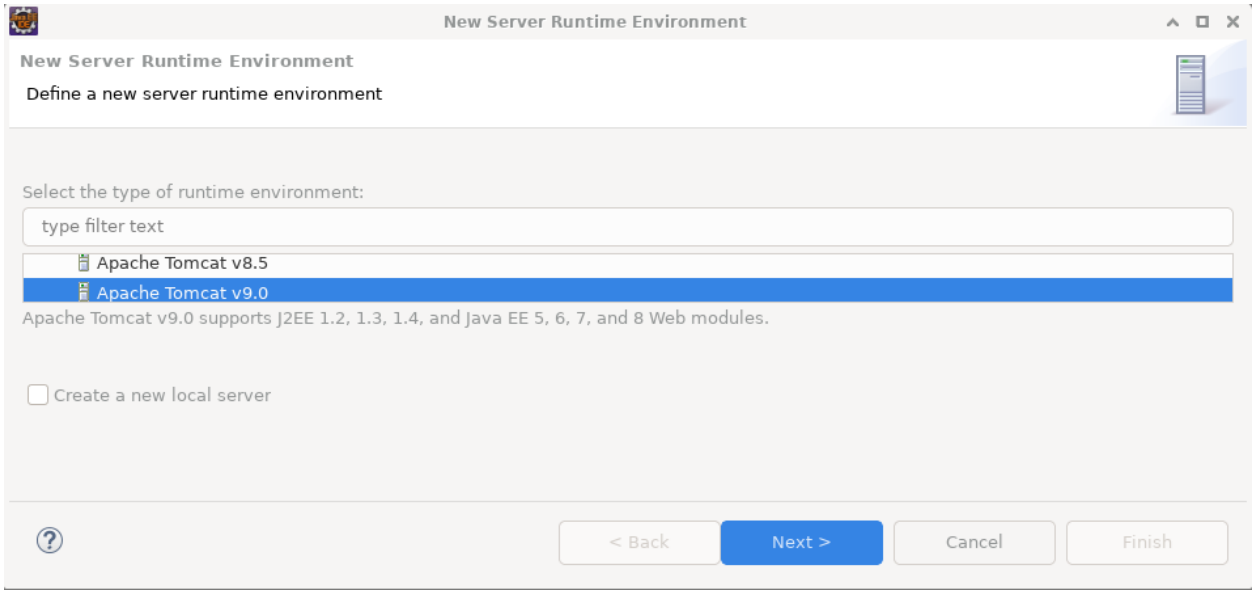


Рисунок 1.19 — Выбор версии сервера Apache Tomcat

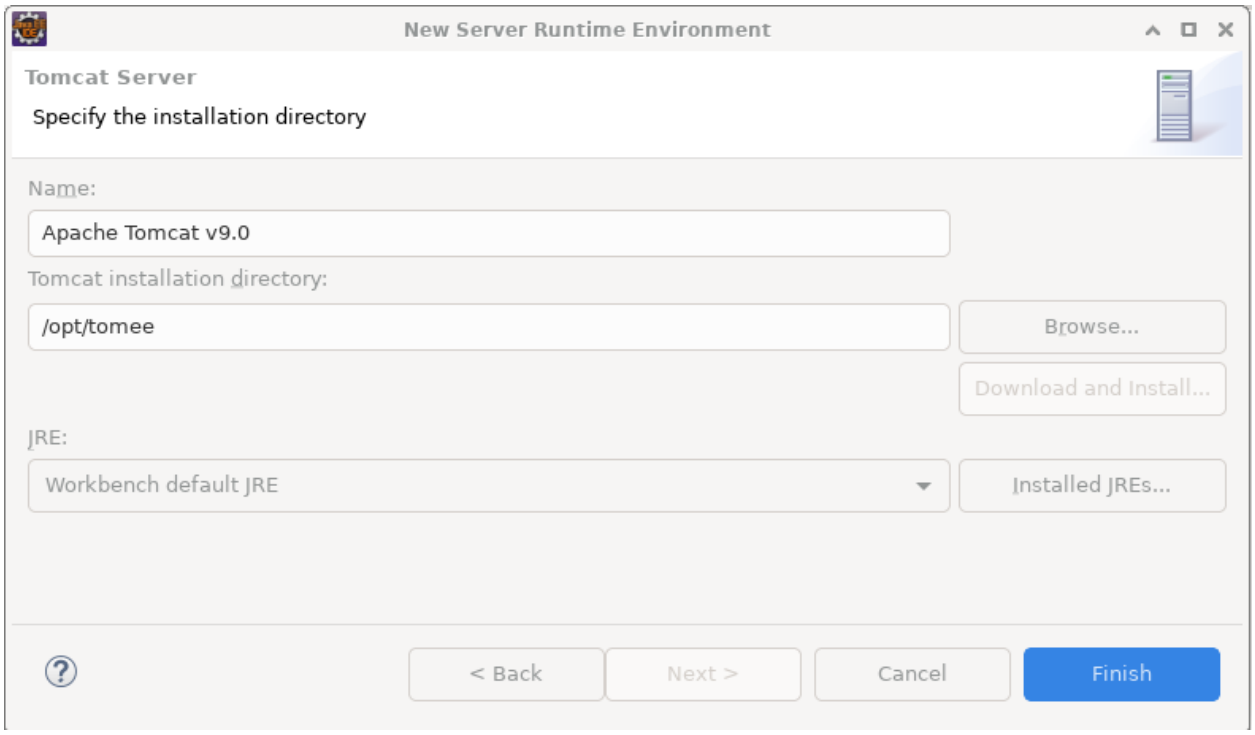


Рисунок 1.20 — Установка переменных среды сервера и ПО Java

Далее, нажимая кнопки «***Finish***» на всех окнах, мы завершаем создание проекта ***test***, как это показано на рисунке 1.21.

Заметим, что на рисунке 1.21 еще не отображен сервер Apache Tomcat, который появится после первого запуска проекта на исполнение.

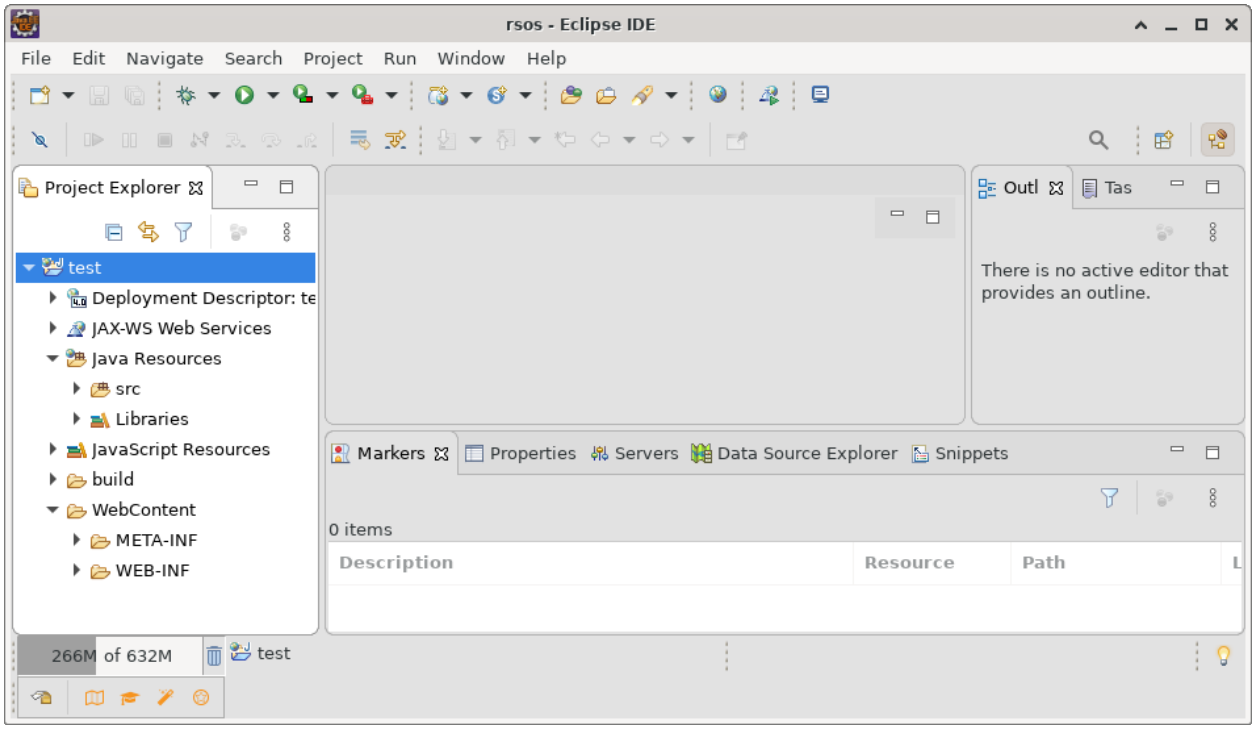


Рисунок 1.21 — Базовая структура проекта test

Столь подробное описание тестового проекта вызвано тем, что первый проект требует правильного задания местоположения и версии сервера.

При создании оследующих проектов будет предлагаться уже выбранный сервер Apache Tomcat.

Создание программного обеспечения проведем стандартным способом, который уже известен студентам по бакалаврскому курсу:

1) в проекте ***test*** выделим каталог ***src*** и создадим шаблон сервлета с именем ***TestTomee*** и именем пакета ***asu.rsos***; текст шаблона сервлета заменим содержимым листинга 1.1;

2) в проекте ***test*** выделим каталог ***WEB-INF*** и создадим шаблон JSP-страницы с именем ***test.jsp***; текст шаблона JSP-страницы заменим содержимым листинга 1.2.

Завершив создание исходного текста проекта, переходим на вкладку сервлета ***TestTomee.java*** и выполняем запуск проекта.

Результат запуска показан на рисунке 1.22. Студенту следует продолжить тестирование этого примера, вводя произвольный текст в соответствующее окно формы и нажимая кнопку «***Отправить***».

Листинг 1.1 — Исходный текст сервлета TestTomee.java проекта test

```
package asu.rsos;

import java.io.IOException;
import java.util.Date;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class TestTomee
 */
@WebServlet("/TestTomee")
public class TestTomee extends HttpServlet {
                private static final long serialVersionUID = 1L;

    // Разделяемый ресурс
    private String msgs = "";

    /**
     * @see HttpServlet#HttpServlet()
     */
    public TestTomee() {
        super();
        // TODO Auto-generated constructor stub
     }

      /**
       * @see HttpServlet#doGet(HttpServletRequest request,
       * HttpServletResponse response)
       */
      protected void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                              throws ServletException, IOException
      {
            /**
            * Явная установка кодировок объектов запроса и ответа.
            * Стандартная установка контекста ответа.
            */

            response.setCharacterEncoding("UTF-8");
```

```
            response.setContentType("text/html");
            /**
             * Стандартное подключение ресурса сервлета.
             */
            RequestDispatcher disp =
                  request.getRequestDispatcher("/WEB-INF/test.jsp");
                  disp.forward(request, response);
                  }

            /**
             * @see HttpServlet#doPost(HttpServletRequest request,
             * HttpServletResponse response)
             */
            protected void doPost(HttpServletRequest request,
                        HttpServletResponse response)
                              throws ServletException, IOException
            {
                  request.setCharacterEncoding("UTF-8");
                  // Сохраняем сообщение
                  Date dt = new Date();
                  msgs += dt.toString() + "<br>"
                        + request.getParameter("text") + "<hr>";
                  request.setAttribute("msgs", msgs);

                  // Переходим к методу doGet()
                  doGet(request, response);
            }
}
```

Листинг 1.2 — Исходный текст JSP-страницы test.jsp проекта test

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
      <head>
                  <meta charset="UTF-8">
                  <title>Тест сервера tomee</title>
      </head>
      <body>
            <hr>
                  <b>Вызван метод: <%= request.getMethod() %>;</b>
                  ID сессии:       <%= session.getId() %>
            <hr>
      <%
            if (request.getMethod().equals("POST")){
```

```
                out.println(request.getAttribute("msgs"));
            }
        %>

            <form action="TestTomee" method="post" accept-charset="UTF-8">
                <p> Введи текст: <br>
                    <textarea rows="5" cols="40" name="text"></textarea>
                </p>
                <p>
                    <input type="submit">
                </p>
            </form>
        <hr>
        </body>
</html>
```

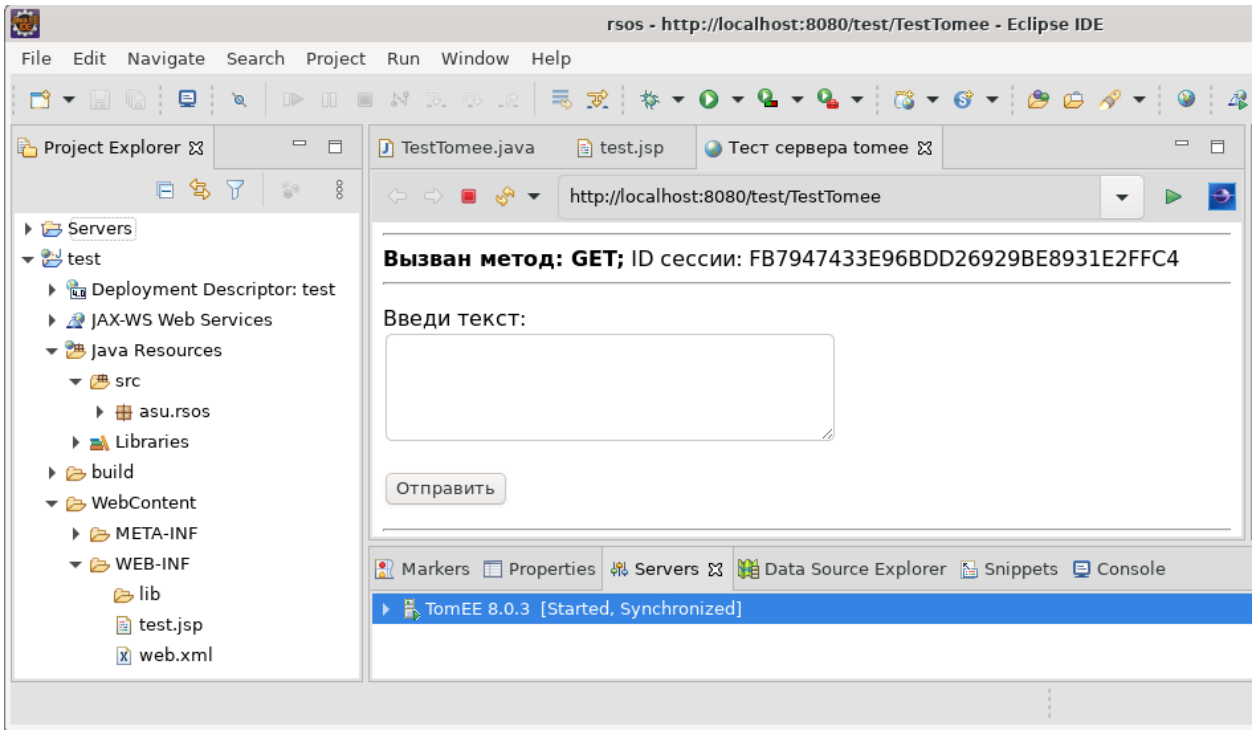


Рисунок 1.22 — Результат начального запуска сервлета TestTomee

Рисунок 1.22 показывает результат первоначального запуска сервлета ***TestTomee***. Обратите внимание, что приложение выводит метод, котрым сервлет обработал обращение к нему (метод GET), и номер сессии, которая присвоена обращению клиентской программы (браузера) к сервлету.

После ввода произвольного сообщения в поле текста формы и отправки сообщения серверу, сервлет сохранит переданное сообщение и отправит браузеру новый вариант формы, например, как показано на рисунке 1.23.

Обратите внимание, что в новой форме указан метод POST, которым сервлет обработал запрос, а номер сессии остался прежним.

Таким образом, мы проверили правильную установку и работоспособность взаимодействия Eclipse EE и сервера приложений Apache TomEE.

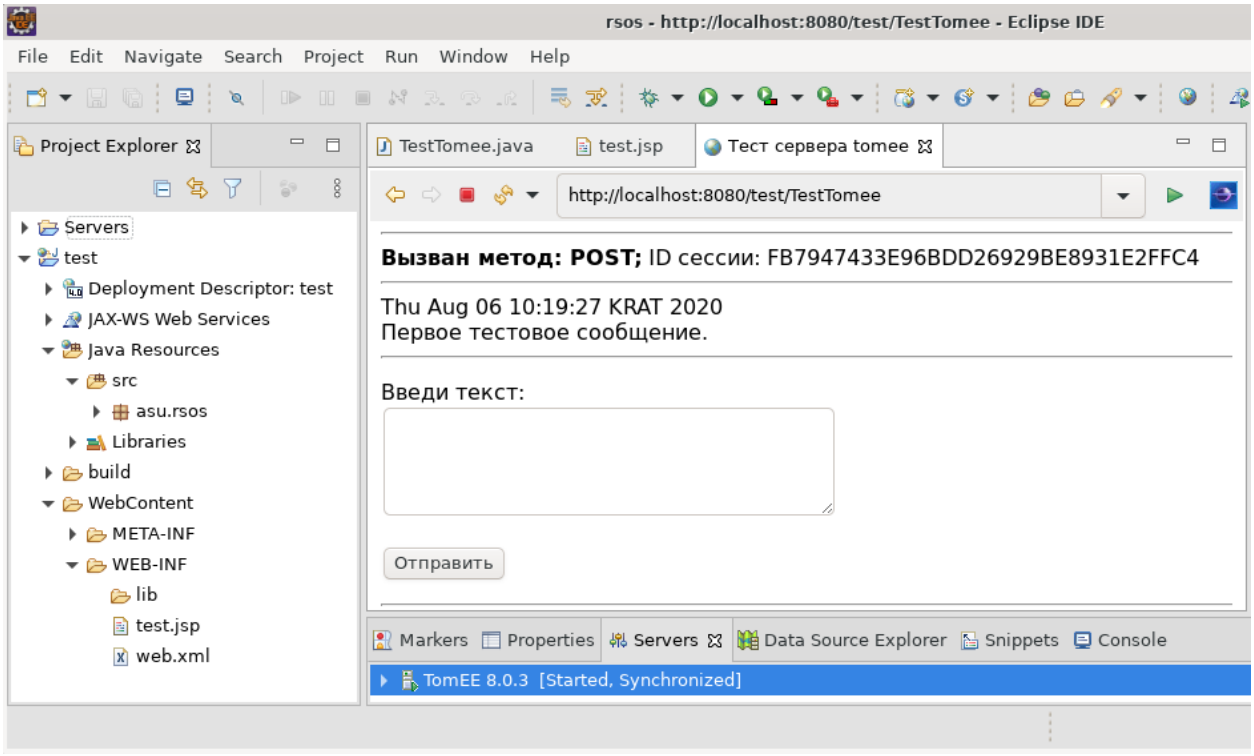


Рисунок 1.23 — Результат вывода сервлета после ввода сообщения

Завершив тестирование работы сервлета, студенту следует выполнить следующие операции:

1) запустить окно виртуального терминала, включая приложение Midnight Commander;
2) перейти в каталог ***$HOME/rsos*** и изучить структуру каталогов ***Servers*** и ***test***;
3) отразить результаты проделанной работы в личном отчете;
4) дополнительно сохранить файл отчета на личном flashUSB и быть готовым к проверке отчета преподавателем.

2 Работа 2. Использование компоненты JavaServer Faces

Лабораторная работа №2 посвящена практическому закреплению теоретического материала главы 2 «*Использование компоненты JSF контейнера Web*» учебного пособия [1]. В целом, этой теме посвящены две работы, полностью раскрывающие тематику ***JSF*** и тесно связанную с ней тематику ***CDI*** (*Context and Dependency Injection*).

Формальное название данной работы «*Использование компоненты JavaServer Faces*», которая построена на учебном материале первых трех подразделов главы 2 [1]:

- ***2.1*** — Web-сервис представления бизнес-информации;
- ***2.2*** — Шаблон проектирования MVC;
- ***2.3*** — Реализация тестового примера средствами JSF.

В такой последовательности и рекомендуется выполнять данную лабораторную работу.

2.1 Анализ проекта test средствами технологии JSF

Учебная цель первой части лабораторной работы — разобраться со структурой проекта типа ***Dynamic Web Project***, его файлами конфигурации (***дескрипторами развертывания***) и основными отличиями технологии ***JSF*** от технологии ***HTTP***-сервлетов.

Для этого следует:

1) изучить архитектуру проекта ***test***, реализованного в предыдущей лабораторной работе;
2) разобраться с назначением каталогов ***WebContent***, ***META-INF*** и ***WEB-INF***;
3) познакомиться и при необходимости отредактировать обязательный дескриптор развертывания приложения в виде файла ***web.xml***.

Завершив указанные действия, следует обратить внимание на шаблон проектирования ***MVC*** (***Model-View-Controller***), в котором контроллер, соответствующий HTTP-сервлету, программируется «*вручную*» программистом посредством реализации его методов ***doGet(...)*** и ***doPost(...)***. Далее, следует разобраться с различием и общностью форматов ***HTML*** и ***XHTML***, обеспечивающих представление отображаемой информации.

На этом подготовительную часть работы можно считать законченной.

2.2 Реализация Facelets-шаблона проекта labs

Учебная цель второй части лабораторной работы — практическое закрепление теоретических построений технологии JSF в проекциях на шаблон проектирования MVC и проектной структуры типа ***Dynamic Web Project*** для инструментальных средств Eclipse EE.

Достигается указанная цель посредством изучения студентом учебного материала второго подраздела главы 2 источника [1], где шаблон MVC интерпретируется рисунком 2.12 как:

1) ***FacesServlet*** — контроллер, управляемый сервером приложений;
2) ***компонент-подложка*** — интерфейстная часть модели приложения;
3) ***XHTML-ресурс*** — представление желаемой клиентом (*браузером*) информационной части технологии «*Клиент-сервер*», которая активно взаимодействует с моделью (*компонентом-подложкой*).

Обратите внимание, что в технологии JSF адресуется XHTML-ресурс, а в технологии HTTP-сервлетов — сам сервлет. Но в любом случае, HTTP-запросы перехватываются контроллером (*сервлетом*).

Дополнительно для практики являются важными: представления о жизненном цикле обработки запроса, наличие и методы доступа к контексту запроса ***FacesContext***, понятия JAVA-классов ***JavaBeans*** (*POJO*), областей действия и их аннотаций, средства представления с помощью технологии ***Facelets***, а также понятия и обозначения языка ***EL*** (*Expression Language*).

После освоения теоретического материала, студент создает в Eclipse EE проект типа ***Dynamic Web Project*** с именем ***labs***, который подготавливает для реализации приложения с помощью технологии JSF ***OmniFaces***. Как это делается — описано в пункте 2..2.5 источника [1].

Студент должен тщательно освоить подготовку шаблона проекта типа ***Dynamic Web Project*** для успешной реализации реализации технологии JSF применительно к разрабатывамым приложениям. Особое внимание следует уделить подготовку файлов дескрипторов развертывания ***context.xml***, ***web.xml*** и ***beans.xml***, без правильной реализации которых дальнейшая разработка приложений становится невозможной.

В завершении этой части лабораторной работы, следует создать и протестировать исполнение всех рекомендованных XHTML-файлов ***Facelets***-шаблона проекта.

Затем, можно переходить к выполнению третьей части работы.

2.3 Реализация тестового примера проекта labs

Учебная цель третьей части лабораторной работы — сугубо практическая. Она ориентирована на демонстрацию конкретных действий по применению инструментов JSF-технологи применительно к простейшему приложению, которое уже было реализовано в учебном материале предыдущей главы средствами технологии HTTP-сервлетов.

Исходным состоянием тестового примера является JAVA-POJO-класс проекта ***labs*** с именем ***TestTomee***, представленный на листинге 2.1 источника [1] как CDI-компонента-подложка с обрастью действия ***@ApplicationScoped***.

Полное описание процедуры реализации этого тестового примера приведено в позразделе 2.3 того же источника [1]. Эта процедура состоит из реализации девяти XHTML-файлов таблицы 2.5, составляющих ***Facelets-шаблон*** общего представления интерфеса приложения, который будет виден в окне браузера пользователю этого приложения.

Идейная часть предложенного Faces-шаблона отражает интерфейс изучаемой дисциплины, что показано на рисунке 2.20 источника [1], что демострирует потенциал возможностей самой JSF-технологии. В целом, этот шаблон является избыточным для рассматриваемого примера, но его реализация демонстрирует использование многих инструментальных средств, применяемых для широкого круга приложений. Технология создания таких шаблонов должна пополнить будущий багаж навыков студента.

Косвенно, представленный Facelets-шаблон учебной дисциплины демонстрирует технологию декомпозиции сложных приложений, в которых обозначаются отдельные прикладные задачи, уже интегрированные в общий интерфейс, но еще не реализованные программно. Такой подход позволяет реализовывать приложение последовательно, не искажая общий интерфейс его представления или включить элементы параллельной разработки всего проекта.

Прямая реализация тестового примера предполагает использование языка ***EL*** для разработки XHTML-ресурса ***lab3.xhtml***, который в качестве компоненты-подложки использует JAVA-класс ***TestTomee***. Как и какими средствами это делается, подробно описано в пункте 2.3.2 второй главы основного учебного пособия [1].

По результатам выполнения лабораторной работы студент должен оформить отчет, где описать результаты своих действий и отдельно обозначить отличия технологий реализации тестового примера средствами JSF и HTTP-сервлета.

3 Работа 3. Области действия технологии JSF

Лабораторная работа №3 является продолжением процесса закрепления практической тематики учебного материала главы 2 «*Компонент JSF контейнера Web*» учебного пособия [1].

Учебная цель работы — дальнейшее освоение студентами инструметальных средств технологии JSF, в плане проектирования функциональности типовых приложений, на области действия с разным жизненным циклом компонент-подложек.

Теоретический и практический материал для выполнения данной работы подробно описан в подразделе 2.4 методического пособия [1], а также основан на результатах выполнения лабораторной работы №2.

Идейная часть данной работы основана на реализации служебного функционала учебного приложения проекта ***labs***, дополняющего его двумя средствами:

1) ***авторизации пользователя*** в пределах всего приложения проекта;
2) ***возможностью переключения*** авторизованного пользователя между различными работами изучаемой дисциплины.

Указанные приложения должны обогатить студента навыками разработки приложений крупного масштаба и одновременно продемонстрировать возможности самой технологии JSF.

3.1 Учебная задача авторизации пользователя

В предыдущей лабораторной работе, компонента-подложка ***TestTomee*** использовала аннотацию ***@ApplicationScopped***. Это означает, что, когда сервер стартовал приложение проекта ***labs***, указанная компонента-подложка обрабатывает запросы от всех пользователей, которые подключились к этому приложению. В таком варианте, программист должен сам реализовывать средства разделения запросов между различными пользователями, включая все иные условия безопастности приложения.

Технология JSF предоставляет программисту различные области действия создаваемых им компонент, три из которых являются классическими и вполне обеспечивают программиста нужными инструментальными средствами, не заставляя его программировать уже реализуемые контейнером функции. Для таких областей действия и предусмотренны соответствующие три аннотации ***@RequestScoped***, ***@SessionScoped*** и ***@ApplicationScoped***. Чтобы наглядно показать их применение, студенту предлагается учебная реализация задачи авторизации пользователя в пределах приложения, которая является типичной служебной функцией большинства крупных проектов.

Общая схема взаимодействия браузера и компонент-подложек для нашей учебной задачи показана на рисунке 3.1.

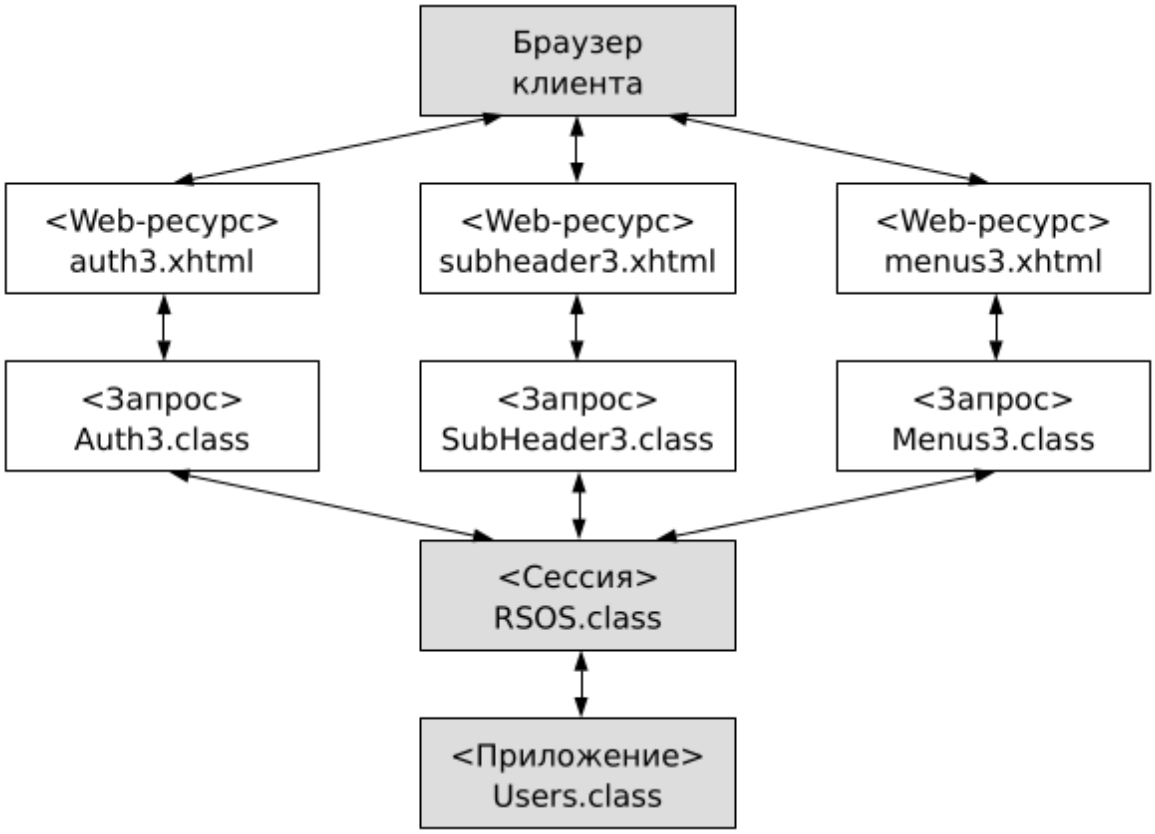


Рисунок 3.1 — Схема взаимодействия браузера и компонент-подложек JSF проекта labs (см. рисунок 2.26 [1])

Представленная на рисунке схема взаимодействия классов учебной задачи демонстрирует всю «*иерархитескую глубину*» любого типичного приложения, использующего технологию JSF. Такая схема обязательно должна быть создана, прежде чем программист начнет непосредственную реализацию самого приложения. Если это требование не будет соблюдено, то программист будет нарушать основную концепцию сервис-ориентированных технологий — построение слабосвязанных систем.

Чтобы полностью соответствовать парадигме сервис-ориентированных систем (***PCOC***), использующих слабосвязанную модель взаимодействия (***loose coupling***) между потребителями и поставщиками сервисов, необходимо тщательно проработать и закрепить функциональное назначение всех элементов представленной схемы, а затем на основе ее проводить реализацию приложения, двигаясь от нижних уровней к внешним. В такой последовательности и должна выполняться данная лабораторная работа, опираясь на полученные результаты уже реализованной части проекта ***labs***. Опишем указанные установки

в виде условной последовательности шагов.

Шаг 1. Реализация компоненты ***Users*** уровня ***@ApplicationScopped***.

Любое приложение должно иметь уровень, обеспечивающий его старт вместе с запуском сервера приложений. В нашей учебной задаче оно реализовано в виде класса ***Users***, в котором хранятся имена и пароли всего приложения в целом. В пункте 2.4.2 (см. [1, листинг 2.19]) описано как это делается.

Используемая информация хранится в коллектиции типа ***HashMap***, что предоставляет студенту возможность познакомиться с очень полезным инструментом программной платформы языка Java, обеспечивающим сохранение структурированной информации в памяти ЭВМ.

Конечно, информацию можно хранить в базах данных или извлекать из файлов, но в учебных целях используется минимально затратный вариант, демонстрирующий суть решаемой задачи.

Шаг 2. Реализация компоненты ***RSOS*** уровня ***@SessionScopped***.

Когда пользователь (с помощью браузера) соединяется с сервером приложений, сервер открывает сессию, которая остается открытой и после того, как пользователь (браузер) получил затребованный ресурс (HTML или XHTML-страницу). В результате создается не только иллюзия присутствия пользователя на сервере, но и реальная возможность сохранения на сервере индивидуальных данных пользователя, а также возможность индивидуального управления этой информацией. Таким образом, создание сессионной компоненты, предоставляет пользователю сервиса индивидуальное обслуживание, которое обеспечивается сервером приложений и не требует его отдельного программирования.

В нашем учебном примере, компонента ***RSOS*** сохраняет в коллекции типа ***HashMap*** имена доступных пользователю работ и связанные с этими работами имена XHTML-ресурсов, к которым пользователь принудительно возвращается после выполнения своих запросов. Дополнительно, этот компонент сохраняет:

1) имя авторизованного пользователя и выполняемую им работу;
2) объект типа ***Users***, методами которого проверяется авторизация пользователя или ее отсутствие.

Детали реализации этой компоненты описаны в пункте 2.4.3 (см. [1, листинг 2.20]).

Шаг 3. Реализация компоненты ***Auth3*** уровня ***@RequestScopped***.

Непосредственное решение отдельных задач пользователя обслуживаются компонентами-подложками типа запросов. Именно эти компоненты обычно обслуживают XHTML-ресурсы технологии JSF.

В нашем учебном примере, мы решаем задачу авторизации пользователя с помощью компоненты ***Auth3***, которая напрямую обслуживает HTML-ресурс в виде файла ***auth3.xhtml***. Как это делается, — описано в пунктах 2.4.4 и 2.4.5

(см. [1, листинги 2.21 и 2.22]).

Этим шагом мы реализовали заявленную функциональность учебной задачи. Далее переходим к ее использованию.

3.2 Переключение работ пользователя

Данная часть лабораторной работы посвящена использованию функционала авторизации пользователя.

Шаг 4. Реализация компоненты ***SubHeader31*** проекта ***labs***.

Данный шаг лабораторной работы призван наглядно показать возможность применения реализованного функционала задачи авторизации. Для этой цели проводится модификация компонента-подложки ***SubHeader31***, которая также имеет уровень ***@RequestScopped***.

Хотя данный пример можно отнести к «*украшательствам*» проекта ***labs***, тем не менее он реализует важную оформительскую цель, которая очень важна для приложений, осуществляющих диалог к конечным пользователем.

Как это делается, — описано в пункте 2.4.6 учебного пособия (см. [1, листинги 2.23 и 2.24]).

Шаг 5. Реализация компоненты ***Menus31*** уровня ***@SessionScopped***.

Данный шаг лабораторной работы делает проект ***labs*** еще более «*адекватным*» с точки зрения его прикладного использования, поскольку касается одного из вариантов реализации компоненты боковой панели.

Как это делается, — описано в пунктах 2.4.7 учебного пособия (см. [1, листинги 2.25 и 2.26]).

Шаг 6. Реализация компоненты ***Menus32*** уровня ***@SessionScopped***.

Данный шаг является завершающим для всей лабораторной работы. Он реализует второй вариант использования боковой панели учебной задачи. Данный вариант решения, как и проект ***labs*** в целом, будут использованы при изучении шестой темы дисциплины (см. лабораторная работа №9).

Как это делается, — описано в пунктах 2.4.8 учебного пособия (см. [1, листинги 2.27 - 2.29]).

Студенту следует выполнит все описанные шаги лабораторной работы, ***кроме шага 5***, который можно изучить только теоретически, как один из возможных вариантов реализации учебного примера.

4 Работа 4. Современные способы доступа к данным

Лабораторная работа №4 посвящена практическому закреплению теоретического материала главы 3 «*Современные способы доступа к данным*» учебного пособия [1].

Познавательная цель данной лабораторной работы совпадает с теоретической целью главы 3 — изучение альтернативных подходов к реализации достаточно масштабных приложений уровня предприятий, использующих язык SQL для доступа к одной или нескольким базам данных.

Практическая цель лабораторной работы — освоить достаточно современную технологию ***JPA***, которая является частью программной платформы Java EE и применяется для реализации прикладных компонентов ***EJB***, управляемых контейнерами серверов приложений.

Содержание и последовательность выполнения работы рекомендуется строить в соответствии с учебным материалом учебного пособия [1] и разделить на три части:

1) выполнить постановку прикладной задачи, открыть проект и подготовить соответствующую инфраструктуру для реализации проекта;
2) изучить теоретическую часть технологии ***JPA*** и реализовать простейший пример с использованием ***не-JTA***-транзакций;
3) реализовать пример с использованием ***JTA***-транзакций.

4.1 Постановка учебной задачи

В предыдущей работе, изучив технологию JSF, мы реализовали проект с именем ***labs***, реализацию которого в принципе можно наполнять материалом других лабораторных работ. Но мы откажемся от такого подхода до лабораторной работы №9, потому что он требует излишней реализации программного обеспечения, сопуствующего самой работе.

В данной лабораторной работе, мы будем использовать технологию HTTP-сервлетов, уделяя основное внимание ***EJB***-компонентам и технологии объектно-реляционного отбражения (***ORM***). В целом, данная часть работ основана на учебном материале подраздела 3.1 учебного пособия [1].

4.1.1 Учебная задача Letters

Выполнение лабораторной работы следует начинать и изучения пункта 3.1.1 учебного пособия [1].

Несмотря на краткость изложения этой части учебного материала, данный пункт содержит краткое описание двух классов ***Letter*** и ***Letters***, которые с рядом

модификаций используются во всех последующих главах изучаемой дисциплины.

4.1.2 Корпоративные EJB-компоненты

Эта часть выполнения лабораторной работы основано на материале пункта 3.1.2 учебного пособия [1]. В ней, на основании определения EJB-компоненты и используемых аннотаций:

1) создается новый проект типа Dynamic Web Project с именем ***lab4***;
2) создается формальное описание класса ***Letter***, предположительно сохраняющая данные в будущей базе данных;
3) создается EJB-компонента ***Letters***, реализующая удаленные и локальные интерфейсы;
4) создается и тестируется HTTP-сервлет с именем ***JpaServlet***, который демонстрирует способы работы с EJB-компонентами.

4.1.3 Варианы тестирования EJB-компоненты Letters

В этой части лабораторной работы, студент тестирует различные варианты применения EJB-компоненты ***Letters***, напрямую используя учебный материал пункта 3.1.3 учебного пособия [1].

4.1.4 Создание учебной базы данных

Данная часть работы посвящена созданию учебной базы данных в среде СУБД Apache Derby. Здесь студент использует учебный материал пункта 3.1.4 пособия [1]. В дальнейшем, созданная база данных данных будет использоваться во всех примерах данной дисциплины, предполагая сохранение в ней данных ассоциированных с классом ***Letter***.

Особое внимание студент должен уделить описанию файла ресурсов ***resources.xml*** и файлу ***persistence.xml***, представляющему дескриптор развертывания технологии ***JPA***. В дальнейшем, указанные файлы должны быть размещены во всех проектах, использующих указанную учебную базу данных.

В результате выполнения действий по всем четырем пунктам данного методического описания, у студента должны быть:

1) проект ***lab4*** со всеми дескрипторами развертывания;
2) реализованные классы ***Letter***, ***Letters*** и сервлет ***JpaServlet***;
3) база данных ***lab4db*** с учебной таблицей ***t_letter***.

4.2 Использование фабрики менеджера сущностей

Вторая подраздел лабораторной работы основан на учебном материале подраздела 3.2 учебного пособия [1].

Успешное выполнение этой части работы требует реализации трех последовательных проектных действий:

1) ***подготовку сущности*** (*сущностей*) адекватно отображаемых в базе данных;
2) ***обеспечение доступа*** к менеджеру сущностей;
3) ***использование менеджера сущностей*** для доступа к инструментальным средствам технологии ORM и отображения результатов, например, в окне браузера.

4.2.1 Создание сущности Letter

Используя учебный материал пунктов 3.2.1 и 3.2.2 учебного пособия [1], студент должен разобраться с понятием сущности и усвоить назначение аннотаций @Entity, @Table, @GeneratedValue, @Column, @Temporal, @Transient и @Id.

Указанные аннотации должны быть примененны к POJO-классу ***Letter***, с целью адекватного отображения его в структуру таблицы ***t_letter*** уже созданной в предыдущей части работы базы данных ***lab4db***.

4.2.2 Освоить технологии менеджера сушностей

Следует внимательно изучить учебный материал пункта 3.2.3 учебного пособия [1], в котором кратко описан класс ***javax.persistence.EntityManager***. Правильное использование этого класса во многом зависит от используемой СУБД и доступного инструментария сервера приложений.

В целом, доступность и функционирование менеджера сущностей опирается на два файла: файл ресурсов ***resources.xml*** и дескриптор развертывания ***persistence.xml***.

Первый из них описывает параметры местоположения используемой базы данных и строку соединения с ней, включая имя и пароль пользователя.

Второй — ***persistence.xml*** содержит информацию о типе используемых транзакций для самого менеджера сущностей и имеет ссылку на файл ресурсов.

В общем случае, эта часть технологии JPA требует изучения специальных источников информации, ориентированных на применение конкретной СУБД. Студенту предоставляется уже подготовленная информация, ориентированная на использование СУБД Apache Derby.

В этой части работ, студенту также следует изучить методы мененджера сущностей, покрайней мере в объеме, представленном в таблице 3.2 учебного пособия [1].

4.2.3 Использование не-JTA-типа транзакций

Для ORM-программирования взаимодействия EJB-компонент платформы Java EE используется два типа транзакций, определяемые в дескрипторе развертывания технологии JPA — файл ***persistence.xml***:

1) ***не-JTA-тип***, определяемый как «***RESOURCE_LOCAL***»;
2) ***JTA-тип***, определяемый как «***JTA***».

Считается, что инструментарий транзакций ***не-JTA***-типа реализован для большего количества СУБД и серверов приложений. Тем не менее, он менее удобен для программиста, так как требует инкапсуляции в EJB-компоненту фабрики менеджера сущностей. Дополнительно также требуется «*ручное управления*» самими транзакциями.

Студенту рекомендуется теоретически изучить использование этой технологи по учебному материалу пункта 3.2.4 учебного пособия [1].

4.3 Использование контекста менеджера сущностей

Этот подраздел лабораторной работы №4 является завершающим. Он полностью основан на учебном материале подраздела 3.3 пособия [1].

Учебная цель этой части работ — создание прототипа EJB-компоненты, использующей ORM-отображение сущности ***Letter*** на таблицу ***t_letter*** базы данных ***lab4db***, которую далее можно было бы использовать в темах, непосредственно описывающих технологии объектно-ориентированных систем (темы 5 и 6). Работа этого подраздела выполняется в два этапа:

1) ***изучение*** методов запроса к данным типа ***Criteria API***;
2) ***реализация*** сущности, EJB-компоненты и сервлета, демонстрирующих функциональность технологии JPA.

4.3.1 Методы запросов типа Criteria API

По содержимому пункта 3.3.1 источника [1], студенту необходимо познакомиться с пятью типами запросов, используемых технологией JPA. Затем следует изучить основную схему ***Criteria***-запросов и их методов, включающих восемь этапов формарования и обработки запросов к хранилищам объектных данных. После этого можно приступить к выполнению работы.

4.3.2 Реализация и исследование примера технологии JPA

Реализация демонстрационного примера опирается на уже созданную сущность ***Letter***, адекватно обеспечивающую ORM-отображение данных в таблицу ***t_letter*** базы данных ***lab4db***.

В качестве EJB-компоненты используется класс ***Lets2***, который является модификацией EJB-компоненты ***Letters***, но использует тип транзакции ***JTA***. Подобные изменения сделаны для того, чтобы студент мог изучить все варианты технологии ***JPA*** и не запутаться в содержимом JAVA-классов.

При выполнении данной части работы, необходимо сначала реализовать EJB-компоненту ***Lets2***, в которой менеджер сущностей инкапсулируется в объект с помощью аннотации ***@PersistenceContext***(name = "***lab4-unit2***"). Это сразу делает доступным менеджер сущностей для всех методов класса и указывает контейнеру сервера приложений использовать тип транзакций ***JTA***.

Обратите внимание (см. [1, листинг 3.24]), что теперь программисту нет необходимости получать доступ к менеджеру сущностей через фабрику, а также самостоятельно открывать и закрывать транзакции. Все эти действия автоматически выполняются контейнером сервера приложений.

Прочитав пункт 3.3.2 источника [1], студент может убедиться, что EJB-компонента ***Lets2*** полностью реализует все пять методов для работы с сущностью ***Letter***, причем все запросы к хранилищу информации выполняются только на языке Java.

Далее, на основе учебного материала (см.[1, пункт 3.3.3]), необходимо выполнить следующие работы:

1) реализовать JSP-страницу ***jpa_test2.jsp***, согласно листингу 3.25;
2) реализовать HTTP-сервлет ***JpaServlet2***, согласно листингу 3.26.

В результате указанных действий, студент получает полностью реализованное демонстрационное приложение, показывающее наиболее оптимальную стратегию реализации EJB-компонент, использующих различные СУБД.

Методом запуска проекта ***lab4*** студенту необходимо убедиться и продемонстрировать работу учебного приложения, а затем изложить полученные результаты в своем личном отчете.

После указанных действий, лабораторная работа №4 считается законченой и можно переходить к следующей.

5 Работа 5. Представление информации с помощью XML

Лабораторная работа №5 посвящена только одной технологии JAXB, которая обеспечивает работу с документами в формате XML на языке Java. Она основана на учебном материале подраздела 4.1 главы 4: «Обработка XML и JSON» учебного пособия [1].

Учебная цель данной лабораторной работы — изучение технологии JAXB для преобразования объектов JAVA-классов в формат представления XML и обратно.

Методически, работу рекомендуется разбить на две части, каждая из которых решает свою частную задачу:

1) первая часть — ***технология JAXB*** ориентирована на закрепление навыков описания и оформления классов Java для подготовки их к преобразованиям в формат представления XML и обратно;
2) вторая часть — ***примеры реализации технологии JAXB*** исключительно посвящена тестированию и исследованию проведенных преобразований.

В организационном плане, практическая часть работы выполняется в отдельном проекте с именем ***lab5***. Такой подход позволяет использовать и преобразовывать исходные тексты уже известных студенту классов, не искажая полученные результаты предыдущих работ.

5.1 Инструментальные средства технологии JAXB

Java Architecture for XML Binding (*JAXB*), как и любая подобная технология, основывается на двух функциональных возможностях:

1) ***маршалинге*** JAVA-объектов в формат документа XML;
2) ***демаршалинге*** из документа XML обратно в JAVA-объект.

5.1.1 Классы и методы технологи JAXB

В этом пункте лабораторной работы изучению подлежат три основных класса технологии JAXB: ***JAXBContext***, ***Marshaller*** и ***Unmarshaller***. Обычно их достаточно для простейших операций по преобразованию объектов сериализуемых классов языка Java в форматы XML-файлов и обратно. Краткое описание этих классов и их методов изложено в базовом учебном пособии (см. [1, пункте 4.1.1]).

Обратите внимание, что технология JAXB имеет вполне самостоятельное значение, которое не обязательно привязано к технологии сервис-ориентированных систем. Например, очень часто объекты сериализуемых классов используются для хранения данных, например, в файлах. Учитывая это обстоятельство,

студенту рекомендует повторить инструментальные средства ввода/вывода языка Java, содержащиеся в базовом программном пакете ***java.io***.

5.1.2 Аннотации технологии JAXB

Инструментальные средства JAXB имеют достаточно развитые средства представления объектов языка Java в формате XML. За основу такого представления берется содержимое классов, представленное в виде байт-кодов. Дополнительно, для изменения имени самого класса и имен его полей используется технология аннотаций.

Список базовых аннотаций и их назначение приведены в пункте 4.1.2 учебного пособия [1], а примеры их применения показаны на основе уже известного студентам класса ***Letter***.

Обратите внимание, что аннотации технологии ***JAXB*** являются вполне совместимыми в использовании с аннотациями технологии ***JPA***.

5.2 Примеры реализации технологии JAXB

Примеры использования технологии JAXB представлены и описаны в пункте 4.1.3 учебного пособия [1]. Студенту следует реализовать все эти примеры в среде проекта ***lab5***. Для этой цели рекомендуется использовать листинги 4.1 — 4.5, отображающие один из вариантов реализации этих примеров.

Особое внимание следует уделить сериализации списков объектов. Данная задача решается созданием специально ориентированных для этого классов. Например, класс ***ListLetters***, представленный на листинге 4.4 учебного пособия [1] предназначен для представления в формате XML списка объектов типа ***Letter***.

Поскольку данная лабораторная работа имеет исключительно познавательный характер, студент может придумать свои классы и воспользоваться их преобразованиями для демонстрации усвоенного материала.

Результаты проведенной студентом работы должны быть обязательно отражены в его личном отчете.

6 Работа 6. Представление информации с помощью JSON

Лабораторная работа №6 продолжает практическое закрепление учебного материала по способам представления объектов сериализованных классов, изложенных в четвертой главе «*Обработка XML и JSON*» учебного пособия [1, подраздел 4.2].

Учебная цель данной лабораторной работы — изучение технологии JSON, как альтернативного подхода, стремящегося устранить имеющиеся недостатки технологии JAXB.

Методически, данная работа опирается на учебный материал пункта 4.2.1 учебного пособия [1] и выполняется в отдельном проекте с именем ***lab6***, чтобы устранить возможные зависимости между использованными ранее описаниями классов ***Letter*** и ***ListLetters***.

Общую последовательность выполнения данной работы рекомендуется разделить на изложенные далее три части.

6.1 Инструментальные средства JSON-P

Вся технология JSON ориентирована на упрощение результатов представления сериализованной информации для эффективной передачи ее между взаимодействующими сторонами. В этом ее преимущество и, соответственно, — недостатки:

1) используя только два структурных типа — ***объекты*** и ***массивы***, удается избежать большого объема пределений тегов, присущих технологии JAXB;

2) использование только четырех примитивных типов данных — ***число***, ***строка***, ***двоичное значение*** и ***null***, приводит к необходимости применять дополнительные их преобразования, что, в конечном итоге, ложится на плечи программистов и, естественным образом, порождает ошибки.

В целом, используя технологию JSON, программист должен очень хорошо представлять себе структуру преобразуемых объектов, возможно предварительно представив ее в формате XML.

Выполняя эту часть лабораторной работы, которая использует объект типа ***ListLetters***, студенту следует:

1) разобраться с содержимым листинга 4.5 (см. [1, пункт 4.2.1]);

2) реализовать тестовые приложения ***Test1*** и ***Test2*** (см. [1, пункт 4.2.2]).

6.2 Представление данных на уровне преобразуемых классов

Используя технологию JSON, программист сталкивается с необходимостью удовлетворения двух противоречивых требований:

1) максимально упростить представление и объем сериализуемых данных;
2) обеспечить надежность правильного представления как передаваемых данных, так и надежность правильного восстановления объектов из принятых данных.

Независимо от того какую задачу решает программист, ему необходимо хорошо знать стуктуру передаваемых и принимаемых объектов. К сожалению инструментарий платформы Java EE не предоставляет для этого удобных средств, с пмощью которых программист мог бы легко обнаружить допущенные ошибки.

Одним из средств, облегчающих работу программиста, является дополнение используемых классов методами, позволяющими обеспечивать форматированное представление результатов сериализации передаваемых данных. Этому вопросу и посвящена вторая часть лабораторной работы.

Студенту следует на основе описания классов ***Letter*** и ***ListLetters***, реализовать примеры, представленные в пункте 4.2.3 учебного пособия [1]. Приведенный в этом пункте подход имеет гораздо большее значение, чем просто частное решение в рамках технологии JSON, поэтому необходимо его подробно изложить в своем личном отчете.

6.3 Сравнительный анализ технологий JAXB и JSON

Выполнив задания первых двух частей данной лабораторной работы, студент должен завершить изучение темы 4 сравнительным анализом подходов, представленных технологиями JAXB и JSON.

Желательно, выполнить краткий анализ по всем предыдущим лабораторным работам, каждая из которых предоставляла свои инструментальные средства используемые в распределенных сервис-ориентированных системах.

Считается, что на данном этапе студент полностью готов к реализации простейших сервис-ориентированных систем, которые будут им выполняться в последующих трех работах основанных на учебном материале глав 5 и 6 учебного пособия [1].

7 Работа 7. Классические средства описания Web-сервисов

Лабораторная работа №7 является первой частью учебной работы, которая посвящена практическому закреплению теоретического материала главы 5 учебного пособия [1]: «*Web-службы SOA*».

Приступая к выполнению данной работы, студент должен четко понимать, что основное отличие сервис-ориентированных систем от других подходов, использующих парадигму «*Клиент-сервер*», состоит в принципиальном разделении поставщиков сервисов и потребителей сервисов. Это разделение касается как инструментальных средств, используемых каждой из сторон, так и целевой установкой на полученный результат.

Учебная цель данной работы — создание учебной Web-службы, которая реализуется инструментальными средствами ***поставщиков сервисов***.

Методически, данная работа разделена на две части, которые студент должен выполнить последовательно, согласно учебному материалу подразделов 5.1 и 5.2 главы 5 учебного пособия [1]:

1) подготовку инструментальных средств, необходимых для реализации Web-службы SOAP;
2) непосредственную реализацию учебной Web-службы на базе инструментальных средств программной платформы Java EE.

7.1 Инструментальные средства Web-служб SOAP

Теоретическая концепция Web-служб SOAP основана на классическом развитии инструментальных средств распределенных программных систем, ведущих свое начало от технологий объектого подхода. Парадигма такого подхода опиралась на создание некоторого промежуточного программного обеспечения (***middleware***), которое берет на себя роль посредника между поставщиком и потребителем Web-служб, одновременно выполняя и обязанности реестра Web-служб.

7.1.1 Теоретические составляющие Web-служб SOAP

Простейшим реестром Web-служб стала система ***UDDI*** (*Universal Description Discovery & Integration*), предоставляющая взаимодействующим сторонам две возможности:

1) язык ***WSDL*** (*Web Service Description Language*), обеспечивающий описание интерфейсов Web-сервисов обслуживаемых поставщиком;
2) протокол ***SOAP*** (*Simple Object Access Protocol*), обеспечивающий прямое взамодействие поставщика и потребителя Web-сервиса.

Подраздел 5.1 источника [1] дает краткое описание указанных выше средств и связанных с ними определений, а также указывает программные пакеты платформы Java EE, необходимые для реализации этой технологии.

7.1.2 Инфраструктура учебного примера Web-службы

Поставщик сервиса, реализуя Web-службу, которая будет по его мнению интересна ***потребителю сервиса***, решает две задачи:

1) проектирует и реализует функционал некоторого приложения, которое предполагается использовать в сети ЭВМ;
2) реализует сервлет Web-службы, который является контейнером созданного приложения и одновременно сам управляется контейнером сервера приложений.

Первая задача всегда предполагает некоторое уникальное решение, для реализации которого могут использоваться все возможные технологии программирования, но, в пределах нашей дисциплины, мы именуем ее как ***EJB-компоненту***.

Технология Web-служб SOAP позволяет реализовывать приложения уровня предприятия, ориентированные на произвольный масштаб. И хотя этот масштаб имеет значение с позиции потребляемых приложением ресурсов, он принципиально не влияет на концепцию решения второй задачи.

Чтобы не создавать дополнительные трудности, порождаемые уникальностью первой задачи, воспользуемся уже готовым решением EJB-компоненты, рассмотренной в главе 3 учебного пособия [1] и реализованной в проекте ***lab4***, (см. более подробно [1, подраздел 3.3] и результаты лабораторной работы №4 данного пособия).

Более конкретно, в данной работе используются результаты разработки учебного приложения, основанного на двух классах:

1) классе ***Letter*** преобразованного в сущность, отображаемую в базе данных ***lab4db***;
2) классе ***Lets2*** представляющего EJB-компоненту, реализующую методы доступа и работы с сущностью класса ***Letter***.

Таким образом, рассматриваемый в данной работе учебный пример Web-сервиса, будет использовать инструментальные средства не только сервера приложений Apache TomEE, но и СУБД Apache Derby.

Вторая задача, которая и является основной целью данной лабораторной работы, имеет достаточно универсальный инструментарий своего решения и описана в следующем подразделе данного пособия.

7.2 Создание учебной Web-службы SOAP

Процесс создания учебной Web-службы SOAP подробно описан в подразделе 5.2 главы 5 учебного пособия [1], которого и следует придерживаться при выполнении данной части лабораторной работы.

В целом, студент должен концептуально ориентироваться на три этапа реализации данной части работы:

1) подготовку отдельного проекта в инструментальной среде Eclipse EE;
2) непосредственное создание Web-службы;
3) исследование Web-службы.

7.2.1 Подготовка проекта lab7 в среде Eclipse EE

Использование отдельного проекта обеспечивает адресное разделение классов, уже использованных в других проектах (см. [1, пункт 5.2.1]):

1) класс ***Letter*** переносится без всяких изменений, кроме имени пакета, которое принимает значение ***rsos.lab7***;
2) файлы ***derbyclient.jar*** и ***resources.xml*** не содержат изменений;
3) в файле ***pesistence.xml*** изменяется путь к классу ***Letter*** (см. [1, листинг 5.3,]);
4) класс ***Lets2*** переносится с новым именем ***Lets7***, именем пакета и описанием интерфейса ***RemoteLets*** (см. [1, листинги 5.4 и 5.5]).

7.2.2 Непосредственное создание Web-сервиса Lets7

Непосредственное создание Web-сервиса можно проводить разными способами, например:

1) для опытных программистов, хорошо знающих WSDL, рекомендуется сначала описать интерфесы на этом языке;
2) для болших проектов, использующих множество классов-сущностей и EJB-компонент, рекомендуется создать отдельный класс, агрегирующий все составляющие приложения; этот класс затем должен быть преобразован в соответствующий сервис.

В любом случае, необходимо хорошо разобраться с аннотациями поставщика Web-сервисов, которые описаны в источнике (см. [1,пункте 5.2.2]).

В нашем учебном примере, имея единственную сущность ***Letter*** и одну EJB-компоненту ***Lets7***, имеет смысл преобразовать ее в Web-сервис, что и нужно сделать студенту, используя соответствующие аннотации.

С проектной точки зрения, для создания Web-службы минимально необходимы две аннотации:

1) ***@WebService*** — обязательная аннотация, имеющая набор параметров по умолчанию (см листинг 5.6 источника [1]);
2) ***@javax.ejb.Stateless*** или ***@javax.ejb.Singleton*** — аннотации, определяющие тип конечной точки Web-сервиса.

Рекомендуется без необходимости вообще не использовать аннотацию ***@Singleton***, поскольку она требует дополнительного программирования, например, для реализации многопользовательского доступа к сервису.

Поскольку класс ***Lets7*** является одновременно и EJB-компонентой, то рекомендуется использовать также аннотацию ***@PersistenceContext***, как это показано на листинге 5.7 источника [1]. Эта аннотация обеспечивает инкапсуляцию в Web-сервис объекта менеджера сущностей, что необходимо для эффектиной работы с классом ***Letter***.

В заключение, необходимо правильно описать дескриптор развертывания проекта ***web.xml*** (см. [1, листинг 5.8]), в котором класс ***Lets7*** описан как сервлет.

Теперь можно переходить к исследованию Web-службы.

7.2.3 Исследование Web-службы SOAP

В нашем учебном примере, тестирование Web-службы осуществляется достаточно просто. Необходимо:

1) запустить СУБД Apache Derby в сетевом варианте;
2) запустить сервер приложений Apache TomEE;
3) запустить браузер и выполнить соединение с Web-ресурсом по адресу: http://localhost:8080/lab7/Lets7?wsdl.

В окне браузера должно появится описание сервиса на языке WSDL, как это показано на рисунке 5.3 учебного пособия [1].

Студенту также настоятельно рекомендуется изучить примеры обработки исключений и контекста, описанные в пунктах 5.2.3 и 5.2.4 того же учебного пособия.

После описания результатов работы в личном отчете, студент может считать данную лабораторную работу выполненной.

8 Работа 8. Классические средства реализации Web-сервисов

Лабораторная работа №8 является второй частью учебной работы, которая посвящена практическому закреплению теоретического материала главы 5 учебного пособия [1]: «Web-службы SOA». В этом отношении данная работа является продолжением предыдущей.

В контексте выполнения учебного задания, студент должен помнить, что основное отличие сервис-ориентированных систем от других подходов, использующих парадигму «*Клиент-сервер*», состоит в принципиальном разделении поставщиков сервисов и потребителей сервисов. Эта лабораторная работа посвящена созданию программных агентов потребителей сервиса.

Учебная цель данной работы — создание учебной Web-службы, которая реализуется инструментальными средствами ***потребителей сервисов***.

Методически, данная работа разделена на две части, которые студент должен выполнить последовательно, согласно учебному материалу пятой главы учебного пособия (см. [1, подраздела 5.3]):

1) подготовку инструментальных средств, необходимых для реализации агента Web-службы SOAP;
2) непосредственную реализацию агента учебной Web-службы на базе инструментальных средств программной платформы Java EE.

8.1 Подготовка проекта для агента потребителя сервиса

В техническом плане, реализация агента потребителя сервиса — достаточно проста и завист от размещения поставщика и потребителя сервиса.

Если агент потребителя сервиса размещается в том же адресном пространстве, что и поставщик сервиса, то можно воспользоваться соответствующей аннотацией ***@javax.xml.ws.WebServiceRef*** и инкапсулировать объект типа ***Lets7***, обращаясь к которому — уже и получить нужный результат. Как это делается, — описано в пункте 5.3.1 учебного источника [1].

Нас в данной работе интересует более общий подход, предполагающий что агент потребителя сервиса размещается на другом сервере приложений чем поставщик сервиса. Для таких случаев имеются инструментальные средства, которые поставляются даже на платформе Java SE.

8.1.1 Исползование утилиты wsimport

Необходимые действия по использованию утилиты ***wsimport*** описаны в пункте 5.3.2 источника [1].

В общем случае, для генерации необходимых исходных текстов, требуется выполнение трех условий:

1) наличие и доступность сервера приложений, публикующего ***WSDL***-описание поставщика Web-сервиса;

2) наличие самой утилиты ***wsimport***;

3) конкретное определение адресного пространства создаваемого агента потребителя сервиса, что обычно задается оператором ***package***.

Применение утилиты ***wsimport*** позволяет любому программисту, который даже не изучал язык WSDL, сгенерировать необходимый набор программного обеспечения, обеспечивающего создание прокси-объекта для доступа к методам поставщика Web-сервиса (Web-службы).

8.1.2 Подготовка проекта lab8

Для реализации агента потребителя сервиса создается отдельный проект с именем ***lab8***. Цель отдельного проекта — разделить пространства имен поставщика и потребителя сервиса. Таким образом моделируется распределенное размещение взаимодействующих сторон.

Студент должен также понимать, что агент потребителя сервиса может быть реализован в виде АРМ (*Автоматизированного Рабочего Места*) или в виде утилиты средствами программной платформы Java SE. В соответсвии с этим утверждением, реализация агента потребителя сервиса на сервере Apache Tomee является одним из частных случаев возможных проектных решений.

8.2 Реализация тестового агента потребителя сервиса

Поскольку реализация агента потребителя сервиса осуществляется в инструментальной среде Eclipse EE, то в соответствующее место проекта ***lab8*** необходимо скопировать исходные тексты классов, сгенерированных утилитой ***wsimport***. Откомпилированные классы переносить не нужно, чтобы не создавать конфликты в среде разрабоки Eclipse EE. Количество и размеры созданных утилитой ***wsimport*** классов может быть большим. Это в свою очередь зависит от количества используемых поставщиком сервиса EJB-компонент и их методов.

Для нашего Web-сервиса, представленного поставщиком сервиса в виде Web-услуги ***Lets7***, необходимыми и достаточными для реализации потребителя сервиса являются описания классов, содержащихся в файлах:

1) ***Lets7.java*** — описание интерфейс класса ***Lets7***, по которому можно определить методы предоставляемые поставщиком сервиса;

2) ***Lets7Service.java*** — описание класса ***Lets7Service*** (см. [1, листинг 5.16]), содержащего методы для подключения агента потребителя сервиса к поставщику Web-услуги ***Lets7***.

8.2.1 Тестовый класс для потребителя сервиса

Целью создания тестового класса для потребителя Web-сервиса является демонстрация работоспособности классических средств реализации Web-служб на основе протокола SOAP.

Студенту предлагается создать, в рамках проекта ***lab8***, тестовый класс ***ClientLets***, с помощью которого демострируется использование методов:

1) чтения списка записей объектов типа ***Letter***;
2) чтения отдельного объекта типа ***Letter*** по его идентификатору;
3) удаления отдельного объекта типа ***Letter*** по его идентификатору.

Пример такого класса представлен на листинг 5.17 в учебном пособии [1, пункт 5.3.3].

Студенту следует реализовать данный тестовый пример и привести описание полученных результатов в личном отчете.

8.2.2 Выводы по лабораторным работам №7 и №8

Данная лабораторная работа фактически завершает практическое освоение заявленной классической теоретической части изучаемой дисциплины. Кроме того, она является конечным результатом разработки распределенной сервис-ориентированной системы, начатой в лабораторной работе №7.

Студенту необходимо дать совокупное описание проделанных работ, которое должно быть изложено в контексте разделения технологии разработки распределенных Web-сервисов (Web-услуг) на две части:

1) часть, соответствующая технологиям поставщика сервиса;

2) часть, соответствующая технологиям создания агентов потребителей сервиса.

9 Работа 9. Web-службы в стиле REST

Лабораторная работа №9 посвящена практическому закреплению теоретического материала, соответствующего направлению технологий, считающихся альтернативными классическим технологиям, которые основанны на протоколе SOAP. Учитывая многоплановость и отсутсвие должной стандартизации этих направлений, студенту рекомендуется опираться на теоретические положения, изложенные в шестой главе базового учебного пособия [1]: «*Web-службы в стиле REST*».

Учебная цель данной работы — создание учебной Web-службы в стиле RESTfull, которая сама по себе демонстрирует полную реализацию учебного примера инструментальными средствами программной платформы Java EE.

Методически, данная работа разделена на три части, которые студент должен выполнить последовательно, согласно учебному материалу шестой главы учебного пособия [1, подразделы 6.1 — 6.3]:

1) проектирование Web-службы в стиле REST;
2) реализация поставщика сервиса;
3) реализация потребителя сервиса.

9.1 Проектирование Web-службы в стиле REST

Стиль REST (***R**epresentational **S**tate **T**ransfer*), введенный Роем Филдингом в 2000 году для создания сервис-ориентированных систем, стал популярным благодаря недостаткам классического подхода, основанного на протоколе SOAP. К таким недостаткам следует отнести излишнюю универсальность представления объектов, влекущих за собой излишнюю сложность, громоздкость и медленную работу создаваемых систем.

Как альтернатива архитектуре SOA и универсальному протоколу SOAP, RESTfull-системы построены на различных ограничениях, основные из которых описаны в шестой главе учебного источника [1, подраздел 6.1]. Студенту следует внимательно изучить эти ограничения и выполнить два тестовых упражнения.

9.1.1 Проектные средства технологии RESTfull

Проектные средства технологии RESTfull определяются основными ограничениями, налагаемыми на это понятие, которое должно быть реализовано производителем сервиса. Важнейшими из них является требования применяемые к ***универсальному идентификатору ресурса*** (*URI*), что подробно описано в источнике [1, пункт 6.1.1].

Хотя само определение URI является достаточно формализованным, его практическое использование определяется прикладной частью создаваемого Web-сервиса (*Web-услуги*).

Студенту следует разработать набор запросов к учебному сервису для работы с объектами уже известной ему сущности ***Letter***, предполагая что:

1) реализация сервиса будет проводиться его поставщиком в проекте с именем ***lab9***;
2) сервис будет предоставлять стандартный набор из пяти услуг, подразумевающих: чтение всего списка объектов типа ***Letter***, создание нового объекта этого типа, а также набор операций чтения, удаления и модификации отдельных объектов типа ***Letter*** по заданному для них уникальному идентификатору.

9.1.2 Тестирование простейшего RESRfull-сервлета

Завершив задание по предыдущему пункту, студенту следует создать и протестировать простейший RESTfull-сервлет. Программная платформа Java EE предлагает для этих целей набор пакетов, определенных общей спецификацией JAX-RS.

В качестве конкретного задания студенту предлагается реализовать JAVA-класс с именем ***LetsRestService***, который далее будет рассматриваться как шаблон, демонстрирующий работу со списком записей типа ***Letter***. Вариант реализации этого сервлета подробно описан в источнике [1, пункт 6.1.4].

Особое внимание студент должен обратить на правильное использование базовой аннотации ***@Path(...)*** и сопуствующим им аннотациям ***@GET*** и ***@POST*** (см. [1, листинг 6.1]).

Тестирование работы сервлета следует провести посредством чтения и отображения браузером HTML-страницы ***index.html*** (см. [1, листинг 6.2]).

Несмотря на кажущуюся простоту и прикладную «***бесполезность***» этого теста, студент должен тщательно провести анализ выполненной работы и отразить результаты этого анализа в личном отчете. Особое внимание следует уделить способам адресации самих запросов и учесть, что:

1) одинаковые HTTP-адреса соотвествуют разным прикладным запросам к сервисам;
2) имеется широкая возможность вызова одного и того же сервиса различными способами адресации.

Это упражнение должно обеспечить лучшее понимание проектных решений, изложенных в следующем подразделе данной работы.

9.2 Реализация поставщика сервиса

В предыдущем подразделе был представлен тестовый вариант сервлета ***LetsRestService***, на котором показано, что Web-услуга, соответствующая инструментальным средствам RESTfull-технологии, действительно работает на запросах браузера типа GET и POST.

Данная часть лабораторной работы посвящена реализации функционала той части Web-услуги поставщика сервиса, которую можно протестировать, используя только возможности браузеров, ограниченные запросами типа GET и POST.

В прикладном плане, создаваемая Web-услуга должна полностью соответствать прикладным возможностям сервиса, реализованного в лабораторных работах №7 и №8. Такой подход должен облегчить саму реализацию, поскольку позволяет использовать значительную часть уже разработанного программного обеспечения и полностью сосредоточить внимание студента на особенностях использования технологии RESTfull.

Конкретная последовательность работ, требующих выполнения данного задания и описанная в следующих двух пунктах, опирается на учебный материал пособия [1, подраздел 6.2].

9.2.1 Подготовительная часть проекта lab9

Подготовительная часть проекта ***lab9*** проводится также, как это было сделано ранее в лабораторной работе №7:

1) в проект переностися исходный текст класса ***Letter***, представляющего собой сущность отображаемую в таблице ***t_letter*** базы данных ***lab4db***;
2) в проект следует перенести класс ***Lets7***, являющийся EJB-компонентой для работы с сущностью ***Letter***, который следует преобразовать в класс ***Lets9***;
3) дополнительно переносятся и редактируются: ***derbyclient.jar*** — драйвер для работы с СУБД Apache Derby; ***resources.xml*** — десриптор ресурсов для соединения с СУБД Apache Derby; ***pesistence.xml*** — дескриптор описания юнитов технологи JPA для работы с СУБД Apache Derby;
4) дополнительно создается сериализуемый класс ***ListLets***, предназначенный для пересылки объектов типа ***Letter*** в виде списка.

Как это все делается, — достаточно подробно описано в учебном пособии [1, пункты 6.2.1 — 6.2.2].

Выполняя работу, студенту следует добиться условия, чтобы среда разработки Eclipe EE не показывала наличие ошибок.

Студент также должен понимать, что подготовительная часть проекта, реализующая прикладную часть Web-услуги (Web-сервиса), может быть очень

большой. Ее объем (размер) зависит от количества и сложности EJB-компонент, включенных поставщиком сервиса в обеспечение общей функциональности.

9.2.2 Последовательная реализация сервлета LetsRestService

По замыслу выполнения лабораторной работы реализация Web-услуги проводится посредством модификации сервлета ***LetsRestService***, который содержит заготовки шести методов (см. [1, листинг 6.1]): ***testRS()***, ***getLets()***, ***getLetter()***, ***addLetter()***, ***deleteLetter()*** и ***modLetter()***. Назначение этих методов — достаточно очевидно и не требует дополнительных комментариев. В такой последовательности и следует выполнять реализацию и тестирование методов указанного RESTfull-сервлета.

С другой стороны, каждый разработанный метод, реализующий отдельную услугу, должен быть протестирован самим поставщиком сервиса, прежде чем предоставлять его потребителям. В общем случае, желательно, чтобы поставщик сервиса предоставлял примеры клиентских программ для каждого сервиса.

Учитывая указанные выше условия, в данной части лабораторной работы реализуется только часть методов сервлета ***LetsRestService***, которая может быть протестирована с помощью браузера. Как это делается, — подробно описано в учебном пособи [1, пункты 6.2.3 — 6.2.5].

При выполнении этой части работы, студент должен обратить особое внимание на следующие ее особенности, которые следует правильно отразить в своем индивидуальном отчете.

Метод ***testRS()*** (см. [1, листинг 6.6]) предназначен для проверки наличия объектов инкапсулируемых EJB-компонент. Очевидно, что если какие-либо компоненты не подключаются во время запроса, осуществляемого потребителем сервиса, то сервис не может считаться работоспособным. Неисключено, что с целью повышения надежности Web-услуги, наиболее критичные методы должны самостоятельно проверять результат инкапсуляции нужных им EJB-компонент.

Методы ***getLets()*** и ***getLetter()*** (см. [1, подразделы 6.2.3 и 6.2.4]), использующие метод GET HTTP-запроса, могут быть реализованы, протестированы и использованы без последующего изменения.

Метод ***addLetter()*** (см. [1, листинг 6.9]), ориентированный на метод POST HTTP-запроса, приведен в качестве примера добавления новой записи из браузера, посредством использования аннотации ***@FormParam()***.

Следующая и последняя часть лабораторной работы демонстрирует реализацию всех методов рассматриваемого учебного сервиса.

9.3 Реализация потребителя сервиса

До сих пор мы разрабатывали и тестировали ту часть Web-услуги, которая разрабатывается поставщиком сервиса, но наиболее интересные и полезные решения возникают, когда подключаются инструментальные средства, предоставляемые спецификацией JAX-RS для потребителей сервиса.

Все дело в том, что сам сервис может быть востребован, если он не только удобен для использования потребителем, но и содержит более мощные инструменты реализации приложений, чем это могут предоставлять общие возможности браузеров. И здесь речь идет о методах доступа по протоколам HTTP, которые не ограничены только запросами GET и POST, а позволяют легко использовать запросы типа PUT и DELETE. В целом, данной части лабораторной работы соответствует учебный материал, изложенный в пособии [1, подраздел 6.3].

Чтобы полностью раскрыть основные возможности стиля RESTfull, данная часть лабораторной работы разделена на четыре этапа:

1) тест инструментальных средств потребителя сервиса знакомит студента с основным шаблоном его применения;
2) полная реализация сервиса проекта ***lab9*** показывает реализацию Web-услуги с учетом новых инструментальных средств потребителя сервиса;
3) использование технологии JSF показывает возможности сервера приложений для реализации программного агента потребителя сервиса;
4) последний этап показывает реализаци запросов типа GET, POST, PUT и DELETE на стороне агента потребителя сервиса.

9.3.1 Тест инструментальных средств потребителя сервиса

Основная часть инструментальных средств потребителя сервиса, содержащаяся в пакете ***javax.ws.rs.client***, представлена объектами классов ***Client***, ***WebTarget***, ***Invocation*** и ***Response***. Для демонстрации этого инструмента использован отдельный проект среды разработки Eclipse EE с именем ***jaxrs***, в котором на примере класса ***RunGetLetter*** показано формирование HTTP-запроса типа GET (см. [1, пункт 6.3.1]).

Сам запрос расписан достаточно подробно (см. [1, листинг 6.10]), чтобы студент на этом примере освоил основную идею использования инструментария потребителя сервиса.

Отражая результат тестирования класса ***RunGetLetter*** в личном отчете, студент должен описать и его варианты, указанные в листинге 6.10 подсказками в виде комментариев.

9.3.2 Полная реализация сервиса проекта lab9

Вторым этапом работы идет полная реализация проекта ***lab9***, соответствующая функционалу поставщика сервиса, который учитывает новые возможности инструментальных средств потребителя сервиса.

Обратите внимание, что, прежде чем приступать к реализации методов класса ***LetsRestService***, необходимо провести предварительное проектирование адресной части запросов и кодов возврата, на которые должен будет реагировать потребитель сервиса. А как это делается, — описано в учебном пособии [1, пункт 6.3.2].

В отчете студент должен особо отметить каким образом изменилось представление о сервисе, когда поставщик сервиса учел инструментальные возможности потребителя сервиса.

9.3.3 Использование технологии JSF

Полная реализация RESTfull-сервиса требует размещение функционала потребителя сервиса на сервере приложений или в виде отдельной программмы, использующей инструментарий клиента RESTfull-сервиса. В свою очередь, программная платформа Java EE предоставляет технологию JSF, которая размещается в специальном контейнере сервера приложений и предоставляет широкие возможности по реализации программных агентов потребителей сервиса (см. [1, глава 2]).

Данный этап лабораторной работы предполагает использование уже полученныз результатов проекта ***labs***, реализованных при выполнени лабораторных работ №2 и №3. эти результаты позволяют на достаточно профессиональном уровне реализовать программный агент потребителя RESTfull-сервиса, а как это делается, — описано в учебном пособии [1, пункт 6.3.3].

Студенту следует реализовать в проекте ***labs*** шаблон потребителя сервиса, модифицировав соотвествующим образом XHTML-файл ***lab9.xhtml*** и создав EJB-компоненту ***lab9.java***, которая содержит шаблоны методов для запросов потребителя сервиса.

9.3.4 Реализация запросов клиентов

Завершающим этапом лабораторной работы является реализация пяти базовых методов: ***mGetList()***, ***mGetLetter()***, ***mPostLetter()***, ***mDeleteLetter()*** и ***mPutLetter()***. Как это делается, — описано в учебном пособии [1, пункты 6.3.4 — 6.3.7].

В личном отчете, студент должен отразить результаты и выводы по выполненной работе.

Заключение

Данное учебно-методическое пособие является дополнением к учебному пособию [1] и обеспечивает магистранта всей необходимой информацией для успешного выполнения полного цикла из девяти лабораторных работ по дисциплине «*Распределенные сервис-ориентированные системы*».

Методика выполнения работ предполагает, что студент выполняет их последовательно в указанном порядке, предварительно изучая теоретический материал закрепляемый соответствующей работой, согласно таблице В.1.2 представленной во введении.

Особо важное значение имеет успешное завершение первой лабораторной работы, в которой студент должен установить, настроить и протестировать достаточно большой объем инструментальных программных средств, без правильной работы которых дальнейшее продвижение учебного процесса становится невозможным. Учитывая это обстоятельство, студенту даются рекомендации по самостоятельному поиску, приобретению и установке соответствующих программных пакетов, необходимых для создания целостной учебной системы.

Последующие пять лабораторных работ (№2 - №6) посвящены закреплению теоретического материала и получению навыков работы с инструментальными средствами, которые подготавливают студента к созданию целостных сервис-ориентированных систем. После их успешного завершения, студент может приступать к выполнению последних трех работ.

Лабораторные работы №7 и №8 позволяют студенту реализовать простейшую учебную распределенную сервис-ориентированную систему, которая соответствует классической парадигме теоретических представлений о предмете изучения, основанной на использовании протокола SOAP. Успешное завершение этих работ рассматривается как успешное практическое освоение учебного материала изучаемой дисциплины.

Лабораторная работа №9 содержит учебное задание по альтернативным подходам к реализации сервис-ориентированных систем, известных как системы, реализованные в стиле RESTfull. Ее описание расширено по отношению к требуемым временным характеристкам учебного процесса. Это сделано с целью более полной демонстрации студенту возможностей этого направления развития современных технологий. Даже частичное выполнение данного задания оценивается как положительный результат работы студента.

Список использованных источников

1. Резник, В. Г. Распределенные сервис-ориентированные системы: Учебное пособие [Электронный ресурс] / В. Г. Резник. — Томск: ТУСУР, 2020. - 305 с.
2. Учебный программный комплекс кафедры АСУ на базе ОС ArchLinux [Электронный ресурс]: Учебно-методическое пособие для студентов направления 09.03.01, Направление подготовки "Программное обеспечение средств вычислительной техники и автоматизированных систем" / В. Г. Резник - 2016. 33 с. — Режим доступа: https://edu.tusur.ru/publications/6238.
3. Резник, В. Г. Распределенные вычислительные сети: Учебное пособие [Электронный ресурс] / В. Г. Резник. — Томск: ТУСУР, 2019. — 211 с. — Режим доступа: https://edu.tusur.ru/publications/9072.
4. Apache Derby [Электронный ресурс]: Режим доступа: http://db.apache.org/derby/.
5. Apache TomEE [Электронный ресурс]: Режим доступа: http://tomee.apache.org/.
6. Eclipse Foundation [Электронный ресурс]: Режим доступа: https://www.eclipse.org/.

Оглавление