Лабораторная работа №3

Разработка Web-приложения для обмена сообщениями с использованием JMS 2.0

Цель работы

Разработать простое Web-приложение, демонстрирующее процесс обмена сообщениями с использованием технологии JMS 2.0.

Общие сведения

Вместе с релизом Java EE (Enterprise Edition) 7 в 2013 году, появилась версия технологии JMS 2.0, которая не обновлялась с 2002 года.

API (Application Programming Interface) JMS 1.1 требует для работы достаточно много кода, но успело себя хорошо зарекомендовать, потому, с 2002 года не изменялось. В JMS 2.0 новое API призвано упростить отправку и прием JMS-сообщений. При этом API JMS 1.1 не было упразднено и продолжает работать наравне с новым.

В API JMS 2.0 появились новые интерфейсы: JMSContext, JMSProducer и JMSConsumer:

1) JMSContext заменяет Connection и Session в API JMS 1.1;

2) JMSProducer – это легковесная замена MessageProducer, которая позволяет задавать настройки доставки сообщений, заголовки, свойства, через вызов цепочки методов (паттерн Builder);

3) JMSConsumer заменяет MessageConsumer и используется по такому же принципу.

Для сравнения рассмотрим отправку сообщения с помощью JMS 1.1:

```
public void sendMessageJMS11(ConnectionFactory connectionFactory, Queue queueString text) {
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session =connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageProducer messageProducer = session.createProducer(queue);
         TextMessage textMessage = session.createTextMessage(text);
         messageProducer.send(textMessage);
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      // handle exception (details omitted)
   }
}
```

А также отправку сообщения с помощью JMS 2.0:

```
public void sendMessageJMS20(ConnectionFactory connectionFactory, Queue queue,
String text) {
   try (JMSContext context = connectionFactory.createContext();){
      context.createProducer().send(queue, text);
   } catch (JMSRuntimeException ex) {
      // handle exception (details omitted)
   }
}
```

Отличительными особенностями использования JMS 2.0 для отправки сообщений являются следующие:

1) в версии 2.0 используется конструкция try-with-resources, появившаяся в Java SE 7;

2) параметр Session.AUTO_ACKNOWLEDGE устанавливается по умолчанию в JMSContext. Если требуется установить другое значение (CLIENT_ACKNOWLEDGE или DUPS_OK_ACKNOWLEDGE), оно передается как отдельный параметр;

3) используется JMSContext вместо объектов Connection и Session;

4) чтобы создать TextMessage, достаточно просто передать в метод send строку.

Отличительной особенностью нового API JMS 2.0 является то, что его методы выбрасывают RuntimeException – JMSRuntimeException, вместо

checked-исключения JMSException. Это дает возможность при желании не обрабатывать JMS-исключения.

Сравним синхронное получение сообщения с помощью JMS 1.1:

```
public String receiveMessageJMS11(ConnectionFactory connectionFactory,Queue queue){
   String body=null;
   try {
      Connection connection = connectionFactory.createConnection();
      try {
         Session session =connection.createSession(false,Session.AUTO_ACKNOWLEDGE);
         MessageConsumer messageConsumer = session.createConsumer(queue);
         connection.start();
         TextMessage textMessage = TextMessage)messageConsumer.receive();
         body = textMessage.getText();
      } finally {
         connection.close();
      }
   } catch (JMSException ex) {
      // handle exception (details omitted)
   }
   return body;
}
```

А также синхронное получение сообщения с помощью JMS 2.0:

```
public String receiveMessageJMS20(
ConnectionFactory connectionFactory,Queue queue){
   String body=null;
   try (JMSContext context = connectionFactory.createContext();){
      JMSConsumer consumer = session.createConsumer(queue);
      body = consumer.receiveBody(String.class);
   } catch (JMSRuntimeException ex) {
      // handle exception (details omitted)
   }
   return body;
}
```

Отличительными особенностями использования JMS 2.0 для синхронного получения сообщений являются следующие:

1) используется конструкция try-with-resources для автоматического закрытия соединения;

2) используется JMSContext вместо объектов Connection и Session;

3) AUTO_ACKNOWLEDGE устанавливается по умолчанию;

4) обрабатывается JMSRuntimeException, который можно не обрабатывать, вместо JMSException в JMS 1.1;

5) вызов метода start() у объекта connection выполняется автоматически;

6) строка получается автоматически с помощью вызова метода receiveBody(String.class) у объекта consumer, вместо получения объекта Message, приведения его к TextMessage и вызова метода getText().

Сравним асинхронное получение сообщения с помощью JMS 1.1:

```
MessageConsumer messageConsumer = session.createConsumer(queue);
messageConsumer.setMessageListener(messageListener);
connection.start();
```

А также асинхронное получение сообщения с помощью JMS 2.0:

```
JMSConsumer consumer = context.createConsumer(queue);
consumer.setMessageListener(messageListener);
```

Вместо MessageConsumer используется JMSConsumer. Соединение стартует автоматически.

В Java EE при построении Web-приложений или EJB приложений, как и прежде, необходимо использовать message-driven bean, вместо метода setMessageListener().

В Java EE приложении JMSContext можно использовать посредством применения аннотации @Inject. После чего, JMSContext будет находиться под управлением сервера приложений.

Следующий фрагмент кода позволяет вставлять JMSContext в session bean или сервлет:

```
@Inject @JMSConnectionFactory(
"jms/connectionFactory") private JMSContext context;

@Resource(lookup = "jms/dataQueue") private Queue dataQueue;

public void sendMessageJavaEE7(String body) {
   context.send(dataQueue, body);
}
```

Закрытие JMSContext производится автоматически сервером приложений. Если во время запроса выполняется JTA-транзакция, то JMSContext закроется автоматически после коммита, если без транзакции, то закроется в конце запроса.

Процесс выполнения работы

Перед тем, как приступить к разработке веб-приложения, необходимо загрузить сервер GlassFish 4.1.2, доступный по следующей ссылке http://repo1.maven.org/maven2/org/glassfish/main/distributions/glassfish/4.1.2/.

После того, как сервер будет загружен, необходимо распаковать содержимое архива. Таким образом, сервер будет располагаться, например, по такому пути D:/glassfish4.

После чего, можно перейти к созданию веб-приложения с помощью среды разработки Eclipse IDE.

1. В среде Eclipse IDE создать новый проект, выбрав File -> New -> Other -> Web -> Dynamic Web Project, в поле Project Name ввести название создаваемого проекта.

2. Ниже, в секции Target runtime, нажать New Runtime, выбрать GlassFish и нажать Next (рисунок 1).

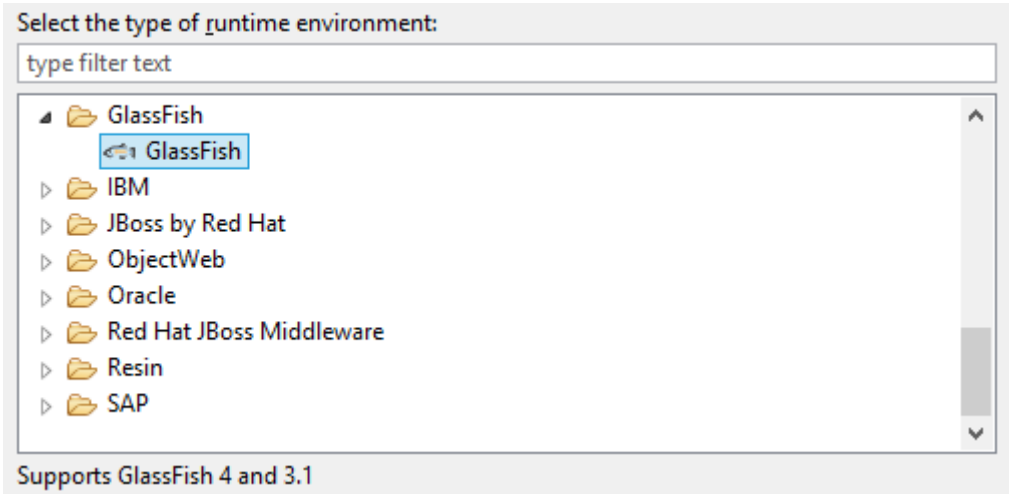

Рисунок 1

3. В поле GlassFish location ввести путь к папке с сервером, распакованной на стадии подготовки к созданию веб-приложения, поле Java location будет заполнено автоматически, после чего нажать Finish (рисунок 2).

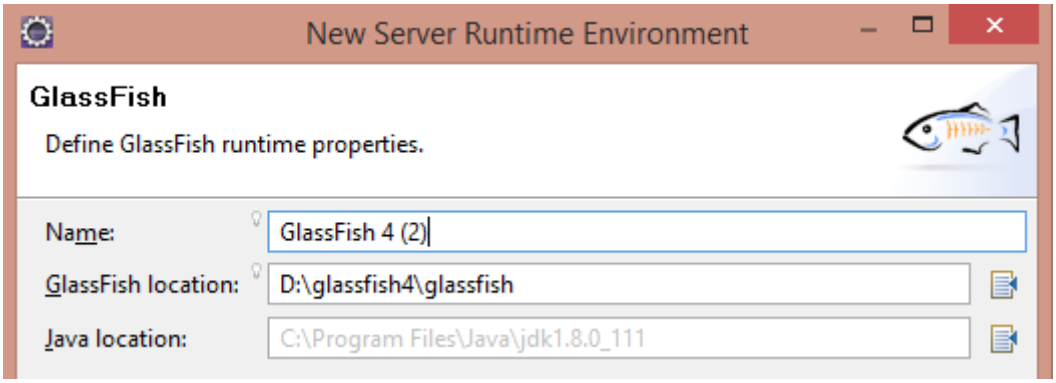

Рисунок 2

4. Для завершения этапа по созданию проекта, необходимо нажать кнопку Finish.

5. После того, как проект будет создан, в папке src необходимо создать следующие пакеты (рисунок 3):

1) ua.khpi.constants;

2) ua.khpi.service;

3) ua.khpi.servlet.

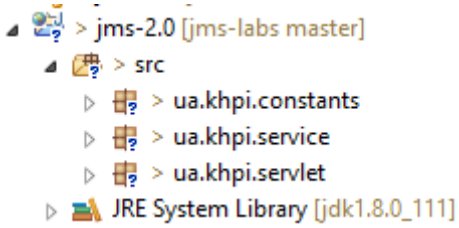

Рисунок 3

6. В пакете ua.khpi.constants необходимо создать интерфейс Constants, используя меню New -> Interface (рисунок 4).

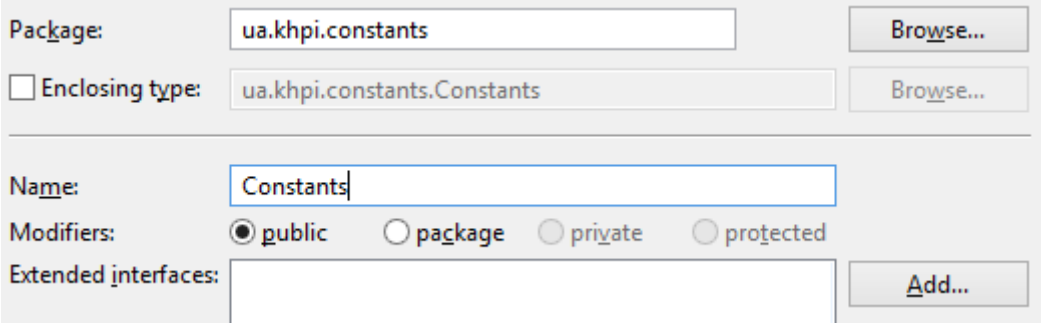

Рисунок 4

7. В созданном интерфейсе необходимо добавить следующие поля (рисунок 5).

```
public static final String QUEUE = "jms/Lab03";

public static final String SENDER = "sender";
public static final String RECEIVER = "receiver";

public static final String SERVICE = "service";
public static final String TEXT = "text";

public static final String ERROR = "error";

public static final String HOME = "index.jsp";
```

Рисунок 5

8. После чего, в пакете ua.khpi.service необходимо создать два класса – JMSSender и JMSReceiver соответственно (рисунок 6).

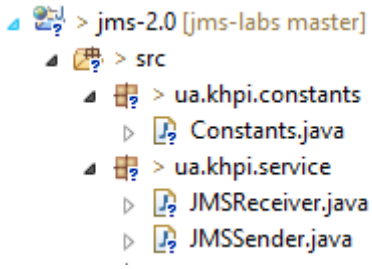

Рисунок 6

9. В классе JMSSender необходимо создать поля context и queue, указав для данных полей аннотации Inject и Resource(mappedName = Constants.QUEUE) соответственно (рисунок 7).

```
@Inject
private JMSContext context;

@Resource(mappedName = Constants.QUEUE)
private Queue queue;
```

Рисунок 7

10. Аналогичные поля необходимо создать также и в классе JMSReceiver.

11. После того, как поля для созданных классов будут добавлены, в классе JMSSender необходимо описать метод send() (рисунок 8).

```
public void send(String text) {
    context.createProducer().send(queue, text);
}
```

Рисунок 8

12. В классе JMSReceiver, в свою очередь, необходимо описать метод receive() (рисунок 9).

```
public String receive() {
    JMSConsumer consumer = context.createConsumer(queue);
    return consumer.receiveBody(String.class);
}
```

Рисунок 9

13. Для обоих созданных классов JMSSender и JMSReceiver необходимо указать аннотации ApplicationScoped (рисунок 10, 11).

```
@ApplicationScoped
public class JMSSender {
```

Рисунок 10

```
@ApplicationScoped
public class JMSReceiver {
```

Рисунок 11

14. Далее, в пакете ua.khpi.servlet, необходимо создать класс JMSController, унаследовав его от класса HttpServlet (рисунок 12).

```
public class JMSController extends HttpServlet {
```

Рисунок 12

15. В созданном классе необходимо объявить поля sender и receiver, для каждого из которых необходимо указать аннотации Inject (рисунок 13).

```
@Inject
private JMSSender sender;

@Inject
private JMSReceiver receiver;
```

Рисунок 13

16. Наконец, необходимо создать метод doPost(), предназначенный для обработки HTTP запросов (рисунок 14).

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String serviceName = request.getParameter(Constants.SERVICE);

    if (serviceName == null) {
        request.setAttribute(Constants.ERROR, "Service name can't be empty!");
    } else {
        if (serviceName.equals(Constants.SENDER)) {
            String text = request.getParameter(Constants.TEXT);

            if (text == null || text.isEmpty()) {
                request.setAttribute(Constants.ERROR, "Text can't be empty!");
            } else {
                sender.send(text);
            }
        } else if (serviceName.equals(Constants.RECEIVER)) {
            String text = receiver.receive();

            request.setAttribute(Constants.TEXT, text);
        } else {
            request.setAttribute(Constants.ERROR, "Invalid service name!");
        }
    }

    request.getRequestDispatcher(Constants.HOME).forward(request, response);
}
```

Рисунок 14

Далее необходимо создать веб-страницу, на которой будут располагаться элементы простого интерфейса, предназначенного, для отправки и получения сообщений с использованием технологии JMS 2.0.

1. Для этого в папке WebContent необходимо создать файл, используя меню New -> File, указав для него название index.jsp (рисунок 15).

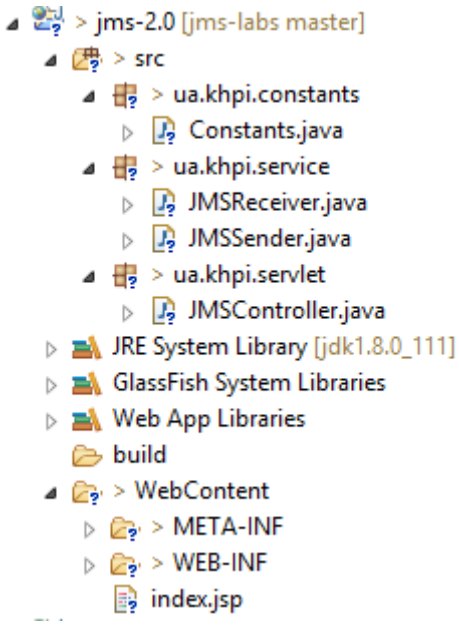

Рисунок 15

2. Содержимое данного файла будет иметь следующий вид:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
      pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JMS 2.0</title>
</head>
<body>
      <div>
            <form action="./jms" method="post">
                  <input type="hidden" name="service" value="sender" />
                  <p>
                        Text: <input type="text" name="text" />
                  </p>
                  <p>
                        <input type="submit" value="Send" />
                  </p>
            </form>
      </div>
```

```
	<div>
		<form action="./jms" method="post">
			<input type="hidden" name="service" value="receiver" />
			<p>
				<input type="submit" value="Receive" />
			</p>
		</form>
	</div>
	<p>
		<c:if test="${text ne null}">
			Received message: ${text}
		</c:if>
	</p>
	<p>
		<c:if test="${error ne null}">
			${error}
		</c:if>
	</p>
</body>
</html>
```

3. Для того чтобы в веб-приложении использовались теги JSTL, необходимо загрузить соответствующую библиотеку, доступную по ссылке https://mvnrepository.com/artifact/jstl/jstl/1.2, и поместить jar файл в папку WebContent/WEB-INF/lib (рисунок 16).

Рисунок 16

4. Наконец, необходимо добавить дескриптор развертывания. Для этого в папке WEB-INF требуется создать файл web.xml (рисунок 17).

Рисунок 17

5. Содержимое данного файла будет иметь следующий вид:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      id="WebApp_ID" version="3.0">
      <display-name>JMS 2.0</display-name>
      <welcome-file-list>
            <welcome-file>index.jsp</welcome-file>
      </welcome-file-list>
      <servlet>
            <servlet-name>JMSController</servlet-name>
            <servlet-class>ua.khpi.servlet.JMSController</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>JMSController</servlet-name>
            <url-pattern>/jms</url-pattern>
      </servlet-mapping>
</web-app>
```

6. Таким образом, структура созданного проекта будет выглядеть, как это показано на рисунке 18.

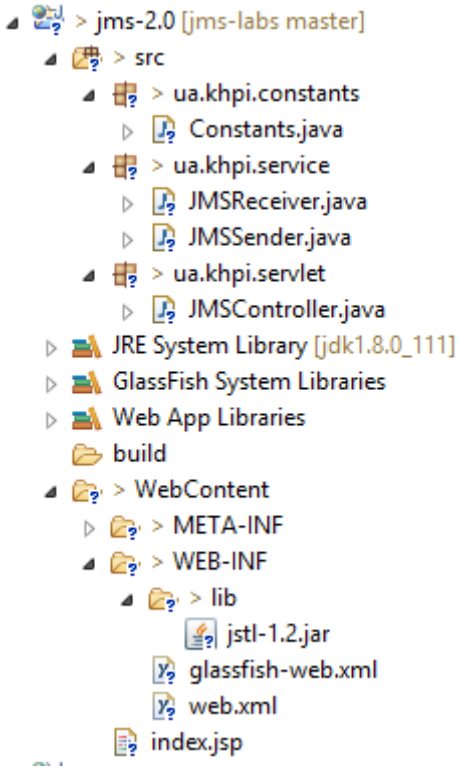

Рисунок 18

Для запуска веб-приложения необходимо воспользоваться меню Run As -> Run On Server, выбрать сервер GlassFish 4 и нажать кнопку Finish (рисунок 19).

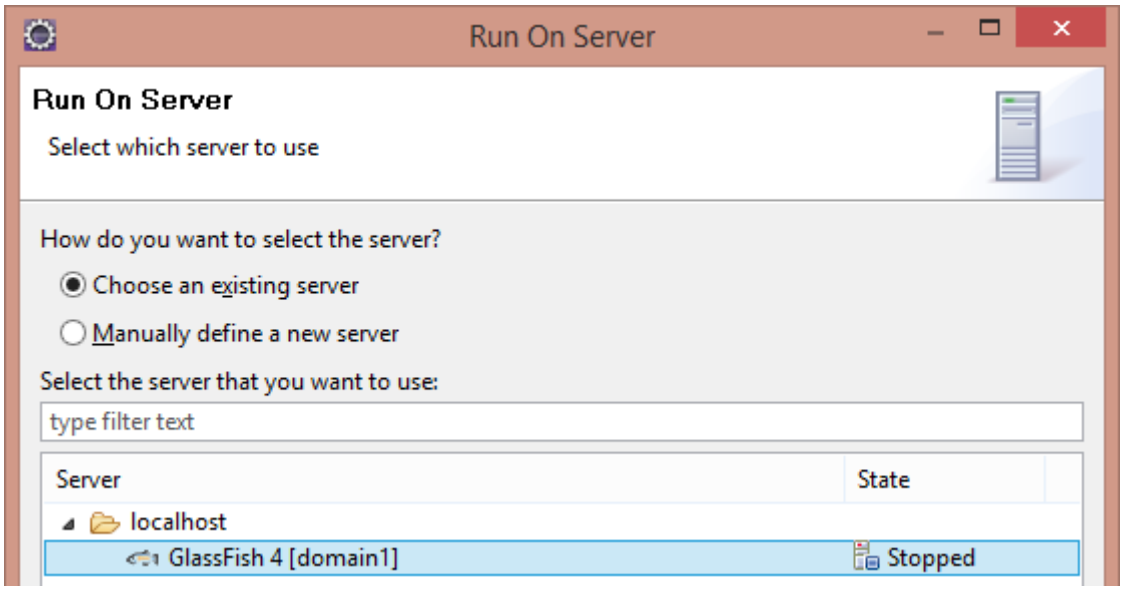

Рисунок 19

1. После того, как сервер будет запущен и приложение будет развернуто, необходимо перейти по адресу http://localhost:4848 (панель управления сервером GlassFish), в разделе Resources выбрать JMS Resources -> Destination Resources и создать очередь, нажав кнопку New (рисунок 20).

Рисунок 20

2. Далее, необходимо ввести следующие значения (таблица 1).

Таблица 1

Поле	Значение
JNDI Name	jms/Lab03
Physical Destination Name	Lab03
Resource Type	javax.jms.Queue
Deployment Order	100
Description	Lab03 Queue
Status	Enabled

3. После того, как описанные значения будут введены, необходимо нажать кнопку Save (рисунок 21).

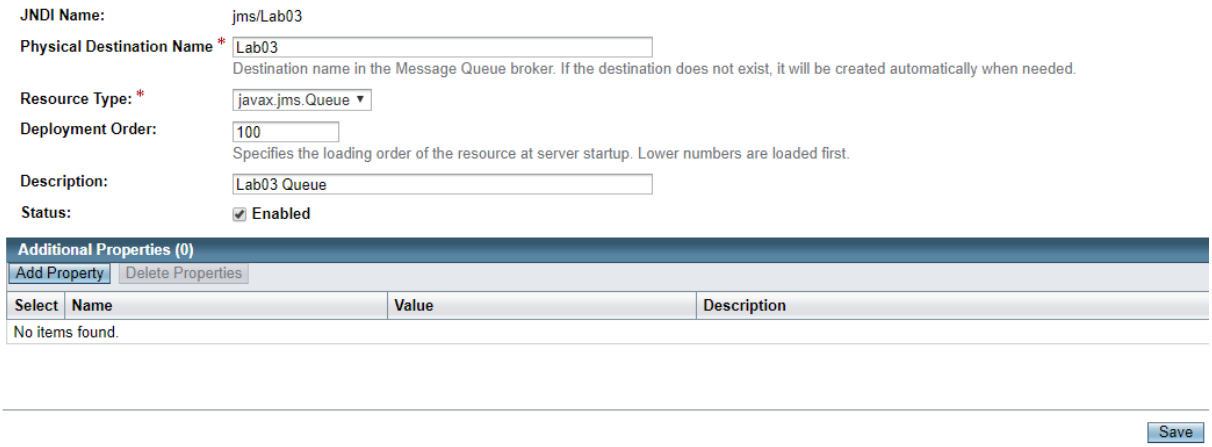

Рисунок 21

4. После чего, можно приступить непосредственно к работе с созданным веб-приложением, перейдя по адресу http://localhost:8080/jms-2.0/ (рисунок 22).

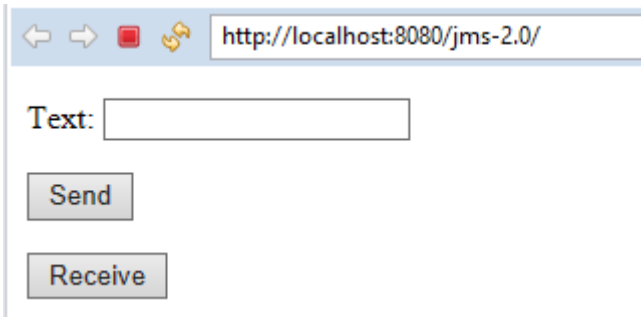

Рисунок 22

5. Для отправки сообщения необходимо ввести его текст в поле Text, после чего нажать кнопку Send (рисунок 23).

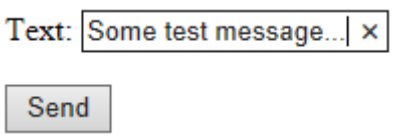

Рисунок 23

6. Для получения сообщения, необходимо нажать кнопку Receive (рисунок 24).

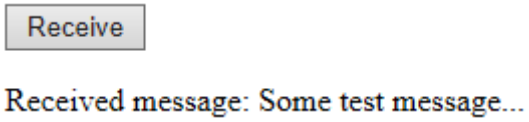

Рисунок 24

Таким образом, исходный код разработанного приложения будет следующим:

1. Интерфейс Constants:

```
package ua.khpi.constants;

public interface Constants {
	public static final String QUEUE = "jms/Lab03";

	public static final String SENDER = "sender";
	public static final String RECEIVER = "receiver";

	public static final String SERVICE = "service";
	public static final String TEXT = "text";

	public static final String ERROR = "error";

	public static final String HOME = "index.jsp";
}
```

2. Класс JMSSender:

```
package ua.khpi.service;

import javax.annotation.Resource;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.jms.JMSContext;
import javax.jms.Queue;

import ua.khpi.constants.Constants;

@ApplicationScoped
public class JMSSender {
	@Inject
	private JMSContext context;
```

```
	@Resource(mappedName = Constants.QUEUE)
    private Queue queue;

	public void send(String text) {
		context.createProducer().send(queue, text);
	}
}
```

3. Класс JMSReceiver:

```
package ua.khpi.service;

import javax.annotation.Resource;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;
import javax.jms.JMSConsumer;
import javax.jms.JMSContext;
import javax.jms.Queue;

import ua.khpi.constants.Constants;

@ApplicationScoped
public class JMSReceiver {
	@Inject
	private JMSContext context;

	@Resource(mappedName = Constants.QUEUE)
	private Queue queue;

	public String receive() {
		JMSConsumer consumer = context.createConsumer(queue);
		return consumer.receiveBody(String.class);
	}
}
```

4. Класс JMSController:

```
package ua.khpi.servlet;

import java.io.IOException;

import javax.inject.Inject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import ua.khpi.constants.Constants;
import ua.khpi.service.JMSReceiver;
import ua.khpi.service.JMSSender;

public class JMSController extends HttpServlet {
	private static final long serialVersionUID = 1L;
```

```
	@Inject
	private JMSSender sender;

	@Inject
	private JMSReceiver receiver;

	@Override
	protected void doPost(HttpServletRequest request, HttpServletResponse
response)
			throws ServletException, IOException {
		String serviceName = request.getParameter(Constants.SERVICE);

		if (serviceName == null) {
			request.setAttribute(Constants.ERROR, "Service name can't
be empty!");
		} else {
			if (serviceName.equals(Constants.SENDER)) {
				String text = request.getParameter(Constants.TEXT);

				if (text == null || text.isEmpty()) {
					request.setAttribute(Constants.ERROR, "Text
can't be empty!");
				} else {
					sender.send(text);
				}
			} else if (serviceName.equals(Constants.RECEIVER)) {
				String text = receiver.receive();

				request.setAttribute(Constants.TEXT, text);
			} else {
				request.setAttribute(Constants.ERROR, "Invalid
service name!");
			}
		}

		request.getRequestDispatcher(Constants.HOME).forward(request,
response);
	}
}
```

В отчете необходимо кратко описать основные этапы выполнения лабораторной работы, дать ответы на контрольные вопросы.

Контрольные вопросы

1. Какие интерфейсы появились в JMS 2.0?

2. Назовите основные отличительные особенности использования JMS 2.0 для отправки сообщений?

3. Назовите основные отличительные особенности использования JMS 2.0 для синхронного получения сообщений?

4. Назовите основные отличительные особенности использования JMS 2.0 для асинхронного получения сообщений?

5. С какой целью можно применять аннотацию @Inject при работе с JMS 2.0 в Java EE приложении?

6. Каким образом происходит закрытие JMSContext?

Дополнительные источники информации

1. Deakin, N. What's New in JMS 2.0, Part One: Ease of Use. http://www.oracle.com/technetwork/articles/java/jms20-1947669.html

2. Richards, M., Richard M. H., David A. C. (2009). Java Message Service, Second Edition. O'Reilly.

3. GlassFish Wiki. https://wikis.oracle.com/display/GlassFish/GlassFish Wiki

4. Арун Гупта. Java EE 7. Основы = Java EE 7 Essentials. – М.: «Вильямс», 2014. – 336 с. – ISBN 978-5-8459-1896-3.

5. Браун К.; Крейг Г.; Хестер Г. и др. Создание корпоративных Java-приложений для IBM WebSphere. – Кудиц-Образ, 2005. – С. 860. – ISBN 5-9579-0061-3, 0-321-18579-X.