**ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ**

**ПО ДИСЦИПЛИНЕ**

**«РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ»**

Оглавление

[ЛАБОРАТОРНАЯ РАБОТА 1 4](#_Toc94719579)

[ВВЕДЕНИЕ 4](#_Toc94719581)

[1 Основы XML 4](#_Toc94719582)

[2 Основные составляющие XML-документа 4](#_Toc94719583)

[3 Пространства имён XML 5](#_Toc94719584)

[4 Основы XSLT и XPATH 6](#_Toc94719585)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 9](#_Toc94719586)

[ЛАБОРАТОРНАЯ РАБОТА 2 12](#_Toc94719587)

[ВВЕДЕНИЕ 12](#_Toc94719589)

[5 Понятие сервлета. Упрощённая схема работы сервлета 12](#_Toc94719590)

[6 Создание сервлета средствами NetBeans 12](#_Toc94719591)

[7 Некоторые особенности написания сервлетов 14](#_Toc94719592)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 16](#_Toc94719593)

[ЛАБОРАТОРНАЯ РАБОТА 3 17](#_Toc94719594)

[ВВЕДЕНИЕ 17](#_Toc94719596)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 17](#_Toc94719597)

[ЛАБОРАТОРНАЯ РАБОТА 4 18](#_Toc94719598)

[ВВЕДЕНИЕ 18](#_Toc94719600)

[8 Основные понятия 18](#_Toc94719601)

[9 Алгоритм создания программ, работающих с JMS и типы JMS-сообщений 18](#_Toc94719602)

[10 Создание простейшего приложения, использующего JMS средствами NetBeans (модель «от пункта к пункту») 19](#_Toc94719603)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 21](#_Toc94719604)

[ЛАБОРАТОРНАЯ РАБОТА 5 22](#_Toc94719605)

[ВВЕДЕНИЕ 22](#_Toc94719607)

[11 Основные понятия 22](#_Toc94719608)

[12 Создание и тестирование веб службы в IDE NetBeans 23](#_Toc94719609)

[ЛАБОРАТОРНАЯ РАБОТА 6 25](#_Toc94719610)

[ВВЕДЕНИЕ 25](#_Toc94719612)

[13 Понятие AJAX. Сравнение стандартного подхода и AJAX-подхода 25](#_Toc94719613)

[14 AJAX-запрос 25](#_Toc94719614)

[15 Создание приложения, использующего AJAX, средствами NetBeans 25](#_Toc94719615)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 26](#_Toc94719616)

[JAVAFX 27](#_Toc94719618)

[ВВЕДЕНИЕ 27](#_Toc94719619)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 27](#_Toc94719620)

[ЛАБОРАТОРНАЯ РАБОТА 8 28](#_Toc94719621)

[ВВЕДЕНИЕ 28](#_Toc94719623)

[16 Понятие EJB. EJB как компонент и как фреймворк 28](#_Toc94719624)

[17 Архитектура Enterprise-приложения 28](#_Toc94719625)

[18 Создание бина средствами NetBeans 29](#_Toc94719626)

[19 Создание веб-приложения для уже созданного бина средствами NetBeans 30](#_Toc94719627)

[20 Создание клиентского приложения для уже созданного бина средствами NetBeans 31](#_Toc94719628)

[ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ 31](#_Toc94719629)

# ЛАБОРАТОРНАЯ РАБОТА 1

# ПРИМЕНЕНИЕ ТЕХНОЛОГИЙ XML/XSLT ДЛЯ ОФОРМЛЕНИЯ HTML-СТРАНИЦ

# ВВЕДЕНИЕ

## Основы XML

XML - расширяемый язык разметки, описывающий целый класс объектов данных, называемых XML-документами. [1] Назначение XML состоит в описании данных. Как и HTML, XML содержит теги, однако тэги XML не заданы в этом языке с самого начала и определяются пользователем при помощи DTD (Document Type Definition - определение типа документа).

XML документ описывает данные, их значение и структуру. Для определения данных используются тэги. Тэги определяются пользователем.

Простейший тег имеет вид (угловые скобки – элемент синтаксиса):

<имя>значение</имя>

Название тега может быть произвольным. После названия тега могут быть записаны произвольные атрибуты в виде пар <имя>="<значение>". Несколько атрибутов отделяются пробелом друг от друга. Комментарии заключаются в «скобки» <!-- и -->.

Для того, чтобы включить в документ символ, используемый для определения каких-либо конструкций языка (например, символ угловой скобки) и не вызвать при этом ошибок в процессе разбора такого документа, нужно использовать его специальный символьный либо числовой идентификатор. Например, &lt; , &gt; &quot; или &#036;(десятичная форма записи), &#x1a (шестнадцатеричная) и т.д.

## Основные составляющие XML-документа

В общем случае XML-документ должен удовлетворять следующим требованиям:

1. В заголовке документа помещается объявление XML, в котором указывается язык разметки документа, номер его версии и дополнительная информация;
2. Каждый открывающий тэг, определяющий некоторую область данных в документе обязательно должен иметь своего закрывающего "напарника", т.е., в отличие от HTML, нельзя опускать закрывающие тэги;
3. В XML учитывается регистр символов;
4. Все значения атрибутов, используемых в определении тэгов, должны быть заключены в кавычки;
5. Вложенность тэгов в XML строго контролируется, поэтому необходимо следить за порядком следования открывающих и закрывающих тэгов;
6. Вся информация, располагающаяся между начальным и конечными тэгами, рассматривается в XML как данные и поэтому учитываются все символы форматирования ( т.е. пробелы, переводы строк, табуляции не игнорируются, как в HTML).

Обычно XML-документ начинается со строки (первая и последняя угловая скобка – элементы синтаксиса)

<?xml version="1.0" encoding="<кодировка>"?>

В качестве кодировки обычно используют UTF-8.

После этой строки может идти объявление типа документа и разметки

<! DOCTYPE <имя корневого тега> [

<!ELEMENT <имя тега> (<список имен дочерних тегов>)>

...

<!ATTLIST <имя тега> <имя атрибута> <тип> <значение по умолчанию (опц)> <обязательность> >

]>

Если дочерних тегов нет, то вместо списка имен дочерних тегов ставится #PCDATA.

В качестве типа может выступать перечисление, элементы которого указываются в скобках и отделяются друг от друга символом |. В качестве типа также могут выступать CDATA - любые символьные данные, ID - уникальный идентификатор элемента в документе и некоторые другие, представленные в [2].

В качестве обязательности может выступать #REQUIRED – атрибут обязателен, #IMPLIED – атрибут не обязателен.

Разметка может быть объявлена в другом файле:

<!DOCTYPE <имя корневого тега> SYSTEM "<имя dtd-файла>">

Вообще говоря, объявление типа документа и разметки может быть опущено.

После объявления типа документа идёт корневой тег, внутри которого располагаются все остальные теги документа. Несколько корневых тегов не допускается. XML-документ фактически является древовидной структурой.

## Пространства имён XML

Пространство имён – поименованная область, служащая для логического группирования данных.

Пространства имён используются, чтобы избежать совпадения (конфликта) имён.

Формат объявления пространства имён в XML (внешние угловые скобки – элемент синтаксиса)

<<префикс>:<элемент> xmlns:<префикс>="<URI>">

<!--...-->

</<префикс>:<элемент>>

Префикс служит для обращения к элементам пространства имён. Префиксы не определяют элемент к той или иной схеме. Это делают уникальные идентификаторы, которые поставлены в соответствие этим префиксам - URI. Допускается несколько префиксов, соответствующих одной схеме (т.е. одному и тому же URI).

Стоит заметить, что не обязательно, чтобы по адресу, указанному в URI, что-то находилось. [3]

Элементы, находящиеся внутри пространства имён, снабжаются его префиксом:

<<префикс>:<элемент>>

<!--...-->

</<префикс>:<элемент>>

На имя префикса пространства имен накладываются следующие ограничения: [4]

* не может иметь значение xml, т.к. это значение зарезервировано за префиксом, привязанным к имени пространства имен http://www.w3.org/XML/1998/namespace;
* не может иметь значение xmlns, т.к. данное имя зарезервировано за атрибутом объявления пространств имен;
* не должно начинаться последовательностью трех символов x, m, l (независимо от регистра).

## Основы XSLT и XPATH

XSLT - язык преобразования XML-документов.

Таблица стилей XSLT (XSLT-преобразование) – XML-документ, с использованием которого осуществляется преобразования исходного XML-документа в конечное дерево, которое может быть сериализовано в виде XML-документа, XHTML-документа (только для XSLT 2.0), HTML-документа или простого текстового файла.

Все элементы, относящиеся к XSLT, имеют префикс "xsl:". Это инструкции процессору, осуществляющему преобразование. Корневым тегом любого XSLT-преобразования является тег

<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

Преобразования XSLT основаны на шаблонах. Шаблон определяется инструкцией xsl:template, поэтому первым тегом внутри xsl:stylesheet является тег

<xsl:template match="/">

указывающий правило, по которому должен быть преобразован корневой узел. Атрибут match="/" указывает на то, что значение пути адресации соответствует корневому узлу.

XSLT-процессор анализируют исходный документ и пытается найти подходящий XSL-шаблон. Если такой шаблон найден, то выполняются инструкции внутри него.

Правила выбора (и, отчасти, преобразования) данных из исходного дерева пишутся на языке запросов XPATH. [5] Самые простые выражения XPATH похожи на пути в файловых системах, только в случае XPATH имеется в виду «путь» от корневого тега к требуемому.

Построение XSLT-преобразования XML-документа в HTML-документ состоит из двух этапов:

1. Построение «каркаса» HTML-документа при помощи HTML-тегов;
2. Добавление в теги информации из XML-документа путём использования XPATH-запросов.

При выполнении заданий учитывается, что кодировка – UTF-8. Поэтому все преобразования имеют вид

<xsl:stylesheet version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

<xsl:template match="/">

<html>

<head>

<meta charset="UTF-8"></meta>

</head>

<body>

<!-- содержимое HTML-страницы, полученное -->

<!-- при помощи XPATH из исходного XML -->

</body>

</html>

</xsl:template>

</xsl:stylesheet>

Некоторые запросы языка XPATH представлены в таблице 1.

Таблица 1. Некоторые запросы языка XPATH

|  |  |
| --- | --- |
| Тег, содержащий запрос XPATH | Назначение |
| <xsl:value-of select="<путь от корневого тега>"/> | Вставляет значение выбранного тега в виде текста [6]. Если требуется вставка значения атрибута, то к имени тега добавляется /@<имя атрибута>. |
| <xsl:text><строка></xsl:text> | Подставляет указанную строку. |
| <xsl:copy-of select="<путь от корневого тега>"/> | Выполняет копирование поддерева в виде текста. Если обрамляющий тег поддерева не нужен, то к имени тега добавляется /\*. |
| {<путь>} | Выводит содержимое, указанное в пути. Используется для ввода атрибутов. |
| <xsl:for-each select="<множество>"> | Осуществляет перебор по элементам заданного множества (по поводу множеств см. далее). |
| <xsl:if test = <логическое выражение>> |  |

В качестве значений атрибутов match и select используются выражения, представленные в таблице 2, синтаксис которых похож на маршрут файловой системы.

Таблица 2. Атрибуты для match и select

|  |  |
| --- | --- |
| Выражение | Описание |
| / | Корневой узел |
| . | Текущий узел |
| .. | Родительский узел текущего узла |
| fruit | Узел fruit |
| fruit/lime | Подузел lime узла fruit |
| fruit/\* | Все потомки узла fruit |
| /fruit | Узел fruit, являющийся прямым потомком корневого узла |
| ./@taste | Атрибут taste текущего узла |
| ./@\* | Все атрибуты текущего узла |
| fruit@taste | Атрибут taste узла fruit |
| fruit/lime/@taste | Атрибут taste узла lime, являющегося подузлом узла fruit |
| ../@taste | Атрибут taste родительского узла |
| // | Любое количество промежуточных узлов |
| fruit//lime | Все узлы lime, имеющие предка fruit |
| | | Знак разделения конкретных узлов |
| lime|grape | Узел lime и узел grape |
| [] | Предикатное выражение |
| fruit[lime] | Узел fruit, имеющий потомка lime |
| fruit[lime="fine"] | Узел fruit, имеющий потомка lime, значение которого равно fine |
| fruit[@taste] | Узел fruit, имеющий атрибут taste |
| fruit[@taste="5"] | Узел fruit, имеющий атрибут taste, значение которого равно 5 |
| count(fruit/\*) | Количество потомков узла fruit |
| name() | Возвращает имя текущего узла |
| position() | Возвращает позицию (индекс) текущего узла |
| first() | Проверяет, является ли текущий узел первым во множестве |
| last() | Проверяет, является ли текущий узел последним во множестве |

Команды типа fruit/\* возвращают множества. К конкретному элементу множества можно обратиться при помощи операции индексации (нумерация с 1), например fruit/\*[2] возвратит второго по счёту потомка узла fruit.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Построить XSLT-преобразования, осуществляющие преобразование исходных xml-документов в html-документы согласно заданиям 1 - 7. Убедится в том, что сгенерированные html-документы корректно отображаются браузером.

Задание 1. Применить указанный цвет к надписи «название книги»

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <library>  <book>Название книги</book>  <color>red</color>  </library> | <font color="red">Название книги</font> |

Задание 2. Выделить все элементы первого порядка (элементом нулевого порядка считать корневой элемент). Один абзац – один элемент.

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <root>  <XXX id="x1">  <BBB id="b1"/>  <BBB id="b2"/>  </XXX>  <AAA id="a2">  <BBB id="b3"/>  <BBB id="b4"/>  <CCC id="c1">  <DDD id="d1"/>  </CCC>  <BBB id="b5">  <CCC id="c2"/>  </BBB>  </AAA>  </root> | <p>XXX id=x1</p>  <p>AAA id=a2</p> |

Задание 3. Вывести имена после id. Выделить id курсивом и жирным.

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <employees>  <employee id="js0034">  Joe Smith</employee>  <employee id="ma0012">  Marry Ann</employee>  </employees> | <b><i>js0034</i></b> Joe Smith  <b><i>ma0012</i></b> Marry Ann |

Задание 4. Вывести таблицу, в которой указывается id родителя (если родителя нет, то ничего не выводить).

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <root>  <AAA id="a1">  <BBB id="b1"/>  <BBB id="b2"/>  </AAA>  <AAA id="a2">  <BBB id="b3"/>  <BBB id="b4"/>  <CCC id="c1">  <DDD id="d1"/>  </CCC>  <BBB id="b5">  <CCC id="c2"/>  </BBB>  </AAA>  </root> | <table border="1" cellpadding="6">  <tr><th colspan="2">Axis: child</th></tr>  <tr><th>Element</th><th>Node-set</th></tr>  <tr><td>AAA id = a1</td><td> </td></tr>  <tr><td>BBB id = b1</td><td>a1 </td></tr>  <tr><td>BBB id = b2</td><td>a1 </td></tr>  <tr><td>AAA id = a2</td><td> </td></tr>  <tr><td>BBB id = b3</td><td>a2 </td></tr>  <tr><td>BBB id = b4</td><td>a2 </td></tr>  <tr><td>CCC id = c1</td><td>a2 </td></tr>  <tr><td>CCC id = c2</td><td>c1 </td></tr>  <tr><td>BBB id = b5</td><td>a2 </td></tr>  <tr><td>CCC id = c3</td><td>b5 </td></tr>  </table> |

Задание 5. Вывести id, относящиеся к BBB красным цветом, к CCC – синим цветом, к DDD – зелёным цветом.

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <root>  <AAA id="a1">  <BBB id="b1"/>  <BBB id="b2"/>  </AAA>  <AAA id="a2">  <BBB id="b3"/>  <BBB id="b4"/>  <CCC id="c1">  <DDD id="d1"/>  </CCC>  <BBB id="b5">  <CCC id="c2"/>  </BBB>  </AAA>  </root> | <font color="red">BBB id=b1</font>  <font color="red">BBB id=b2</font>  <font color="red">BBB id=b3</font>  <font color="red">BBB id=b4</font>  <font color="red">BBB id=b5</font>  <font color="blue">CCC id=c1</font>  <font color="green">DDD id=d1</font> |

Задание 6. Раскрасить название цвета в соответствующий цвет.

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <root>  <color>blue</color>  <color>navy</color>  <color>yellow</color>  <color>green</color>  <color>lime</color>  <color>red</color>  </root> | <TABLE><TR><TD style="color:blue">blue</TD></TR></TABLE>  <TABLE><TR><TD style="color:navy">navy</TD></TR></TABLE>  <TABLE><TR><TD style="color:yellow">yellow</TD></TR></TABLE>  <TABLE><TR><TD style="color:green">green</TD></TR></TABLE>  <TABLE><TR><TD style="color:lime">lime</TD></TR></TABLE>  <TABLE><TR><TD style="color:red">red</TD></TR></TABLE> |

Задание 7. Добавить точку после последнего имени.

|  |  |
| --- | --- |
| Исходник (XML) | Результат (HTML) |
| <list>  <entry name="A"/>  <entry name="B"/>  <entry name="C"/>  <entry name="D"/>  </list> | <p>A, B, C, D.</p> |

Преобразования выполнять с помощью кода:

import javax.xml.transform.\*;

import javax.xml.transform.stream.StreamResult;

import javax.xml.transform.stream.StreamSource;

import java.io.File;

public class XmlToHtmlWithXSLT

{

public static void main(String[] args)

{

Source xmlInput =

new StreamSource(new File("src/input.xml"));

Source xsl =

new StreamSource(new File("src/transformer.xsl"));

Result xmlOutput =

new StreamResult(new File("src/output.html"));

try

{

Transformer transformer =

TransformerFactory.newInstance().

newTransformer(xsl);

transformer.transform(xmlInput, xmlOutput);

}

catch (TransformerException e)

{

System.out.println("Tranform has failure");

}

}

}

# ЛАБОРАТОРНАЯ РАБОТА 2

# СЕРВЛЕТЫ

# ВВЕДЕНИЕ

## Понятие сервлета. Упрощённая схема работы сервлета

Сервлет – небольшая программа, которая выполняется на серверной стороне Web-соединения.

Сервлет является классом Java, реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ. [1]

Контейнер сервлетов — программа, представляющая собой сервер, который занимается системной поддержкой сервлетов и обеспечивает их жизненный цикл в соответствии с правилами, определёнными в спецификациях.[2]

Жизненный цикл сервлета состоит из следующих шагов:

1. В случае отсутствия сервлета в контейнере класс сервлета загружается контейнером, после чего контейнер создает экземпляр класса сервлета. После создание объекта класса сервлета контейнер вызывает метод init(). Этот метод инициализирует сервлет и вызывается в первую очередь, до того, как сервлет сможет обслуживать запросы. За весь жизненный цикл метод init() вызывается только один раз.
2. Обслуживание клиентского запроса. Каждый запрос обрабатывается в своем отдельном потоке. Контейнер вызывает метод service() для каждого запроса. Этот метод определяет тип пришедшего запроса и распределяет его в соответствующий этому типу метод для обработки запроса. Разработчик сервлета должен предоставить реализацию для этих методов. Если поступил запрос, метод для которого не реализован, вызывается метод родительского класса и обычно завершается возвращением ошибки инициатору запроса.
3. В случае если контейнеру необходимо удалить сервлет, он вызывает метод destroy(), который снимает сервлет из эксплуатации. Подобно методу init(), этот метод тоже вызывается единожды за весь цикл сервлета.

## Создание сервлета средствами NetBeans

Создать сервлет в NetBeans можно, выполнив следующие шаги:

1. Выбрать создание нового проекта командой Файл | Создать новый проект;
2. В открывшемся окне выбора проектов выбрать категорию «Java Web» и проект «веб-приложение», после чего нажать кнопку «Далее»;
3. В открывшемся окне выбора имени и расположения ввести имя и расположение проекта в соответствующие поля, после чего нажать кнопку «Далее»;
4. В окне выбора сервера и параметров настройки указываем в поле «Сервер» путь до каталога, в котором расположен сервер веб-приложений, после чего нажать кнопку «Готово».

После создания проекта осуществляем добавление сервлета в проект, выполнив следующие шаги:

1. Кликнуть ПКМ по названию проекта в окне «Проекты» и выполнить команду Новый | Сервлет;
2. В открывшемся окне выбора имени и положения сервлета ввести имя класса и имя пакета в соответствующие поля, после чего нажать кнопку «Далее»;
3. В открывшемся окне настройки развертывания сервлета поставить галочку напротив строки «Добавление строки к дескриптору развёртывания» , после чего нажать кнопку «Готово».

Созданный проект содержит специальный файл, отвечающий за развертывание приложения на сервере - дескриптор развёртывания. Этот файл хранится по адресу <имя проекта>\Веб-страницы\WEB-INF\web.xml. При использовании сервера GlassFish 4.1 этот файл после добавления в пакет pkg сервлета NewServlet имеет вид:

<?xml version="1.0" encoding="UTF-8"?>

<web-app

version="3.1"

xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee

http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd">

<servlet>

<servlet-name>NewServlet</servlet-name>

<servlet-class>pkg.NewServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>NewServlet</servlet-name>

<url-pattern>/NewServlet</url-pattern>

</servlet-mapping>

<session-config>

<session-timeout>

30

</session-timeout>

</session-config>

</web-app>

Стоит выделить несколько моментов [3]:

1. Тэг web-app является корневым тэгом. Он может содержать дополнительные аттрибуты, но нам они в данный момент не нужны. Все остальыне тэги находятся внутри него.
2. Тэг servlet содержит внутри себя тэги, которые описывают сервлет. Тэг servlet-name содержит внутри себя какое-то имя – вы можете использовать то, что вам больше нравится. Тэг servlet-class должен содержать реальное имя класса. В нашем случае мы поместили наш класс в пакет students.web и полное имя класса будет students.web.HelloWorldServlet.
3. Тэг servlet-mapping. Здесь мы определяем URL, который пользователь будет запрашивать для получения данных от нашего сервлета. Точнее будет сформулировать так: мы определяем какой сервлет будет обрабатывать введенный URL.

Для того, чтобы после запуска веб-приложения запустить именно сервлет NewServlet, следует дописать в адресную строку / NewServlet.

## Некоторые особенности написания сервлетов

В задачу метода service() класса HttpServlet входит анализ полученного через запрос HTTP-метода доступа к ресурсам и вызов метода, имя которого сходно с названием метода доступа к ресурсам, но перед именем добавляется префикс do: doGet() или doPost(). Кроме этих методов, могут использоваться методы doHead(), doPut(), doDelete(), doOptions() и doTrace(). Разработчик должен переопределить нужный метод, разместив в нем функциональную логику. [4]

Вообще говоря, достаточно переопределить нужные нам из методов doGet(), doPost(),doHead(), doPut(), doDelete(), doOptions(), doTrace(). Метод service() можно не переопределять. Метод service() имеет смысл переопределять только в том случае, если реакция сервлета одинакова, например, и на метод GET и на метод POST. [5]

Таким образом, каркас простейшего сервлета имеет вид:

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Enumeration;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

@WebServlet(name = "MS", urlPatterns = {"/MS"})

public class MS extends HttpServlet {

@Override

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws IOException {

request.setCharacterEncoding("Cp1251");

response.setContentType("text/html;charset="

+ "Windows-1251");

PrintWriter out = response.getWriter();

// Пользовательский код

}

@Override

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws IOException {

request.setCharacterEncoding("Cp1251");

response.setContentType("text/html;charset="

+ "Windows-1251");

Enumeration paramNames = request.getParameterNames();

// Пользовательский код

}

}

При вызове двух главных методов сервлета HttpServlet (doPost() или doGet()) контейнер сервлета создает объекты javax.servlet.http.HttpServletRequest и javax.servlet.http.HttpServletResponse и передает их в качестве параметров этим методам обработки запроса. Объект HttpServletRequest представляет исходный запрос, а HttpServletResponse инкапсулирует ответ сервлета на этот запрос.

Для ликвидации проблем с неправильным отображением русского языка используются строки:

request.setCharacterEncoding("Cp1251");

response.setContentType("text/html;charset="Windows-1251");

PrintWriter – класс, метод println(String) которого используется для вывода HTML-текста на веб-станицу.

Перечисление request.getParameterNames() содержит названия имён передаваемых методу сервлета параметров. Значения самих параметров можно получить по их имени при помощи метода request.getParameter(String).

Занести имя параметра ptrname можно при помощи метода println("name=\" ptrname\"") класса PrintWriter, а значение ptrval - println("value=\" ptrval\"").

Практика включения HTML-кода в код сервлета не считается хорошей, так как эти действия “уводят” сервлет от его основной роли – контроллера приложения. Это приводит к разрастанию размеров сервлета, которое на определенном этапе становится неконтролируемым и реализует вследствие этого анти-шаблон “Волшебный сервлет”. Даже приведенный выше маленький сервлет имеет признаки анти-шаблона, так как содержит метод print(), используемый для формирования кода HTML. Сервлет должен использоваться только для реализации бизнес-логики приложения и обязан быть отделен как от непосредственного формирования ответа на запрос, так и от данных, необходимых для этого. Обычно для формирования ответа на запрос применяются возможности JSP, JSPX или JSF.

Примеры сервлетов даны в 18 главе книги [6].

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Реализовать сервлет, который при запуске приложения (метод GET) отображал страницу, позволяющую ввести своё имя, фамилию и e-mail в текстовые поля и по нажатию кнопки «отправить» (метод POST) выводил бы страницу, где показывалось, что именно было введено. Если какое-то поле было оставлено пустым, то выводить сообщение о том, что данное поле не заполнялось.

# ЛАБОРАТОРНАЯ РАБОТА 3

# JAVA SERVER PAGES

# ВВЕДЕНИЕ

JSP (JavaServer Pages) — технология, позволяющая веб-разработчикам создавать содержимое, которое имеет как статические, так и динамические компоненты. Страница JSP содержит текст двух типов: статические исходные данные, которые могут быть оформлены в одном из текстовых форматов HTML, SVG, WML, или XML, и JSP- элементы, которые конструируют динамическое содержимое.[1]

JSP при первом обращении преобразуется в сервлет и работает уже как сервлет. [2]

JSP позволяет встраивать в себя код на языке Java. Встраиваемый код заключается в символы <% и %>. После <% следует @page – если нужно использовать import, ! – для функций на Java и = - для добавления в тег или атрибут тега значения переменной Java или результата выполнения Java-функции.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Создать JSP, который печатает строку текста, использующую тэг <H1>. Установить цвет этого текста случайным образом, используя код, встроенный в JSP страницу.

# ЛАБОРАТОРНАЯ РАБОТА 4

# JAVA MESSAGE SERVICE

# ВВЕДЕНИЕ

## Основные понятия

Java Message Service - стандарт промежуточного ПО для рассылки сообщений, позволяющий приложениям, выполненным на платформе Java EE, создавать, посылать, получать и читать сообщения. [1]

Коммуникация между компонентами, использующими JMS, асинхронна (процедура не дожидается ответа на своё сообщение) и независима от исполнения компонентов.

JMS поддерживает две модели обмена сообщениями: «от пункта к пункту» (объект Queue) и «издатель-подписчик» (объект Topic).

Модель «от пункта к пункту» характеризуется следующим:

1. Каждое сообщение имеет только одного адресата
2. Сообщение попадает в «почтовый ящик», или «очередь» адресата и может быть прочитано когда угодно. Если адресат не работал в момент отсылки сообщения, сообщение не пропадёт.
3. После получения сообщения адресат посылает извещение.

Модель «издатель-подписчик» характеризуется следующим:

1. Подписчик подписывается на определённую «тему»
2. Издатель публикует своё сообщение. Его получают все подписчики этой темы
3. Получатель должен работать и быть подписан в момент отправки сообщения

## Алгоритм создания программ, работающих с JMS и типы JMS-сообщений

Алгоритм создания программ, работающих с JMS: [2]

1. Подключаемся к серверу, используя ConnectionFactory.
2. Получаем соединение Connection из ConnectionFactory.
3. Создаем однопоточный контекст Session из соединения.
4. Получаем буфер Destination привязанный к определенному адресу для создания интерфейсов отправки и получения сообщений.
5. Создание объектов MessageProducer для отправки или MessageConsumer для получения сообщений.
6. Отдельно идет этап создания сообщения для отправки.

Существуют следующие типы JMS-сообщений:

1. StreamMessage - поток примитивных типов Java, считывать можно со стандартных интерфейсов ввода/вывода;
2. MapMessage - содержит информацию на подобии коллекций в виде ключ-значение (String, Object);
3. TextMessage - обычное, текстовое сообщение;
4. ObjectMessage - для передачи Serializable-объектов, можно передавать сразу классы;
5. BytesMessage - список не интерпретированных байт, можно передавать файлы.

## Создание простейшего приложения, использующего JMS средствами NetBeans (модель «от пункта к пункту»)

Для создания простейшего приложения, использующего JMS средствами NetBeans в первую очередь следует настроить сервер сообщений, для этого следует:

1. Перейти на вкладку «Службы»;
2. Найти в узле «Серверы» нужный сервер (в случае NetBeans это, скорее всего, GlassFish);
3. Щелкнуть по серверу ПКМ, выбрать пункт «Просмотр консоли администратора домена»;
4. Выбрать в открывшемся окне узел Resources\JMS Resources\Connection Factories
5. Добавить новую Connection Factory, причём Resource Type - javax.jms.TopicConnectionFactory;
6. Выбрать в консоли администратора домена узел Resources\JMS Resources\Destination Resources;
7. Добавить новый Destination Resource, причём Resource Type - javax.jms.Queue.

После настройки сервера, создаём новый проект Enterprise Application, при создании указываем создать модули EJB и веб-приложения.

Следующим шагом является создание MDB. Переходим к модулю EJB созданного проекта и добавляем туда компонент, управляемый сообщениями, причём выбираем Server Destination, в качестве которого выбираем созданный ранее Destination Resource.

Передача текстового сообщения реализуется в методе onMessage класса созданного бина:

@Override

public void onMessage(Message message) {

TextMessage tmsg = null;

tmsg = (TextMessage)message;

try {

System.out.println(tmsg.getText());

} catch(JMSException ex){

Logger.getLogger(MBean.class.getName()).

log(Level.SEVERE, null, ex);

}

Здесь MBean – имя бина.

После создания бина создаём JSP, которая будет обрабатывать введённое нами сообщение. Она должна содержать теги form с атрибутом action="NewServlet" (NewServlet – имя сервлета, передающего введённое сообщение-строку MDB из JSP) и input, одним из атрибутов является name="msg".

Наконец, пишем сервлет, передающий введённое сообщение-строку MDB из уже созданной JSP, для чего следует:

1. Перейти к уже созданному модулю веб-приложения и добавить туда сервлет;
2. Добавляем пустую строку после строки response.setContentType("text/html;charset=UTF-8") метода processRequest класса созданного сервлета;
3. Кликаем ПКМ по этой строке, выбираем пункт «Вставка кода…»;
4. В открывшемся окне выбираем «Отправка сообщения JMS…»;
5. В окне отправки сообщения JMS выбираем Message Driven Bean, куда вводим имя ранее созданного бина, а в качестве Connection Factory вводим имя созданной ранее Connection Factory.

Для передачи MDB введённого в JMS сообщения следует добавить в метод processRequest после строки response.setContentType("text/html;charset=UTF-8") следующий код:

String str = request.getParameter("msg");

try {

sendJMSMessageToDest(str);

} catch (JMSException ex) {

Logger.getLogger(NewServlet.class.getName()).

log(Level.SEVERE, null, ex);

}

После запуска Enterprise Application и ввода строки-сообщения в предложенную форму будет выведена HTML-страница, сгенерированная разработанным сервлетом, передающим введённое сообщение-строку MDB, а само введённое сообщение появится в логе сервера.

Замечание: в реальных "боевых" условиях, поставляемый стандартный функционал JMS от GlassFish практически не используется. Как правило, сервер приложений и сервер сообщений - это абсолютно два разных процесса и расположены на разных машинах. Чаще всего вам придется сталкиваться с другими популярными MQ серверами, такими как Apache ActiveMQ Server или WebShpere MQ Server.

По большому счету самой программе (не MQ серверу) неважно какому из серверов отправлять сообщения, внутренний код программы не изменится, или изменится незначительно, ведь JMS - это спецификация, и методы работы с JMS меняться не должны.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Cоздать приложение, отправляющее сообщение в queue, и message-driven bean, читающий отправленное в queue сообщение.

# ЛАБОРАТОРНАЯ РАБОТА 5

# ВЕБ-СЛУЖБЫ

# ВВЕДЕНИЕ

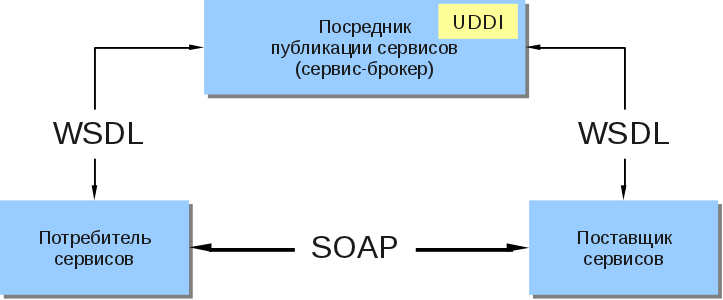
## Основные понятия

Веб-служба (веб-сервис, web service) - идентифицируемая веб-адресом (URI) программная система со стандартизированными интерфейсами. [1]

Главные особенности веб-служб:

1. Веб-службы - самодостаточны, т.е. с клиентской стороны не требуется никакого дополнительного программного обеспечения кроме языка программирования, поддерживающего работу с XML и HTTP, а на серверной стороне требуются только HTTP-сервер, поддерживающий работу с посланиями.
2. Веб-службы являются самоописываемыми, поскольку метаданные, описывают сообщения передается вместе с сообщением и не требуется никаких внешних хранилищ метаданных.

Концепция веб-служб представлена на рисунке ниже [2]



Можно выделить три инстанции, взаимодействующие в рамках веб-службы:

* заказчик;
* исполнитель;
* каталог.

Когда служба разработана, исполнитель регистрирует её в каталоге, где её могут найти потенциальные заказчики. Заказчик, найдя в каталоге подходящую службу, импортирует оттуда её WSDL-спецификацию и разрабатывает в соответствии с ней свое программное обеспечение. WSDL описывает формат запросов и ответов, которыми обмениваются заказчик и исполнитель в процессе работы.

Для обеспечения взаимодействия используются следующие стандарты [1]:

* XML: Расширяемый язык разметки, предназначенный для хранения и передачи структурированных данных;
* SOAP: Протокол обмена сообщениями на базе XML;
* WSDL: Язык описания внешних интерфейсов веб-службы на базе XML;
* UDDI: Универсальный интерфейс распознавания, описания и интеграции - каталог веб-служб и сведений о компаниях, предоставляющих веб-службы во всеобщее пользование или конкретным компаниям.

## Создание и тестирование веб службы в IDE NetBeans

Создать веб-службу в NetBeans можно путём создания нового веб-приложения и добавления к созданному проекту веб-службы. При создании проекта на сервере GlassFish следует выбрать «Реализовать веб-службу в качестве сеансового компонента сеанса без сохранения состояния».

Созданная веб-служба имеет вид

import javax.ejb.Stateless;

import javax.jws.WebService;

import javax.jws.WebMethod;

import javax.jws.WebParam;

@WebService(serviceName = "NewWebService") // (1)

@Stateless()

public class NewWebService {

@WebMethod(operationName = "hello") // (2)

public String hello(

@WebParam(name = "name") // (3)

String txt) {

return "Hello " + txt + " !";

}

}

Функционал веб-службы описывается в виде набора операций класса, перед описанием которого содержится строка, подобная строке (1). Перед методами, представляющими функционал веб-службы, записывается строка, подобная строке (2) листинга. Перед каждым аргументом метода записывается строка, подобная строке (3) листинга.

После того, как код веб-службы написан, её можно протестировать, если используемый сервер приложений поддерживает подобный функционал (GlassFish, идущий с NetBeans, поддерживает тестирование веб-служб). Для этого следует:

1. Кликнуть ПКМ по проекту, в котором находится тестируемая веб-служба и выполнить команду «развернуть»;
2. Если разворачивание прошло успешно, то в дереве проекта найти нужную службу среди списка веб-служб, кликнуть ПКМ по ней и выполнить команду «Тестировать веб-службу».
3. После этого в браузере откроется окно, где можно осуществить непосредственное тестирование.

Клиент веб-службы представляет собой реализованный в отдельном проекте сервлет, в который требуемая веб-служба была добавлена как «клиент веб-службы».

При написании кода сервлета-клиента в редакторе следует перетащить требуемые методы веб-службы из узла «Ссылки на веб-службы» в любое место тела класса сервлета-клиента. Это добавит к классу сервлета путь к WSDL веб-службы, новое поле service и новый метод, заголовок которого совпадает с заголовком метода веб-службы. Именно через этот метод и осуществляется обращение к веб-службе.

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Разработать веб-службу, осуществляющую конвертацию некоторой целой величины amount в величину amount\*rate, целое amount и вещественное rate задаётся пользователем.

Разработать сервлет-клиент для данной службы.

# ЛАБОРАТОРНАЯ РАБОТА 6

# AJAX

# ВВЕДЕНИЕ

## Понятие AJAX. Сравнение стандартного подхода и AJAX-подхода

AJAX - подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. [1]

В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения работают быстрее.

В классической модели веб-приложения:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
2. Браузер формирует и отправляет запрос серверу;
3. В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д. После чего браузер полностью перегружает всю страницу.

При использовании AJAX:

1. Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент;
2. Скрипт (на языке JavaScript) определяет, какая информация необходима для обновления страницы;
3. Браузер отправляет соответствующий запрос на сервер;
4. Сервер возвращает только ту часть документа, на которую пришел запрос;
5. Скрипт вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

## AJAX-запрос

В основе работы всех методов AJAX лежит метод jQuery.ajax(), который и осуществляет запрос к серверу без перезагрузки страницы. [2]

Форматы данного метода:

jQuery.ajax(url,[settings]):jqXHR

jQuery.ajax(settings):jqXHR

Отличие второй формы метода от первой заключается лишь в том, что свойство url здесь является частью настроек, а не отдельным параметром.

Список настроек с описанием приведён в [2].

## Создание приложения, использующего AJAX, средствами NetBeans

Приложение, использующего AJAX, создаётся средствами NetBeans как обычное приложение.

К приложению, использующему AJAX, следует добавить библиотеку jQuery. После этого в JSP после тега title следует прописать следующее

<script

src="<путь к jquery.js>"

type="text/javascript">

</script>

<script

src="<путь к js-скрипту>"

type="text/javascript">

</script>

Причём корнем пути является узел «Веб-страницы» проекта.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Написать веб-приложение, поддерживающее автодополнение c использованием AJAX.

# ЛАБОРАТОРНАЯ РАБОТА 7

# JAVAFX

# ВВЕДЕНИЕ

JavaFX — платформа для создания RIA, позволяет строить унифицированные приложения с насыщенным графическим интерфейсом пользователя для непосредственного запуска из-под операционных систем, работы в браузерах и на мобильных телефонах, в том числе работающих с мультимедийным содержимым. [1]

Каркас приложения на JavaFX имеет вид

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

primaryStage.setTitle("Simple Application");

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}

Здесь класс Application является главным классом приложения, именно в его потоке можно использовать компоненты JavaFX.

В методе main, где находится точка входа в программу, вызывается метод launch(), он в свою очередь, является точкой входа в FX-приложение. Метод start() вызывается при создании потока приложения, в его параметрах можно увидеть объект класса Scene. Этот класс связан с экземпляром окна, которое будет видеть пользователь. [2]

Именно в метод start добавляется кнопки, текстовые поля и т.п. В остальном написание простейших FX-приложений напоминает написание обычного java приложения, например на Swing.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Создать простую формы для аутентификации с использованием JavaFX.

# ЛАБОРАТОРНАЯ РАБОТА 8

# EJB

# ВВЕДЕНИЕ

## Понятие EJB. EJB как компонент и как фреймворк

Enterprise JavaBeans (EJB) — спецификация технологии написания и поддержки серверных компонентов, содержащих бизнес-логику. [1]

С точки зрения компонента EJB - это всего лишь надстройка над классом, описываемая с помощью аннотации. Существует три типа компонентов EJB: [2]

* session beans - используется для описания бизнесс-логики приложения;
* message-driven beans - так же используется для бизнесс-логики;
* entities - используется для хранения данных.

С точки зрения фреймворка EJB - это технология, предоставляющая множество готовых решений (управление транзакциями, безопасность, хранение информации и т.п.) для приложения.

## Архитектура Enterprise-приложения

Архитектура Enterprise-приложения представлена на рис.1.

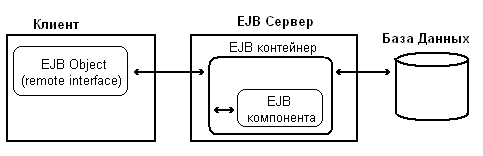


Рис 1. Архитектура Enterprise-приложения.

EJB-компонента представляет из себя Java-класс, который реализует некоторую бизнес-логику. Все остальные классы в EJB-системе либо реализуют поддержку клиент/сервер взаимодйествий между компонентами, либо реализуют некоторые сервисы для компонент. [3]

EJB-контейнер - это то место, где «живет» EJB-компонент. EJB-контейнер управляет жизненным циклом EJB-компонент, разработчик только использует соответствующие методы EJB-контейнера. Как правило, в одном EJB-контейнере живет несколько однотипных EJB-компонент.

Клиентские приложения вызывают методы на удаленных EJB-компонентах через EJB-объект, который реализует лишь бизнес-интерфейс для EJB-компоненты, являясь, в некотором смысле, «промежуточным» звеном между клиентом и EJB-компонентой (EJB-интерфейс).

В качестве поясняющего примера рассмотрим видеомагнитофон. Предположим, что он является EJB-компонентой. EJB-объект, в таком случае, является аналогом пульта управления этого магнитофона. У пульта управления есть все те же кнопки, что и на передней панели видеомагнитофона. Нажатие кнопки на пульте управления приведет к такому же эффекту, что и нажатие кнопки, расположенной на самом магнитофоне, но в результате только магнитофон, а не пульт управления, начнет выполнять функции.

Приложение-клиент соединяется с EJB-сервером и посылает ему запрос на создание бина для обработки своих запросов. Сервер отвечает на такой запрос созданием объекта на стороне сервера (экземпляр EJB-компоненты) и возвращает клиенту прокси-объект (EJB-объект), чей интерфейс совпадает с интерфейсом созданной EJB-компоненты и чьи методы перенаправляют вызовы собственно экземпляру компоненты. После этого приложение-клиент работает с EJB-объектом как с локальным объектом, даже и не подозревая, что всю работу выполняет не EJB-объект, а удаленная компонента на сервере. Необходимо заметить, что созданием и удалением EJB-компонент на сервере занимается EJB-контейнер.

С помощью EJB-компоненты нельзя:

* создавать и манипулировать с нитями;
* осуществлять ввод-вывод;
* выполнять код на C.

## Создание бина средствами NetBeans

EJB-интерфейс создаётся аналогично созданию библиотеки класса Java. Простейший EJB-интерфейс имеет вид:

package ejb;

import javax.ejb.Remote;

@Remote

public interface MySessionRemote {

// Заголовки методов

}

После создания EJB-интерфейса создаем Enterprise Application, при создании которого создаём EJB-модуль. В проект для EJB-модуля добавляем сеансовый компонент. При добавлении в качестве типа сессии указываем Stateless, для Create Interface выбираем Remote in project и указываем проект, содержащий EJB-интерфейс.

После создания сеансового компонента следует кликнуть ПКМ по строке внутри класса сеансового компонента, выбрать пункт контекстного меню «Вставить код», после чего ввести имя метода и тип возвращаемого значения. Имя и тип должны соответствовать таковым в интерфейсе (т.е. в классе сеансового компонента следует реализовывать методы EJB-интерфейса).

## Создание веб-приложения для уже созданного бина средствами NetBeans

После создания бина можно создать использующее его веб-приложение. Создаём веб-приложение, добавляем в него EJB-интерфейс, а само приложение добавляем в Enterprise Application.

JSP созданного веб-приложения имеет вид:

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@page import="ejb.MySessionRemote, javax.naming.\*"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<%!

private MySessionRemote s = null;

public void jspInit()

{

try {

InitialContext ic = new InitialContext();

s = (MySessionRemote) ic.lookup(

MySessionRemote.class.getName());

} catch (Exception ex)

{

System.out.println(

"Couldn't create bean."

+ ex.getMessage());

}

}

public void jspDestroy()

{

s = null;

}

%>

<!—код HTML-страницы --!>

Здесь ejb.MySessionRemote – имя EJB-заглушки, код которой лежит в пакете ejb.

В коде HTML-страницы используются методы EJB-объекта s.

После этого в узел Веб-страницы\WEB-INF веб-приложения следует добавить файл glassfish-web.xml со следующим содержанием:

<!DOCTYPE glassfish-web-app PUBLIC

"-//GlassFish.org//DTD

GlassFish Application Server 3.1 Servlet 3.0//EN"

"http://glassfish.org/dtds/glassfish-web-app\_3\_0-1.dtd">

<glassfish-web-app error-url="">

<context-root>/index.jsp</context-root>

</glassfish-web-app>

Перед запуском веб-приложения следует развернуть Enterprise Application на сервере. Для этого следует кликнуть ПКМ по Enterprise Application и выбрать в появившемся контекстном меню пункт «Развернуть».

## Создание клиентского приложения для уже созданного бина средствами NetBeans

После создания бина можно создать использующее его клиентское приложение. Оно создаётся в NetBeans, как Enterprise Application Client.

После создания проекта клиентского приложения следует добавить библиотеку классов, содержащую удаленный интерфейс, к пути классов проекта, чтобы позволить клиенту приложения ссылаться на EJB.

Затем, в клиентском приложении следует щелкнуть ПКМ по коду главного класса и выбрать в контекстном меню пункт «Вставка кода». В открывшемся окне следует выбрать пункт «Вызвать компонент корпоративного уровня», после чего указать, какой именно бин будет вызываться.

Перед запуском клиентского приложения следует развернуть Enterprise Application на сервере.

# ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

Продемонстрировать работу Enterprise Java Bean.