1. JSP基本语法
2. JSP运行原理

**1. JSP基本语法**

**1.1. JSP的由来**

**1.1.1. 为什么有JSP规范**

Servlet技术产生以后，在使用过程中存在一个很大的问题，即为了表现页面的效果而需要输出大量的HTML标签，这些标签在Servlet中表现为一个个的字符串常量。这种输出页面的方式不仅仅增加了开发时对页面的控制难度，也不利于后期的维护。为了能够将Servlet中用于表现的功能分离出来，提高视图的开发效率，推出了JSP这种技术，主要用于将Servlet中负责显示的语句抽取出来。

**1.1.2. 什么是JSP**

JSP（Java Server Page）是Sun公司制定的一种服务器端动态页面技术的组件规范，以“.jsp”为后缀的文件中既包含HTML静态标记用于表现页面，也包含特殊的代码，用于生成动态内容。

JSP作为简化Servlet开发的一种技术，实质上最终依然要转变为Servlet才可能运行，只不过这个转变过程由Servlet容器来完成。所以遵循JSP的编写规范才能使得JSP转变为需要的Servlet。

**1.2. JSP编写规范**

**1.2.1. 如何编写JSP**

步骤一、创建一个以“jsp”为后缀的文件

步骤二、在文件中添加用于控制页面显示的HTML代码、样式及JavaScript脚本。

步骤三、在需要动态生成的部分添加Java代码的编程逻辑

**1.2.2. JSP页面中的HTML代码**

JSP页面中的HTML代码与静态的HTML页面代码没有区别，可以有基本标记，可以有用于控制样式的CSS，可以有控制页面的JS脚本。这些原本在Servlet中表现为字符串常量的内容最终依然会在JSP转换成Servlet时由系统使用out.write方法自动输出。

**1.2.3. JSP页面中的注释**

在JSP页面中可以添加如下两种类型的注释

1. **<!--** 注释内容 **-->**
2. **<%--** 注释内容 **-->**

第一种注释也叫HTML注释，可以出现在JSP页面之中，注释内容中可以包含了一些Java代码，但这些代码会被执行

第二行注释是JSP注释，不允许注释的内容出现Java代码，写了Java代码也会被忽略，不会执行。

**1.2.4. JSP页面中的Java代码**

JSP页面之中可以嵌入一些Java代码，以此动态生成页面中的一些内容。而JSP规范恰恰是针对这些嵌入的Java代码的一些约束，各种不同类型的代码的作用以及在转变为Servlet一部分时的规则都是不一样的。JSP页面中可以包含如下三种类型的Java代码：

* JSP表达式（方便输出）
* JSP小脚本（完成相对较长的逻辑运算）
* JSP声明（添加属性或方法）

这些Java代码可以在页面的任何位置进行编写，当他们被执行时就可以控制页面上产生可变化的内容了。

**1.2.5. JSP表达式**

使用表达式可以方便的在JSP页面中输出运算的结果，代码形式如下所示：

1. **<%=3+5%>**
2. **<%=**add**()%>**
3. **<%=**xx**.**getName**()%>**
4. **<%=**xx**.**getName**()+**“abc”**%>**

***注意：表达式结束不需要写分号。***

这种形式的Java代码在转译成Servlet时，会成为service（）方法中使用out.print语句的输出。

1. out**.**print**(3+5);**
2. out**.**print**(**add**());**
3. out**.**print**(**xx**.**getName**());**
4. out**.**print**(**xx**.**getName**()+**“abc”**));**

**1.2.6. JSP小脚本**

JSP小脚本可以编写Java代码段，从而实现相对较长的一段运算逻辑。这些Java代码最终都会成为Servlet中Service方法的一部分。由于HTML与Java可以进行混合使用，所以需要注意括号的匹配。

1. **<**table**>**
2. **<%**
3. List<User> allUser **=** **(**List**<**User**>)**request**.**getAttribute**(**“users“**);**
4. **for(**User u **:** allUser**){**
5. **%>**
6. **<**tr**>**
7. **<**td**>** **<%=**u**.**getId**()%>** **</**td**>**
8. **<**td**>** **<%=**u**.**getName**()%>** **</**td**>**
9. **</**tr**>**
10. **<%**
11. **}**
12. **%>**
13. **</**table**>**

以上代码使用小脚本控制生成一个表格。<tr>行标记作为要循环输出的一部分出现在了循环体内。需要注意循环体结束时的大括号的匹配。这些代码最终都会成为service方法的一部分，转换结果如下：

1. **public** **void** service**(**…**){**
2. out**.**write**(**“**<**table**>**”**);**
3. List<User> allUser **=** **(**List**<**User**>)**request**.**getAttribute**(**“users“**);**
4. **for(**User u **:** allUser**){**
5. out**.**write**(**“**<**tr**>** **<**td**>**”**);**
6. out**.**print**(**u**.**getId**());**
7. out**.**write**(**“**</**td**><**td**>**”**);**
8. out**.**print**(**u**.**getName**());**
9. out**.**write**(**“**</**td**></**tr**>**”**);**
10. **}**
11. out**.**write**(**“**</**table**>**”**);**
12. **}**

**1.2.7. JSP声明**

JSP声明可以为对应的Servlet添加属性和方法。这种形式的代码使用的很少。语法规则如下：

1. **<%!**
2. //属性或方法的声明
3. **%>**

如编写下列代码

1. **<%!**
2. **public** **void** fun**(){**
3. //… 方法体
4. **}**
5. **%>**

转换为Servlet之后的结果如下：

1. **public** **class** XXX\_JSP **extends** JSPBase**{**
2. **public** **void** fun**(){**
3. // … 方法体
4. **}**
5. **public** **void** service**(**… …**){**
7. **}**
8. **}**

**1.2.8. JSP页面中的指令**

指令在JSP页面中通常起到转译成Servlet时的格式控制的作用。基本语法为：

1. **<%**@ 指令名 属性**=**值 **%>**

常用指令包含以下三种

* page指令
* include指令
* taglib指令

这些指令都有自己的属性来实现不同的控制功能。taglib指令会在讲解JSP标签时详细介绍。

**1.2.9. page指令**

page指令可以实现在JSP页面中导入要用到的Java包，也可以对页面的一些属性进行设置。

**导包**

1. **<%--** 导包 **--%>**
2. **<%**@ page **import=**“java**.**util**.\***“**%>**
3. **<%**@ page **import=**“java**.**util**.\*,**java**.**sql**.\***“**%>**

使用page指令导包时，需要用到import属性。如果需要导入多个包，可以分成多条page指令来编写，也可以在一条page指令中，使用“，“逗号作为分隔来实现。注意，page指令要放在页面的最上面编写

**设置response.setContentType（）方法的参数值**

1. **<%--** 设置response**.**setConentType方法的参数值 **--%>**
2. **<%**@ page contentType**=**“text**/**html**;**charset**=**utf**-8**“**%>**

使用page指令可以设置输出内容的编码方式，这样就可以设置浏览器使用正确的解码方式来显示页面。

**设置容器读取该文件时的解码方法**

1. **<%--** 设置容器读取该文件时的解码方式 **--%>**
2. **<%**@ page pageEncoding**=**“UTF**-8**“**%>**

为了保证页面中编写的中文能够保存，以及容器在加载文件时能正确的解码文件中的中文，需要通过page指令的pageEncoding属性来完成。这段代码可以保证页面在加载到内存时正确的解码中文。

**1.2.10. include指令**

include指令主要用于将其他页面包含在另一个页面之中。同一个应用中的很多页面都会有相同的导航头、版权声明，在每一个页面中重复编写是没有必要的，而且如果导航头发生变化时不利于所有页面的更新。如果将这样共用的部分单独做到一个文件中，只需要在不同的页面中添加引用，就可以实现页面内容的重用。

include指令的语法如下：

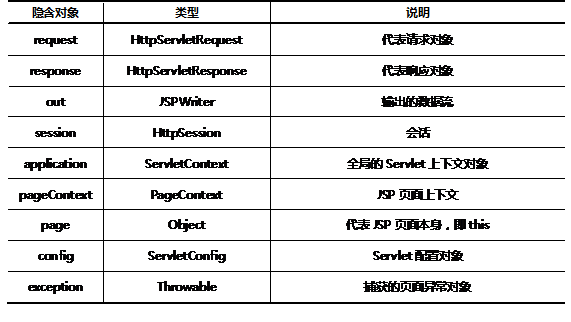
1. **<%**@ include file**=**“header**.**html” **%>**

**1.2.11. JSP页面中的隐含对象**

由于JSP页面最终要转化为一个Servlet，而作为即将转变为Servlet的一个前期版本，会有很多系统提供给页面能够直接使用的对象，这些对象的创建由系统完成，对象名也有系统命名，根据页面中要实现的功能直接使用即可。具体的隐含对象信息见 表-1。

在Servlet中需要自己创建获取的对象在JSP页面中使用表-1中的对象名即可，原有的方法通过对象来调用。

表 – 1 JSP页面总的隐含对象



**2. JSP运行原理**

**2.1. JSP的运行原理**

**2.1.1. JSP是如何运行的**

JSP页面的运行流程遵循如 图 -1 所以的过程。整个过程在Servlet容器的控制下将请求的JSP资源转变为Servlet之后，同样会经历Servlet的例化、初始化、就绪、销毁四部分的生命周期。从图中可以看到，之所以第一次请求一个JSP页面会比之后的请求响应慢一些，就是要经过转译成Servlet，编译成class这样一个过程。

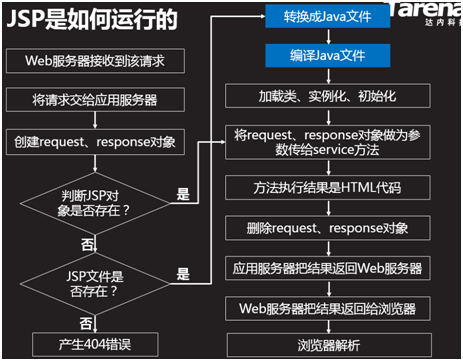


图 - 1

**2.1.2. JSP如何转换为Java**

JSP页面最终是要转译为一个Servlet，所以JSP页面中的HTML代码及Java代码、指令等内容都会按照固定的方式变为Servlet中的一部分。

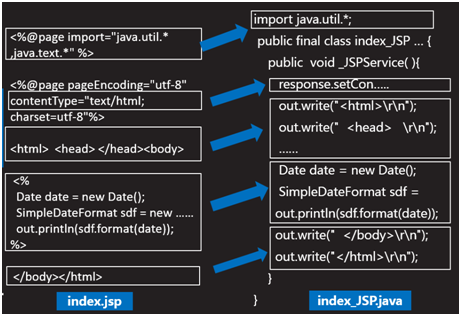


图 – 2

如 图- 2 所示，用于导包的page指令最终成为import语句，page指令中的contentType属性转变为了\_JSPService()方法中的response.setContentType()方法，HTML标记都会有out.write方法输出，<%%>的小脚本原封不动成为了\_JSPService()方法内的一部分。当HTMl标记与小脚本混合在一起时，转变过程遵循从上至下的顺序逐行转变。

**2.1.3. 如何将静态页面转换为动态页面**

在将一个静态HTML页面转变为动态的JSP页面时，首先需要分析页面的整体结构，找到页面中不变的公共部分，如导航、页脚等信息。将静态页面的脚本拷贝到JSP页面之后，一定要添加page指令pageEncoding属性，保证页面中的中文能够被正确编码。添加page指令的import属性导入页面中需要的Java包。将页面中与目标页面不一致的地方进行修改，如修改表头与实际字段一致。最后将页面中需要动态生成的内容删除，使用小脚本的Java代码来实现运算逻辑。

**JAVA SERVLETJSP DAY04**

1. JSP基本元素练习
2. JSP指令练习
3. 员工管理——使用JSP实现员工信息列表1

**1 JSP基本元素练习**

**1.1 问题**

在JSP页面中输出20行“Hello JSP”。

**1.2 方案**

使用表达式、小脚本、注释配合HTML标记的方式实现动态页面的生成。

**1.3 步骤**

**步骤一：新建hello.jsp页面**

新建web Project，在webRoot下右键，新建File，名称为hello.jsp。点击Finish。如图-1所示：

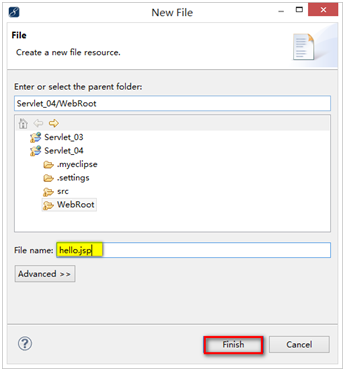


图 - 1

**步骤二：添加页面代码**

在hello.jsp页面内编写如下代码，实现循环输出10行Hello JSP。再使用表达式输出10行Hello JSP。并且为两种输出添加注释。如图-2所示：

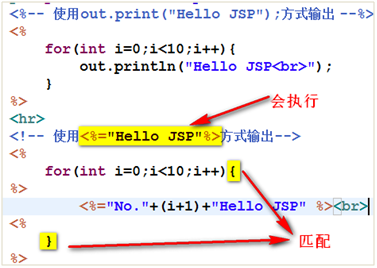


图 - 2

**步骤三：部署，访问应用**

部署应用，在地址栏中输入地址“http://localhost:8080/部署应用名/hello.jsp”查看页面效果以及页面源码，会发现第二种注释中的java代码会执行也会出现在源代码中，但第一种注释既不会运行也不会出现在结果的源代码中。

**1.4 完整代码**

hello.jsp文件代码如下：

**<%**@ page pageEncoding**=**"UTF-8" **%>**

**<**html**>**

**<**head**></**head**>**

**<**body style**=**"font-size:24px"**>**

**<%--** 使用out**.**print**(**"Hello JSP"**);**方式输出 **--%>**

**<%**

**for(**int i**=0;**i**<10;**i**++){**

                out**.**println**(**"Hello JSP<br>"**);**

**}**

**%>**

**<**hr**>**

**<!--** 使用**<%=**"Hello JSP"**%>**方式输出**-->**

**<%**

**for(**int i**=0;**i**<10;**i**++){**

**%>**

**<%=**"No."**+(**i**+1)+**"Hello JSP" **%><**br**>**

**<%**

**}**

**%>**

**</**body**>**

**</**html**>**

**2 JSP指令练习**

**2.1 问题**

在页面中输出当前系统时间，并且该时间能够嵌入到其他页面中。

**2.2 方案**

使用page指令的import导包，contentType控制编码，pageEncoding控制显示中文，使用include指令实现页面内包含其他页面的效果。

**2.3 步骤**

**步骤一：新建date.jsp页面**

在WebRoot节点上右键(New(File,文件名称为date.jsp。在页面内添加page指令及小脚本，如图-3所示：



图 - 3

**步骤二：新建includeDemo.jsp页面**

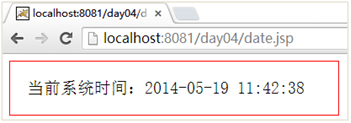
webRoot节点上右键，创建New File为 includeDemo.jsp页面，用于包含其他页面。使用include指令添加，file属性说明文件的位置。如图-4：



图 - 4

**步骤三：部署应用，查看结果**

部署后输入地址访问页面，查看结果如图-5：



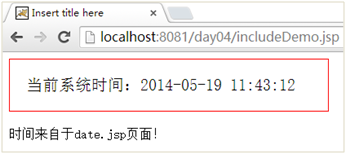
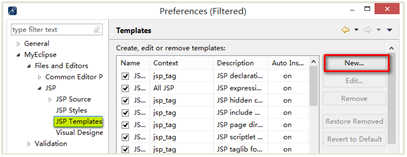


图 - 5

**步骤四：小技巧，创建自定义的JSP模板**

在随便哪一个JSP页面上，右键，选择Preferences，选择JSP Template，点击New，为模板命名并填写模板内容，步骤如图-6：



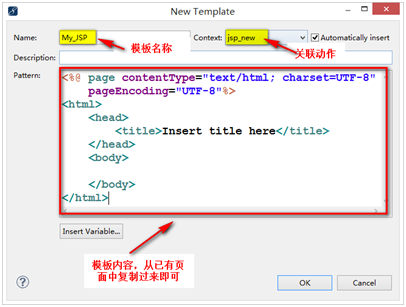
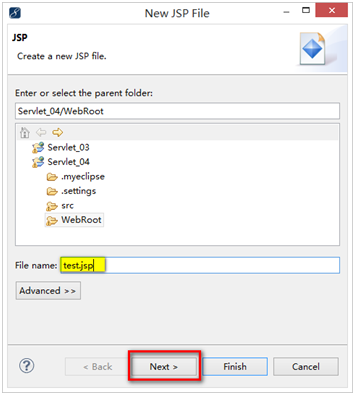


图 - 6

**步骤五：使用自定义的模板创建JSP页面**

在webRoot节点上右键选择New(JSP，填写文件名称后，点击Next，如图-7：



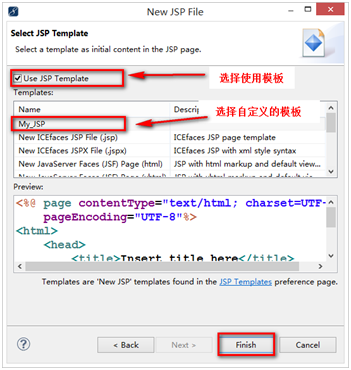


图 - 7

**2.4 完整代码**

date.jsp文件代码：

**<%**@page import**=**"java.util.\*,java.text.SimpleDateFormat"

    pageEncoding**=**"UTF-8"

    contentType**=**"text/html;charset=UTF-8"**%>**

**<**div style**=**"font-size:20px;border:1px solid red;

            width:350px;padding:20px"**>**

**<%**

        Date date **=** **new** Date**();**

        SimpleDateFormat sdf **=**

**new** SimpleDateFormat**(**"yyyy-MM-dd HH:mm:ss"**);**

**%>**

    当前系统时间：**<%=**sdf**.**format**(**date**)%>**

**</**div**>**

includeDemo.jsp文件代码：

**<%**@ page pageEncoding**=**"UTF-8"**%>**

**<**html**>**

**<**head**>**

**<**title**>**Insert title here**</**title**>**

**</**head**>**

**<**body**>**

**<%**@include file**=**"date.jsp" **%><**br**>**

    时间来自于date**.**jsp页面！

**</**body**>**

**</**html**>**

**3 员工管理——使用JSP实现员工信息列表1**

**3.1 问题**

在页面中显示员工信息列表，并且能够实现奇偶行背景交错变换的效果。

* 1. 创建实体类：entity
  2. 创建一个工具类：DBUtil.java(链接数据库的类)
  3. 数据列表的逻辑代码类
  4. Jsp页面调用相关的方法显示数据

**3.2 方案**

在JSP页面中，通过创建EmployeeDAO的实例来获取所有雇员信息，再使用循环语句以表格的形式输出雇员信息。交错行变色为不同的背景样式决定的，所以在输出行标记时根据行坐标输出不同的样式即可。

**3.3 步骤**

**步骤一：新建listEmp1.jsp页面**

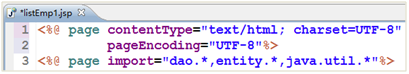


图 - 8

**步骤二：添加循环输出表格**

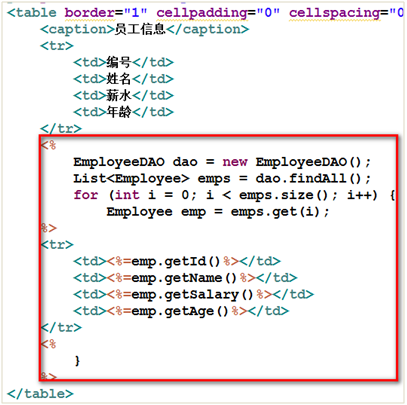


图 - 9

**步骤三：部署查看结果**

运行结果如图-10所示：



图 - 10

**步骤四：添加两个CSS类样式**

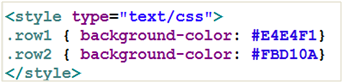


图 - 11

**步骤五：为表格添加类样式**

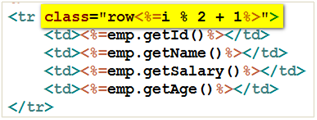


图 - 12

**3.4 完整代码**

dao包中的DBUtil.java、EmployeeDAO.java和entity包中的Employee.java文件源码参考。

Employee.java文件代码：

package entity;

/\*\*

\* 实体类：员工属性与t\_emp表中的字段匹配

\*/

public class Employee {

private int id;

private String name;

private double salary;

private int age;

public Employee() {

super();

}

public Employee(int id, String name, double salary, int age) {

super();

this.id = id;

this.name = name;

this.salary = salary;

this.age = age;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

@Override

public String toString() {

return id+" "+name+" "+salary+" "+age;

}

}

DBUtil文件代码：

package dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

/\*\*

\* JDBC管理连接的工具类，可以获取连接和关闭连接

\*/

public class DBUtil {

public static Connection getConnection() throws ClassNotFoundException, SQLException{

Class.forName("com.mysql.jdbc.Driver");

Connection con = DriverManager.getConnection("jdbc:mysql:///db1803", "root", "root");

return con;

}

public static void close(Connection con) throws SQLException{

if(con!=null){

con.close();

}

}

public static void main(String[] args) throws ClassNotFoundException, SQLException {

System.out.println(DBUtil.getConnection());

}

}

EmployeeDao文件代码：

package dao;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import entity.Employee;

public class EmployeeDao {

public List<Employee> findAll() throws ClassNotFoundException, SQLException{

ArrayList<Employee> emps = new ArrayList<Employee>();

Connection con=null;

PreparedStatement ps=null;

ResultSet rs=null;

con=DBUtil.getConnection();

ps=con.prepareStatement("select \* from t\_emp");

rs=ps.executeQuery();

while(rs.next()){

Employee emp=new Employee(rs.getInt("id"),rs.getString("name"),rs.getDouble("salary"),rs.getInt("age"));

emps.add(emp);

}

DBUtil.close(con);

return emps;

}

}

listEmp1.jsp文件代码：

**<%**@ page contentType**=**"text/html; charset=UTF-8"

pageEncoding**=**"UTF-8"**%>**

**<%**@ page import**=**"dao.\*,entity.\*,java.util.\*"**%>**

**<**html**>**

**<**head**>**

**<**style type**=**"text/css"**>**

**.**row1 **{**

    background**-**color**:** #E4E4F1

**}**

**.**row2 **{**

    background**-**color**:** #FBD10A

**}**

**</**style**>**

**</**head**>**

**<**body style**=**"font-size:24px"**>**

**<**table border**=**"1" cellpadding**=**"0" cellspacing**=**"0" width**=**"500px"**>**

**<**caption**>**员工信息**</**caption**>**

**<**tr**>**

**<**td**>**编号**</**td**>**

**<**td**>**姓名**</**td**>**

**<**td**>**薪水**</**td**>**

**<**td**>**年龄**</**td**>**

**</**tr**>**

**<%**

            EmployeeDAO dao **=** **new** EmployeeDAO**();**

            List**<**Employee**>** emps **=** dao**.**findAll**();**

**for** **(**int i **=** **0;** i **<** emps**.**size**();** i**++)** **{**

                Employee emp **=** emps**.**get**(**i**);**

**%>**

**<**tr **class=**"row<%=i % 2 + 1%>"**>**

**<**td**><%=**emp**.**getId**()%></**td**>**

**<**td**><%=**emp**.**getName**()%></**td**>**

**<**td**><%=**emp**.**getSalary**()%></**td**>**

**<**td**><%=**emp**.**getAge**()%></**td**>**

**</**tr**>**

**<%**

**}**

**%>**

**</**table**>**

**</**body**>**

**</**html**>**