

知识点列表

编号	名称	描述	级别
1	改进 DAO 工厂模式	掌握 DAO 模式的一般用法	**
2	关于类加载器 (ClassLoader)	了解类加载过程, 可能在面试中出现。	*
3	Servlet 处理多种请求	掌握使用一个 Servlet 处理多个请求的技巧	**
4	Servlet 处理多种请求应用	掌握	**
5	核心接口与类	了解 Servlet 中的核心类与接口, 在学习中重点学习的是 ServletRequest 和 ServletResponse	*
6	Servlet 的生命周期	理解并能复述出 Servlet, 面试中出现的频率较高。	***
7	如何写一个 jsp 文件	了解 JSP 文件的结构, 掌握 JSP 开发步骤	**
8	jsp 文件的组成	了解 JSP 文件的组成。	**
9	JSP 实现员工增删改查	熟练练习该案例	**

注: **"理解级别 ***"掌握级别 ****"应用级别

目录

1. 改进【案例】DAO 及工厂模式*	错误！未定义书签。
1.1. 改进 DAO 工厂模式**	3
1.2. 关于类加载器 (ClassLoader) *	错误！未定义书签。
2. 处理请求资源路径**	10
3. Servlet 处理多种请求**	14
3.1. 【案例 1】Servlet 处理多种请求**	15
3.2. 【案例 1】Servlet 处理多种请求应用**	17
4. servlet 生命周期及核心接口与类**	23
4.1. 核心接口与类*	23
4.2. servlet 的生命周期***	24
5. Jsp**	36
5.1. 如何写一个 jsp 文件**	37
5.2. jsp 文件的组成**	43

1. 改进【案例】DAO 及工厂模式 **

1.1.改进 DAO 工厂模式 **

在昨天的案例中，我们使用了工厂模式，但遗留了一个小问题

提问：

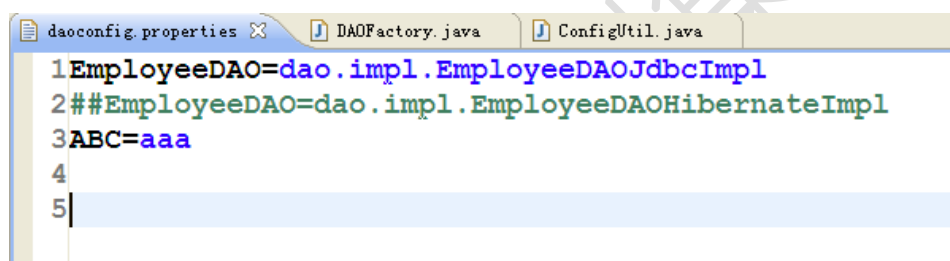
我们在实现“数据库访问技术由 JDBC 更改为 Hibernate”的时候，还需要修改 DAOFactory，如果我们不想修改 DAOFactory 该如何实现？

使用配置文件。

演示案例

步骤 1

新建配置文件 daoconfig.properties



```
daoconfig.properties X DAOFactory.java ConfigUtil.java
1 EmployeeDAO=dao.impl.EmployeeDAOJdbcImpl
2 ##EmployeeDAO=dao.impl.EmployeeDAOHibernateImpl
3 ABC=aaa
4
5 |
```

步骤 2

新建读取配置文件的工具类 ConfigUtil.java

```

1 package util;
2
3 import java.io.IOException;
4
5
6
7 /**
8  * 读properties文件工具
9  * @author teacher
10  *
11  */
12 public class ConfigUtil {
13     private static Properties props =
14         new Properties();
15     static{
16         ClassLoader loader =
17             ConfigUtil.class.getClassLoader();
18         InputStream ips =
19             loader.getResourceAsStream(
20                 "util/daoconfig.properties");
21         try {
22             props.load(ips);
23         } catch (IOException e) {
24             e.printStackTrace();
25         }
26     }
27
28     public static String getValue(String key){
29         return props.getProperty(key);
30     }
31
32     public static void main(String[] args) {
33         System.out.println(getValue("ABC"));
34     }
35
36 }
37

```

步骤 3

修改 DAOFactory.java

```

DAOFactory.java X
1 package util;
2
3 /**
4  * dao工厂，为调用者提供dao实现类的实例
5  * @author teacher
6  */
7 public class DAOFactory {
8     public static Object getInstance(String type){
9         Object obj = null;
10        //依据接口名找到对应的类名
11        String className =
12            ConfigUtil.getValue(type);
13        //使用反射，创建实例
14        try {
15            obj =
16                Class
17                    .forName(className)
18                    .newInstance();
19        } catch (Exception e) {
20            e.printStackTrace();
21        }
22        return obj;
23    }
24 }
25

```

在以后的修改，我们只需要修改配置文件 daoconfig.properties 即可

```

daoconfig.properties X
1##EmployeeDAO=dao.impl.EmployeeDAOJdbcImpl
2EmployeeDAO=dao.impl.EmployeeDAOHibernateImpl
3ABC=aaa

```

工厂模式完成。

1.2. 关于类加载器 (ClassLoader) *

在 CoreJava 基础部分，我们对程序执行过程的内存流程图做了详细解析，在之前知识的基础上，今天做一些扩展，通过如下一段代码

```
Student s = new Student();
s.play();
Student s2 = new Student();
```

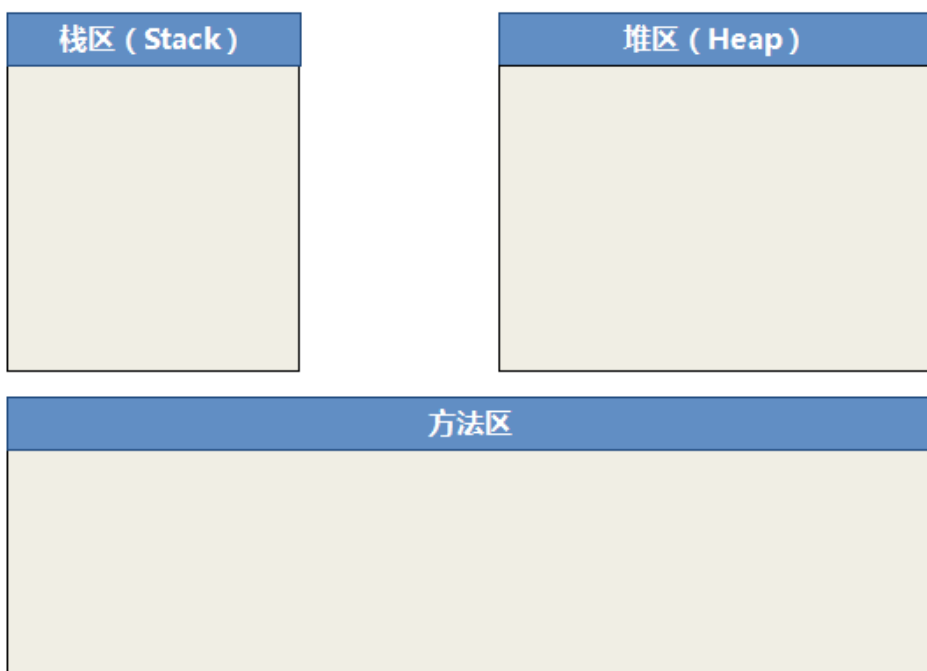
我们将解释

ClassLoader 是干什么用的？

在 JVM 中类（Class）是怎么执行的？

执行流程

JVM 作为操作系统的一个进程在系统中执行，那么系统会为 JVM 分配一块内存空间，这块内存空间被 JVM 分为 3 大块（**栈区**、**堆区**、**方法区**）



一般而言，对象在堆（Heap）中创建，但是一些特殊的对象会在方法区中创建。

第 1 步

当 JVM 执行第一行代码 “Student s = new Student();” 时

JVM 先碰到了 Student 类， “**Student** s = new Student();”

此时，JVM 将查看**方法区**中是否有 Student 对应的 Class 对象（我们学习过反射，都知道 Class 对象，在同一个 JVM 中，可以有很多的 Student 实例，但是 Student 的 Class 对象只有一个）。

因为是第一次执行，方法区中没有 Student 的 Class 对象，此时 JVM 就会调用**类加载器**（ClassLoader）。

类加载器有 2 大类：

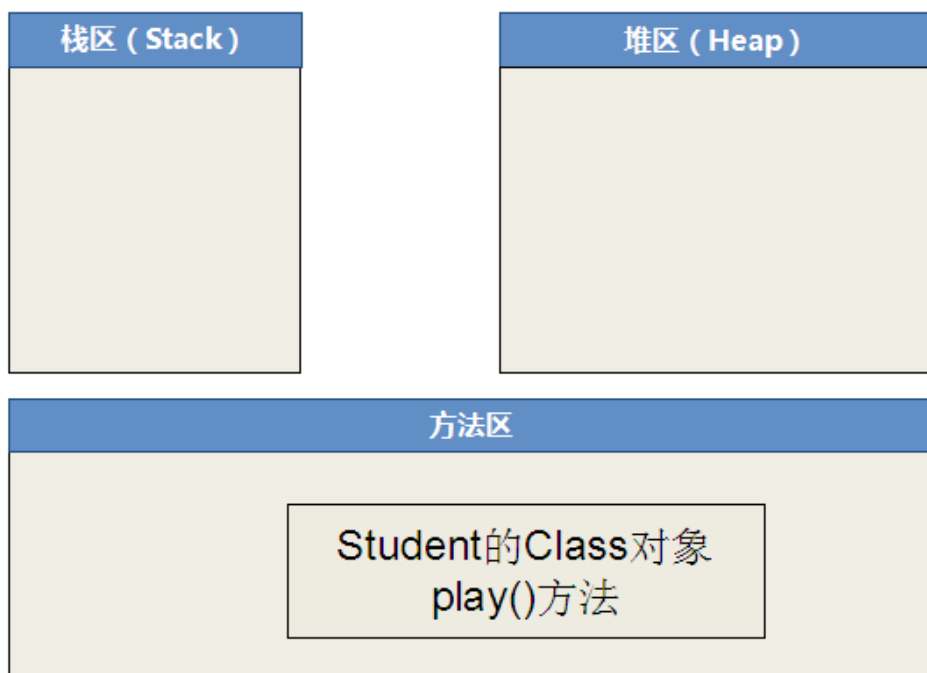
第 1 种是虚拟机本身提供的，第 2 种是程序员自定义的（像 Tomcat 本身也有自己的类加载器）

类加载器（ClassLoader）要加载 Student 类的过程，就是要在物理位置找到 Student 类的字节码文件（如 D：/workspace/web03/classes/Student.class）。

怎样才能找到？JVM 会根据 ClassPath 搜索。

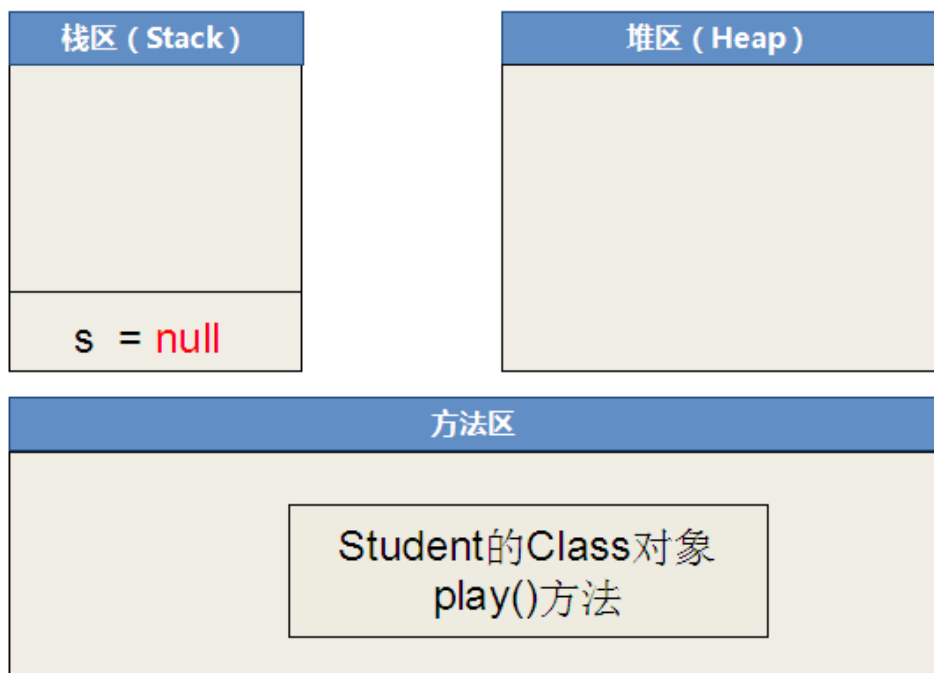
当 JVM 找到 Student 类的字节码文件后，JVM 会将该字节码文件转换为一个 Student 的 Class 对象，放入**方法区**。

当这个 Student 的 Class 对象构造完毕，类加载过程就完成了。



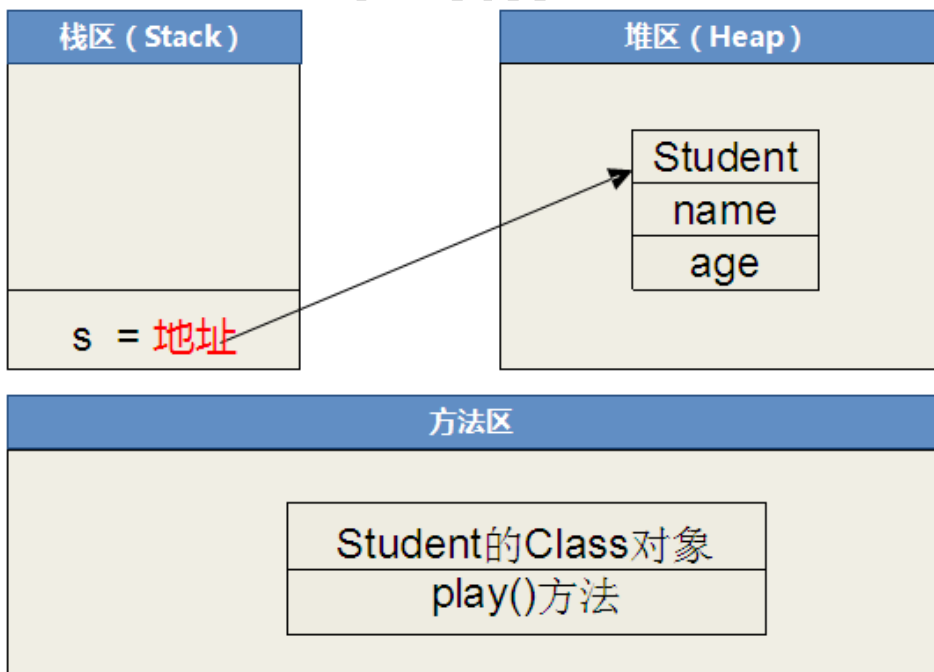
第 2 步

将 Student 类型的变量 s 放入栈 (Stack) 中；“Student s = new Student();”



第 3 步

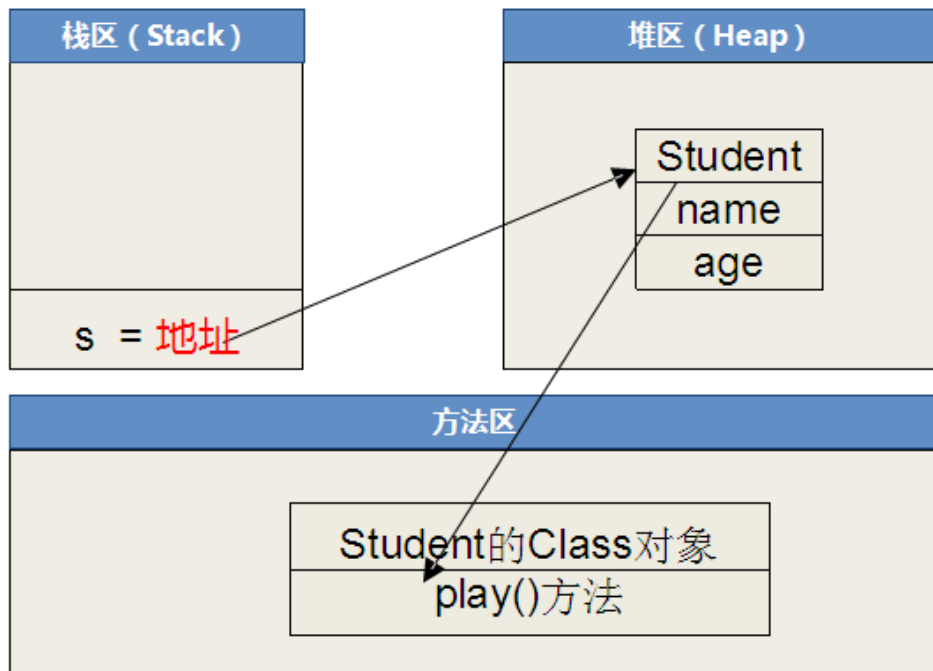
"Student s = **new Student()**;" 在堆 (heap) 中创建一个 Student 对象，变量 s 指向该对象。



第 4 步

play()方法放在代码区中 Student 的 Class 对象中 对象的方法在 JVM 中只有 1 份 对象的属性(每个对象都有独有的属性) 可以有多个。

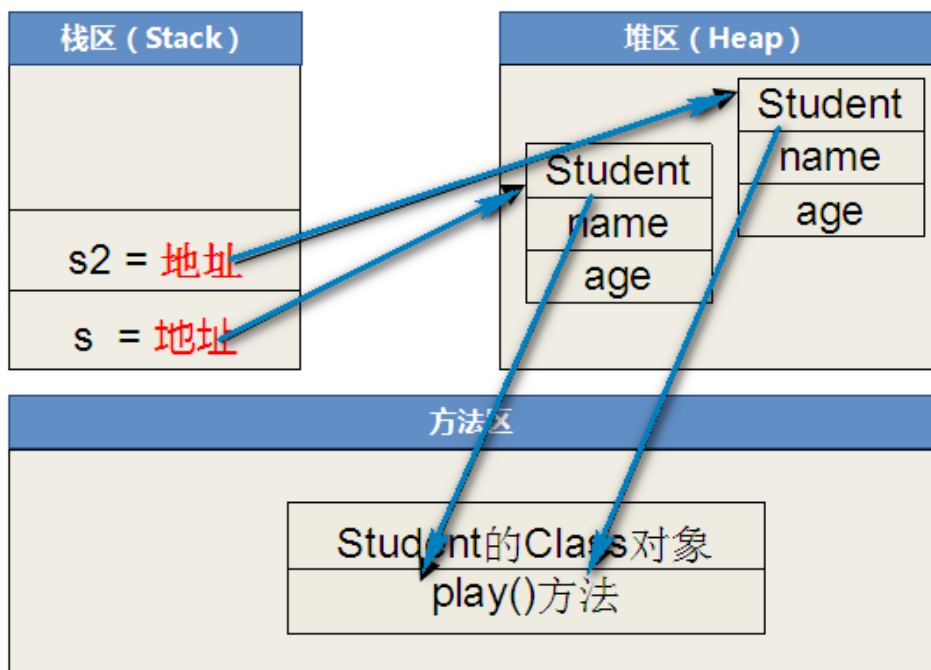
执行 “s.play();” 方法时，Student 对象到方法区中找到 play()方法并执行。



第 5 步

执行 “**Student** s2 = new Student();” 时，JVM 到方法区中找到了 Student 的 Class 对象，所以 JVM 不再调用 ClassLoader 加载 Class 对象。

将直接在堆中创建。



2. 处理请求资源路径 **

Servlet 容器如何处理请求资源路径？

什么是请求资源路径？

比如输入 `http://ip:port/appname/abc.html/appname/abc.html` 称为请求资源路径

Servlet 容器处理请求资源路径的步骤

step1

依据/appname 查找该应用对应的文件夹(比如查找 webapps 下面的文件夹是否与其匹配)。

step2

在找到的文件夹下面，找到 web.xml 文件，然后使用 `<url-pattern>` 与 `/abc.html` 去做匹配。

匹配规则：

- 1) **精确匹配:** 即要求 `<url-pattern>` 与 `/abc.html` 完全一致。
- 2) **通配符匹配:** 使用 `"*"` 代表任意的字符串。

比如在 `<url-pattern>` 中使用:

`/*` 表示任意字符串
`/abc/*` 表示有 `/abc/` 的字符串

- 3) **后缀匹配:** 不能使用 `"/"` 开头，要求 `"*.任意的字符串"`。

比如

*.do	要求以.do 结尾
*.action	要求以.action 结尾

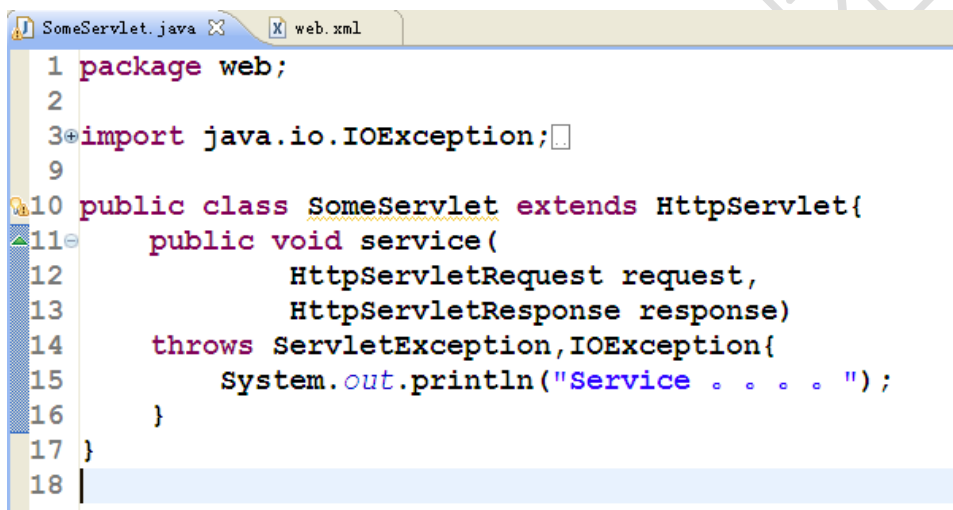
step3

尝试查找/abc.html 文件,如果找到,则返回该文件,找不到,则返回 404 状态码。

演示 1

通配符匹配

1) SomeServlet.java



```

1 package web;
2
3 import java.io.IOException;
4
5
6
7
8
9
10 public class SomeServlet extends HttpServlet{
11     public void service(
12         HttpServletRequest request,
13         HttpServletResponse response)
14         throws ServletException, IOException{
15         System.out.println("Service ....");
16     }
17 }
18

```

2) web.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
5     xsi:schemaLocation="http://java.sun.com/xml/ns
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xs
7 <servlet>
8     <servlet-name>someServlet</servlet-name>
9     <servlet-class>web.SomeServlet</servlet-cl
10 </servlet>
11 <servlet-mapping>
12     <servlet-name>someServlet</servlet-name>
13     <url-pattern>/*</url-pattern>
14 </servlet-mapping>
15 </web-app>
16

```

3) 访问 <http://localhost:8080/web04/aabbccddeedd> , OK

4) 访问 <http://localhost:8080/web04/aa/bb/cc/dd> , OK

5) 修改 web.xml

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
5     xsi:schemaLocation="http://java.sun.com/xml/ns
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xs
7 <servlet>
8     <servlet-name>someServlet</servlet-name>
9     <servlet-class>web.SomeServlet</servlet-cl
10 </servlet>
11 <servlet-mapping>
12     <servlet-name>someServlet</servlet-name>
13     <url-pattern>/abc/*</url-pattern>
14 </servlet-mapping>
15 </web-app>
16

```

6) 访问 <http://localhost:8080/web04/aabbccddeedd> , 404

7) 访问 <http://localhost:8080/web04/abc/aabbcc> , OK

8) 访问 <http://localhost:8080/web04/abc/aa/aabbcc> , OK

后缀匹配

9) 修改 web.xml

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
7 <servlet>
8     <servlet-name>someServlet</servlet-name>
9     <servlet-class>web.SomeServlet</servlet-class>
10 </servlet>
11 <servlet-mapping>
12     <servlet-name>someServlet</servlet-name>
13     <url-pattern>*.do</url-pattern>
14 </servlet-mapping>
15 </web-app>
16
```

- 10) 访问 <http://localhost:8080/web04/aabbcc> , 404
- 11) 访问 <http://localhost:8080/web04/aabbcc.do> , OK
- 12) 访问 <http://localhost:8080/web04/abc/aabbcc.do> , OK

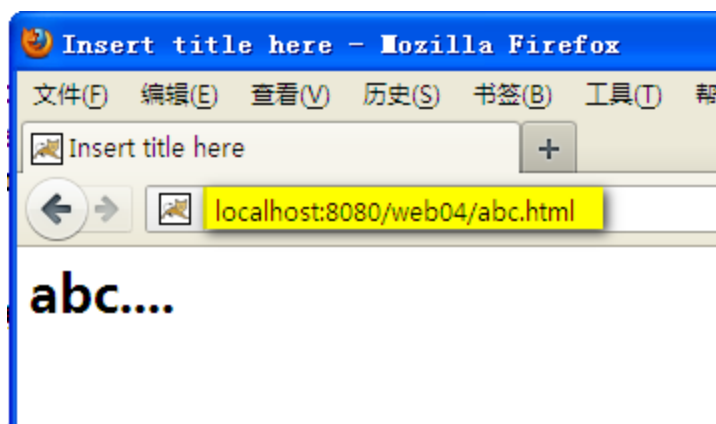
<url-pattern>*.action</url-pattern>同理。

精确匹配

13) 新建 abc.html

```
abc.html
1 <html>
2 <head>
3 <meta http-equiv="Content-Type"
4     content="text/html; charset=UTF-8">
5 <title>Insert title here</title>
6 </head>
7 <body>
8     <h1>abc...</h1>
9 </body>
10 </html>
11
```

- 14) 访问 <http://localhost:8080/web04/abc.html> , 找到 html 页面



15) 修改 web.xml



16) 访问 <http://localhost:8080/web04/abc.html> ,

注意：访问的是 SomeServlet，而不是 abc.html 页面

3. Servlet 处理多种请求 **

一个 servlet 如何处理多种请求？

step1 使用后缀匹配模式。

step2 在 service()方法里加入如下代码

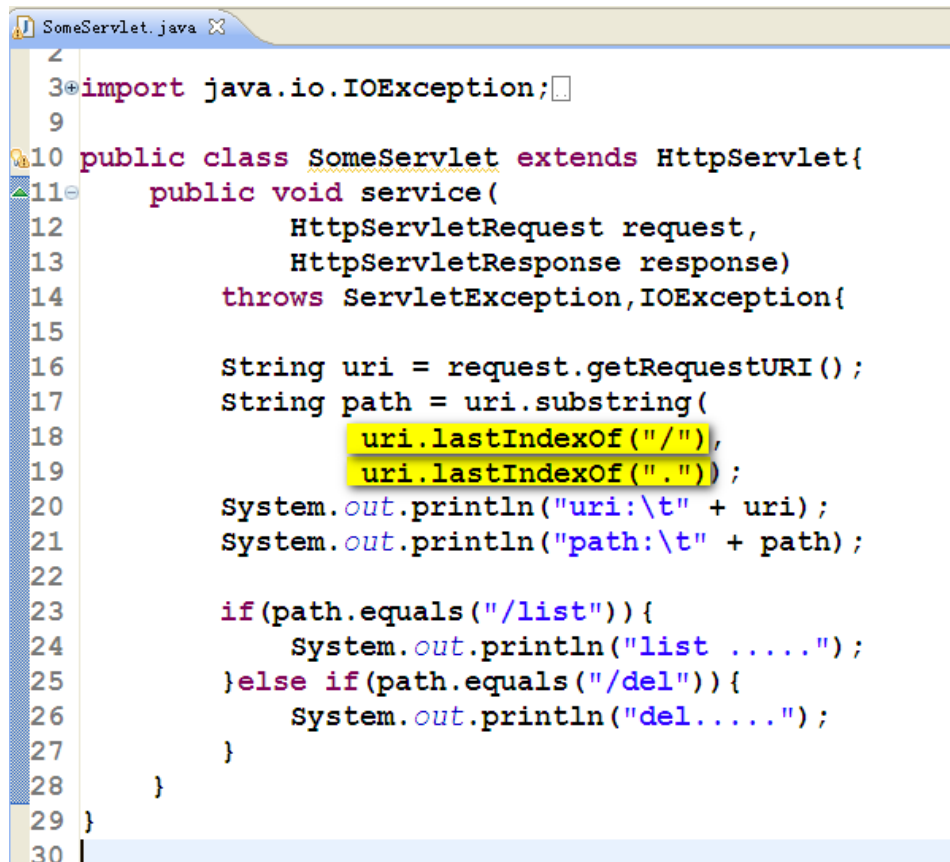
//获得请求资源路径

String uri = request.getRequestURI();

//分析请求资源路径，来决定做何种处理。

【案例 1】Servlet 处理多种请求 **

1) SomeServlet.java



```

1  SomeServlet.java
2
3  import java.io.IOException;
4
5
6
7
8
9
10 public class SomeServlet extends HttpServlet{
11     public void service(
12         HttpServletRequest request,
13         HttpServletResponse response)
14         throws ServletException, IOException{
15
16         String uri = request.getRequestURI();
17         String path = uri.substring(
18             uri.lastIndexOf("/"),
19             uri.lastIndexOf("."));
20         System.out.println("uri:\t" + uri);
21         System.out.println("path:\t" + path);
22
23         if(path.equals("/list")){
24             System.out.println("list .....");
25         }else if(path.equals("/del")){
26             System.out.println("del .....");
27         }
28     }
29 }
30

```

2) web.xml

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
5     xsi:schemaLocation="http://java.sun.com/xml/ns
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xs
7 <servlet>
8     <servlet-name>someServlet</servlet-name>
9     <servlet-class>web.SomeServlet</servlet-cla
10 </servlet>
11 <servlet-mapping>
12     <servlet-name>someServlet</servlet-name>
13     <url-pattern>*.do</url-pattern>
14 </servlet-mapping>
15 </web-app>
16
```

3) 访问 <http://localhost:8080/web04/list.do>

```
Console
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 30, 20
2012-1-30 17:43:56 org.apache.catalina.startup.HostCon
信息: Reloading context [/web04]
uri:    /web04/list.do
path:   /list
list .....
```

4) 访问 <http://localhost:8080/web04/del.do>

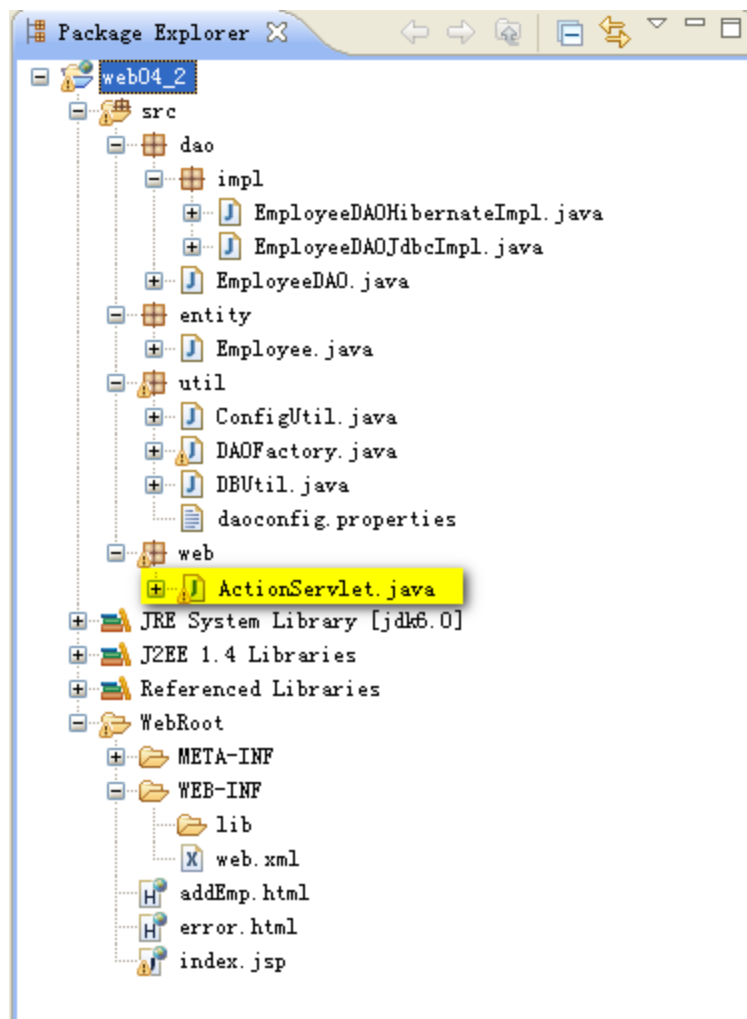
```
Console
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 30, 201
uri:    /web04/del.do
path:   /del
del.....
```

如上实现了一个 Servlet 处理多种请求的功能。

【案例 2】Servlet 处理多种请求应用 **

使用一个 Servlet 处理增删改查操作

项目结构



参考代码

1) ActionServlet.java

```
package web;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import util.DAOFactory;
import dao.EmployeeDAO;
import dao.impl.EmployeeDAOJdbcImpl;
import entity.Employee;

public class ActionServlet extends HttpServlet {
    public void service(
        HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        request.setCharacterEncoding("utf-8");
        response.setContentType(
            "text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        String uri = request.getRequestURI();
        String path =
            uri.substring(
                uri.lastIndexOf("/"),
                uri.lastIndexOf("."));
        if (path.equals("/list")) {
            // 使用 dao 访问数据库
            try {
                EmployeeDAO dao =
                    (EmployeeDAO) DAOFactory
                        .getInstance("EmployeeDAO");
                List<Employee> employees = dao.findAll();
                // 使用查询得到的结果，生成一个表格
                out.println(
                    "<table border='1' " +
                    "width='60%' " +
                    + "cellpadding='0' " +
                    "cellspacing='0'>");
                out.println(
                    "<tr><td>id</td><td>" +
                    "姓名</td>"
```

```

        + "<td>" +
        "薪水</td><td>" +
        "年龄</td><td>" +
        "操作</td></tr>");
    for (int i = 0; i < employees.size(); i++) {
        Employee e = employees.get(i);
        out.println(
            "<tr><td>" +
            e.getId() +
            "</td><td> "
            + e.getName() +
            "</td><td>" +
            e.getSalary()
            + "</td><td> " +
            e.getAge() + "</td><td>"
            + "<a href='del.do?id=" +
            e.getId() + "'>删除</a>"
            + "&nbsp;<a href='load.do?id=" +
            e.getId()
            + "'>修改</a></td></tr>");
    }
    out.println("</table>");
    out.println(
        "<a href='addEmp.html'>" +
        "增加新雇员</a>");
    out.close();
} catch (Exception e) {
    e.printStackTrace();
    throw new ServletException(e);
}
} else if (path.equals("/add")) {
    String name = request.getParameter("name");
    double salary =
        Double.parseDouble(
            request.getParameter("salary"));
    int age = Integer.parseInt(
        request.getParameter("age"));
    // 访问数据库

```

```

        try {
            EmployeeDAO dao = new EmployeeDAOJdbcImpl();
            Employee e = new Employee();
            e.setName(name);
            e.setSalary(salary);
            e.setAge(age);
            dao.save(e);
            response.sendRedirect("list.do");
        } catch (Exception e) {
            e.printStackTrace();
            throw new ServletException(e);
        }
    } else if (path.equals("/del")) {
        long id = Long.parseLong(
            request.getParameter("id"));
        try {
            EmployeeDAO dao = new EmployeeDAOJdbcImpl();
            dao.delete(id);
            response.sendRedirect("list.do");
        } catch (Exception e) {
            e.printStackTrace();
            throw new ServletException(e);
        }
    } else if (path.equals("/load")) {
        long id = Long.parseLong(
            request.getParameter("id"));
        try {
            EmployeeDAO dao = new EmployeeDAOJdbcImpl();
            Employee e = dao.findById(id);
            out.println(
                "<form action='modify.do?id=" + id
                + "' method='post'>");
            out.println("id:" + id + "<br/>");
            out.println(
                "姓名:<input name='name' value='"
                + e.getName()
                + "'/><br/>");
            out.println(

```

```

        "薪水:<input name='salary' value='"
        + e.getSalary()
        + "'/><br/>");
    out.println(
        "年龄:<input name='age' value='"
        + e.getAge()
        + "'/><br/>");
    out.println(
        "<input type='submit' " +
        "value='确认'/>");
    out.println("</form>");
    out.close();
} catch (Exception e) {
    e.printStackTrace();
    throw new ServletException(e);
}
}else if(path.equals("/modify")){
    long id =
        Long.parseLong(
            request.getParameter("id"));
    String name = request.getParameter("name");
    double salary =
        Double.parseDouble(
            request.getParameter("salary"));
    int age =
        Integer.parseInt(
            request.getParameter("age"));

    try {
        EmployeeDAO dao = new EmployeeDAOJdbcImpl();
        Employee e = new Employee();
        e.setId(id);
        e.setName(name);
        e.setSalary(salary);
        e.setAge(age);
        dao.update(e);
        response.sendRedirect("list.do");
    }catch(Exception e){

```

```
e.printStackTrace();
throw new ServletException(e);
    }
}
}
```

2) web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <servlet>
    <servlet-name>actionServlet</servlet-name>
    <servlet-class>web.ActionServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>actionServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

3) 拷贝其它相关代码到项目中

请参考 day03 课堂练习：【案例 3】DAO 及工厂模式

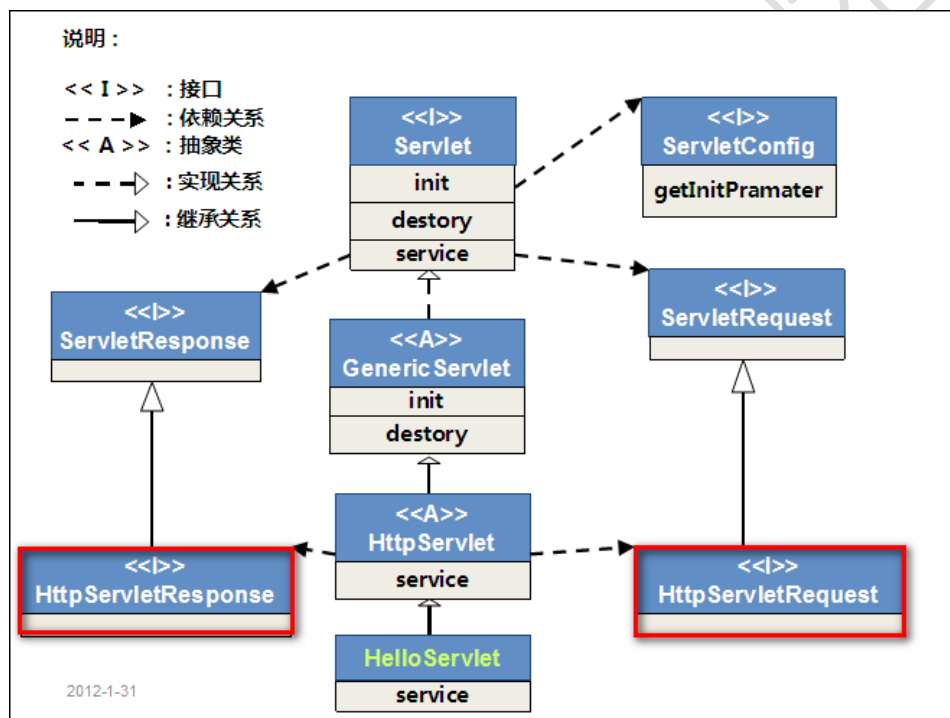
4. servlet 生命周期及核心接口与类 **

4.1. 核心接口与类 *

1) Servlet 接口

- ✓ init(ServletConfig config)

- ✓ destroy()
- ✓ service(ServletRequest res, ServletResponse rep)
- 2) **GenericServlet 抽象类**
实现了 Servlet 接口中的 init、destroy 方法。
- 3) **HttpServlet 抽象类**
继承了 GenericServlet，实现了 service 方法。
- 4) **ServletRequest 与 ServletResponse 接口**
- 5) **HttpServletRequest 与 HttpServletResponse 接口**
- 6) **ServletConfig 接口**
String getInitParameter(String paraName);



4.2. servlet 的生命周期 ***

所谓生命周期，指的是 servlet 容器如何创建 servlet 实例、分配其资源、调用其方法、并销毁其实例的整个过程。

阶段一：实例化（就是创建 servlet 对象,调用构造器）

在如下两种情况下会进行对象实例化。

第一种情况：

当请求到达容器时，容器查找该 servlet 对象是否存在，如果不存在，才会创建实例。

第二种情况：

容器在启动时，或者新部署了某个应用时，会检查 web.xml 当中，servlet 是否有 load-on-startup 配置。如果有，则会创建该 servlet 实例。

load-on-startup 参数值越小，优先级越高（最小值为 0，优先级最高）。

阶段二：初始化

为 servlet 分配资源，调用 `init(ServletConfig config)` 方法

`config` 对象可以用来访问 servlet 的初始化参数。

初始化参数是使用 `init-param` 配置的参数。

`init` 可以 override。

阶段三：就绪/调用

有请求到达容器，容器调用 servlet 对象的 `service()` 方法。

`HttpServlet` 的 `service()` 方法，会依据请求方式来调用 `doGet()` 或者 `doPost()` 方法。

但是，这两个 `do` 方法默认情况下，会抛出异常，需要子类去 override。

阶段四：销毁

容器依据自身的算法，将不再需要的 servlet 对象删除掉。

在删除之前，会调用 servlet 对象的 `destroy()` 方法。

`destroy()` 方法用于释放资源。

在 servlet 的整个生命周期当中，`init`, `destroy` 只会执行一次，而 `service` 方法会执行多次。

演示

Servlet 生命周期

- 阶段一：实例化

- 1) 新建 `OtherServlet.java`


```

1 package web;
2
3 import java.io.IOException;
4
5
6
7
8
9 public class OtherServlet extends HttpServlet{
10
11     public OtherServlet() {
12         System.out.println(
13             "OtherServlet's constructor...");
14     }
15
16     public void service(
17         HttpServletRequest request,
18         HttpServletResponse response)
19         throws ServletException, IOException{
20         System.out.println("service...");
21     }
22 }
23

```

2) 修改 web.xml

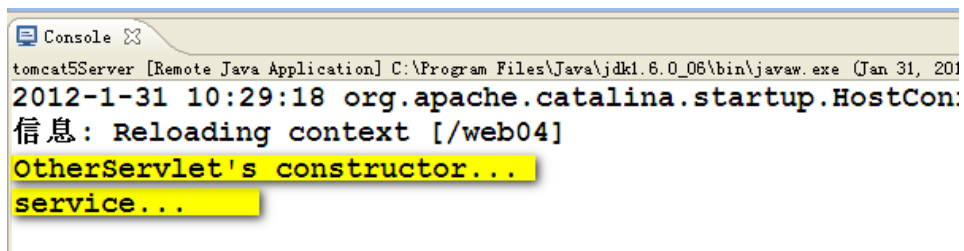
```

1
2
3
4 xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
5 xsi:schemaLocation="http://java.sun.com/xml/ns,
6 http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd
7
8 <servlet>
9     <servlet-name>someServlet</servlet-name>
10    <servlet-class>web.SomeServlet</servlet-cl
11</servlet>
12<servlet>
13    <servlet-name>otherServlet</servlet-name>
14    <servlet-class>web.OtherServlet</servlet-cl
15</servlet>
16<servlet-mapping>
17    <servlet-name>someServlet</servlet-name>
18    <url-pattern>*.do</url-pattern>
19</servlet-mapping>
20<servlet-mapping>
21    <servlet-name>otherServlet</servlet-name>
22    <url-pattern>/other</url-pattern>
23</servlet-mapping>
24</web-app>
25

```

3) 访问 <http://localhost:8080/web04/other>

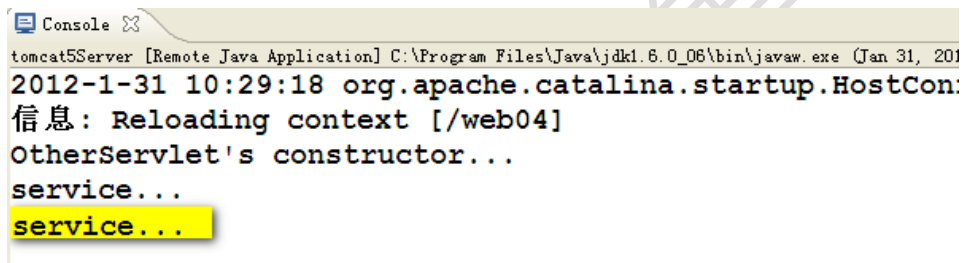
第 1 次请求，首先构造 Servlet 对象，其次调用 service()方法



```
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 2012-1-31 10:29:18 org.apache.catalina.startup.HostCon:
信息: Reloading context [/web04]
OtherServlet's constructor...
service...
```

4) 再次访问 <http://localhost:8080/web04/other>

当第 2 次发送请求时，直接调用 service()方法，因为 Servlet 对象已经创建



```
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 2012-1-31 10:29:18 org.apache.catalina.startup.HostCon:
信息: Reloading context [/web04]
OtherServlet's constructor...
service...
service...
```

5) 修改 web.xml

```

web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="2.4"
3     xmlns="http://java.sun.com/xml/ns/j2ee"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
5     xsi:schemaLocation="http://java.sun.com/xml/ns
6     http://java.sun.com/xml/ns/j2ee/web-app_2_4.xs
7 <servlet>
8     <servlet-name>someServlet</servlet-name>
9     <servlet-class>web.SomeServlet</servlet-cla
10 </servlet>
11 <servlet>
12     <servlet-name>otherServlet</servlet-name>
13     <servlet-class>web.OtherServlet</servlet-c
14     <!-- 每个servlet都可以配置该参数,
15          按参数从小到大顺序加载每个Servlet
16     -->
17     <load-on-startup>1</load-on-startup>
18 </servlet>
19 <servlet-mapping>

```

6) 重新部署

在没有访问 <http://localhost:8080/web04/other> 之前，程序部署之后就加载了 OtherServlet

容器在启动时（或者新部署了某个应用时）会检查 web.xml 当中的 servlet 是否有 load-on-startup 配置。如果有，则会创建该 servlet 实例。

load-on-startup 参数值越小，优先级越高（最小值为 0，优先级最高）

```

Console
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 20
2012-1-31 10:57:39 org.apache.catalina.startup.HostCon
信息: Reloading context [/web04]
OtherServlet's constructor...

```

7) 访问 <http://localhost:8080/web04/other>

将不再创建 OtherServlet

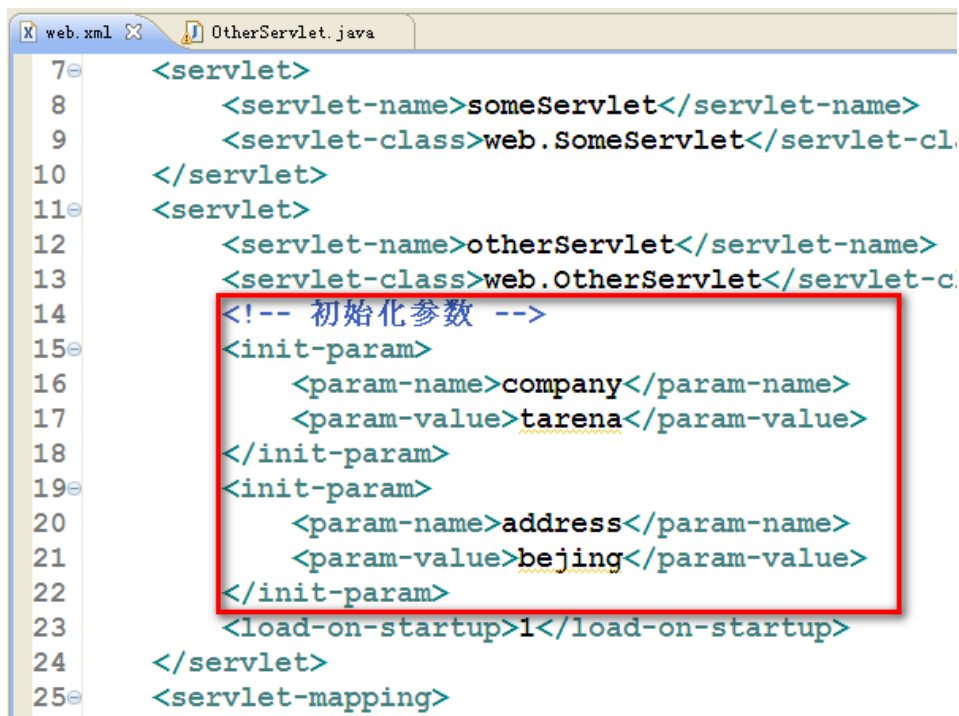
```

Console
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 20
2012-1-31 10:57:39 org.apache.catalina.startup.HostCon
信息: Reloading context [/web04]
OtherServlet's constructor...
service...

```

- 阶段二：初始化

8) 修改 web.xml，为 OtherServlet 增加初始化参数



```

7    <servlet>
8        <servlet-name>someServlet</servlet-name>
9        <servlet-class>web.SomeServlet</servlet-cl
10    </servlet>
11    <servlet>
12        <servlet-name>otherServlet</servlet-name>
13        <servlet-class>web.OtherServlet</servlet-c
14        <!-- 初始化参数 -->
15        <init-param>
16            <param-name>company</param-name>
17            <param-value>tarena</param-value>
18        </init-param>
19        <init-param>
20            <param-name>address</param-name>
21            <param-value>beijing</param-value>
22        </init-param>
23        <load-on-startup>1</load-on-startup>
24    </servlet>
25    <servlet-mapping>

```

9) 修改 OtherServlet.java，在 OtherServlet 中访问初始化参数

```

1 package web;
2
3 import java.io.IOException;
10
11 public class OtherServlet extends HttpServlet{
12
13     public OtherServlet() {
14         System.out.println(
15             "OtherServlet's constructor...");
16     }
17
18     public void service(HttpServletRequest request,
19                         HttpServletResponse response)
20     throws ServletException, IOException{
21         System.out.println("service...");
22         //获得初始化参数使用ServletConfig对象
23         ServletConfig config = getServletConfig();
24         String company =
25             config.getInitParameter("company");
26         System.out.println(company);
27     }
28 }
29

```

10) 重新部署&& 访问 <http://localhost:8080/web04/other>
打印获得的初始化参数

```

Console X
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 20
2012-1-31 11:32:00 org.apache.catalina.startup.HostCon
信息: Reloading context [/web04]
OtherServlet's constructor...
service...
tarena

```

11) 修改 OtherServlet.java , 覆盖 init 方法
init 方法是可以覆盖的

```

34
35 @Override
36 public void init(ServletConfig config)
37     throws ServletException {
38     /* super.init() 必须留着、
39     * 删除意味着父类的init方法没有调用,
40     * 则无法调用ServletConfig
41     */
42     super.init(config);
43     System.out.println(
44         "你自己的初始化代码...");
45 }
46 }
47

```

12) 重新部署

在初始化时，可以自定义一些操作，如下输出了一句话

```

Console X
tomcat5Server [Remote Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Jan 31, 20
2012-1-31 13:54:22 org.apache.catalina.startup.HostCon
信息: Reloading context [/web04]
OtherServlet's constructor...
你自己的初始化代码...

```

13) 修改 OtherServlet.java，覆盖无参数的 init() 方法

此时不需要加 super.init() 语句了，覆盖此方法更方便

```

28
29 @Override
30 public void init() throws ServletException {
31     System.out.println(
32         "你自己的初始化代码...");
33 }
34
35 // @Override
36 // public void init(ServletConfig config)
37 //     throws ServletException {
38 //     /* super.init()必须留着、
39 //     * 删除意味着父类的init方法没有调用,
40 //     * 则无法调用ServletConfig
41 //     */
42 //     super.init(config);
43 //     System.out.println(
44 //         "你自己的初始化代码...");
45 // }
46 }
47

```

- 阶段三：就绪/调用

- 14) 修改 OtherServlet.java

- 注释 service 方法，覆盖 doGet()和 doPost()方法

有请求到达容器，容器调用 servlet 对象的 service()方法，

HttpServlet 的 service()方法，会依据请求方式来调用 doGet()或者 doPost()方法，

但是，这两个 do 方法默认情况下，会抛出异常，需要子类去 override

```

17
18 // public void service(HttpServletRequest request
19 //     HttpServletResponse response)
20 //     throws ServletException, IOException {
21 //     System.out.println("service...");
22 //     //获得初始化参数使用ServletConfig对象
23 //     ServletConfig config = getServletConfig();
24 //     String company =
25 //         config.getInitParameter("company");
26 //     System.out.println(company);
27 // }
28
29 @Override
30 protected void doGet(
31     HttpServletRequest req,
32     HttpServletResponse resp)
33     throws ServletException, IOException {
34     System.out.println("doGet...");
35     ServletConfig config = getServletConfig();
36     String company =
37         config.getInitParameter("company");
38     System.out.println(company);
39 }
40
41 @Override
42 protected void doPost(
43     HttpServletRequest req,
44     HttpServletResponse resp)
45     throws ServletException, IOException {
46     doGet(req, resp);
47 }

```

● 阶段四：销毁

容器依据自身的算法，将不再需要的 servlet 对象删除掉。

在删除之前，会调用 servlet 对象的 destroy()方法。

destroy()方法用于释放资源。

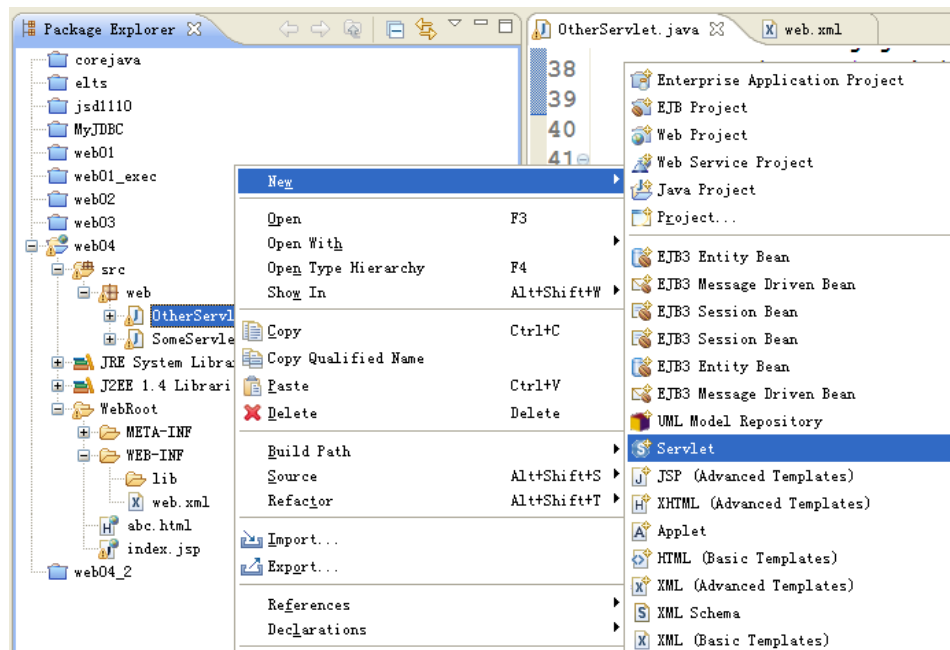
在 servlet 的整个生命周期当中，init 和 destroy 只会执行一次，而 service 方法会执行多次

演示

MyEclipse 工具生成 Servlet

1) 右键“new”Servlet

注：如果没有，则点“other”，在其中找



2) "Create a new Servlet"

Create a new Servlet.

Servlet Wizard
Create a new Servlet class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ default ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Template to use:

Options:

<input type="checkbox"/> Create Inherited Methods	<input checked="" type="checkbox"/> Create doGet
<input type="checkbox"/> Create Constructors	<input type="checkbox"/> Create doPost
<input type="checkbox"/> Create init and destroy	<input type="checkbox"/> Create doPut
<input type="checkbox"/> Create doDelete	<input type="checkbox"/> Create getServletInfo

3) MyEclipse 工具帮助配置 web.xml

Create a new Servlet.

XML Wizard

☒ Generate/Map web.xml file

Servlet/JSP Class Name:

Servlet/JSP Name:

Servlet/JSP Mapping URL:

File Path of web.xml:

Display Name:

Description:

4) 生成 SomeServlet2.java&&自动配置了 web.xml

```

1 package web;
2
3 import java.io.IOException;
4
10
11 public class SomeServlet2 extends HttpServlet {
12
13     /**
14      * The doGet method of the servlet. <br>
15      *
16      * This method is called when a form has its t
17      *
18      * @param request the request send by the client
19      * @param response the response send by the server
20      * @throws ServletException if an error occurred
21      * @throws IOException if an error occurred
22      */
23     public void doGet(HttpServletRequest request, HttpServletResponse response)
24         throws ServletException, IOException {
25
26         response.setContentType("text/html");

```

5. Jsp **

1) 什么是jsp? *

java server page(java 服务器端页面技术), 是 sun 公司制订的一种服务器端动态页面生成技术的规范。

因为直接使用 servlet 生成页面, 如果页面比较复杂, 则代码过于繁琐, 并且难以维护, 所以对于比较复杂的页面, 使用 jsp 来编写, 更容易编写和维护。

2) 如何写一个 jsp 文件 **

写一个以.jsp 为后缀的文件, 然后, 在该文件当中, 添加 html 和 java 代码。编写完成之后, 不需要编译, 当客户端请求访问某个.jsp 文件, 则服务器会自动将.jsp 文件转换成一个.java 文件(该.java 文件其实是一个 servlet)。

3) jsp 文件的组成 **

a. html(包括 css、javascript)

直接写在.jsp 文件里即可

b. java 代码

➤ 第一种形式: java 代码片断

语法: `<% java 代码 %>`

- **第二种形式：** jsp 表达式
语法： **<%= java 表达式%>**

c. 指令

- 所谓指令，就是告诉 jsp 引擎(容器当中，负责将.jsp 文件转换成.java 文件，并在运行时，为 jsp 提供一些 辅助支持的模块),在将.jsp 文件转换成.java 文件时，做一些额外的处理。
- 语法: **<%@ 指令名 属性名=属性值%>**
- **page 指令**
##常用指令共 3 个，今天了解并掌握 page 指令##
 - **import 属性：**
用于导包。
比如<%@page import="java.util.*,java.text.*"%>
 - **contentType 属性:**
等价于 response.setContentType();
 - **pageEncoding 属性:**
告诉 jsp 引擎，.jsp 文件保存时的编码。

d. 隐含对象

所谓隐含对象，指的是在.jsp 文件当中，不用声明和创建该对象，就可以直接使用的对象。原因是，.jsp 文件对应的.java 文件当中，已经自动生成了该对象的代码。

- **out**
- **request**
- **response**

4) jsp 文件如何转换成.java 文件 *

- ✓ html 转换为 jsp
放到 service()方法里，使用 out.write()输出。
- ✓ java 代码片断
放到 service()方法里，照搬

5.1. 如何写一个 jsp 文件 **

演示 1

JSP 演示

1) 新建工程 web04_jsp

2) 新建 SayHelloServlet

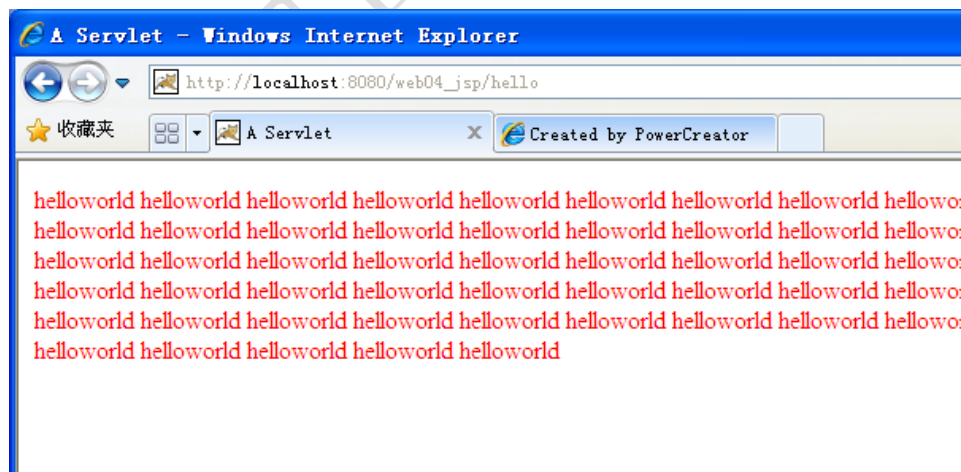
打印 100 次 helloworld，使用 Servlet 实现。

```

1 import java.io.IOException;
2
3
4
5
6
7
8 public class SayHelloServlet extends HttpServlet {
9     public void service(
10         HttpServletRequest request,
11         HttpServletResponse response)
12         throws ServletException, IOException {
13
14         response.setContentType("text/html");
15         PrintWriter out = response.getWriter();
16         out.println("<HTML>");
17         out.println("<HEAD>" +
18             "<TITLE>A Servlet</TITLE></HEAD>");
19         out.println(
20             "<BODY style='color:red;'>");
21         for(int i=0;i<100;i++){
22             out.println("helloworld");
23         }
24         out.println("</BODY>");
25         out.println("</HTML>");
26         out.flush();
27         out.close();
28     }
29 }

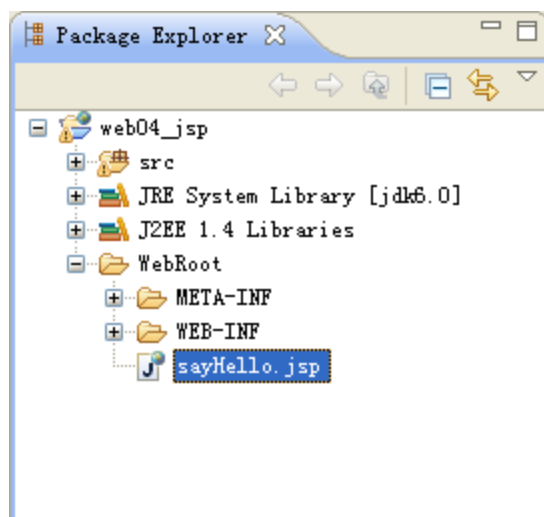
```

3) 访问 http://localhost:8080/web04_jsp/hello



4) 新建 sayHello.jsp

打印 100 次 helloworld，使用 JSP 实现。



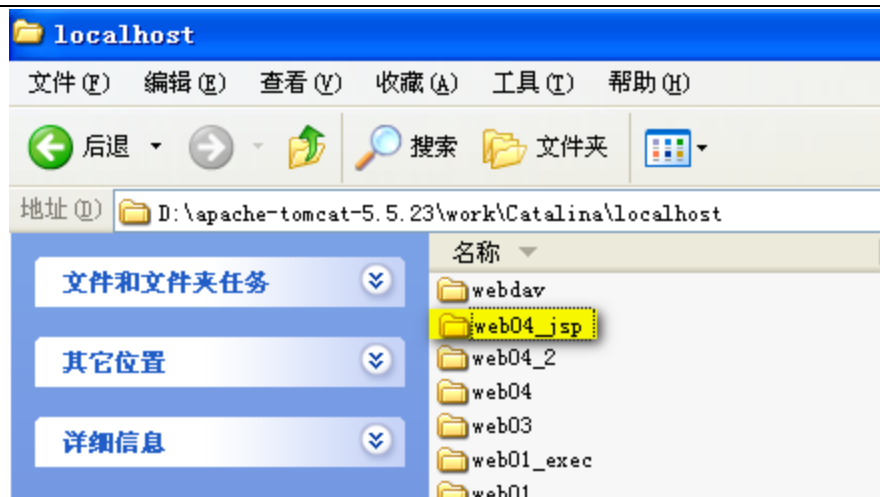
5) 访问 <http://localhost:8080/sayHello.jsp>



6) 进入 D:\apache-tomcat-5.5.23\work\Catalina\localhost 目录下

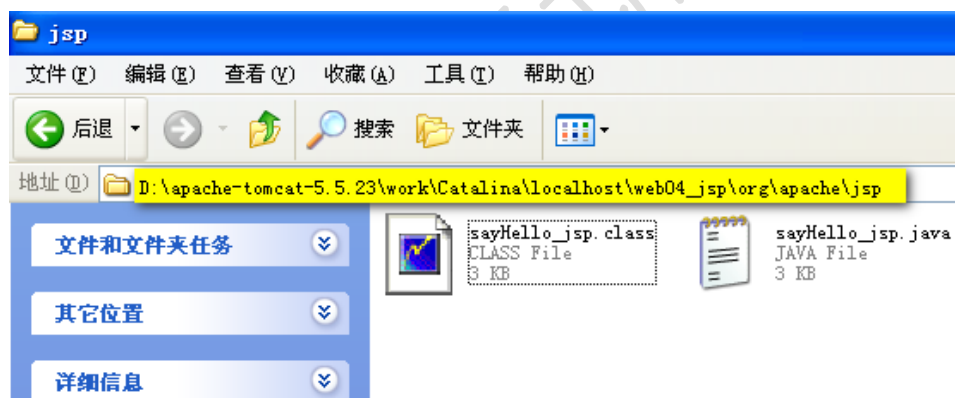
该目录下也有一个工程叫 web04_jsp

work 目录主要是用于存放 JSP 页面生成的 Servlet



7) 进入目录

D:\apache-tomcat-5.5.23\work\Catalina\localhost\web04_jsp\org\apache\jsp
sayHello.jsp 生成的 Servlet 源文件 (sayHello_jsp.java) 及编译后的二进制文件
(sayHello_jsp.class)



8) 打开 sayHello_jsp.java

结构与 Servlet 相似


```
sayHello_jsp.java
0 10 20 30 40 50
1 package org.apache.jsp;
2
3 import javax.servlet.*;
4 import javax.servlet.http.*;
5 import javax.servlet.jsp.*;
6 import java.util.*;
7
8 public final class sayHello_jsp
9     extends
10         org.apache.jasper.runtime.HttpJspBase
11     implements
12         org.apache.jasper.runtime.JspSourceDependent {
13
14     private static java.util.List _jspx_dependants;
15
16     public Object getDependants() {
17         return _jspx_dependants;
18     }
19
20     public void _jspService(
21         HttpServletRequest request,
22         HttpServletResponse response)
23         throws java.io.IOException, ServletException {
24
```

演示 2

多态

1) 新建 A.java

在 service 方法中调用 _jspService 方法

```

A.java X B.java Test.java
1 package test;
2
3 public class A {
4     public void service(){
5         System.out.println("service...");
6         _jspService();
7     }
8
9     protected void _jspService(){
10        System.out.println(
11            "A's _jspService...");
12    }
13 }
14

```

2) 新建 B.java

继承类 A，并覆盖类 A 的 _jspService 方法

```

A.java B.java X Test.java
1 package test;
2
3 public class B extends A{
4
5     @Override
6     protected void _jspService() {
7         System.out.println(
8             "B's _jspService...");
9     }
10 }
11

```

3) 新建 Test.java

创建声明为父类（A）的子类（B）对象 a，调用 a 的 service 方法

```

1 package test;
2
3 public class Test {
4     public static void main(String[] args) {
5         A a = new B();
6         a.service();
7     }
8 }
9

```

4) 运行 Test.java

父类方法被子类覆盖

```

1 package test;
2
3 public class Test {
4     public static void main(String[] args) {
5         A a = new B();
6         a.service();
7     }
8 }

```

Console

```

<terminated> Test (5) [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Feb 1, 201
service...
B's _jspService...

```

5.2.jsp 文件的组成 **

1) html(包括 css、javascript)

直接写在.jsp 文件里即可

2) java 代码

第一种形式： java 代码片断

语法： `<% java 代码 %>`

第二种形式： jsp 表达式

语法： `<%= java 表达式 %>`

3) 指令

所谓指令，就是告诉 jsp 引擎(容器当中，负责将.jsp 文件转换成.java 文件，并在运行时，为 jsp 提供一些 辅助支持的模块),在将.jsp 文件转换成.java 文件时，做一些额外的 处理。

语法: **<%@ 指令名 属性名=属性值%>**

page 指令

##常用指令共 3 个，今天了解并掌握 page 指令##

- ✓ **import 属性：**
用于导包。比如<%@page import="java.util.*,java.text.*"%>
- ✓ **contentType 属性:**
等价于 response.setContentType();
- ✓ **pageEncoding 属性:**
告诉 jsp 引擎，.jsp 文件保存时的编码。

4) 隐含对象

所谓隐含对象，指的是在.jsp 文件当中，不用声明和创建该对象，就可以直接使用的对象。原因是，.jsp 文件对应的.java 文件当中，已经自动生成了该对象的代码。

- ✓ **out**
- ✓ **request**
- ✓ **response**

【案例 3】JSP 演示 *

1) sayHello.jsp

2) 访问 http://localhost:8080/web04_jsp/sayHello.jsp



【案例 4】JSP 实现员工增删改查 **

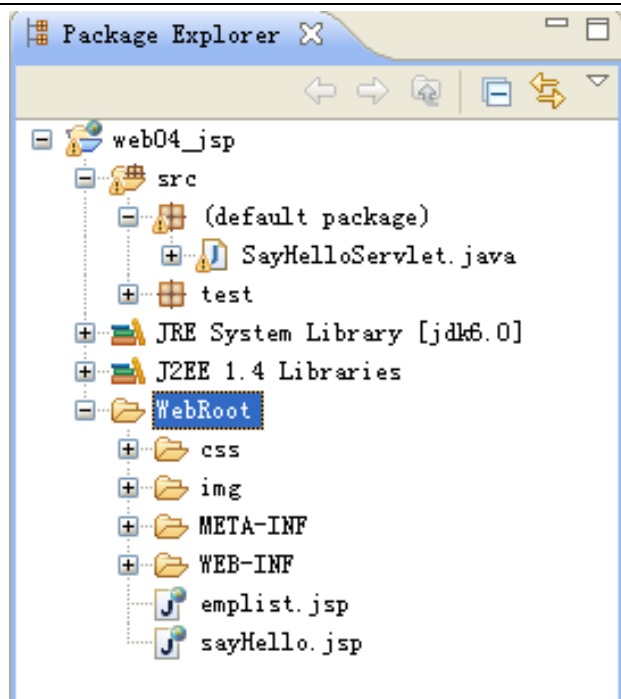
1) 准备静态页面

请下载 [emsdemo.zip](#)



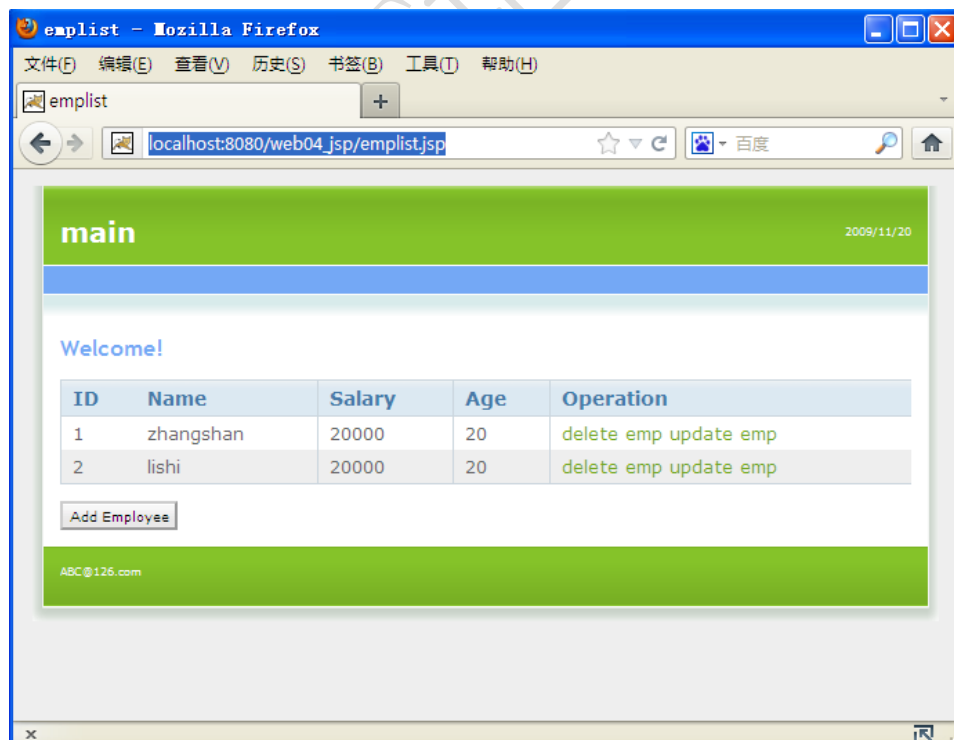
2) 拷贝 css、img 两个目录到项目下

3) 拷贝 emplist.html 到 WebRoot 目录下，并改名为 emplist.jsp



4) 测试

访问 http://localhost:8080/web04_jsp/emplist.jsp



5) 修改 emplist.jsp

加入 page 指令

```
<%@page pageEncoding="utf-8"
contentType="text/html;charset=utf-8" %>
<html>
  <head>
    <title>emplist</title>
    <meta http-equiv="Content-Type"
      content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css"
      href="css/style.css" />
  </head>
  <body>
    <div id="wrap">
      <div id="top_content">
        <div id="header">
          <div id="righthead">
            <p>
              2009/11/20
            <br />
          </p>
        </div>
        <div id="topheader">
          <h1 id="title">
            <a href="#">main</a>
          </h1>
        </div>
        <div id="navigation">
        </div>
      </div>
      <div id="content">
        <p id="whereami">
        </p>
        <h1>
          欢迎
        </h1>
        <table class="table">
```



```

<tr class="table_header">
  <td>ID</td>
  <td>姓名</td>
  <td>薪水</td>
  <td>年龄</td>
  <td>操作</td>
</tr>
<%
  //访问数据库代码
%>
<tr class="row1">
  <td>
    1
  </td>
  <td>
    zhangshan
  </td>
  <td>
    20000
  </td>
  <td>
    20
  </td>
  <td>
    <a href="emplist.html">
      delete emp</a>&nbsp;  
    <a href="updateEmp.html">
      update emp</a>
    </td>
</tr>
</table>
<p>
  <input type="button"
    class="button"
    value="添加雇员"
    onclick="location='addEmp.html'"/>
</p>
</div>

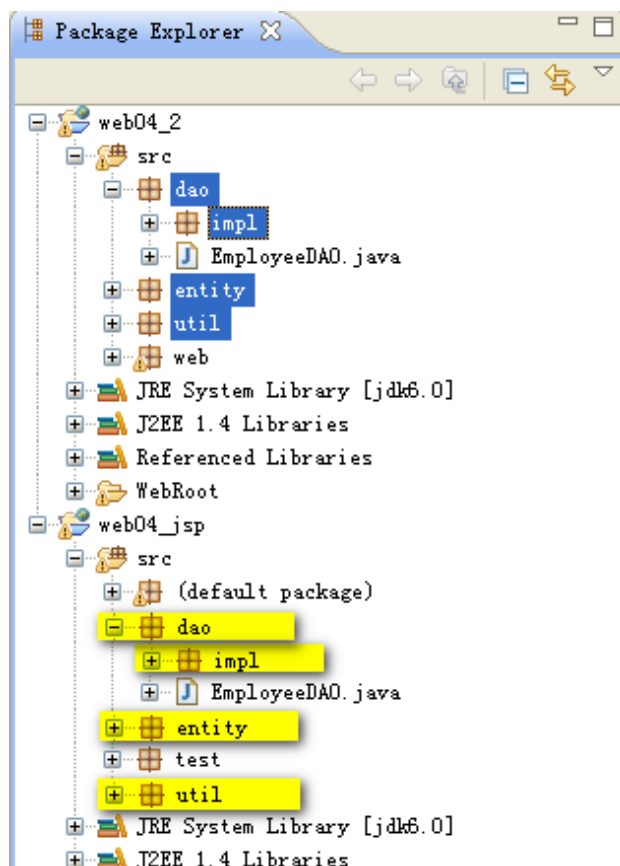
```

```

</div>
<div id="footer">
    <div id="footer_bg">
        ABC@126.com
    </div>
</div>
</div>
</body>
</html>

```

6) 将 DAO 操作相关的类拷贝到 web04_jsp 项目下



7) 导入 mysql 驱动

8) 测试

测试 DAO 操作是否 OK

```

1 package test;
2
3 import java.util.List;
4
5
6
7
8 public class DAOTest {
9     public static void main(String[] args)
10        throws Exception{
11        EmployeeDAO dao =
12            (EmployeeDAO) DAOFactory
13                .getInstance("EmployeeDAO");
14        List<Employee> employees = dao.findAll();
15
16        System.out.println("##test findAll##");
17        for(int i=0;i<employees.size();i++){
18            Employee e = employees.get(i);
19            System.out.println(e.getName());
20        }
21    }
22 }

```

Console

```

<terminated> DAOTest [Java Application] C:\Program Files\Java\jdk1.6.0_06\bin\javaw.exe (Feb 1, 2012)
##test findAll##
ls
ww
you
wwq

```

9) 修改 emplist.jsp

加入 Java 代码

```

<%@page pageEncoding="utf-8"
    contentType="text/html; charset=utf-8" %>
<%@page import="dao.*,util.*,java.util.*,entity.*" %>
<html>
    <head>
        <title>emplist</title>
        <meta http-equiv="Content-Type"
            content="text/html; charset=UTF-8">
        <link rel="stylesheet" type="text/css"
            href="css/style.css" />
    </head>
    <body>
        <div id="wrap">

```

```

<div id="top_content">
    <div id="header">
        <div id="righthead">
            <p>
                2009/11/20
            <br />
            </p>
        </div>
        <div id="topheader">
            <h1 id="title">
                <a href="#">main</a>
            </h1>
        </div>
        <div id="navigation">
        </div>
    </div>
    <div id="content">
        <p id="whereami">
        </p>
        <h1>
            欢迎
        </h1>
        <table class="table">
            <tr class="table_header">
                <td>ID</td>
                <td>姓名</td>
                <td>薪水</td>
                <td>年龄</td>
                <td>操作</td>
            </tr>

<%
EmployeeDAO dao =
    (EmployeeDAO) DAOFactory
        .getInstance("EmployeeDAO");
List<Employee> employees = dao.findAll();

for(int i=0;i<employees.size();i++){
Employee e = employees.get(i);

```

```

%>
    <tr class="row1">
    <td>
        <%=e.getId()%>
    </td>
    <td>
        <%=e.getName()%>
    </td>
    <td>
        <%=e.getSalary()%>
    </td>
    <td>
        <%=e.getAge()%>
    </td>
    <td>
        <a href="emplist.html">
            删除</a>&nbsp;
        <a href="updateEmp.html">
            更新</a>
        </td>
    </tr>
<%
}
%>
</table>
<p>
    <input type="button"
        class="button"
        value="Add Employee"
        onclick="location='addEmp.html'"/>
</p>
</div>
</div>
<div id="footer">
    <div id="footer_bg">
        ABC@126.com
    </div>
</div>
</div>

```

```
</div>
</body>
</html>
```

10) 测试

访问 http://localhost:8080/web04_jsp/emplist.jsp

main

2009/11/20

欢迎

ID	姓名	薪水	年龄	操作
6	ls	8887.0	21	删除 更新
7	ww	3001.0	22	删除 更新
10	you	5000.0	23	删除 更新
12	wwq	20000.0	30	删除 更新

Add Employee

ABC @ 126.com

11) 小技巧

```
<%
EmployeeDAO dao =
    (EmployeeDAO) DAOFactory
        .getInstance("EmployeeDAO");
List<Employee> employees = dao.findAll();

for(int i=0;i<employees.size();i++){
Employee e = employees.get(i);
%>
    <tr class="row<%= (i%2+1) %>">
    <td>
        <%=e.getId() %>
    </td>
    <td>
        <%=e.getName() %>
    </td>
```

显示效果如下

隔条记录显示指定样式

main2009/11/20

欢迎

ID	姓名	薪水	年龄	操作
6	ls	8887.0	21	删除 更新
7	ww	3001.0	22	删除 更新
10	you	5000.0	23	删除 更新
12	wwq	20000.0	30	删除 更新

Add Employee

ABC@126.com