# Servlet

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【应用】能够独立编写Servlet入门程序**
2. **【应用】能够阐述Servlet的生命周期**
3. **【应用】能够独立完成Servlet开发应用——登录案例**
4. **【应用】了解ServletConfig的使用**

# Servlet开发入门

## Servlet由来

目前我们已经可以让浏览器沟通服务器，访问服务器资源，但是之前都静态的html，服务器中的东西不肯都是静态的

第一天我们就说过，需要服务器，使用Java代码，访问数据库。

因此，我们需要书写java程序，去访问数据库，那么java程序在哪里运行？

用户的请求从浏览器发出，由服务器接收，因此，用户需要什么服务器才知道

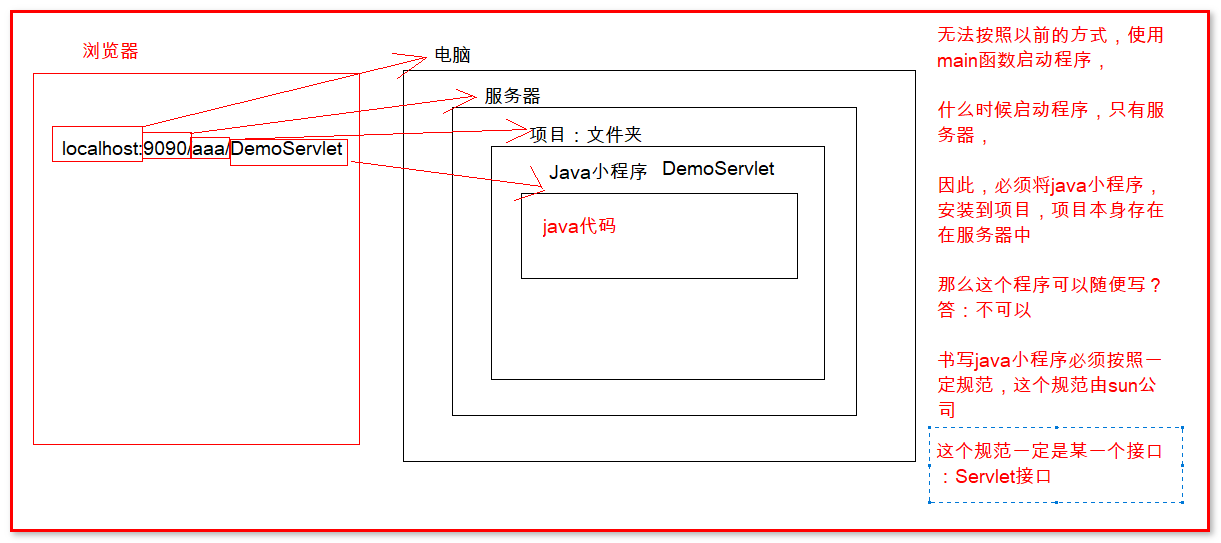
由此可见，这些java程序必须安装到服务器中，让服务器调用执行。

那么我们可以随意写java程序码？

不能，因为就像螺丝和螺帽一样，必须按照一定大小尺寸规定，生成，后期才可以匹配使用。

因此sun公司为了让程序员书写的java程序可以严丝合缝的安装到服务器中，定义了一个接口Servlet：

Servlet示意图



可以查阅JavaEE手册（帮助文档）阅读Servlet规范。



文档总结：

什么是servlet？

处理请求和响应的java程序。

怎么创建servlet？

继承HttpServlet类，在web.xml中配置

## Servlet开发的步骤：（重点）

1、在cn.igeek.web包下 创建一个类 实现 HttpServlet 类

|  |
| --- |
| **package** cn.igeek.web;  **import** javax.servlet.http.HttpServlet;  **public** **class** DemoServlet **extends** HttpServlet{  } |

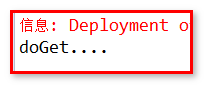
1. 重写doGet和doPost方法

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** DemoServlet **extends** HttpServlet{  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("doGet....");  }    @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("doPost....");  }  } |

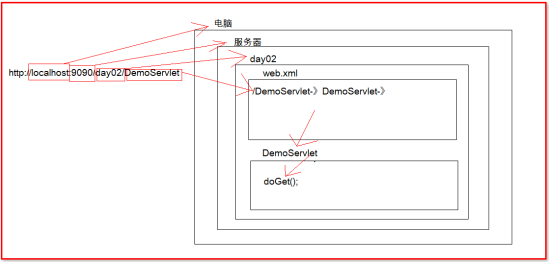
1. 在当前项目中的web.xml文件中配置当前开发好的Servlet程序

|  |
| --- |
| <servlet>  <!-- servlet的名称 -->  <servlet-name>DemoServlet</servlet-name>  <!-- servlet的具体实现类 -->  <servlet-class>cn.igeek.web.DemoServlet</servlet-class>  </servlet>  <servlet-mapping>  <!-- servlet的名称 -->  <servlet-name>DemoServlet</servlet-name>  <!-- 访问这个servlet程序的请求路径 -->  <url-pattern>/DemoServlet</url-pattern>  </servlet-mapping> |

4、发布这个项目，运行服务器，测试



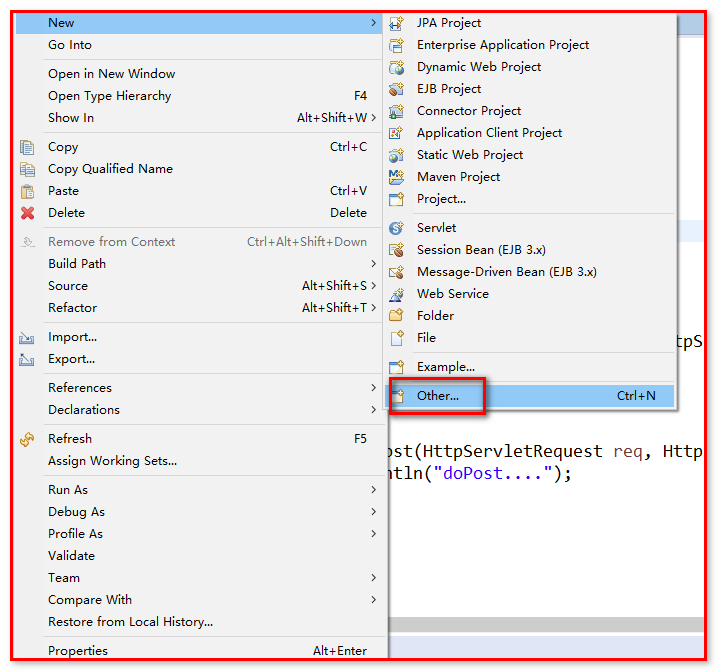
## Servlet中的细节

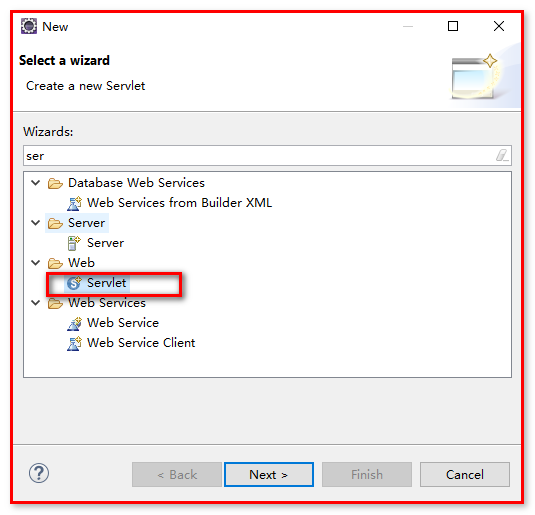


总结：通过请求路径，查询web.xml中servlet配置，获取当前servlet对象，调用java程序。

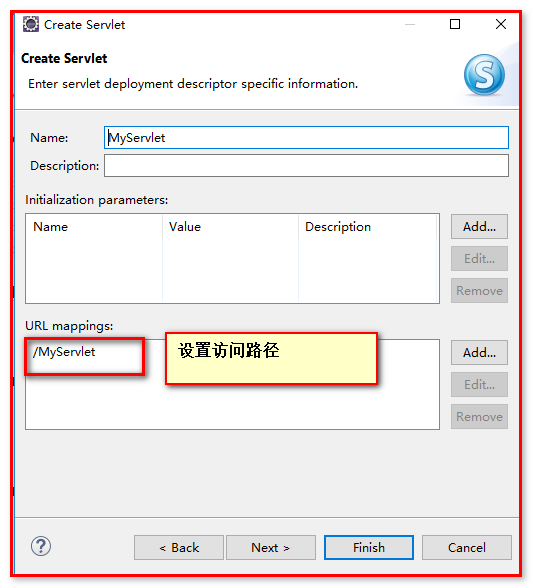
## 使用Eclipse模版开发Servlet（重点）：可以更快捷的创建servlet对象

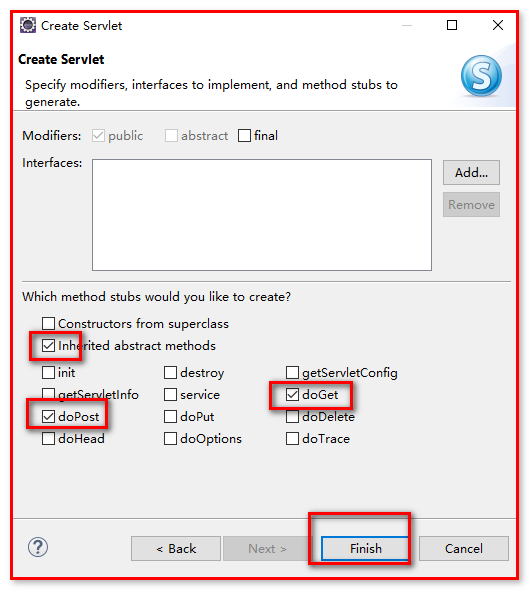
1、选中包名, Ctrl + n 新建servlet











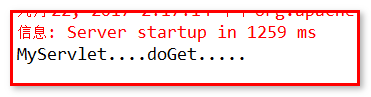
servlet效果：

|  |
| --- |
| package cn.igeek.web;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class MyServlet extends HttpServlet {  private static final long serialVersionUID = 1L;  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  System.out.println("MyServlet....doGet.....");  }  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  System.out.println("MyServlet....doPost.....");  }  } |

Web.xml效果;

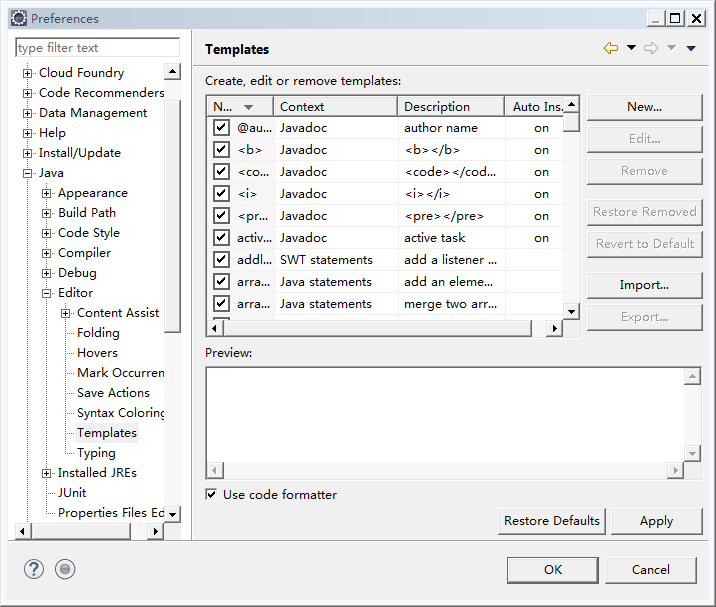
|  |
| --- |
| <servlet>  <description></description>  <display-name>MyServlet</display-name>  <servlet-name>MyServlet</servlet-name>  <servlet-class>cn.igeek.web.MyServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>MyServlet</servlet-name>  <url-pattern>/MyServlet</url-pattern>  </servlet-mapping> |

控制台效果：



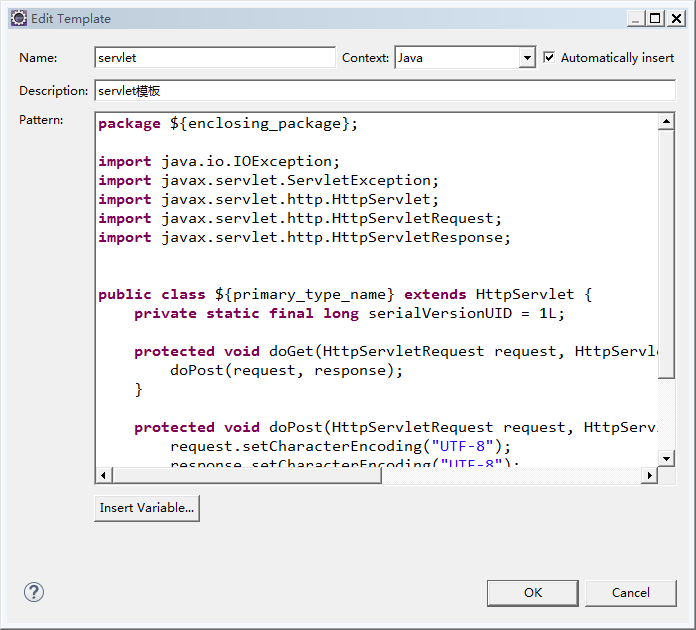
## 新建[servlet模板](http://www.cnblogs.com/qingcong/p/5824413.html)（了解）

1.首先找到菜单条上的Window并点击出现下拉列，选中下拉列中的Preferences选项跳出如下图结果



2.找到其中的java列点开——》找到其中的Editor点开——》找到其中的Templates点击出现如上图结果

点击右侧的新建按钮出现一个类似下图的窗口

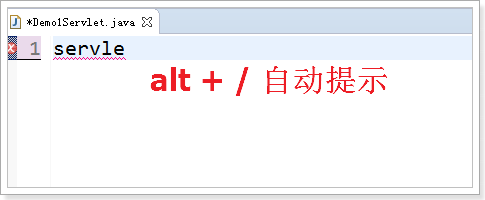


3.在界面上的Name和Description中键入如上内容，在下方的Pattern中输入你自己编写好的servlet模板

|  |
| --- |
| **package** ${enclosing\_package};  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** ${primary\_type\_name} **extends** HttpServlet {    **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {    }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

4.最后一路点击下方的OK按钮，之后回到java ee视图新建一个servlet文件，使用快捷键Ctrl+A全选中代码，

键入“servlet”字样，代码自动提示会出现如下结果，按一下Enter键就得到自己所要的代码。

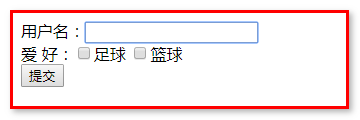


|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** DemoServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

## 获取请求（重点）

准备html

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <form action="/day02/my" method=*"post"*>  用户名：<input type=*"text"* name=*"username"*><br>  爱 好：<input type=*"checkbox"* name=*"hobby"* value=*"football"*>足球  <input type=*"checkbox"* name=*"hobby"* value=*"basketball"*>篮球<br>  <input type=*"submit"* value=*"提交"*>  </form>  </body>  </html> |



发送请求，之后要接收，对于提交上了的用户名，爱好，我们必须获取后，才能处理，那么请求参数如何获取？

使用服务器给我提供的一个对象Request对象！！！！！



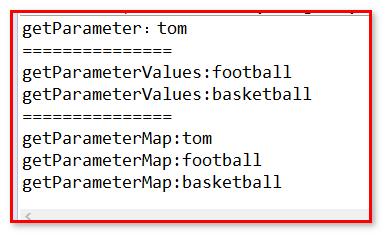
获取参数的方法有哪些？

|  |  |
| --- | --- |
| [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang) | **[getParameter](file:///C:/Users/Administrator/Documents/My%20Knowledge/temp/eac0abc7-8e82-4da1-a835-843b374202a1/128/index.htm" \l "getParameter(java.lang.String))**([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang) name)           根据name 获取对应的值 |
| [Map](http://java.sun.com/j2se/1.5/docs/api/java/util/Map.html" \o "class or interface in java.util) | **[getParameterMap](file:///C:/Users/Administrator/Documents/My%20Knowledge/temp/eac0abc7-8e82-4da1-a835-843b374202a1/128/index.htm" \l "getParameterMap())**()            参数名作为key，参数值作为value，封装到map中。 |
| [String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang)[] | **[getParameterValues](file:///C:/Users/Administrator/Documents/My%20Knowledge/temp/eac0abc7-8e82-4da1-a835-843b374202a1/128/index.htm" \l "getParameterValues(java.lang.String))**([String](http://java.sun.com/j2se/1.5/docs/api/java/lang/String.html" \o "class or interface in java.lang) name)            获取name相同的所有value 例如复选框。 |

需求：获取注册页面的值。

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** java.util.Map;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** MyServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  System.***out***.println("getParameter："+request.getParameter("username"));  System.***out***.println("===============");  String[] values = request.getParameterValues("hobby");  **for** (String string : values) {  System.***out***.println("getParameterValues:"+string);  }  System.***out***.println("===============");  Map<String, String[]> map = request.getParameterMap();  **for**(String key : map.keySet()){  **for** (String value : map.get(key)) {  System.***out***.println("getParameterMap:"+value);  }  }    }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

效果：



## 发送响应（重点）

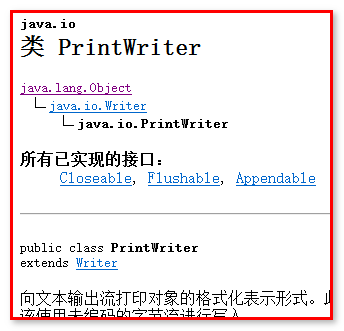
发送响应的对象Response对象：



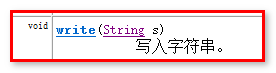
这个对象用于向浏览器发出响应，要使用的方法为：

|  |  |
| --- | --- |
| java.io.PrintWriter | getWriter ()           返回可将字符文本发送到客户端的 PrintWriter 对象 |

PrintWriter：



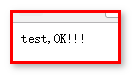
要使用的方法：



代码：

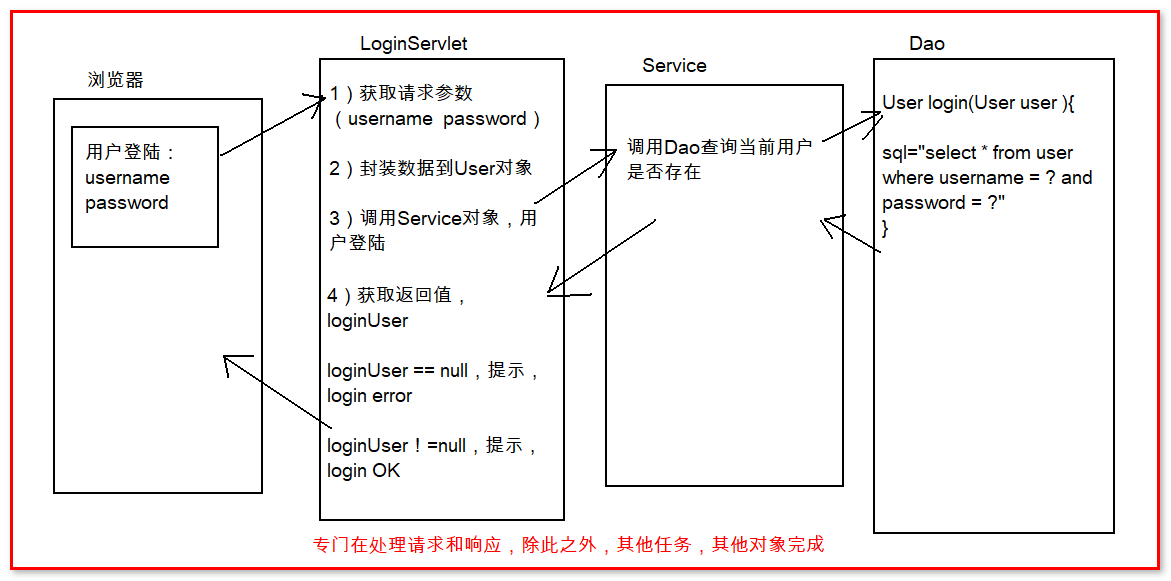
|  |
| --- |
| System.***out***.println("========发出响应===========");  response.getWriter().write("test OK!!!!"); |

效果：



# Servlet开发应用——登录案例(重点)

## 需求分析

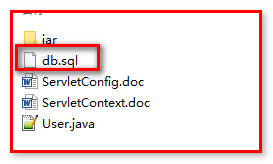


## 准备工作

页面：

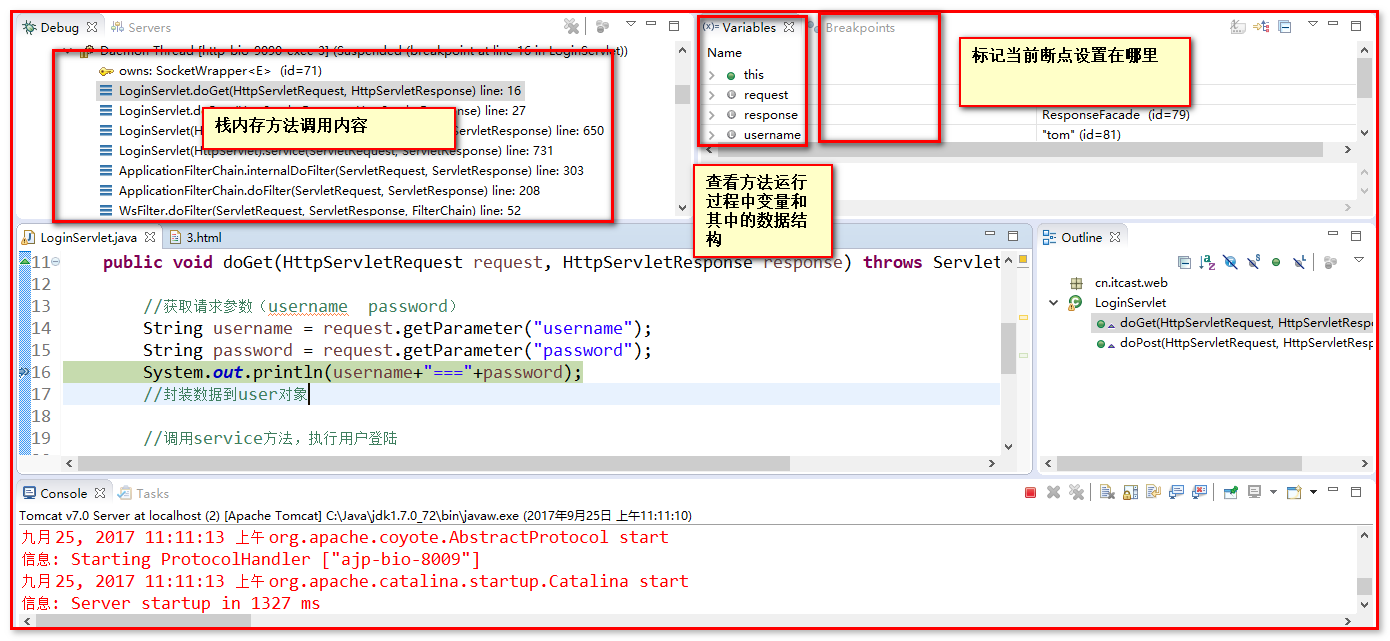
|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>  <title>Insert title here</title>  </head>  <body>  <form action=*"/day02/login"* method=*"post"*>  <table>  <tr>  <td>用户名</td>  <td><input type=*"text"* name=*"username"*/></td>  </tr>  <tr>  <td>密码</td>  <td><input type=*"password"* name=*"password"*/></td>  </tr>  <tr>  <td></td>  <td><input type=*"submit"* value=*"登录"*/></td>  </tr>  </table>  </form>  </body>  </html> |

数据库：资料文件夹中

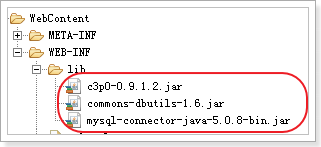


Debug调试模式界面介绍：

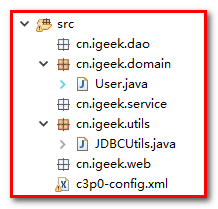
开启debug模式操作：1 服务器启动必须dubug模式 2 需要查看的位置双击设置断点



### 导入jar包

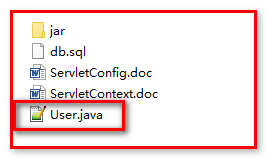


### 新建包结构

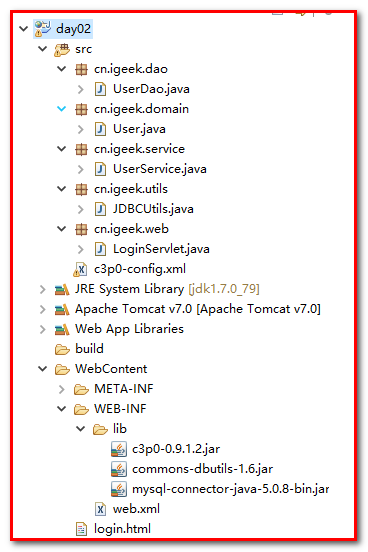


### 新建javabean

资料文件夹中



### 最终结构



## 登录servlet

package cn.igeek.web;

import java.io.IOException;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import cn.igeek.domain.User;

import cn.igeek.service.UserService;

/\*\*

\* @author jerry

\*

\* @time 2017年9月25日

\*

\* 总结：

\* 1 分析流程，画流程图

\* 2 根据流程开启代码 页面 servlet service dao

\* 3 测试 页面发送请求是否正确 servlet（处理请求和响应） service执行具体业务逻辑 dao 执行数据库操作

\*

\* 业务逻辑：

\* 梦境，入职腾讯游戏

\* 注册：

\* 1 将用户信息写入数据库

\* 2 送7天会员

\* 3 送新手装备

\* 4 设置邀请码

\* 5 送sss宝宝

\* 。。。。。

\*

\* 入职，陌陌：

\* 注册：

\* 1 将用户信息写入数据库

\* 2 推送xxx范围内，只看女生 ，只看男生

\* 3 初始化用户活跃度

\* 4 .。。。。

\*

\*

\*/

public class LoginServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

//获取请求参数（username password）

String username = request.getParameter("username");

String password = request.getParameter("password");

//开启debug模式操作：1 服务器启动必须dubug模式 2 需要查看的位置双击设置断点

//封装数据到user对象

User user = new User();

user.setUsername(username);

user.setPassword(password);

//调用service方法，执行用户登陆

UserService userService = new UserService();

User loginUser = userService.login(user);

//根据返回的结果，给出不同提示信息

if(loginUser == null){

//用户名或者密码错误，login error

response.getWriter().write("login error");

}else{

//登陆成功，login OK

response.getWriter().write("login OK");

}

}

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

}

## 业务层

|  |
| --- |
| package cn.igeek.service;  import cn.igeek.dao.UserDao;  import cn.igeek.domain.User;  public class UserService {  private UserDao userDao = new UserDao();  /\*\*  \* 用户登陆的方法  \* @param user  \* @return  \*/  public User login(User user) {  return userDao.login(user);  }  } |

## 数据层

package cn.igeek.dao;

import java.sql.SQLException;

import org.apache.commons.dbutils.QueryRunner;

import org.apache.commons.dbutils.handlers.BeanHandler;

import cn.igeek.domain.User;

import cn.igeek.utils.JDBCUtils;

public class UserDao {

private QueryRunner qr = new QueryRunner(JDBCUtils.getDataSource());

/\*\*

\* 处理用户登陆的方法

\* @param user

\* @return

\*/

public User login(User user) {

String sql = "select \* from user where username = ? and password = ?";

//为什么要抓取异常？

try {

return qr.query(sql, new BeanHandler<User>(User.class), user.getUsername(),user.getPassword());

} catch (SQLException e) {

e.printStackTrace();

//对上面这些异常信息，做了一个中文注释

throw new RuntimeException("用户登陆失败");

}

//try catch finally throw throws

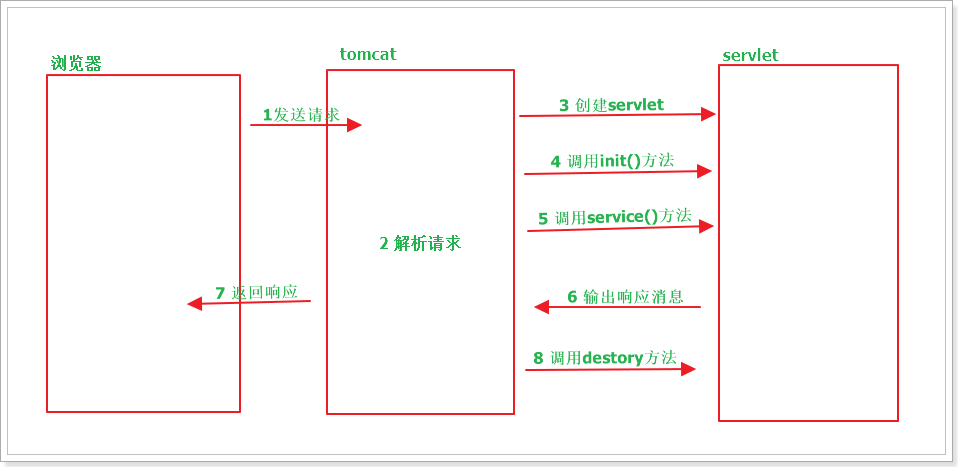
}

}

# servlet其他

## Servlet的生命周期（了解）

生命周期：是说一个对象从创建到销毁的整个过程。称为整个对象的一个生命周期。



上图描述了servlet的生命周期。按照功能的不同，大致分为三个阶段，分别是 初始化阶段，运行阶段（最频繁） 和 销毁阶段。

注意：当服务器关闭的时候，项目中的servlet就调用第8步销毁。

总结：Servlet 的创建，在用户第一次请求，之后，创建。销毁，是发生在服务器关闭的时候。

### 初始化阶段

当客户端向tomcat发送http请求访问servelt程序，tomcat首先会解析请求，检查内存中是否已经有了该servlet对象：

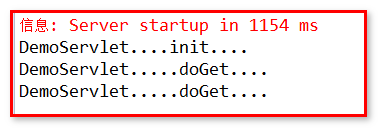
如果有直接使用对应的servlet对象;

如果没有就创建servlet实例对象，然后通过调用init() 方法实现Servlet的初始化工作。

需要注意的是，在整个servlet的生命周期内，init方法只被调用了一次。

测试：添加了init方法

|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** DemoServlet **extends** HttpServlet{  @Override  **public** **void** init() **throws** ServletException {  System.***out***.println("DemoServlet....init....");  **super**.init();  }  @Override  **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("DemoServlet.....doGet....");  }    @Override  **protected** **void** doPost(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  System.***out***.println("DemoServlet.....doPost....");  }  } |



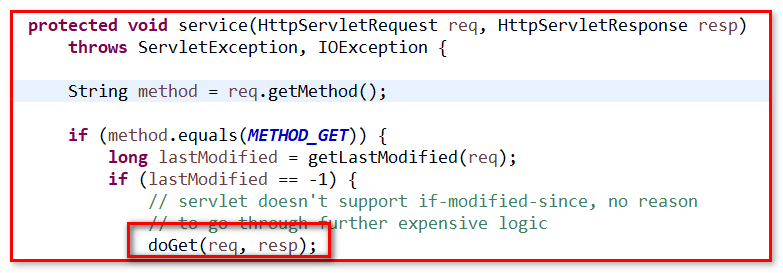
### 运行阶段

在这个阶段，tomcat服务器会为这个请求 创建 代表HTTP请求的ServletRequest对象 和 代表HTTP响应的 ServletResponse对象，然后将他们作为参数传递给service() 方法。

servcie() 方法从ServletRequest对象获取请求的信息并做出处理；通过ServletResponse 对象生成响应的结果。

在servlet的整个生命周期内，对于servlet的每一次访问请求，tomcat都会调用servlet的service方法，并且创建新的ServletRequest对象和ServletResponse对象。也就是说service() 方法会servlet的生命周期内会被调用多次。

doGet和doPost由service方法调用:在我们写的servlet父类HttpServlet中源码：



源码展示：////////////////标记重点////////////////////

|  |
| --- |
| **protected** **void** service(HttpServletRequest req, HttpServletResponse resp)  **throws** ServletException, IOException {  String method = req.getMethod();  /////////////////////////////////////////////  **if** (method.equals(***METHOD\_GET***)) {  **long** lastModified = getLastModified(req);  **if** (lastModified == -1) {  // servlet doesn't support if-modified-since, no reason  // to go through further expensive logic  doGet(req, resp);  //////////////////////////////////////////////////////  } **else** {  **long** ifModifiedSince = req.getDateHeader(***HEADER\_IFMODSINCE***);  **if** (ifModifiedSince < (lastModified / 1000 \* 1000)) {  // If the servlet mod time is later, call doGet()  // Round down to the nearest second for a proper compare  // A ifModifiedSince of -1 will always be less  maybeSetLastModified(resp, lastModified);  doGet(req, resp);  } **else** {  resp.setStatus(HttpServletResponse.***SC\_NOT\_MODIFIED***);  }  }  } **else** **if** (method.equals(***METHOD\_HEAD***)) {  **long** lastModified = getLastModified(req);  maybeSetLastModified(resp, lastModified);  doHead(req, resp);  } **else** **if** (method.equals(***METHOD\_POST***)) {  ///////////////////////////////////////////  doPost(req, resp);  ///////////////////////////////////////////    } **else** **if** (method.equals(***METHOD\_PUT***)) {  doPut(req, resp);    } **else** **if** (method.equals(***METHOD\_DELETE***)) {  doDelete(req, resp);    } **else** **if** (method.equals(***METHOD\_OPTIONS***)) {  doOptions(req,resp);    } **else** **if** (method.equals(***METHOD\_TRACE***)) {  doTrace(req,resp);    } **else** {  //  // Note that this means NO servlet supports whatever  // method was requested, anywhere on this server.  //  String errMsg = *lStrings*.getString("http.method\_not\_implemented");  Object[] errArgs = **new** Object[1];  errArgs[0] = method;  errMsg = MessageFormat.*format*(errMsg, errArgs);    resp.sendError(HttpServletResponse.***SC\_NOT\_IMPLEMENTED***, errMsg);  }  } |

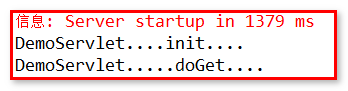
### 销毁阶段

当服务器关闭时，servlet会随着Web应用的销毁而销毁。

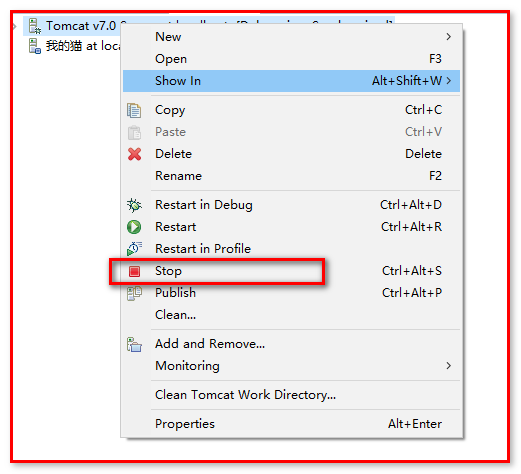
在销毁serlvet之前，tomcat 会调用Servlet的destory方法，以便让Servlet对象释放他所占用的资源。

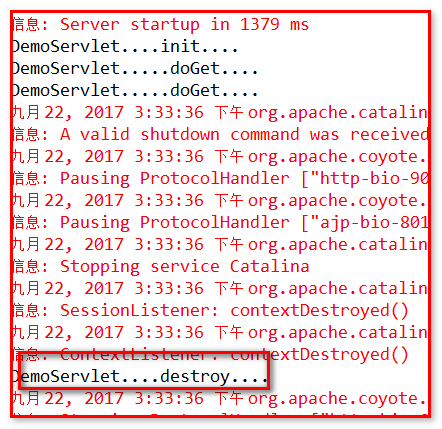
演示destory方法调用：

效果：第一次访问创建servlet



关闭服务器：





## 路径书写问题

### web.xml中url-pattern三种书写



在学习Servlet的时候，每个Servlet程序都需要在当前的这个项目的web.xml文件中注册和映射。

在映射中书写的url-pattern标签的书写方式：

#### 全路径匹配

在书写url-pattern的时候，必须以/开始，后面书写具体浏览器访问时的路径。

配置：<url-pattern>/DemoServlet</url-pattern>

外界的访问方式：

http://localhost:9090/test/DemoServlet

#### 路径通配符匹配

在书写url-pattern 的时候，以/开始，后面可以使用\*号表示任意的匹配

配置：<url-pattern>/DemoServlet/\*</url-pattern>

外界在访问的时候，只要能够和/demo2匹配上，后面写任何东西都可以

http://localhost:9090/test/DemoServlet/11111/aaa

#### 扩展名匹配

在书写url-pattern 的时候，不能以/开始，以\*开始，后面书写扩展名

配置： <url-pattern>\*.do</url-pattern>

常见的扩展名书写：

\*.action \*.do \* .go

访问的方式：

http://localhost:9090/test/xxxx.do

url-pattern标签中的路径可以按照上述的三种书写，它们的优先级：

全路径匹配 > 路径通配符匹配 > 扩展名匹配

测试配置：

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>test</display-name>  <servlet>  <servlet-name>DemoServlet</servlet-name>  <servlet-class>cn.igeek.web.DemoServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>DemoServlet</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>  <welcome-file-list>  <welcome-file>index.html</welcome-file>  <welcome-file>index.htm</welcome-file>  <welcome-file>index.jsp</welcome-file>  <welcome-file>default.html</welcome-file>  <welcome-file>default.htm</welcome-file>  <welcome-file>default.jsp</welcome-file>  </welcome-file-list>  <!-- <servlet>  <servlet-name>MyServlet</servlet-name>  <servlet-class>cn.igeek.web.MyServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>MyServlet</servlet-name>  <url-pattern>/\*</url-pattern>  </servlet-mapping> -->  <!-- <servlet>  <servlet-name>TestServlet</servlet-name>  <servlet-class>cn.igeek.web.TestServlet</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>TestServlet</servlet-name>  <url-pattern>/TestServlet.do</url-pattern>  </servlet-mapping> -->  </web-app> |

测试servlet:

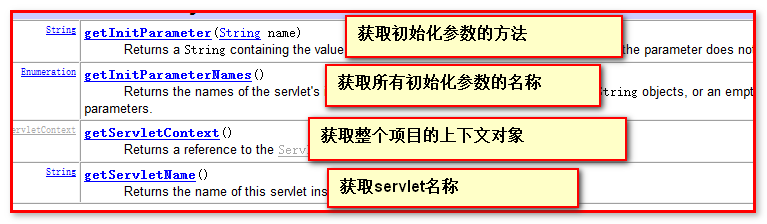
|  |
| --- |
| **package** cn.igeek.web;  **import** java.io.IOException;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServlet;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **public** **class** TestServlet **extends** HttpServlet {  **public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  System.***out***.println("TestServlet.....doGet.....");  }  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  doGet(request, response);  }  } |

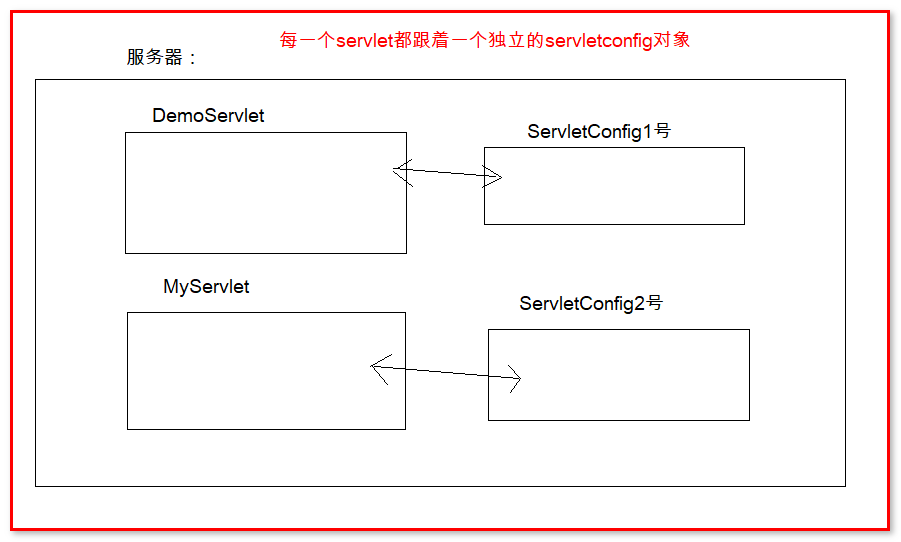
# ServletConfig介绍

应用场景:

为了将一些灵活需要修改的内容，设置到web.xml中，后期 ，可以在不修改代码的情况下，通过修改配置文件，达到，修改内容的目的





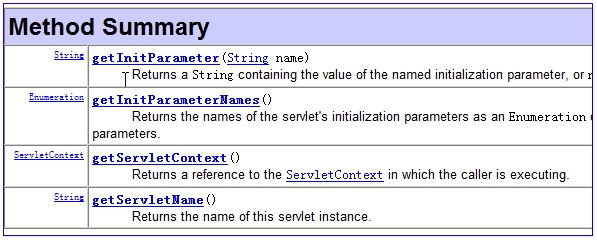


servlet容器: tomcat(java的web服务器)

sun公司要求服务器,创建servlet的同时,再去创建一个servletConfig对象. ServletConfig包含了一些Servlet的配置信息.

再调用servlet的init方法时,将servletCongfig传给servlet

servletConfig的内容是  是自己定义的. 在web.xml中 的servlet标签下书写servletConfig的内容.



ServletConfig它读取某个Servlet的配置信息。

1、如果servlet需要一些参数. 我们可以在web.xml的servlet标签内进行配置

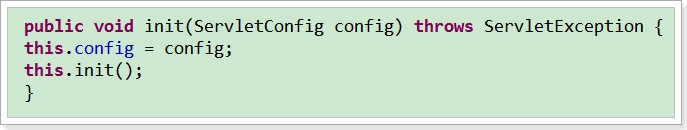


2 tomcat去工作: 读取servlet中init-param这个标签. 把里面的内容封装到ServletConfig中

3 tomcat去工作: 调用servlet的init方法, 把封装好的 servletConfig对象, 传递给servlet中.

调用servelt中的init(ServletConfig config)





4 如何使用

获取servletConfig对象, 通过servlet的getServletConfig() 方法



获取内部的参数值: 通过servletConfig.getInitParameter(key)



1. 重启tomcat，地址栏访问 对应的servlet