Servlet

学习目标

- [ ] 能够使用 IDEA 编写 Servlet
- [ ] 能够使用注解开发 Servlet
- [ ] 能够说出 Servlet 生命周期
- [ ] 掌握 ServletConfig 对象的作用
- [ ] 能够理解 HTTP 协议请求内容
- [ ] 能够使用 Request 对象获取 HTTP 协议请求内容
- [ ] 能够处理 HTTP 请求参数的乱码问题
- [ ] 能够使用 Response 对象操作 HTTP 响应内容
- [ ] 能够处理响应中文乱码问题

第一章 Servlet

1.1 Servlet 概述

1.1.1 Servlet 是什么

Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，具有独立于平台和协议的特性，主要功能在于交互式地浏览和生成数据，生成动态Web内容。

狭义的Servlet是指Java语言实现的一个接口，广义的Servlet是指任何实现了这个Servlet接口的类，一般情况下，人们将Servlet理解为后者。Servlet运行于支持Java的应用服务器中。从原理上讲，Servlet可以响应任何类型的请求，但绝大多数情况下Servlet只用来扩展基于HTTP协议的Web服务器。

最早支持Servlet标准的是JavaSoft的Java Web Server，此后，一些其它的基于Java的Web服务器开始支持标准的Servlet。

总结：**Servlet是运行在Web服务器(如Tomcat服务器)的，使用Java编写的小应用程序。**

1.1.2 Servlet 的作用

1. 开发动态资源：Servlet 也是服务器的一种**资源**，可以供外界(浏览器)去访问。
2. 接收浏览器请求并响应数据给浏览器.

1.1.3 Servlet 的开发步骤

1. 创建一个类实现 javax.servlet.Servlet 接口
2. 实现接口中的所有方法
3. 在 service 方法处理请求和响应数据
4. 配置 Servlet 的访问地址(供外界:浏览器访问)

- 可以通过 web.xml 文件配置
- 可以通过注解配置

1.2 Servlet的第一个程序

1.2.1 Servlet 2.5 开发方式

使用 web.xml 文件配置

1.2.1.1 实现步骤

1. 创建一个 HelloServlet 实现 Servlet 接口
2. 重写接口中的所有方法，在 service 方法中输出内容到控制台
3. 配置 web/WEB-INF/web.xml 文件

1.2.1.2 实现演示

1. 创建web工程

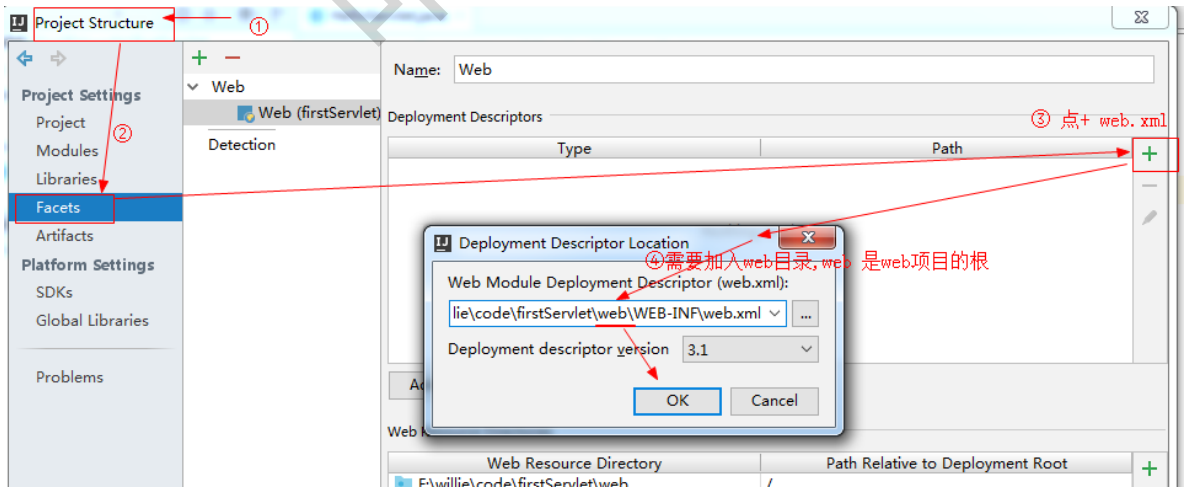
   File -> new Project-> Java Enterprise -> Web Application -> Finish

2. 创建好的Web项目结构：idea 生成或者 tomcat 案例中拷贝 web.xml 文件

标准web项目结构

```
webapp/web          web 项目根目录
    | 静态资源      -- html/js/css
    | WEB-INF      -- 存放的资源不能给外部 (浏览器) 直接访问.
        | lib      -- 存放 web 项目依赖的第三方jar包
        | classes  -- 存放 web 项目的字节码文件
        | web.xml  -- 当前项目的配置文件
```

使用工具生成 web.xml



3. 在 src 创建包，在包下创建 HelloServlet 实现 Servlet 接口重写所有方法，在 service 方法中往控制台输出：Hello Servlet!

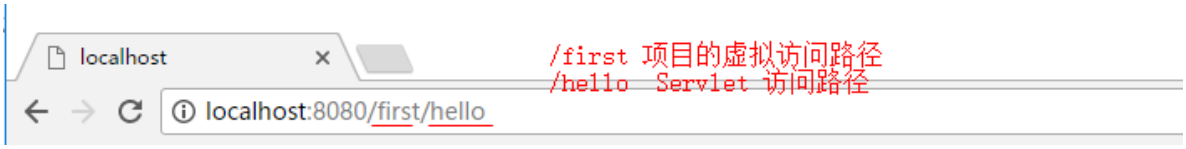
```
package cn.wolfcode._01_servlet;

import javax.servlet.*;
import java.io.IOException;
```

```java
/**
 * 创建一个类实现Servlet接口
 */
public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    // 在该方法中往控制台输出：Hello Servlet!
    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("Hello Servlet!");
    }

    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {

    }
}
```

4. 编辑 web.xml 中配置 servlet，设置访问地址为 /hello

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
           version="2.5">
    <!-- servlet 信息配置 -->
    <servlet>
        <!--servlet 名字-->
        <servlet-name>HelloServlet</servlet-name>
        <!-- servlet类全限定名-->
        <servlet-class>cn.wolfcode._01_servlet.HelloServlet</servlet-class>
    </servlet>
    <!--servlet 访问地址配置 -->
    <servlet-mapping>
        <!--servlet 名字：与上面的名字相同 -->
        <servlet-name>HelloServlet</servlet-name>
        <!--  浏览器访问地址，必须以/开头 -->
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

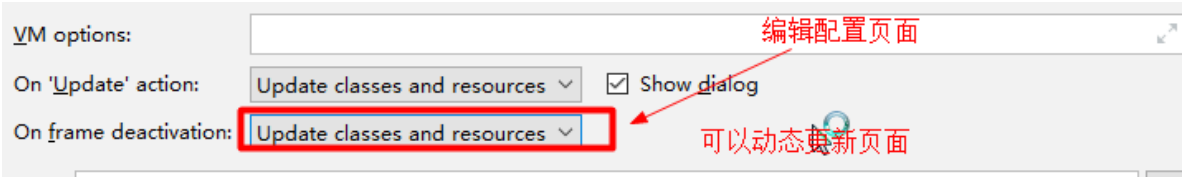
5. 部署项目并通过浏览器访问 Servlet



- 控制台结果：

```
Hello Servlet!
23-Apr-2020 11:37:41.299 INFO [localhost-startStop-1] org.a
23-Apr-2020 11:37:41.340 INFO [localhost-startStop-1] org.a
```

热部署的配置: 可以不用重启服务器直接访问改过的页面.



1.2.2 Servlet 3.0开发方法-注解方式配置

1.2.2.1 @WebServlet 注解

Servlet3.0 之后新增了一些注解，简化的 javaweb 代码开发，可以省略 web.xml配置文件

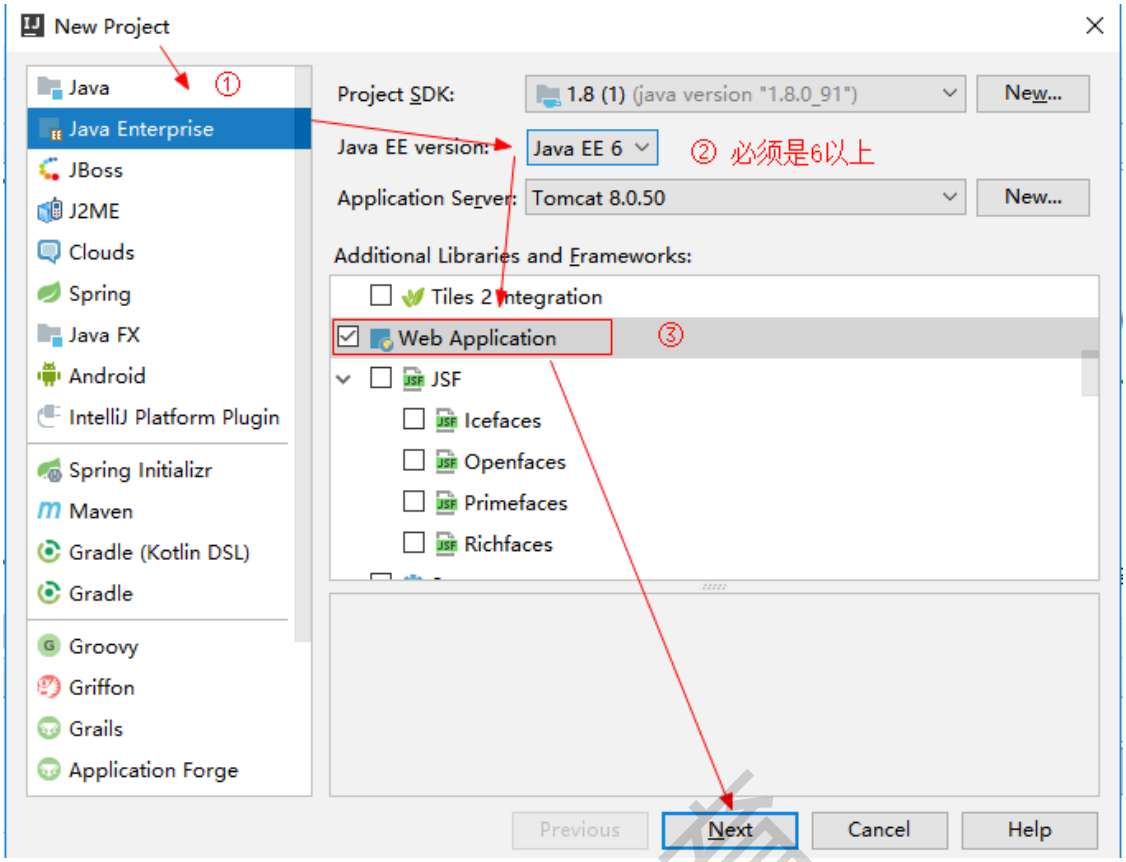
@WebServlet注解属性	说明
name = "HelloServlet"	Servlet名字，等价 `<servlet-name>HelloServlet</servlet-name>`
urlPatterns = "/hello"	访问地址，等价 `<url-pattern>/hello</url-pattern>`
value="/hello"	如果其它属性都不写，可以省略 value，只写访问地址即可。 不能与urlPatterns同时指定

1.2.2.2 注解开发 Servlet 步骤

1. 使用注解，必须是 Servlet3.0 之后才可以，Tomcat8 已经实现了3.0规范
2. 使用注解，必须是 JavaEE 6.0 版本以上
3. 创建类实现 Servlet 接口，在类上使用 @WebServlet 注解中添加 urlPatterns="/hello"或 value="/hello"，作为请求路径

1.2.2.3 注解开发Servlet演示

1. 创建 JavaEE6 模块(版本必须大于 6 才能使用注解)



2. 创建好的 Web 目录结构

3. 在 src 创建包，在包下创建 HttpServlet 实现 Servlet 接口重写所有方法，在 service 方法中往控制台输出：Hello Servlet! **使用注解 @WebServlet 配置访问地址**

```
package cn.wolfcode._01servlet;

import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet(name = "HelloServlet",urlPatterns = "/hello")
public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    // 在该方法中往控制台输出：Hello Servlet!
    @Override
```

```
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("Hello Servlet!");
    }

    @Override
    public String getServletInfo() {
        return null;
    }

    @Override
    public void destroy() {

    }
}
```

4. 部署项目，浏览器地址栏输入：http://localhost:8080/second/hello

- 控制台执行结果：

1.3 Servlet的生命周期

生命周期： 从出生到死亡以及中间所经历的过程则称为一个生命周期

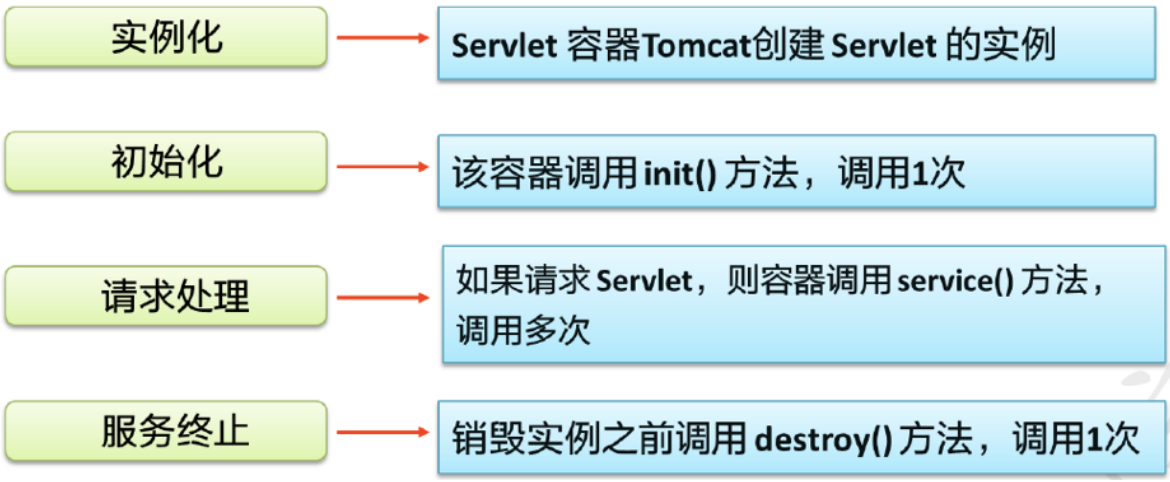
类的生命周期： 创建对象--> 运行操作-->销毁操作

1.3.1 思考三个问题

- [ ] Servlet 对象是谁负责创建的？
- [ ] Servlet 对象是什么时候创建的？
- [ ] Servlet 对象是什么时候销毁的？

1.3.2 Servlet的生命周期

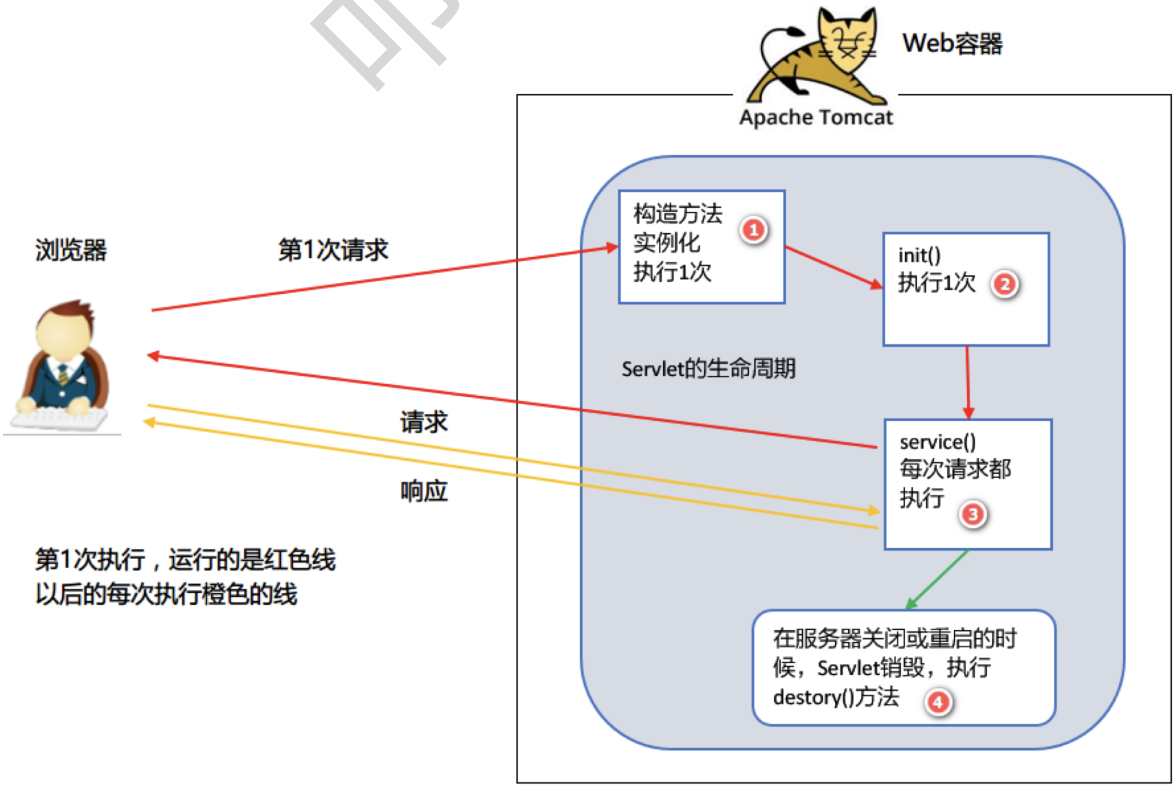
Servlet 对象的生命周期：Servlet 创建对象-->初始化操作--> 运行操作-->销毁操作
Web 服务器管理了 Servlet 的生命周期，Servlet 对象整个过程都是 Web 服务器来管理的。



1.3.3 Servlet 接口中生命周期方法

生命周期方法	作用	运行次数
构造方法	在对象实例化的时候执行 必须有公共的无参数构造方法	1次
void init(ServletConfig config)	在初始化的时候执行	1次
void service(ServletRequest req, ServletResponse res)	每次请求都会执行	n次
void destroy()	在服务器正常关闭的时候	1次

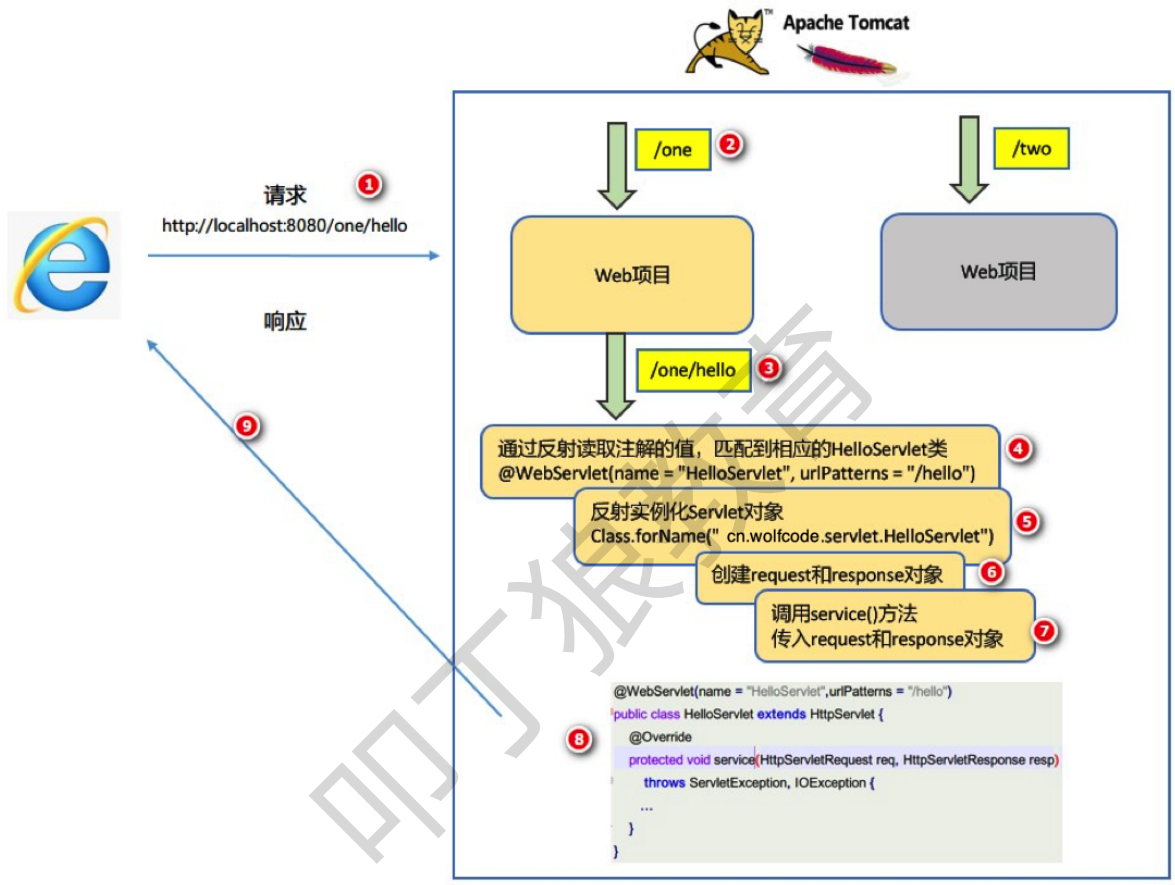
- 生命周期方法执行顺序图



1.3.4 总结回顾

- [ ] Servlet 对象是谁负责创建的？答：Web服务器(Tomcat)负责创建
- [ ] Servlet 对象是什么时候创建的？答：用户第1次访问的时候
- [ ] Servlet 对象是什么时候销毁的？答：服务器正常关闭或重启时销毁
- [ ] Servlet 是单例还是多例的？答：Servlet只在第一次访问时创建了对象，所以是单例的

1.4 Servlet 的请求流程



1. 浏览器发送请求，Tomcat接收到请求并通过解析请求地址获取到要访问的项目和资源路径
   项目访问路径：/one
   资源路径： /hello
2. Tomcat服务器内部会扫描one项目下的所有Servlet：获得每一个Servlet的访问地址并存储到集合中：

```
Map<String,String> map = new HashMap<>();
map.put("/hello","cn.wolfcode.servlet.HelloServlet");
```

3. 将 资源路径 /hello 作为键从 map 集合中获得值：类全限定名
4. 需要判断是否是第一次访问：

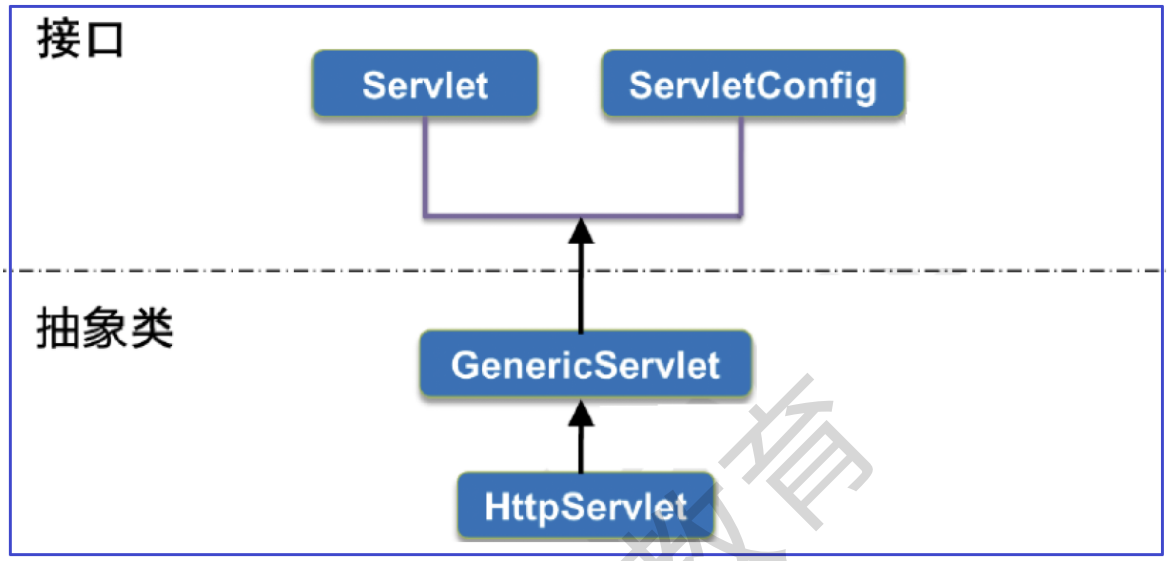
```
Servlet实例缓存池: Map<String, Servlet> map;
if(map.get("全限定名") == null){
    // 第一次访问,执行第5步
} else{
    // 第N次,直接执行第7步
}
```

5. 通过反射实例化这个Servlet对象,存入实例缓存池中.
6. Tomcat 创建 ServletConfig 对象然后调用 init 方法
7. 创建 request 和 response 对象

8. 调用 service 方法，将 request和 response 对象传递进来,在 service 方法中通过 response对象返回输出到浏览器，在浏览器上显示出来。
9. 等待下一次的访问

1.5 Servlet的继承体系

1.5.1 Servlet的继承结构



1.5.2 GenericServlet类

1.5.2.1 GenericServlet类概述

默认实现了 Servlet 和 ServletConfig 这两个接口，它的子类是 HttpServlet，如果我们写的Servlet使用的是Http协议，建议继承于HttpServlet。现在所有的浏览器都是使用http协议，所以我们以后都是继承于HttpServelt类就可以了。

1.5.2.2 继承 GenericServlet 开发 Servlet

```
package cn.wolfcode._02genericServlet;

import javax.servlet.GenericServlet;
```

```
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import java.io.IOException;

@WebServlet(urlPatterns = "/demo")
public class DemoServlet extends GenericServlet {
    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("我是GenericServlet子类");
    }
}
```

- 浏览器访问运行效果

Output

```
我是GenericServlet子类
```

1.5.3 HttpServlet类

1.5.3.1 HttpServlet类概述

1.5.3.2 继承 HttpServlet 开发Servlet

```
package cn.wolfcode._03httpservlet;

import javax.servlet.ServletException;
```

```
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(urlPatterns = "/demo02")
public class DemoServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("我是HttpServlet的子类");
    }
}
```

- 浏览器访问运行效果

Output

```
我是HttpServlet的子类
```

1.5.3.3 service方法源码分析

```
public abstract class HttpServlet extends GenericServlet {
    //定义了一些常量
    private static final String METHOD_GET = "GET";
    private static final String METHOD_POST = "POST";

    public HttpServlet() {
    }

     // 自己的service方法
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
          // 得到请求的方法：返回GET或POST
        String method = req.getMethod();
            // 如果是GET请求，调用doGet
        if (method.equals("GET")) {
              this.doGet(req, resp);
                    // 如果是POST请求，调用POST方法
        } else if (method.equals("POST")) {
            this.doPost(req, resp);
        } else {
                    // 如果所有的都不匹配，发送一个错误
            resp.sendError(501, errMsg);    // 发送一个错误信息
        }
    }

      // Servlet接口中定义的方法
    public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
        HttpServletRequest request;    // 请求子接口
        HttpServletResponse response;    // 响应子接口
        try {
            request = (HttpServletRequest)req;
```

```
            response = (HttpServletResponse)res;
        } catch (ClassCastException var6) {
            throw new ServletException("non-HTTP request or response");
        }
                // 调用重置的service方法
        this.service(request, response);
    }
}
```

1.5.4 小结

1. GenericServlet 是一个通用的 Servlet，可以用来处理各种协议发出的请求。
2. HttpServlet 是专门用来处理 HTTP 协议发送的请求，现在所有的浏览器发请求都是使用 HTTP 协议，因此以后我们开发 Servlet 只需要继承 HttpServlet 即可，可以按照如下步骤开发：

```
1. 创建类继承 HttpServlet
2. 重写参数带着 Http 开头的 service 方法：在该方法中处理请求并响应数据。
     注意事项：不要在该方法中调用父类的 service 方法
```

1.6 ServletConfig 对象

1.6.1 作用

- 用来封装 Servlet 初始化的时候的一些配置信息
- 为什么要配置初始化参数？
    - 解决硬编码的缺点：因为是写在源代码中，一旦写死，很难修改。维护不太方便。建议把一些与Servlet有关的配置信息，写在web.xml中，后期可以方便修改和维护

1.6.2 常用方法

ServletConfig接口常用方法	说明
String getInitParameter("参数名")	通过指定的参数名得到参数

1.6.3 使用演示

- 需求：根据不同的编码格式响应数据给浏览器，编码格式作为Servlet的初始化配置参数，在Servlet的service方法中读取初始化配置参数并输出
- ServletConfigServlet代码

```
public class ServletConfigServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 获得配置对象
        ServletConfig config = this.getServletConfig();
        // 获得初始化参数值
        String encoding = config.getInitParameter("encoding");
        if ("utf-8".equalsIgnoreCase(encoding)){
            // 模拟响应数据给浏览器
            System.out.println("执行utf-8编码操作");
```

```
        } else {
            // 模拟响应数据给浏览器
            System.out.println("执行gbk编码操作");
        }
    }
}
```

- 配置ServletConfigServlet初始化参数

```
<servlet>
    <!--servlet 名字-->
    <servlet-name>ServletConfigServlet</servlet-name>
    <!-- servlet类全限定名-->
    <servlet-class>cn.wolfcode._02servletConfig.ServletConfigServlet</servlet-class>
    <!--配置初始化参数-->
    <init-param>
        <!--参数名-->
        <param-name>encoding</param-name>
        <!--参数值-->
        <param-value>utf-8</param-value>
    </init-param>
</servlet>
<!--servlet 访问地址配置 -->
<servlet-mapping>
    <!--servlet 名字：与上面的名字相同 -->
    <servlet-name>ServletConfigServlet</servlet-name>
    <!--  浏览器访问地址，必须以/开头 -->
    <url-pattern>/config</url-pattern>
</servlet-mapping>
```

- 部署项目并浏览器访问地址如下：

Output

执行utf-8编码操作

1.7 loadOnStartup属性

1.7.1 思考问题

问：如果一个servlet在创建对象或者是初始化的时候需要执行耗时操作：比如加载配置文件并封装信息到对象上，这样就必然导致相第一个访问的用户要等待比较长的时间，用户体验非常差。如何解决这个问题呢？

1.7.2 作用

- 让 web 容器启动的时候创建并初始化 Servlet。

1.7.3 用法

- web.xml配置用法

```
<servlet>
    <!--servlet 名字-->
    <servlet-name>HelloServlet</servlet-name>
    <!-- servlet类全限定名-->
    <servlet-class>cn.wolfcode._01servlet入门.HelloServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

- 注解配置用法

```
@WebServlet(urlPatterns = "/response",loadOnStartup = 1)
public class ResponseServlet extends HttpServlet {
```

说明：取值范围1到10，值越小越先加载。默认值是-1：代表第1次访问时创建和初始化

1.7.4 使用场景

在后续学习的框架中，有些核心的Servlet需要在服务器启动时创建并做初始化操作。比如springMVC 的核心控制器，如下配置：

```
<servlet>
    <servlet-name>dispatchServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

    <!--配置初始参数-->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:springMVC.xml</param-value>
    </init-param>
    <!--启动时候创建servlet(数字越小越先加载-->
    <load-on-startup>1</load-on-startup>
</servlet>
```

小结

1. 理解 Servlet 作用,可以帮咱们完成什么功能
2. 掌握 第一个helloword 程序的开发(2.5 的 web.xml 配置方式,3.0 的注解使用方式)
3. 理解并掌握 Servlet 的生命周期
    1. 首次访问Servlet 时 创建对象并调用init 方法,然后调用 service 方法
    2. 非首次访问,直接调用 service 方法
    3. 正常关闭服务器时调用销毁方法destroy方法
4. 掌握 Servlet 的请求流程 (能独立画流程图或用文字的形式写清请求流程) 浏览器 -- 服务器Servlet 的过程
5. 掌握使用 ServletConfig 对象来解决硬编码问题即可,(会配置web.xml ,能Servlet中获取web.xml 中的配置信息即可)
6. 理解 loadOnStartup 配置的作用即可(配置下,看下效果即可)

1.8 Servlet映射细节

1.8.1 配置多个路径

1.8.1.1 配置方式1

- 一个 `<servlet-mapping>` 中写多个 `<url-pattern>`

```
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/demo1.html</url-pattern>
    <url-pattern>/demo2.html</url-pattern>
</servlet-mapping>
```

1.8.1.2 配置方式2

- 一个`<servlet>`对应多个`<servlet-mapping>`

```
<servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/demo1.html</url-pattern>
</servlet-mapping>
<servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/demo2.html</url-pattern>
</servlet-mapping>
```

注意： `<url-pattern>` 中的内容必须唯一，必须以 / 开头。

1.8.2 通配符映射 *

通配符格式	说明
`/*` `/目录名/*`	/* ：匹配所有的访问地址，必须以 / 开头 /admin/*：匹配 admin 目录下的所有地址
`*.扩展名`	匹配某个扩展名结尾的访问地址。如：*.action *.do

> 注意：不能同时 `/*.扩展名` 的访问路径：会导致整个web项目加载失败，所有的web资源都不能访问。项目启动会报如下错误：

```
Caused by: java.lang.IllegalArgumentException: Invalid <url-pattern>
/admin/*.action in servlet mapping
```


1.8.3 映射注意事项

- 映射路径必须保证唯一性，除通配符结合扩展名使用，否则必须以 / 开头
- servlet-name 不能叫做 default 和 jsp
  - Tomcat内部已有两个Servlet，一个叫default，一个叫 jsp。
  - DefaultServlet 是用来处理静态资源
  - JSPServlet 是用来将 JSP 文件翻译成 Java 文件
  - 如果我们的名字和内部的重名了，则我们的 servlet 会覆盖内部的 servlet。在 tomcat 的配置文件中 conf/web.xml, 该文件是当前服务器所有的项目都可以公用的信息.

1.8.4 面试题

- 创建 2 个Servlet，一个 Servlet1，一个 Servlet2，在下列情况下，访问哪个Servlet

请求URL	Servlet1	Servlet2	访问哪个
/abc/a.html	/abc/*	/*	Servlet1
/abc	/abc/*	/abc	Servlet2
/abc/a.do	/abc/*	*.do	Servlet1
/a.do	/*	*.do	Servlet1
/xxx/yyy/a.do	/*	*.do	Servlet1

- 结论
  - 优先级：/开头的优先级大于扩展名结尾
  - 匹配原则：精确匹配的原则，哪个更匹配就使用哪个

第二章 请求对象

2.1 HttpServletRequest 对象概述

HttpServletRequest 是一个接口，该接口的实现类对象称为请求对象，请求对象**封装了所有的请求信息(请求行，请求头，请求体(请求参数))。**

HttpServletRequest 接口包含了大量的方法。由 Tomcat 去实现这个对象，并且在 servlet 启动的时候调用service() 将请求对象传递进来了。我们在 service 方法中直接使用即可。

2.2 HttpServletRequest 对象常用方法

request 与请求行相关方法	功能描述
String getMethod()	获得请求方式 GET 或 POST
String getRequestURI()	Uniform Resource Identifier统一资源标识符，代表一个资源名字
StringBuffer getRequestURL()	Uniform Resource Locator 统一资源定位符，代表一个可以访问地址
String getProtocol()	获得协议和版本
String getContextPath()	获得上下文路径(项目名path)

request 与请求头相关方法	功能描述
String getHeader(String headName)	得到指定的请求头的值 参数：键的名字 返回：相应请求头的值

request与请求参数相关方法	功能描述
String getParameter(String name)	通过参数名得到一项参数值
String[] getParameterValues(String name)	根据参数名得到一组同名的值 复选框，下拉列表多选
Enumeration getParameterNames()	获得所有的参数名
Map getParameterMap()	得到表单所有的参数键和值，封装成Map对象

Enumeration接口中方法	说明
boolean hasMoreElements()	如果还有其它元素，返回true
E nextElement()	返回下一个元素

2.3 请求对象获取请求行和请求头演示

```
package cn.wolfcode._04request;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/line")
public class RequestLineServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 获得请求行信息
        System.out.println("得到方法：" + request.getMethod());
        System.out.println("统一资源标识符：" + request.getRequestURI());
        System.out.println("统一资源定位符：" + request.getRequestURL());
        System.out.println("协议和版本：" + request.getProtocol());
        System.out.println("当前项目地址：" + request.getContextPath());

        // 得到一个请求头
        System.out.println("得到host的请求头值：" + request.getHeader("host"));
    }
}
```

- 浏览器访问控制台输出效果如下：

Output

```
[2020-04-23 02:07:41,417] Artifact SecondServlet:war expl
[2020-04-23 02:07:41,417] Artifact SecondServlet:war expl
得到方法：GET
统一资源标识符：/second/line
统一资源定位符：http://localhost:8080/second/line
协议和版本：HTTP/1.1
当前项目地址：/second
得到host的请求头值：localhost:8080
```

2.4 请求对象获取请求参数演示

2.4.1 准备注册表单数据

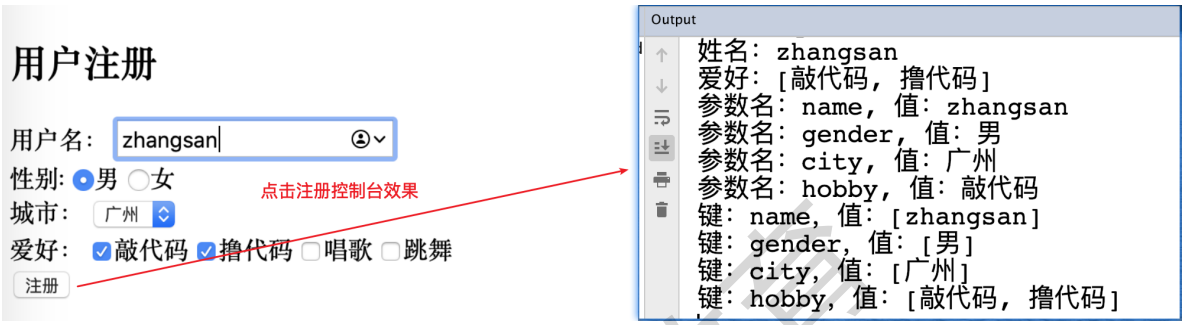
```
<body>
  <h2>用户注册</h2>
  <form action="register" method="post">
  用户名：  <input type="text" name="name"><br/>
  性别：<input type="radio" name="gender" value="男" checked="checked"/>男
  <input type="radio" name="gender" value="女"/>女 <br/>
  城市：
  <select name="city">
    <option value="广州">广州</option>
    <option value="深圳">深圳</option>
    <option value="东莞">东莞</option>
  </select>
  <br/>
  爱好：
  <input type="checkbox" name="hobby" value="敲代码"/>敲代码
  <input type="checkbox" name="hobby" value="撸代码"/>撸代码
  <input type="checkbox" name="hobby" value="唱歌"/>唱歌
  <input type="checkbox" name="hobby" value="跳舞"/>跳舞
  <br/>
  <input type="submit" value="注册"/>
 </form>
</body>
```

2.4.2 处理注册请求获取请求参数

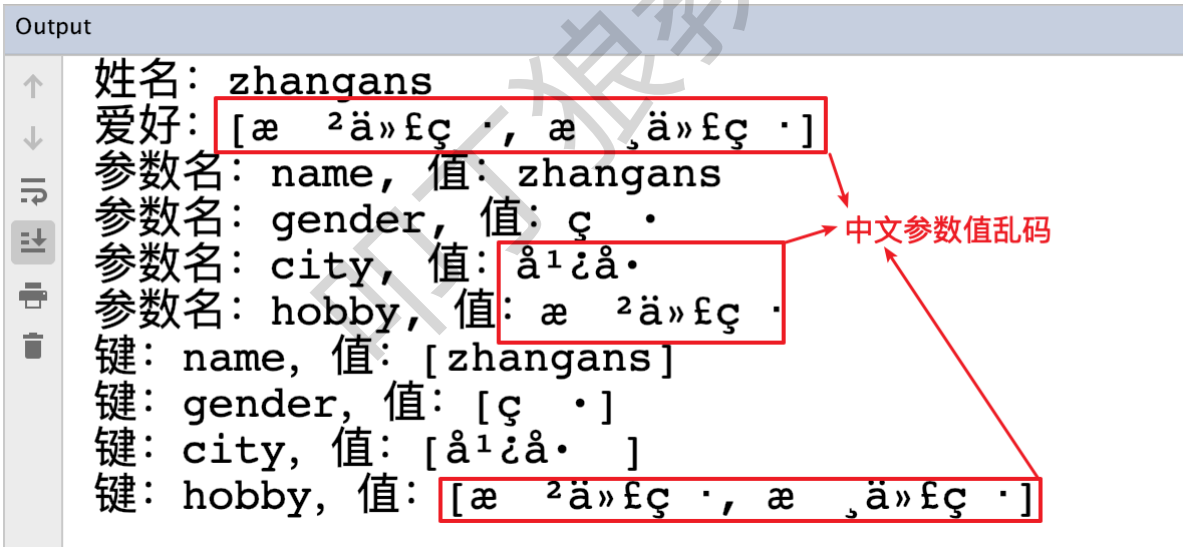
```
@WebServlet(urlPatterns = "/register")
public class RequestParamServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 设置请求参数编码
        request.setCharacterEncoding("utf-8");
        // 根据参数名获得参数值
        System.out.println("姓名：" + request.getParameter("name"));
        // 得到一个字符串数组
        String[] hobbies = request.getParameterValues("hobby");
        // 得到多个参数
        System.out.println("爱好：" + Arrays.toString(hobbies));
        // 得到所有参数的名字
```

```
        Enumeration<String> parameterNames = request.getParameterNames();
        while (parameterNames.hasMoreElements()) {
            // 得到其中一个名字
            String name = parameterNames.nextElement();
            System.out.println("参数名：" + name + "，值：" +
request.getParameter(name));
        }
        // 得到所有的键和值
        Map<String, String[]> map = request.getParameterMap();
        map.forEach((k,v) -> System.out.println("键：" + k + "，值：" +
Arrays.toString(v)));
    }
}
```

- 浏览器访问注册界面：



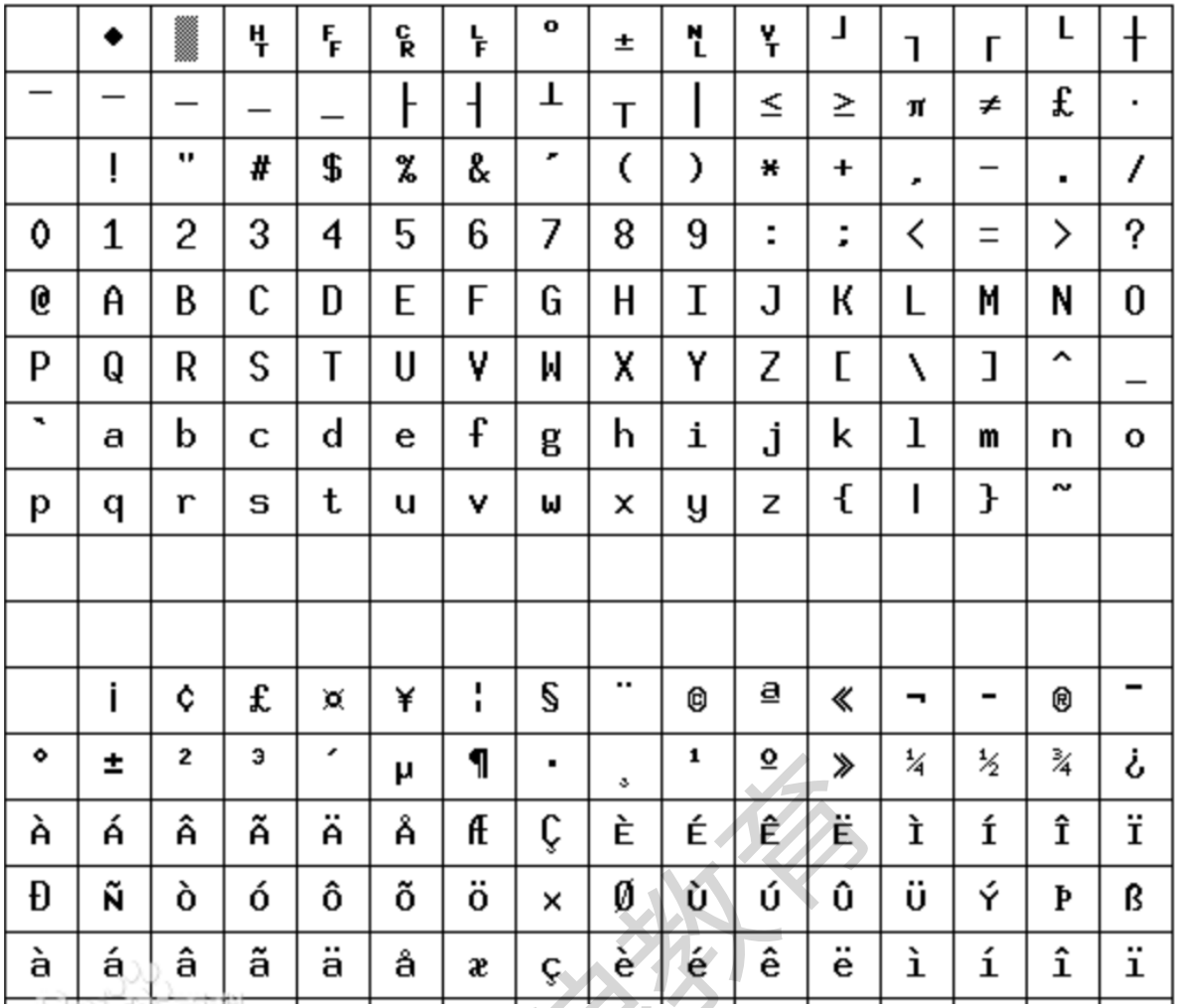
- 如果把 **request.setCharacterEncoding("utf-8");** 该行代码注释到则会出现参数乱码如下：



2.5 请求参数乱码问题

2.5.1 请求参数产生乱码的原因

在浏览器发送数据给服务器的时候，使用 utf-8 编码，但服务器解码默认使用 ISO-8859-1 解码：欧洲码，不支持汉字的。



> 注意：如果是 GET 请求且 Tomcat 版本大于8.0.5，则不需要考虑乱码问题，否则 GET 请求也需要考虑乱码问题。

2.5.2 POST方式乱码解决方案

1. 解决方法：request.setCharacterEncoding("utf-8") 设置请求参数的编码为UTF-8
2. 代码位置：设置请求的编码这句话一定放在获取请求参数之前
3. 页面的编码：这个编码一定要与页面的编码相同。如果页面使用的是 GBK，则这里也要用GBK

第三章 响应对象

3.1 HttpServletResponse对象概述

HttpServletResponse 是一个接口，该接口的实现类对象称为响应对象，**用于响应数据(响应行，响应头，响应体)给浏览器**。HttpServletResponse 接口包含了大量的方法。由 Tomcat 去实现这个对象，并且在 servlet 启动的时候调用 service() 将请求对象和响应对象传递进来了。我们在 service 方法中直接使用即可。

3.2 响应对象响应数据给浏览器

3.2.1 响应数据相关方法

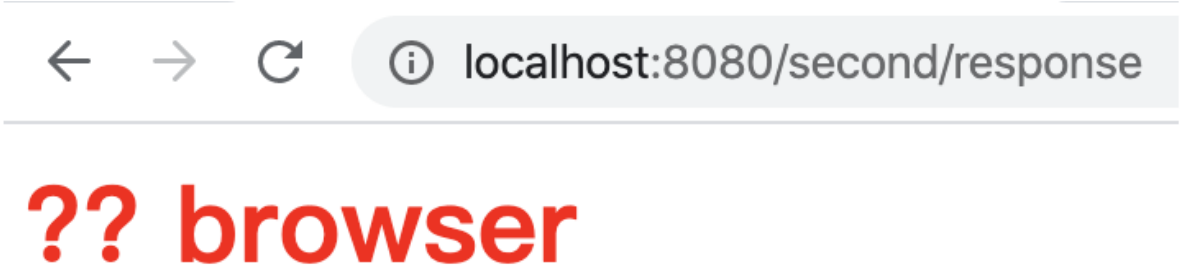
响应体相关的方法	功能描述
OutputStream getOutputStream()	如果服务器端返回的是二进制数据 则使用这个方法，比如图片
PrintWriter getWriter()	如果服务器端返回的是字符的文本数据，使用这个方法

3.2.2 响应数据代码演示

```
@WebServlet(urlPatterns = "/response")
public class ResponseServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 获得字符打印流对象
        PrintWriter out = response.getWriter();
        // 响应数据给浏览器显示
        out.print("<h1 style='color:red'>hello browser<h1>");
    }
}
```

- 浏览器访问效果如下：

- 如果将输出内容改为中文：比如**out.print("你好 browser");**则浏览器效果如下：



- `你好` 已经乱码了。

3.2.3 响应内容中文乱码问题

3.2.3.1 乱码原因

因为 Tomcat 中响应体默认的是欧洲码表，ISO-8859-1 不支持中文。

3.2.3.2 解决方法

1. **在获得打印流对象之前，通过下面方法设置打印流的编码为utf-8**

response方法	说明
response.setCharacterEncoding("字符集")	用于设置响应体的字符集 设置打印流使用的码表

```
@WebServlet(urlPatterns = "/response")
public class ResponseServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 设置打印流编码
        response.setCharacterEncoding("utf-8");
        // 获得字符打印流对象
        PrintWriter out = response.getWriter();
        // 响应数据给浏览器显示
        out.print("<h1 style='color:red'>你好 browser<h1>");
    }
}
```

- 重新部署之后浏览器再次访问效果：

- 问题分析：浏览器还是显示乱码，但比刚刚好一点了，刚刚直接显示的问号，现在至少还可以看出来是中文了，O(∩_∩)O哈哈~，为什么还乱码呢？原因是：浏览器并不知道服务器返回的数据是使用UTF-8编码的，它默认使用了另一种码表进行解码，导致编码和解码的码表不一致。所以还是乱码。那怎么解决？很简单，只需要告诉浏览器返回的数据是使用什么码表编码的，让它使用对应的码表进行解码即可，这样就保证前后码表一致了。

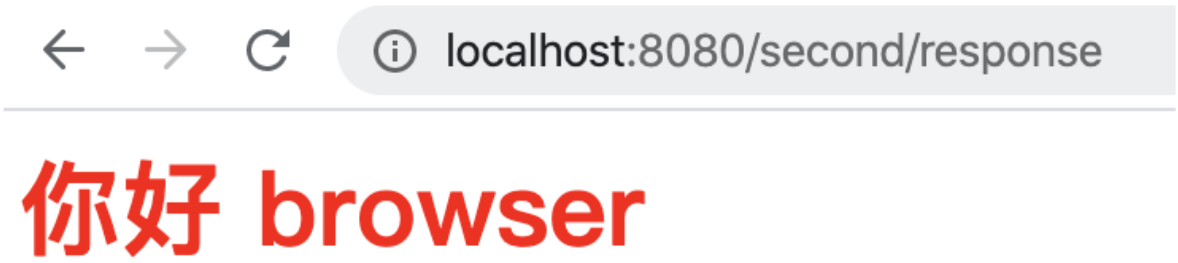
2. **通过下面方法告诉浏览器返回数据类型和编码**

响应对象的方法	功能描述
void setContentType(String type)	1. 告诉浏览器返回内容类型 2. 设置打印流编码 注意: 必须在获取流之前设置,否则无效

```
@WebServlet(urlPatterns = "/response")
public class ResponseServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 设置打印流编码
        // response.setCharacterEncoding("utf-8");
```

```
        // 告诉浏览器返回内容类型并设置打印流编码
        response.setContentType("text/html;charset=utf-8");
        // 获得字符打印流对象
        PrintWriter out = response.getWriter();
        // 响应数据给浏览器显示
        out.print("<h1 style='color:red'>你好 browser<h1>");
    }
}
```

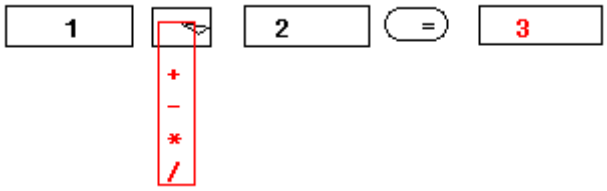
- 重新部署之后浏览器再次访问效果：



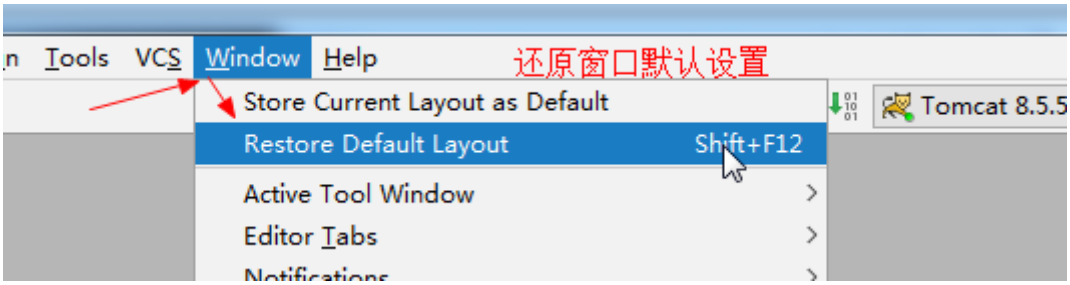
你好 browser

小结

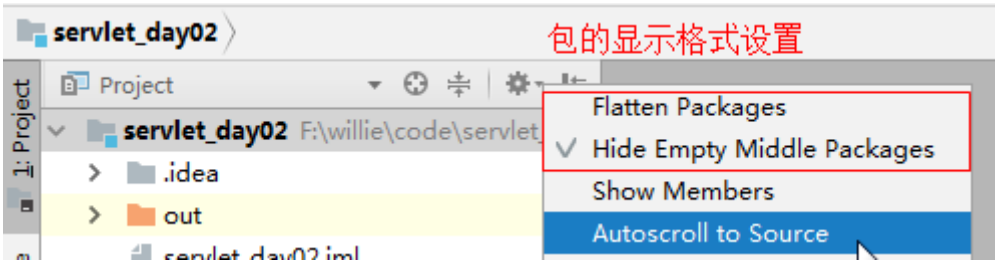
1. 知道 Servlet 可以配置多个访问地址即可(写代码演示)
2. 掌握 * 通配符的作用(写代码去演示* 配置的各种方式)
3. 掌握映射的注意事项 (是否需要使用 / 开头的问题,Servlet name 不能写default 和jsp 的问题)
4. 理解面试题
5. 重点掌握请求对象
    1. 请求对象的作用
    2. 请求对象常用的API (获取请求参数的API)
    3. 解决 请求中参数数据中文乱码的问题
6. 重点掌握响应对象(能够书写响应html 给浏览器代码)
    1. 响应对象的作用
    2. 响应对象的常用 API
    3. 掌握响应数据中文乱码问题解决
    4. 拓展: 在 Servlet 中完成一个简易在线计算器,功能如下图



还原窗口默认设置:



包的显示格式设置:



解决 Tomcat 启动控制台中文乱码问题: