servlet&servletContext

学习目标

1.能够使用idea编写servlet

2.能够使用idea配置tomcat方式发布项目

3.能够使用注解开发servlet

4.能够说出servlet生命周期方法执行流程

5.能够说出servlet运行原理

6.能够使用servletcontext域对象

7.能够完成案例统计访问次数

第1章 Servlet入门

1.1 Servlet2.5实现Hello world例子

1.1.1 servlet的基本概述

Servlet 运行在服务端的Java小程序，是sun公司提供一套规范，用来处理客户端请求、响应给浏览器的动态资源。但servlet的实质就是java代码，通过java的API动态的向客户端输出内容

1. **查阅JavaEE手册（帮助文档）阅读Servlet规范**：

javax.servlet

Interface Servlet

All Known Subinterfaces:
HttpJspPage, JspPage

All Known Implementing Classes:
FacesServlet, GenericServlet, HttpServlet

---

public interface Servlet

Implemented by: FacesServlet, GenericServlet, JspPage

定义所有 servlet 都必须实现的方法。

servlet 是运行在 Web 服务器中的小型 Java 程序。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。

要实现此接口，可以编写一个扩展 javax.servlet.GenericServlet 的一般 servlet，或者编写一个扩展 javax.servlet.http.HttpServlet 的 HTTP servlet。

此接口定义了初始化 servlet 的方法、为请求提供服务的方法和从服务器移除 servlet 的方法。这些方法称为生命周期方法，它们是按以下顺序调用的：

1. 构造 servlet，然后使用 init 方法将其初始化。
2. 处理来自客户端的对 service 方法的所有调用。
3. 从服务中取出 servlet，然后使用 destroy 方法销毁它，最后进行垃圾回收并终止它。

根据文档总结，书写servlet一个三个步骤：

1）创建一个class实现servlet接口

2）重写service方法

3）创建的类必须在web.xml文件中做配置

2. **为什么要做配置?**

   答：必须将请求路径和java程序的对应关系建立起来。

1.1.2 servlet与普通的java程序的区别

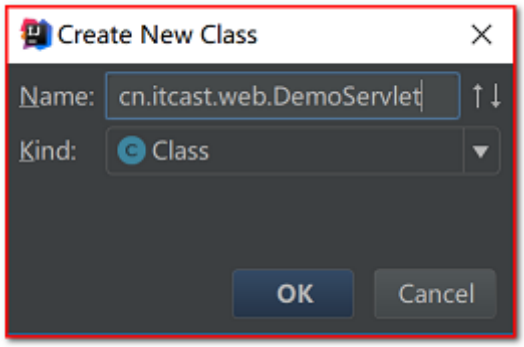
```
1.必须实现servlet接口
2.必须在servlet容器（服务器）中运行
3.servlet程序可以接收用户请求参数以及向浏览器输出数据
```

1.1.3 代码实现servlet的步骤

1. 在工程下创建cn.itcast.web包,在包下创建一个类实现 Servlet接口
2. 实现service方法
3. 在web.xml中配置书写好的servlet

1.1.4 servlet代码实现

在cn.itcast.web包下创建一个类实现Servlet接口



servlet代码：

```
package cn.itcast.web;

import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class DemoServlet implements Servlet{

  @Override
  public void init(ServletConfig servletConfig) throws ServletException {
  }

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

  @Override
  public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
    System.out.println("第一个servlet程序");
```

```
  }

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

  @Override
  public void destroy() {
  }
}
```

web.xml配置（该文件在web/WEB-INF 文件夹下）：

```
<?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>
        <servlet-name>demoServlet</servlet-name>
         <servlet-class>cn.itcast.web.DemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>demoServlet</servlet-name>
        <url-pattern>/demo</url-pattern>
    </servlet-mapping>
</web-app>
```

打开浏览器访问:

http://localhost:9090/day02/demo

就可以在idea的控制台看到输出了"第一个servlet程序"

2.2 Servlet3.0实现Hello world例子

2.2.1 Servlet2.5与Servlet3.0的区别

Servlet3.0相较于Servlet2.5：

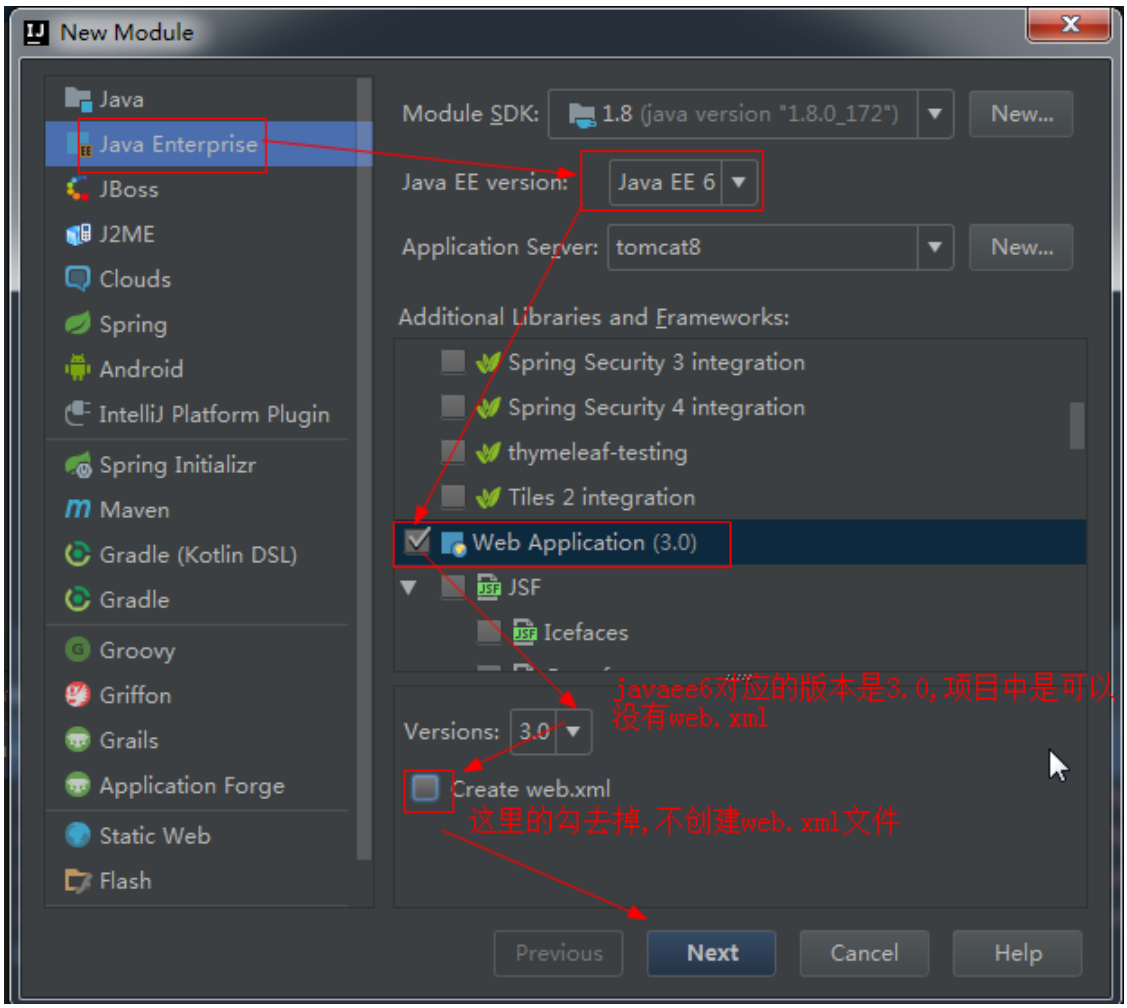
新增了一些注解，简化的javaweb代码开发，可以省略web.xml配置文件 支持异步处理（多线程技术） 支持可插性特性（书写的代码编译后生成的class文件可以直接部署到其他项目的，自动加载执行）

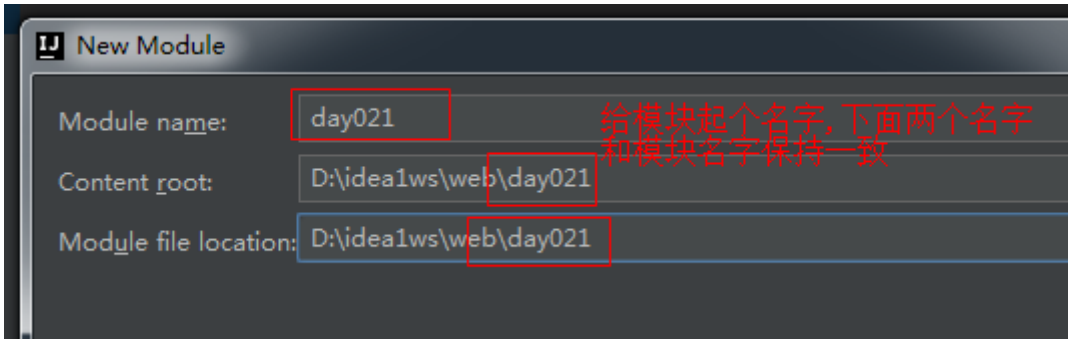
2.2.2 代码实现Servlet3.0步骤（注解配置servlet演示）

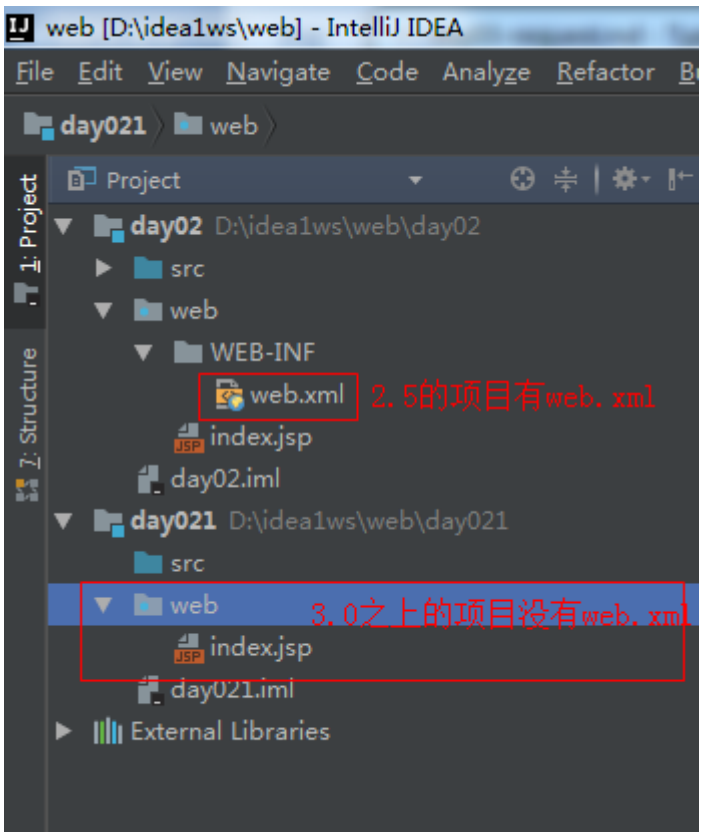
1. 创建JavaEE6(含6)以上的工程
2. 创建servlet，在@WebServlet注解中添加urlPatterns = "/hello"，作为请求路径

2.2.3 注解开发servlet代码演示

1. **创建JavaEE6(含6)以上的工程：**


New Module
Java
Java Enterprise
JBoss
J2ME
Clouds
Spring
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle (Kotlin DSL)
Gradle
Groovy
Griffon
Grails
Application Forge
Static Web
Flash
Module SDK: 1.8 (java version "1.8.0_172")
New...
Java EE version: Java EE 6
Application Server: tomcat8
New...
Additional Libraries and Frameworks:
Spring Security 3 integration
Spring Security 4 integration
thymeleaf-testing
Tiles 2 integration
Web Application (3.0)
JSF
Icefaces
javaee6对应的版本是3.0,项目中是可以没有web.xml
Versions: 3.0
Create web.xml
这里的勾去掉,不创建web.xml文件
Previous
Next
Cancel
Help


New Module
Module name: day021
Content root: D:\idea1ws\web\day021
Module file location: D:\idea1ws\web\day021
给模块起个名字,下面两个名字和模块名字保持一致



2. 注解开发servlet代码演示：

```
package cn.itcast.web;

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

//name = "HelloServlet"：servlet名称，相当于web.xml中的<servlet-name>
//urlPatterns = "/hello"：servlet的访问路径，相当于<url-pattern>
@WebServlet(name = "HelloServlet",urlPatterns = "/hello")
public class HelloServlet implements Servlet {
    @Override
    public void init(ServletConfig servletConfig) throws ServletException {

    }

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

    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("HelloServlet执行.......");
    }

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

    @Override
```

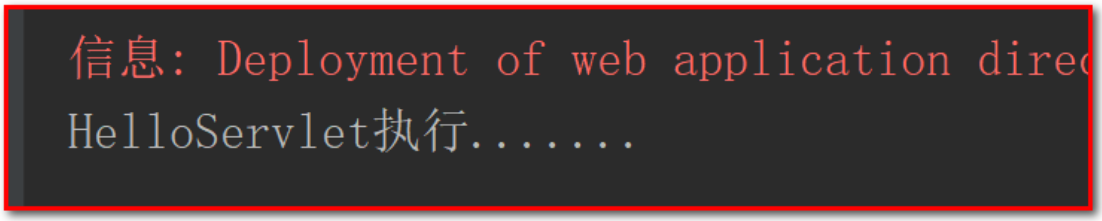
```
    public void destroy() {

    }
}
```

配置项目路径(配置过程参照1.9.2中第二步),启动tomcat服务器测试

浏览器地址栏输入：http://localhost:8080/servlet2/hello

测试成功：



第2章 Servlet生命周期

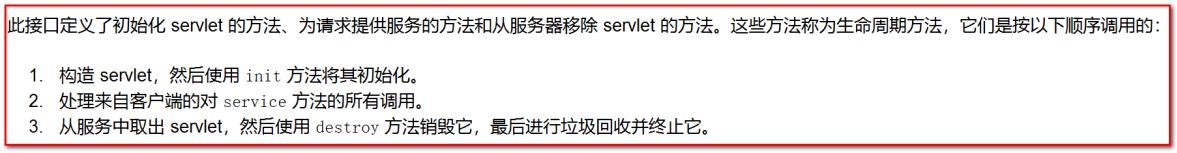
2.1 Servlet生命周期的概述

2.1.1 什么是生命周期

一个对象从创建到消亡的过程，就是生命周期。因此，对Servlet生命周期的学习，我们就是研究Servlet什么时候创建，什么时候销毁。

2.2 servlet生命周期相关的方法

首先我们来回顾servlet接口的文档内容，其中一部分如图所示：


此接口定义了初始化 servlet 的方法、为请求提供服务的方法和从服务器移除 servlet 的方法。这些方法称为生命周期方法，它们是按以下顺序调用的：

1. 构造 servlet，然后使用 `init` 方法将其初始化。
2. 处理来自客户端的对 `service` 方法的所有调用。
3. 从服务中取出 servlet，然后使用 `destroy` 方法销毁它，最后进行垃圾回收并终止它。

2.2.1 API 介绍

上图中，我们注意到两个点，servlet的创建和销毁由两个相关的方法init方法和destroy方法

```
void destroy() 销毁servlet的方法
```

```
void init(ServletConfig config)  初始化servlet的方法
```

我们可以调用测试一下这两个方法：

2.2.2 使用步骤

1. 创建LifeCircleServlet初始化
2. 复写init、service、destroy方法
3. 访问servlet测试初始化LifeCircleServlet
4. 关闭服务器测试销毁LifeCircleServlet

2.2.3 演示代码

```
package cn.itcast.web;

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

public class LifeCircleServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        super.init();
        System.out.println("LifeCircleServlet初始化。。。");
    }

    @Override
    public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException {
        System.out.println("LifeCircleServlet执行。。。");
    }

    @Override
    public void destroy() {
        super.destroy();
        System.out.println("LifeCircleServlet销毁。。。");
    }
}
```

web.xml的配置

```
<?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>
        <servlet-name>LifeCircleServlet</servlet-name>
         <servlet-class>cn.itcast.web.LifeCircleServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>LifeCircleServlet</servlet-name>
        <url-pattern>/life</url-pattern>
    </servlet-mapping>
</web-app>
```

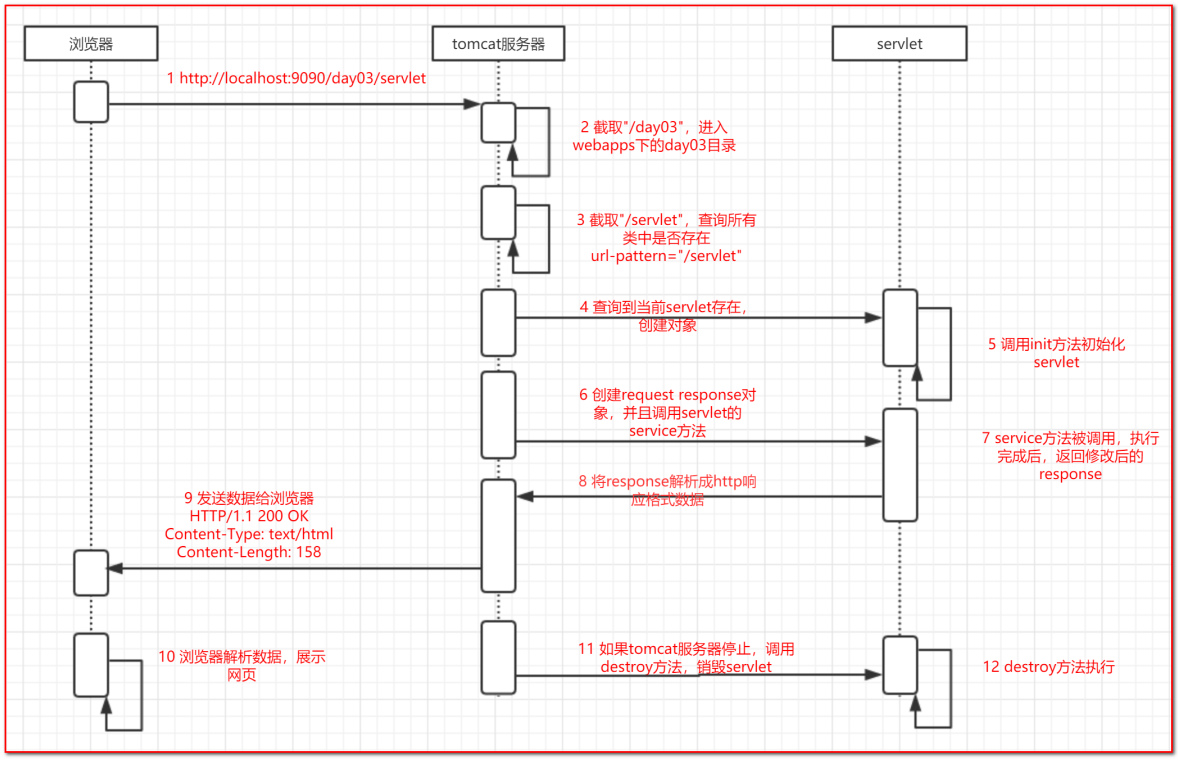
效果:

```
访问servlet：
LifeCircleServlet初始化。。。
LifeCircleServlet执行。。。

关闭tomcat服务器：
LifeCircleServlet销毁。。。
```

2.3 servlet生命周期流程

虽然简单使用过了servlet生命周期相关的方法，但是servlet从创建到销毁的过程对大家来说还是没说清楚，因此，我们以时序图的方式给大家展示了servlet的运行过程，注意图中每一步都由序号，按照序号查看每一个步骤。



附：servlet在初始化一次之后，就不再创建，因此如果多次访问同一个servlet的效果是这样的：

```
LifeCircleServlet初始化。。。
LifeCircleServlet执行。。。
LifeCircleServlet执行。。。
LifeCircleServlet执行。。。
```

因此servlet是一个单例线程不安全的对象。

第3章 Servlet的体系结构

3.1 Servlet的体系结构概述

目前我们已经学会创建一个类实现sevlet接口的方式开发Servlet程序，实现Servlet接口的时候，我们必须实现接口的所有方法。但是，在servlet中，真正执行程序逻辑的是service，对于servlet的初始化和销毁，由服务器调用执行，开发者本身不需要关心。因此，有没有一种更加简洁的方式来开发servlet程序呢？

我们先来查阅API回顾Servlet接口：

由上图可知在servlet接口规范下，官方推荐使用继承的方式，继承GenericServlet 或者HttpServlet来实现接口，那么我们接下来再去查看一下这两个类的API：

GenericServlet：

阅读上图API可知，GenericServlet 是一个类，它简化了servlet的开发，已经提供好了一些servlet接口所需的方法，我们开发者只需要重写service方法即可

我们来使用GenericServlet 创建servlet：

1. 创建一个类
2. 继承GenericServlet
3. 重写service方法

```
package cn.itcast.web;

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;

public class GenericDemoServlet extends GenericServlet {
    @Override
    public void service(ServletRequest servletRequest, ServletResponse
servletResponse) throws ServletException, IOException {
        System.out.println("GenericDemoServlet执行.......");
    }
}
```

web.xml的配置

```
<?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>
        <servlet-name>GenericDemoServlet</servlet-name>
         <servlet-class>cn.itcast.web.GenericDemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>GenericDemoServlet</servlet-name>
        <url-pattern>/generic</url-pattern>
    </servlet-mapping>
</web-app>
```

虽然，GenericServlet已经简化了servlet开发，但是我们平时开发程序需要按照一种互联网传输数据的协议来开发程序——http协议，因此，sun公司又专门提供了HttpServlet，来适配这种协议下的开发。

HttpServlet：

阅读上图的API可知，继承HttpServlet，我们需要重写doGet、doPost等方法中一个即可，根据Http不同的请求，我们需要实现相应的方法。

我们来使用HttpServlet创建servlet：

1. 创建一个类
2. 继承HttpServlet
3. 重写doGet方法

```
package cn.itcast.web;

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;


public class HttpDemoServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("HttpDemoServlet执行.......");
    }
}
```

web.xml配置

```
<?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>
        <servlet-name>HttpDemoServlet</servlet-name>
         <servlet-class>cn.itcast.web.HttpDemoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HttpDemoServlet</servlet-name>
        <url-pattern>/http</url-pattern>
    </servlet-mapping>
</web-app>
```

通过以上两个API阅读，同学们注意一个细节HttpServlet是GenericServlet的子类，它增强了GenericServlet一些功能，因此，在后期使用的时候，我们都是选择继承HttpServlet来开发servlet程序。

虽然目前开发servlet的选择继承类，已经确定，但是另一个问题一直还在我们头脑中，那就是我们学的这些浏览器、服务器、servlet等，到底是如何运行的?

那么接下来，我们来看看servlet的运行原理。

3.2 servlet运行原理

总结

通过上述流程图我们重点需要掌握如下几个点：

1. Servlet对象是由服务器创建
2. request与response对象也是由tomcat服务器创建
3. request对象封装了浏览器过来的所有请求信息，response对象代表了服务器的响应信息。

第4章 Servlet的配置

4.1 url-pattern配置

我们在创建servlet后，如果想要这个servlet可以被我们访问到，必须在web.xml文件中对其进行配置。在其中有一个这个标签是用于确定我们访问一个servlet的路径，接下来，我们详细介绍一下关于这个标签的配置： 它是用于确定我们访问一个servlet的路径. 问题:一个servlet是否可以被不同的路径映射?(多个不同配置是否可以映射同一个servlet) 答案是可以的 我们可以通过下面的示例来说明上面的问题

```
<servlet>
    <servlet-name>Demo1Servlet</servlet-name>
    <servlet-class>cn.itcast.servlet.Demo1Servlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Demo1Servlet</servlet-name>
    <url-pattern>/demo1</url-pattern>
</servlet-mapping>
    <servlet-mapping>
    <servlet-name>Demo1Servlet</servlet-name>
    <url-pattern>/hellodemo1</url-pattern>
</servlet-mapping>
```

上面是关于Demo1Servlet的配置，大家发现我们对于Demo1Servlet它有两个与其映射，那么这时当我们访问 http://localhost:8080/day08/demo1 http://localhost:8080/day08/hellodemo1 都可以访问到Demo1Servlet. 那么对于我们在值的写法到底应该怎样处理呢？ 对于我们在开发中它的写法有以下几种: 1 完全匹配 我们之前写的都是完全匹配方式 ， 它要求必须以”/”开始

2 目录匹配 必须以”/”开始，以”*”结束

3 扩展名匹配 不能以”/”开始，以*.xxx结束 xxx代表的是后缀名

优先级 完全匹配>目录匹配>扩展名匹配

我们现在查看几个例子，我们找到tomcat/conf/web.xml ,在这个文件中配置的所有内容，其实是被我们自己的工 程中的web.xml文件继承了，在这个配置文件中有以下几段内容:

```
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```

对于这段配置，只要访问时后缀名是jsp或jspx就会执行名称叫jsp的servlet内容，这是一个很典型的扩展名匹配效 果，在tomcat/conf/web.xml中还有这样一段配置：

```
<servlet-mapping>
    <servlet-name>default</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

注意：这时它的值就是"/"，那么这时它就是一个默认(缺省)的servlet.。 默认的servlet有什么作用呢? 默认的servlet其作用是用于处理其它的servlet处理不了的请求

4.2 load-on-startup配置

上面我们提到过,它可以让我们在服务器实例化servlet时就调用这个servlet，简单说就是可以让一个servlet可以在 服务器启动时，就加载这个servlet。我们以LifeServlet类为例来说明一下。

Servlet的配置

```
<servlet>
    <servlet-name>LifeServlet</servlet-name>
    <servlet-class>cn.itcast.servlet.LifeServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>LifeServlet</servlet-name>
    <url-pattern>/life</url-pattern>
</servlet-mapping>
```

我们在标签中添加了一个，它的值为2,那么这时LifeServlet就会在服务器启动时，跟随启动。 注意: 值越小代表的是优先级越高。

第5章 Servlet的路径

我们在开发中，经常在页面上通过表单或超连接向服务器发送请求，如果我们访问的是一个servlet，那么这时访问servlet的路径应该如何书写？接下来我们就介绍一下，关于在客户端访问servlet的路径问题. 在介绍之前，我们先看一下，现阶段我们有多少种方式可以访问服务器端的一个资源 1 在地址栏上直接输入url 2 超连接的方式 3 通过表单方式 4 通过js的location.href方式 对于以上方式，只有表单提交的方式才可能有POST请求，其它的都是GET请求 下面我们通过示例来说明一下关于在客户端访问servlet的路径问题:

1. 创建一个Demo2Servlet

```java
public class Demo2Servlet extends HttpServlet {

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

        System.out.println("get请求访问");
    }

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

        System.out.println("post请求访问");
    }

}
```

2.配置文件中的配置

```xml
<servlet>
    <servlet-name>Demo2Servlet</servlet-name>
    <servlet-class>cn.itcast.servlet.Demo2Servlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Demo2Servlet</servlet-name>
    <url-pattern>/demo2</url-pattern>
</servlet-mapping>
```

3.demo页面访问

```xml
<h1>绝对路径访问</h1>
<h2>带协议的绝对路径</h2>
<a href="http://localhost:8080/web-day08/demo2">demo2 servlet</a>
<h2>不带协议绝对路径</h2>
<a href="/web-day08/demo2">demo2 servlet</a>

<hr>
<h1>相对路径访问</h1>
<a href="demo2">demo2 servlet</a>
```

以上操作，访问到demo2servlet时在控制台上输出了get请求访问 4.创建一个目录main,将demo2.html文件赋值到main目录下在次访问 http://localhost:8080/web-day08/main/demo2.html 这时我们在访问时，绝对路径是不会出现问题的，但是当访问相对路径时，页面上报出了404,代表找不到资源。 我们分析一下原因: 相对路径，是指我们的servlet访问路径与当前的页面之间的一个相对关系。 那么我们的servlet访问路径是 http://localhost:8080/web-day08/demo2 而我们访问main/demo2.html的路径是 http://localhost:8080/web-day08/main/demo2.html 这时通过路径对比我们发现demo2.html的路径与demo2的路径它们之间不是在一个层次下，如果想要访问到，可以修改成以下结果:

```xml
<a href="../demo2">demo2 servlet</a>
```

这时，在次访问，会发现访问成功。 结论: 对于我们后续的学习与开发中，我们使用绝对路径会比较多一些，而在使用绝对路径时，我们主要使用的是不带协议的绝对路径，而带协议的绝对路径只有在访问站外资源时才会使用。对于相对路径，我们需要分析它们的相对关系，所以不建议大家使用。

第6章 ServletContext对象

在之前的文件下载案例中，我们使用过了一个对象：servletContext。虽然简单的使用过，但是，想要了解这个对象，我们还需要进一步的学习servletContext。

6.1 ServletContext的概述

ServletContext是一个容器（域对象）可以存储键值对数据（String key,Object value），保存在ServletContext中的数据不仅可以提供给所有的servlet使用，而且可以在整个项目范围内使用（后期的过滤器、监听器也可以使用ServletContext）。

它的主要作用有3个：

1、作为域对象

2、可以获取当前应用下任何路径下的任何资源。

3、获取初始化参数

6.2 ServletContext作为域对象

ServletContext中定义了很多方法，在javaweb阶段我最常用的就是将ServletContext作为容器（域对象）使用，因此，接下来我们要学习这个容器（域对象的）API。

6.2.1 API介绍

GenericServlet：

```
1.  ServletContext getServletContext()
    获取ServletContext对象
```

ServletContext：

```
1.  void setAttribute(String name, Object object)
    往servletcontext容器中存入数据，name为数据名称，object为数据的值
2.  Object getAttribute(String name)
    从ServletContext中获取数据，根据指定的数据名称
3.  void removeAttribute(String name)
    从ServletContext中移除数据，根据指定的数据名称
```

6.2.2 使用步骤

1. 创建ServletContextServlet1和ServletContextServlet2
2. ServletContextServlet1调用存方法
3. ServletContextServlet2调用取方法
4. ServletContextServlet2调用删方法
5. ServletContextServlet2调用取方法

6.2.3 演示代码

ServletContextServlet1：

```
package cn.itcast.web;

import javax.servlet.ServletContext;
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;
```

```
public class ServletContextServlet1 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //获取容器
        ServletContext context = getServletContext();
        //存入数据
        context.setAttribute("addr","北京");
    }
}
```

ServletContextServlet2：

```
package cn.itcast.web;

import javax.servlet.ServletContext;
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;

public class ServletContextServlet2 extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        //获取容器
        ServletContext context = getServletContext();
        //获取数据
        String addr = (String) context.getAttribute("addr");
        System.out.println("存入之后，获取数据："+addr);
        //移除数据
        context.removeAttribute("addr");
        //重新获取数据
        String addr2 = (String) context.getAttribute("addr");
        System.out.println("移除之后，获取数据："+addr2);

    }
}
```

web.xml的配置

```
<?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>
        <servlet-name>ServletContextServlet1</servlet-name>
         <servlet-class>cn.itcast.web.ServletContextServlet1</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletContextServlet1</servlet-name>
        <url-pattern>/context1</url-pattern>
    </servlet-mapping>


    <servlet>
        <servlet-name>ServletContextServlet2</servlet-name>
         <servlet-class>cn.itcast.web.ServletContextServlet2</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletContextServlet2</servlet-name>
        <url-pattern>/context2</url-pattern>
    </servlet-mapping>
</web-app>
```

代码准备好之后，分别访问context1和context2，我们会得到以下结果：

```
存入之后，获取数据：北京
移除之后，获取数据：null
```

通过这个结果，我们发现两次请求虽然访问了不同的servlet，但是依然能通过ServletContext共享数据，而且即使是由一个同学先访问，然后由另一个同学访问也是同样的结果。

```
总结：保存在ServletContext中的数据是项目全局共享的数据。
```

6.2.4 其它作用

```
1. getRealPath(String path)
    获取WEB项目的资源文件路径

2. getResourceAsStream(String path)
    根据WEB项目的文件获取流

3. getInitParameter(String name)
    获取初始化参数
```

6.2.5 演示代码

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

        //1.先获取全局管理者--ServletContext对象
        ServletContext servletContext = getServletContext();
        //2.获取WEB项目的资源文件路径
         String path = servletContext.getRealPath("/文件名");
         System.out.println(path);
        //3.根据WEB项目的文件获取流
         InputStream is = servletContext.getResourceAsStream("/文件名");
         System.out.println(is);
    }
```

6.2.6 项目初始化参数

ServletContext表示当前项目环境,从这个对象中可以获取 web.xml 中配置的 当前项目初始化参数

使用步骤

1.在web.xml中配置项目环境参数值

2.在任意一个servlet中取出当前项目环境参数值

演示代码

1.在web.xml中配置项目环境参数值

```
<!--    当前项目环境中的参数值-->
    <context-param>
        <param-name>contentType</param-name>
        <param-value>text/html;charset=UTF-8</param-value>
    </context-param>

<!--    演示当前项目环境中的参数值-->
    <servlet>
        <servlet-name>context</servlet-name>
        <servlet-class>com.itheima.contxt.ContextServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>context</servlet-name>
        <url-pattern>/context</url-pattern>
    </servlet-mapping>
```

2.在任意一个servlet中取出当前项目环境参数值

```
package com.itheima.contxt;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```java
/**
 * 获取当前项目环境中配置的参数值
 * http://localhost:8080/day06/context
 */
public class ContextServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        ServletContext servletContext = this.getServletContext();//获取当前项目环境
对象
        String contentType = servletContext.getInitParameter("contentType");//取
出项目环境对象中的参数contentType的值
        System.out.println("contentType = " + contentType);



        ServletConfig servletConfig = this.getServletConfig();//获取当前Servlet配
置对象
        String myParam = servletConfig.getInitParameter("myParam");//取出当前
Servlet配置对象中的参数值
        System.out.println("ConfigServlet myParam = " + myParam);//null
    }
}
```

运行结果,控制台打印

```
contentType = text/html;charset=UTF-8
ConfigServlet myParam = null
```

6.2.7 Servlet初始化参数

ServletConfig表示当前Servlet的配置,从这个对象中可以获取 web.xml 中当前Servlet自己配置的当前Servlet初始化参数

使用步骤

1.给一个servlet配置初始化参数值

2.在该servlet中获取参数值

演示代码

1.给一个servlet配置初始化参数值

```
<!--    演示当前Servlet中的参数值-->
    <servlet>
        <servlet-name>config</servlet-name>
        <servlet-class>com.itheima.contxt.ConfigServlet</servlet-class>
        <init-param>
            <param-name>myParam</param-name>
            <param-value>text/plain;charset=UTF-8</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>config</servlet-name>
        <url-pattern>/config</url-pattern>
    </servlet-mapping>
```

2.在该servlet中获取参数值

```
package com.itheima.contxt;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 获取当前Servlet中配置的参数值
 * http://localhost:8080/day06/config
 */
public class ConfigServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

        ServletConfig servletConfig = this.getServletConfig();//获取当前Servlet配
置对象
        String myParam = servletConfig.getInitParameter("myParam");//取出当前
Servlet配置对象中的参数值
        System.out.println("ServletConfig myParam = " + myParam);//有值
    }
}
```

运行结果,控制台打印

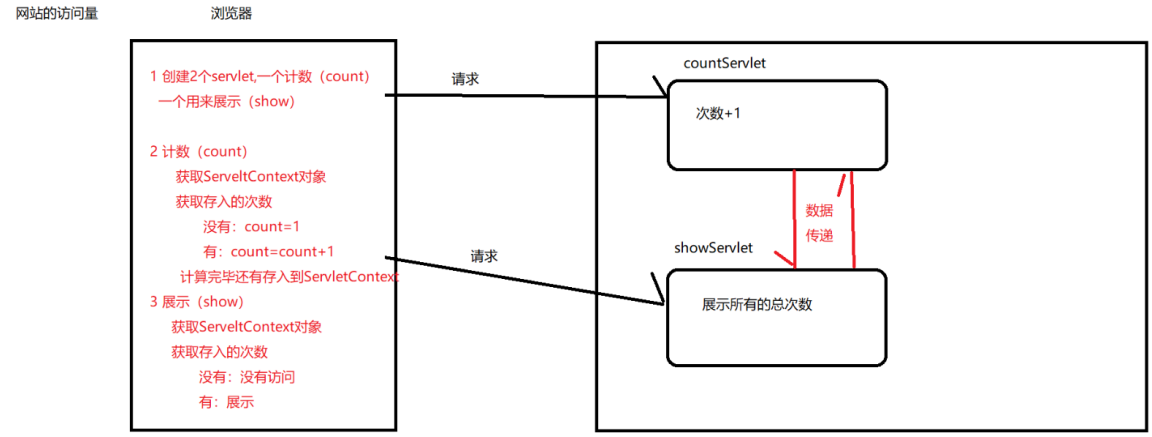
```
ServletConfig myParam = text/plain;charset=UTF-8
```

6.2.8 扩展--案例: 统计访问次数

需求

有一个servlet(countServlet),每访问一次这个servlet,将访问的次数+1,
当我们访问另一个servlet(showServlet),展示访问CountServlet的次数

分析



步骤

```
创建2个servlet(countServlet, showServlet)
    CountServlet中:
                    1 获取servltContext
                    2 从servletContext中先获取一次数据
                            无:
                            count=1
                            有:
                            count=count+1
                    3 将加完的数据重新存取覆盖进去
    ShowServlet中:
                    1 获取servltContext
                    2 从servletContext中获取访问次数
                        无:
                            告诉浏览器没有访问量
                        有:
                            把访问量显示在浏览器上(response)
```

代码实现

CountServlet中的代码

```
package cn.itcast.web;

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

// 统计访问量
public class CountServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        // 获取ServletContext对象
```

```
        ServletContext servletContext = getServletContext();
        // 获取次数
        Integer count =(Integer)servletContext.getAttribute("count");
        // 判断
        if(count==null){
            count=1;
        }else {
            count=count+1;
        }
        // 存进去
        servletContext.setAttribute("count",count);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request,response);
    }
}
```

showServlet中的代码

```
package cn.itcast.web;

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

//展示访问次数
public class ShowServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        response.setContentType("text/html;charset=utf-8");
        // 获取ServletContext对象
        ServletContext servletContext = getServletContext();
        // 获取访问次数
        Object count = servletContext.getAttribute("count");
        // 判断
        if(count==null){
            response.getWriter().print("该网站的访问量为:0");
        }else{
            response.getWriter().print("该网站的访问量为:"+count);
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        doPost(request,response);
    }
}
```

web.xml的配置

```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>
        <servlet-name>CountServlet</servlet-name>
         <servlet-class>cn.itcast.web.CountServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CountServlet</servlet-name>
        <url-pattern>/count</url-pattern>
    </servlet-mapping>


    <servlet>
        <servlet-name>ShowServlet</servlet-name>
         <servlet-class>cn.itcast.web.ShowServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ShowServlet</servlet-name>
        <url-pattern>/show</url-pattern>
    </servlet-mapping>
</web-app>
```