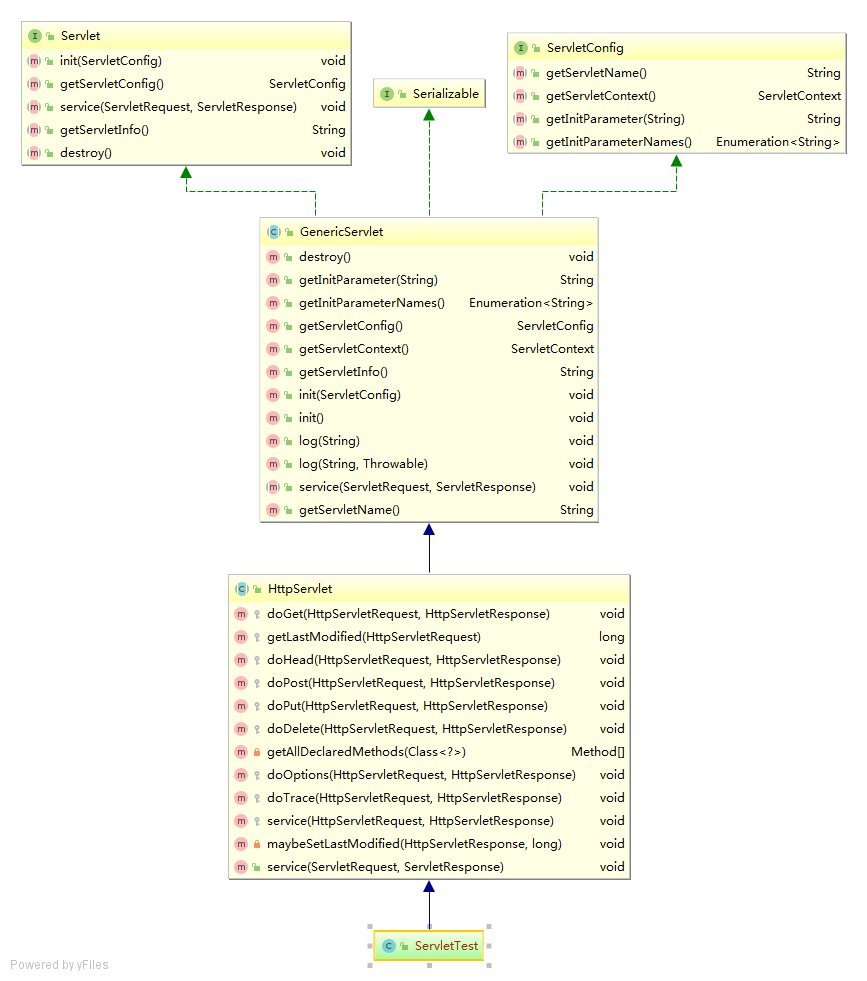
Servlet

## Servlet概念

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。

## 自定义servlet



## 生命周期

### Servlet主要四个生命周期方法

1. 构造方法：客户端请求该 Servlet，加载 Servlet 类到内存，默认情况下 （<load-on-startup>1</load-on-startup>），第一次访问servlet的时候创建servlet对象只调用1次。证明servlet对象在tomcat是单实例的。
2. init方法：实例化并调用init()方法初始化该 Servlet ，创建完servlet对象的时候调用。只调用1次。
3. service方法：根据请求方法不同调用doGet() 或者 doPost()，此外还有doHead()、doPut()、doTrace()、doDelete()、doOptions()、destroy()。 每次发出请求时调用。调用n次。
4. destroy方法： 销毁servlet对象的时候调用。停止服务器或者重新部署web应用时销毁servlet对象。只调用1次。

### 伪代码演示servlet的生命周期

Tomtcat内部代码运行：

1）通过映射找到到servlet-class的内容，全类名com.itmayiedu.a\_servlet.FirstServlet

2）通过反射构造FirstServlet对象

2.1 得到字节码对象

Class clazz = class.forName("com.itmayiedu.a\_servlet.FirstServlet");

2.2 调用无参数的构造方法来构造对象

Object obj = clazz.newInstance(); ---1.servlet的构造方法被调用

3）创建ServletConfig对象，通过反射调用init方法

3.1 得到方法对象

Method m = clazz.getDeclareMethod("init",ServletConfig.class);

3.2 调用方法

m.invoke(obj,config); --2.servlet的init方法被调用

4）创建request，response对象，通过反射调用service方法

4.1 得到方法对象

Method m=clazz.getDeclareMethod("service",HttpServletRequest.class,HttpServletResponse.class);

4.2 调用方法

m.invoke(obj,request,response); --3.servlet的service方法被调用

5）当tomcat服务器停止或web应用重新部署，通过反射调用destroy方法

5.1 得到方法对象

Method m = clazz.getDeclareMethod("destroy",null);

5.2 调用方法

m.invoke(obj,null); --4.servlet的destroy方法被调用

### 时序图演示servlet的生命周期



## 几个核心类

ServletRequest：java对http协议请求内容的封装

HttpServletResponse：java对http协议响应内容的封装

Cookie：java对http协议cookie的封装

HttpSession：java对ttp协议cookie服务端的封装，作用域：一次会话

ServletContext：整个应用相关信息，作用域：作用范围在整个web应用中有效

所有域对象（作用是用于保存数据，获取数据。可以在不同的动态资源之间共享数据）：

HttpServletRequet 域对象

ServletContext域对象

HttpSession 域对象

PageContext域对象

## Servlet 异常处理

当一个 Servlet 抛出一个异常时，Web 容器在使用了 exception-type 元素的 web.xml 中搜索与抛出异常类型相匹配的配置。必须在 web.xml 中使用 error-page 元素来指定对特定异常 或 HTTP 状态码 作出相应的 Servlet 调用。

|  |
| --- |
| *<!-- error-code 相关的错误页面 -->* <**error-page**>  <**error-code**>404</**error-code**>  <**location**>/ErrorHandler</**location**> </**error-page**> <**error-page**>  <**error-code**>403</**error-code**>  <**location**>/ErrorHandler</**location**> </**error-page**> *<!-- exception-type 相关的错误页面 -->* <**error-page**>  <**exception-type**>  javax.servlet.ServletException  </**exception-type** >  <**location**>/ErrorHandler</**location**> </**error-page**> |

## Servlet的多线程并发问题

servlet对象在tomcat服务器是单实例多线程的。

因为servlet是多线程的，所以当多个servlet的线程同时访问了servlet的共享数据，如成员变量，可能会引发线程安全问题。

解决办法：

1. 把使用到共享数据的代码块进行同步（使用synchronized关键字进行同步）
2. 建议在servlet类中尽量不要使用成员变量。如果确实要使用成员，必须同步。而且尽量缩小同步代码块的范围。（哪里使用到了成员变量，就同步哪里！！），以避免因为同步而导致并发效率降低。

## Web三大组件

servlet大致可以分为三个：简单servlet、过滤servlet、监听servlet

### 简单servlet

|  |
| --- |
| public class ServletTest extends HttpServlet {  @Override  protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  super.doGet(req, resp);  }  @Override  protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {  super.doPost(req, resp);  } } |

### 过滤器

过滤器（Filter），过滤器是一种设计模式，主要用来封装Servlet中一些通用的代码。在web.xml中配置哪些URL对应哪些过滤器。如springmvc中字符集过滤器示例：

|  |
| --- |
| <filter>  <filter-name>encodingFilter</filter-name>  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>  <init-param>  <param-name>encoding</param-name>  <param-value>UTF-8</param-value>  </init-param>  <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param> </filter>  public class CharacterEncodingFilter extends OncePerRequestFilter {  private String encoding;  private boolean forceEncoding = false;  public CharacterEncodingFilter() {  }  public void setEncoding(String encoding) {  this.encoding = encoding;  }  public void setForceEncoding(boolean forceEncoding) {  this.forceEncoding = forceEncoding;  }  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  if (this.encoding != null && (this.forceEncoding || request.getCharacterEncoding() == null)) {  request.setCharacterEncoding(this.encoding);  if (this.forceEncoding) {  response.setCharacterEncoding(this.encoding);  }  }  filterChain.doFilter(request, response);  } } |

### 拦截器

它依赖于web框架，在SpringMVC中就是依赖于SpringMVC框架。在实现上,基于Java的反射机制，属于面向切面编程（AOP）的一种运用，就是在service或者一个方法前，调用一个方法，或者在方法后，调用一个方法，比如动态代理就是拦截器的简单实现，在调用方法前打印出字符串（或者做其它业务逻辑的操作），也可以在调用方法后打印出字符串，甚至在抛出异常的时候做业务逻辑的操作。由于拦截器是基于web框架的调用，因此可以使用Spring的依赖注入（DI）进行一些业务操作，同时一个拦截器实例在一个controller生命周期之内可以多次调用。但是缺点是只能对controller请求进行拦截，对其他的一些比如直接访问静态资源的请求则没办法进行拦截处理。

示例：

|  |
| --- |
| public class BaseInterceptor implements HandlerInterceptor{  /\*\*  \* 在DispatcherServlet之前执行  \* \*/  public boolean preHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2) throws Exception {  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*BaseInterceptor preHandle executed\*\*\*\*\*\*\*\*\*\*");  return true;  }  /\*\*  \* 在controller执行之后的DispatcherServlet之后执行  \* \*/  public void postHandle(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, ModelAndView arg3) throws Exception {  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*BaseInterceptor postHandle executed\*\*\*\*\*\*\*\*\*\*");  }  /\*\*  \* 在页面渲染完成返回给客户端之前执行  \* \*/  public void afterCompletion(HttpServletRequest arg0, HttpServletResponse arg1, Object arg2, Exception arg3)  throws Exception {  System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*BaseInterceptor afterCompletion executed\*\*\*\*\*\*\*\*\*\*");  }  } |

**拦截器和过滤器的区别**

1.拦截器是基于动态代理java反射机制的，而过滤器是基于函数回调的。

2.过滤器依赖于servlet容器，而拦截器不依赖于servlet容器。

3.拦截器只对action起作用，而过滤器几乎可以对请求起作用。

4.拦截器可以访问action上下文、值栈里的对象，而过滤器不能。

5.在action的生命周期里，拦截器可以多起调用，而过滤器只能在容器初始化时调用一次。

### 监听器

当某个事件发生时候，监听器里的方法会被调用。例如Tomcat容器启动时、销毁时，session创建时、销毁时。

## servlet容器

### 概念

Java Servlet（Java服务器小程序）是一个基于Java技术的Web组件，运行在服务器端，它由Servlet容器所管理，用于生成动态的内容。 Servlet是平台独立的Java类，编写一个Servlet，实际上就是按照Servlet规范编写一个Java类。Servlet被编译为平台独立 的字节码，可以被动态地加载到支持Java技术的Web服务器中运行。

Servlet容器也叫做Servlet引擎，是Web服务器或应用程序服务器的一部分，用于在发送的请求和响应之上提供网络服务，解码基于 MIME的请求，格式化基于MIME的响应。Servlet没有main方法，不能独立运行，它必须被部署到Servlet容器中，由容器来实例化和调用 Servlet的方法（如doGet()和doPost()），Servlet容器在Servlet的生命周期内包容和管理Servlet。在JSP技术 推出后，管理和运行Servlet/JSP的容器也称为Web容器。

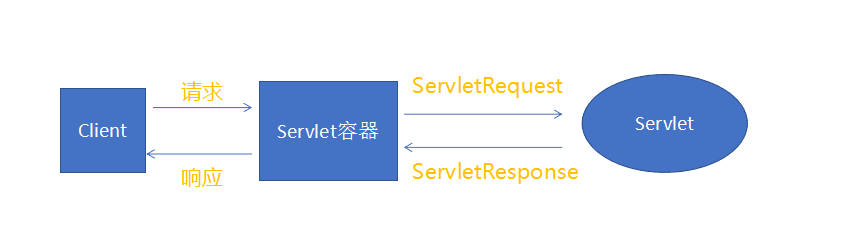
（注：常用的MIME类型：text/html，application/pdf，video/quicktime，application /java，image/jpeg，application/jar，application/octet-stream，application/x- zip）

有了servlet之后，用户通过单击某个链接或者直接在浏览器的地址栏中输入URL来访问Servlet，Web服务器接收到该请求后，并不是将 请求直接交给Servlet，而是交给Servlet容器。Servlet容器实例化Servlet，调用Servlet的一个特定方法对请求进行处理， 并产生一个响应。这个响应由Servlet容器返回给Web服务器，Web服务器包装这个响应，以HTTP响应的形式发送给Web浏览器。

### 种类

目前最流行的Servlet容器软件包括: Tomcat、Jetty、Jboss等。

### Servlet容器响应客户请求过程



### Tomcat 的运行机制

Tomcat是一个servlet容器，其运行机制可参考如下：

1:实现Servlet api规范。这是最基础的一个实现，servlet api大部分都是接口规范。如request、response、session、cookie。为了我们应用端能正常使用，容器必须有一套完整实现。  
2：启动Socket监听端口，等待http请求。  
3：获取http请求，分发请求给不同的协议处理器，如http和https在处理上是不一样的。  
4：封装请求，构造HttpServletRequest。把socket获取的用户请求字节流转换成java对象httprequest。构造httpResponse。  
5：调用(若未创建，则先加载)servlet，调用init初始化，执行servlet.service()方法。  
6：为httpResponse添加header等头部信息。  
7：socket回写流，返回满足http协议格式的数据给浏览器。  
8：实现JSP语法分析器，JSP标记解释器。JSP servlet实现和渲染引擎。  
9：JNDI、JMX等服务实现。容器一般额外提供命名空间服务管理。  
10：线程池管理，创建线程池，并为每个请求分配线程。

## JSP

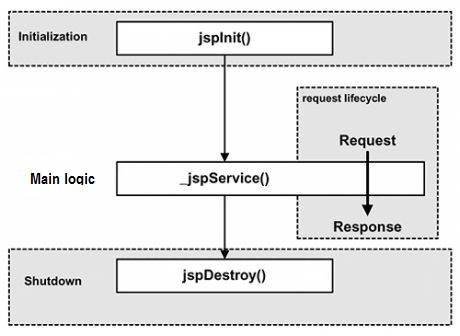
### 简介

JSP全称Java Server Pages，是一种动态网页开发技术。它使用JSP标签在HTML网页中插入Java代码。标签通常以<%开头以%>结束。JSP是一种Java servlet，主要用于实现Java web应用程序的用户界面部分。网页开发者们通过结合HTML代码、XHTML代码、XML元素以及嵌入JSP操作和命令来编写JSP。

JSP通过网页表单获取用户输入数据、访问数据库及其他数据源，然后动态地创建网页。JSP标签有多种功能，比如访问数据库、记录用户选择信息、访问JavaBeans组件等，还可以在不同的网页中传递控制信息和共享信息。

### 生命周期

Jsp本质是servlet，所以先编译成servlet对象后其生命周期和servlet一样



### JSP 隐式对象

JSP隐式对象是JSP容器为每个页面提供的Java对象，开发者可以直接使用它们而不用显式声明。JSP隐式对象也被称为预定义变量。JSP所支持的九大隐式内置对象：

|  |  |
| --- | --- |
| 对象 | 描述 |
| request | HttpServletRequest类的实例 |
| response | HttpServletResponse类的实例 |
| out | PrintWriter类的实例，用于把结果输出至网页上 |
| session | HttpSession类的实例 |
| application | ServletContext类的实例，与应用上下文有关 |
| config | ServletConfig类的实例 |
| pageContext | PageContext类的实例，提供对JSP页面所有对象以及命名空间的访问 |
| page | 类似于Java类中的this关键字 |
| Exception | Exception类的对象，代表发生错误的JSP页面中对应的异常对象 |