## java web简介

### 概念

Java Web，是用Java技术来解决相关web互联网领域的技术总和。web包括：web服务器和web客户端两部分。Java在客户端的应用有java applet，不过[使用](https://baike.baidu.com/item/%E4%BD%BF%E7%94%A8/7741550" \t "https://baike.baidu.com/item/java%20web/_blank)得很少，Java在服务器端的应用非常的丰富，比如Servlet，JSP和第三方框架等等。Java技术对Web领域的发展注入了强大的动力。

Java的Web框架虽然各不相同，但基本也都是遵循特定的路数的：使用[Servlet](https://baike.baidu.com/item/Servlet" \t "https://baike.baidu.com/item/java%20web/_blank)或者[Filter](https://baike.baidu.com/item/Filter" \t "https://baike.baidu.com/item/java%20web/_blank)拦截请求，使用[MVC](https://baike.baidu.com/item/MVC" \t "https://baike.baidu.com/item/java%20web/_blank)的思想设计架构，使用约定，[XML](https://baike.baidu.com/item/XML" \t "https://baike.baidu.com/item/java%20web/_blank)或 Annotation实现配置，运用Java[面向对象](https://baike.baidu.com/item/%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1" \t "https://baike.baidu.com/item/java%20web/_blank)的特点，将各种协议属性封装成对象，如request、response、context等，面向对象实现请求和响应的流程。

### 请求过程

一次完整的http请求过程，以java tomcat为例，涉及的技术主要有网络编程（数据传输、各种协议http、ip/tcp,socket）、servlet、容器等等

1. 浏览器输入url发起
2. 浏览器解析封装http请求报文socket发送
3. tomcat容器接受http请求报文解析封转request、response，调用app服务器程序
4. app服务器接受业务处理再返回
5. tomcat容器解析response生成响应报文socket发送给浏览器
6. 浏览器接受响应报文解析处理数据或页面（响应body里是一个页面，浏览器会解析并展示）。

## servlet-api

### servlet概念

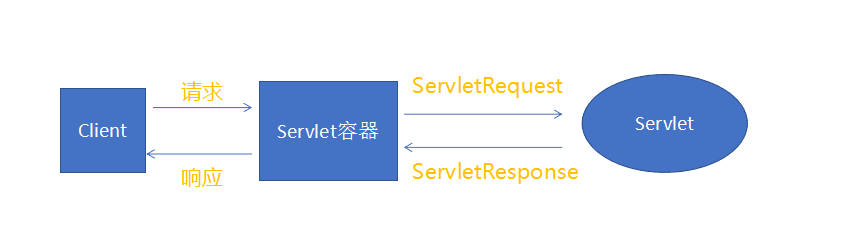
Servlet（Server Applet）是Java Servlet的简称，称为小服务程序或服务连接器，用Java编写的服务器端程序，具有独立于平台和协议的特性，主要功能在于交互式地浏览和生成数据，生成动态Web内容。

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(servlet标准规范servlet-api中)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。

### servlet容器

servlet不是独立的应用程序，没有 [main](https://baike.baidu.com/item/main" \t "https://baike.baidu.com/item/servlet/_blank)() 方法,不是由用户或程序员调用，而是由另外一个应用程序(容器)调用,提供了 Servlet 功能的服务器，叫做 Servlet 容器如tomcat

Servlet容器响应客户请求过程



### servlet生命周期

public interface Servlet {  
 public void init(ServletConfig config) throws ServletException;  
 public ServletConfig getServletConfig();  
 public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;  
 public String getServletInfo();  
 public void destroy();  
}

所有servlet必须实现这个接口，可以继承GenericServlet、HttpServlet等已近实现此接口的类或接口，通过注解@WebServlet或者在部署描述符（web.xml）中用xml配置，在开启web容器时将servlet注册到容器中，一般只在启动项目时注册所以添加修改servlet的url需要重启，当客户端请求某个servlet时，根据注册的url找到相应的servlet，这个servlet会经历一下生命周期

1. 加载 Servlet 类到内存实例化，默认第一次访问servlet的时候创建servlet对象只调用1次，保证servlet对象在tomcat是单实例的。
2. 调用init(ServletConfig config)初始化该 Servlet，实例化后调用，根据config参数初始化该 Servlet ，所以也只调用1次
3. 调用service(req，res)，方法内根据请求方法不同调用doGet() 、doPost()等处理具体业务逻辑，每次发出请求时调用，调用n次。
4. 调用destroy(),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，所以只调用1次。



### 过滤器Filter

public interface Filter {  
 public void init(FilterConfig filterConfig) throws ServletException;  
 public void doFilter(ServletRequest request, ServletResponse response,FilterChain chain)throws IOException, ServletException;  
 public void destroy();

}

过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类，它对资源的请求(servlet或静态内容)或资源的响应执行过滤任务，或者两者都执行，一种设计模式，主要用来封装Servlet中一些通用的代码。可用@WebFilter或xml配置声明配置相关参数

public @interface WebFilter {  
 String description() default "";  
 String displayName() default "";  
 WebInitParam[] initParams() default {};  
 String filterName() default "";   
 String smallIcon() default "";  
 String largeIcon() default ""，  
 String[] servletNames() default {};  
 String[] value() default {};

String[] urlPatterns() default {};

//Filter四种拦截方式：request(默认),forward (拦截转发）,include（拦截包含）,error (只拦截错误页面)，forward和include区别详见下面的请求分派器  
 DispatcherType[] dispatcherTypes() default {DispatcherType.REQUEST};

boolean asyncSupported() default false;  
}

**执行过程/生命周期**

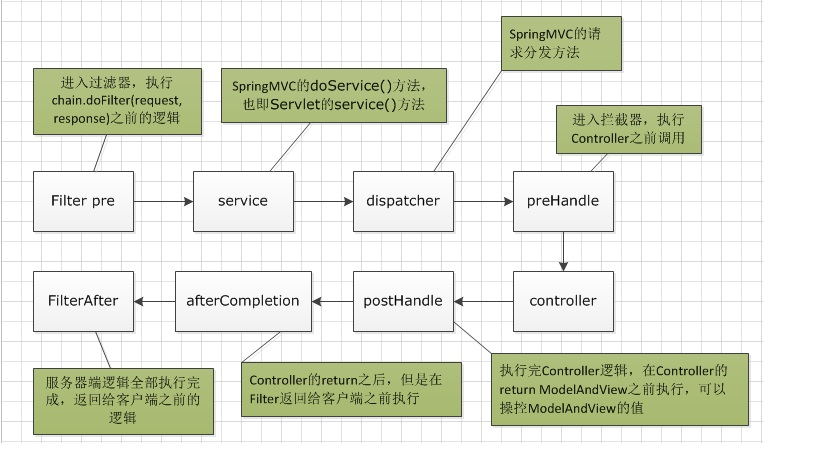


**拦截器**

java里的拦截器是动态拦截Action调用的对象。它提供了一种机制可以使开发者可以定义在一个action执行的前后执行的代码，也可以在一个action执行前阻止其执行，同时也提供了一种可以提取action中可重用部分的方式。在AOP中拦截器用于在某个方法或字段被访问之前或之后，进行拦截然后在之前或之后加入某些操作，如项目中的积分埋点，具体创建学习springmvc拦截器。

**过滤器和拦截器区别**

1. Filter是servlet规范规定的，依赖servlet容器，拦截器是第三方框架支持的如springmvc，可用于web程序、Applicaton、swing等程序中
2. 过滤器是基于函数回调的，拦截器是基于动态代理java反射机制的
3. 过滤器只在servlet前后起作用拦截所有匹配的请求，拦截器能够深入到方法前后、异常抛出前后等可以使用spring里的任何组件、对象更加灵活，在spring项目中优选使用拦截器，在拦截器里注入一个service，可以调用业务逻辑。



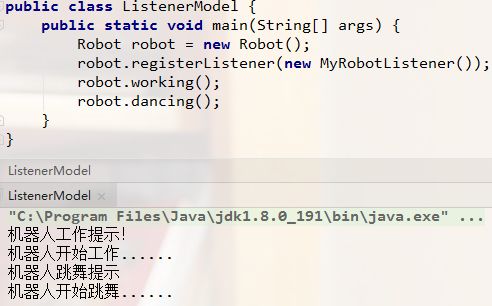
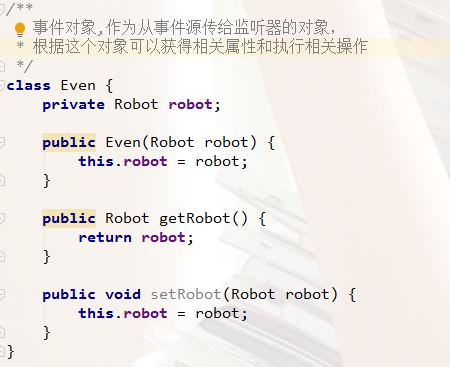
### 监听器Listener

监听观察某个事件（程序）的发生情况，当被监听的事件真的发生了的时候，事件发生者（事件源） 就会给注册该事件的监听者（监听器）发送消息，告诉监听者某些信息，同时监听者也可以获得一份事件对象，根据这个对象可以获得相关属性和执行相关操作。

监听器模型涉及以下三个对象，模型图如下：

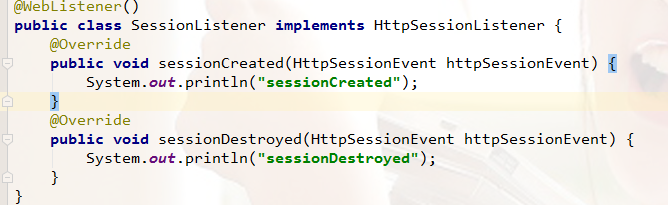
（1）事件：用户对组件的一个操作，或者说程序执行某个方法，称之为一个事件，如机器人程序执行工作。  
（2）事件源：发生事件的组件就是事件源，也就是被监听的对象，如机器人可以工作，可以跳舞，那么就可以把机器人看做是一个事件源。  
（3）事件监听器（处理器）：监听并负责处理事件的方法，如监听机器人工作情况，在机器人工作前后做出相应的动作，或者获取机器人的状态信息。

示例



监听器模式：事件源经过事件的封装传给监听器，当事件源触发事件后，监听器接收到事件对象可以执行监听器要实现的业务功能，也可以回调事件源的方法。  
观察者模式：观察者(Observer)相当于事件监听者，被观察者(Observable)相当于事件源和事件，执行逻辑时通知触发observer的通知方法，将要处理的数据做为参数传递给观察者。

servlet中的监听器接口主要是对request、response、session、context的监听，如对session的监听可以监控到创建session的次数从而知道登录的用户数。



### 请求分派器（转发）

public interface RequestDispatcher{

public void forward(ServletRequest request, ServletResponse response)throws ServletException, IOException;

public void include(ServletRequest request, ServletResponse response)throws ServletException, IOException;

}

RequestDispatcher是一个接口一种规范由servlet容器实现创建对象，其中2个方法代表转发，这就是转发的本质。

RequestDispatcher.forward(request, response)

这个方法将请求从一个 Servlet or JSP目标资源 上 转发到服务器上的另一个资源（servlet、JSP 文件或 HTML 文件，这些资源必须是当前Web上下文中的），让其它的资源去生成响应数据。

例如用户请求的是目标资源A，A接受到请求后，转发到B，真正产生响应数据是被转发的资源B，而A只是起个引导转发作用。浏览器的地址栏不会变，依然是A的URL。 这个方法可以允许被请求的目标资源做一些准备工作后，再让转发的资源去响应请求。

注意事项：

1、在目标资源中调用forward方法时，必须保证此响应没有提交。也就是不要使用 ServletResponse 对象的输出流对象，因为即便你写入了数据到响应缓冲区，最后也会被清空，如果缓冲区数据被刷新提交（out.flush），还会抛出IllegalStateException异常。

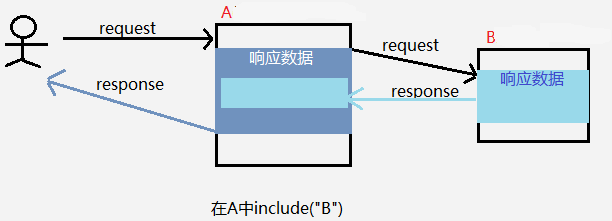
2、对于forward方法传递的request对象：虽然我们从调用上看，好像是将request对象传递给转动的资源上去了，但是我发现目标资源使用的request对象和转发的资源使用的request对象不是同一个request对象，因为分别从这2个request中获取RequestURL，发现是不一样的。但是在目标资源request提取的Paramter 和 Attribute   ，在转发后的资源的request对象中，依然都可以提取到，且是相同的。所以，二者只是在请求路径相关的属性上不同，其它API调用返回的都是一样的。

3、在forward语句的前后，都不应该有响应输出的语句，应该会被忽略。

RequestDispatcher.include(request, response)

此方法用于包含响应中某个资源（servlet、JSP 页面和 HTML 文件）的内容。调用者指定一个被包含的资源，将这个包含的资源（JSP,Servlet，HTML）的响应数据包含到自己的响应体中。即将第二个servlet响应内容包含在第一个响应内容里。

这个过程实质是用一个相同的Request再请求一次被包含的资源，将被包含的资源的响应数据包含到原本的资源中去，构成它的响应数据的一部分。



注意事项：

1、被包含者不能设置ServletResponse的响应状态和响应头（否则并不会产生效果），因为这些都是包含者做的事，被包含者只需要产生响应数据解可以了。

2、不同于 forward中的request的传递特性：在被包含的资源中从request中获取请求路径相关的信息，发现依然是原始请求的路径，也就是浏览器地址栏相关的路径，也就是说被包含的资源获得的request对象的路径属性和原始请求资源的路径一样。其它的API调用也是一样的（Attribute 和Parameter）。

### servlet异常处理

当一个 Servlet 抛出一个异常时，web容器会在web.xml中查找 error-page配置，可以配置异常类型或状态码对应的servlet或页面，如果没有配置那么会直接将异常信息传给客户端。

|  |
| --- |
| <error-page>            <error-code>500</error-code>            <location>/500error.jsp</location>      </error-page>  <error-page>            <exception-type>java.lang.NullException</exception-type>            <location>/Nullerror.jsp</location>    </error-page>  统一的异常页面  <error-page>                  <location>/globalError.jsp</location>    </error-page>  页面也可以是对应的servlet，本质jsp就是一个包含页面信息的servlet |

### servlet多线程并发

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。

## tomcat

### 简介

### 优化