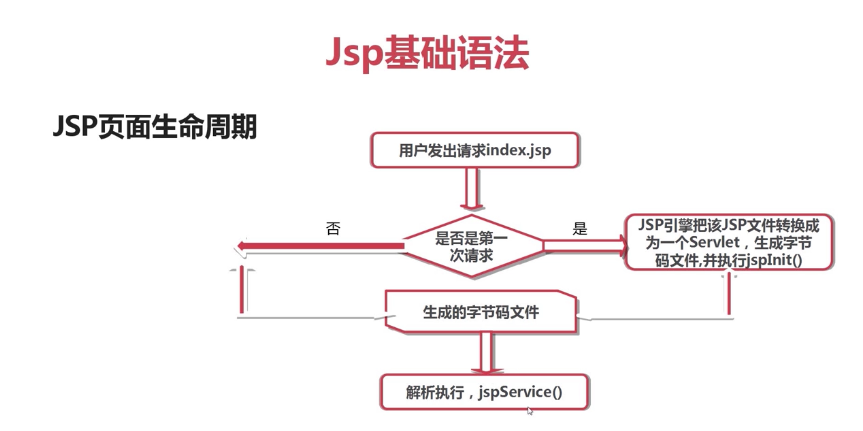
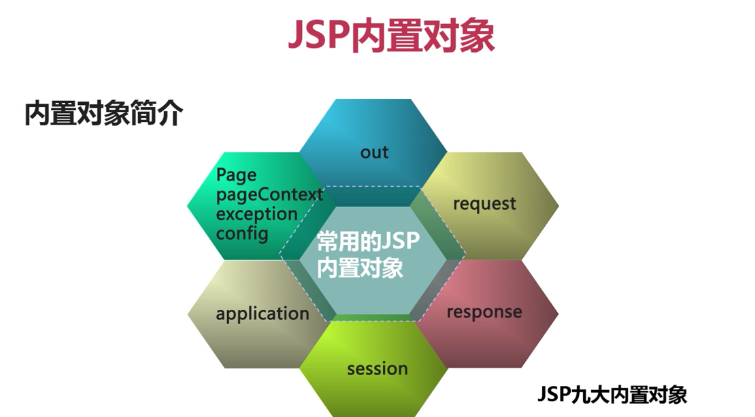
### JSP：



JSP页面生命周期.png

思路：当用户使用客户端浏览器向服务器发送一个请求访问index.jsp页面，当服务器接收到客户的请求之后，开始判断用户所访问的这个资源是否是第一次访问？  
①如果是第一次请求，那么Tomcat的JSP引擎就把这个JSP文件转换成一个Servlet（一个Servlet从本质上而言也是一个Java类），既然是一个Java类就需要通过编译，生成相应的字节码文件，然后执行**jspInit**这个初始化方法。再生成JSP页面的字节码文件，转到③。  
②直接访问所生成的JSP页面的字节码文件，转到③。  
③最后解析执行这个Java类当中的**jspService**方法，这个方法就是专门用来处理用户的请求。

### JSP九大内置对象：



JSP内置对象简介.png

**out对象**：它是JspWriter类的实例，是向客户端输出内容常用的对象。  
**request对象**：它表示客户端的一次请求，这个请求里面的信息是被封装在request对象中的。同时它也是HttpServletRequest类的实例，并且具有请求域，也就是在完成客户端的请求之前，该对象是一直有效的。  
常用方法：

|  |  |
| --- | --- |
| 1 | request.setCharacterEncoding("UTF-8");//设置编码，解决中文乱码问题。 |

|  |  |
| --- | --- |
| 1  2 | request.setAttribute("","");//前者为需要设置的属性，后者为需要的值。  request.getAttribute(); |

|  |  |
| --- | --- |
| 1 | request.getRequestDispatcher("index.jsp").forward(request, response);//请求转发到index.jsp页面，该请求对象是一直有效的。 |

**response对象**：它包含了响应客户请求的相关信息，它也是HttpServletResponse类的实例。和request一样，它具有页面的作用域，也就是在访问一个页面的时候，该页面的response对象只对这次访问有效，往后执行是不存在、无效的。  
常用方法：

|  |  |
| --- | --- |
| 1 | response.setContentType("text/html","charset=utf-8");//设置响应的MIMI类型。 |

|  |  |
| --- | --- |
| 1 | response.sendRedirect("index.jsp");//请求重定向，如果存在多次重定向，那么对象数据是不会保存下来的，即为null。 |

**请求转发和请求重定向的区别分析：**

*请求重定向：客户端行为，response.sendRedirect()，从本质上讲等同于两次请求，前一次的请求对象不会保存，地址栏的URL地址会改变。  
请求转发：服务器行为，request.getRequestDispatcher().forward(req, resp);是一次请求，转发后请求对象会保存，地址栏的URL地址不会改变。*

**session对象：**当我们第一次访问页面的时候，session对象就会被自动装载和创建。它是HttpSession类的实例对象。  
什么是会话？为什么要用session对象？

*会话：****当一个客户端打开浏览器连接到服务器开始，到客户端关闭浏览器服务器结束这个过程，我们称为一个会话。*** *在实际情况下，客户在访问服务器的过程中，避免不了会在服务器的几个页面之间切换，这时候服务端有必要知道当前操作的客户是不是操作前几个页面的客户，也就是必须要清楚知道是否同一个人在进行操作。所以就引入了session对象这个概念。*

**application对象**：它实现了用户间数据的共享，可以存放全局变量，它是ServletContext类的实例。它的生命周期开始于服务器的启动，终止于服务器的关闭。

*注：  
①在用户的前后连接或不同用户之间的连接中，可以对application对象的同一属性进行操作。  
②在任何地方对application对象属性的操作，都将影响到其他用户对此的访问。*

**page对象**就是指向当前JSP页面本身，有点象类中的this指针，它是 java.lang.Object类的实例

**exception对象**是一个例外对象，当一个页面在运行过程中发生了例外，就产生这个对象。如果一个JSP页面要应用此对象，就必须把**isErrorPage设为true，否则无法编译**。他实际上是Java.lang.Throwable的对象

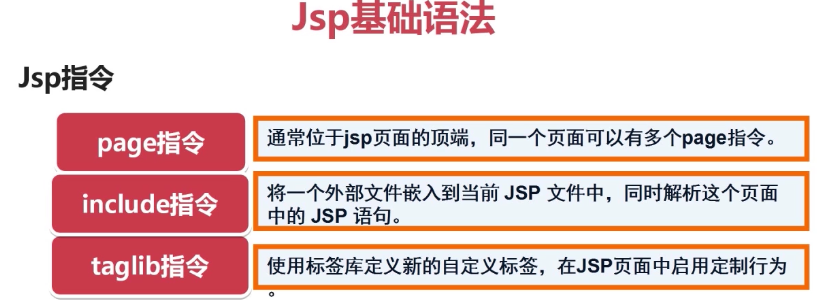
**pageContext对象**提供了对JSP页面内所有的对象及名字空间的访问，也就是说他可以访问到本页所在的SESSION，也可以取本页面所在的application的某一属性值，他相当于页面中所有功能的集大成者，它的本类名也叫pageContext。

**config对象**是在一个Servlet初始化时，JSP引擎向它传递信息用的，此信息包括Servlet初始化时所要用到的参数（通过属性名和属性值构成）以及服务器的有关信息（通过传递一个ServletContext对象）

**JSP的指令和动作：**

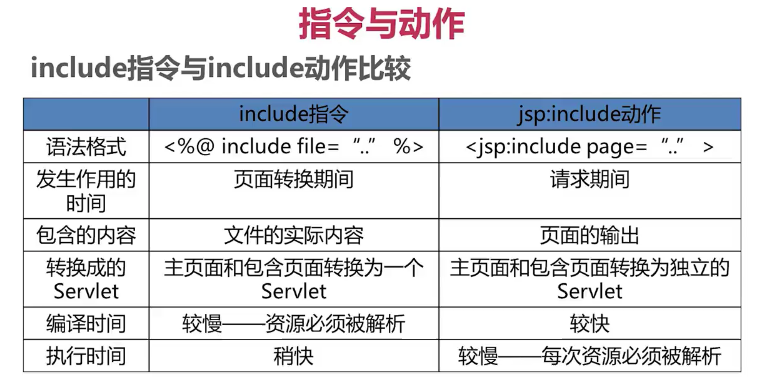
①三个编译指令：page、include、taglib。  
②七个动作指令：jsp:forward、jsp:param、jsp:include、jsp:plugin、jsp:useBean、jsp:setProperty、jsp:getPProperty。

**JSP指令**：



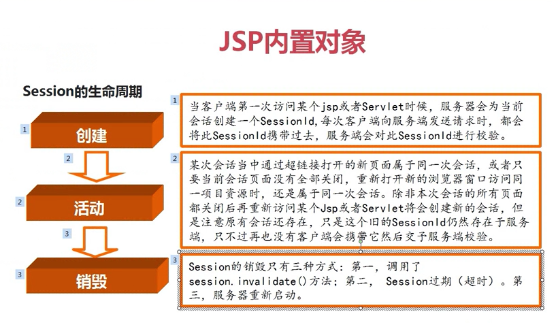
JSP指令.png

**include指令和include动作指令区别：**



include指令与include动作的区别.png

### Session的生命周期



Session生命周期的“活动”思路：

①某次会话当中通过超链接打开的新页面属于同一次会话。

②只有当前会话页面没有全部关闭，重新打开新的浏览器窗口访问同一项目资源时属于同一次会话。

③除非本次会话的所有页面都关闭后再重新访问某个Jsp或者Servlet将会创建新的会话。

注：

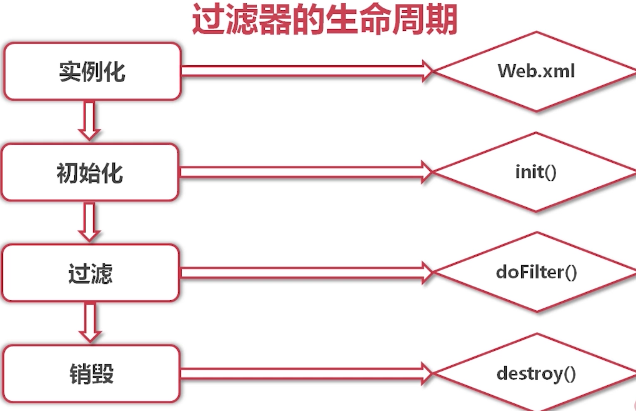
注意原有会话是还存在的，只是这个旧的SessionId仍然存在于服务端，只不过再也没有客户端会携带它然后交予服务端校验。



Session与Cookie的对比.png



过滤器的工作原理.png



过滤器的生命周期.png

*注：  
①****当Web容器启动的时候，过滤器就已经加载并启动了。*** *②实****例化只会实例化一次，也就是说在Web容器中只会使用一次。***



过滤器的分类.png

由于没有配置dispathcer，默认就会按照REQUEST的方式进行。

<filter>

      <filter-name>SecondFilter</filter-name>

      <filter-class>com.test.filter.SecondFilter</filter-class>

  </filter>

<filter-mapping>

      <filter-name>SecondFilter</filter-name>

      <url-pattern>/main.jsp</url-pattern>

      <dispatcher>FORWARD</dispatcher>

  </filter-mapping>

然后看一下ERROR过滤器，通常我们会在web.xml中配置错误页面，如下：

<error-page>

      <error-code>404</error-code>

    <location>/error.jsp</location>

  </error-page>

此时，如果访问不存在的页面，http://localhost:8080/xxx.jsp就会跳转到error.jsp，此时如果过滤器通过REQUEST方式，想要触发，url填写的是/error.jsp并不会起作用，此时就需要把dispathcer改成 ERROR，并且放置在error-page标签下面：

<error-page>

    <error-code>404</error-code>

  <location>/error.jsp</location>

</error-page>

<!-- 需要放在errorpage下面 -->

<filter>

    <filter-name>ErrorFilter</filter-name>

    <filter-class>com.test.filter.ErrorFilter</filter-class>

</filter>

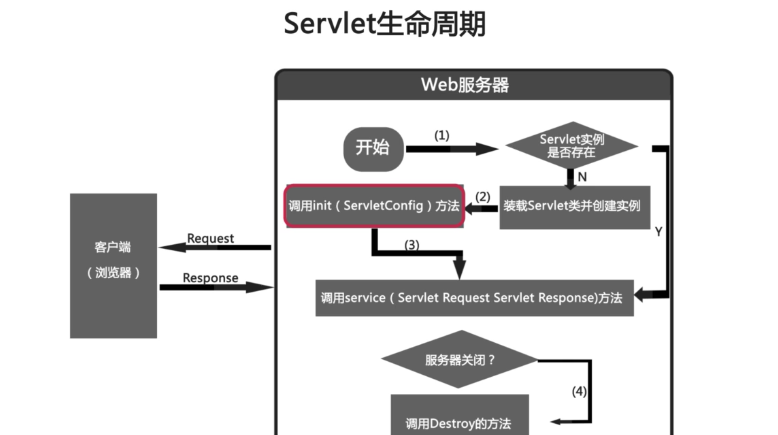
<filter-mapping>

    <filter-name>ErrorFilter</filter-name>

    <rl-pattern>/error.jsp</url-pattern>

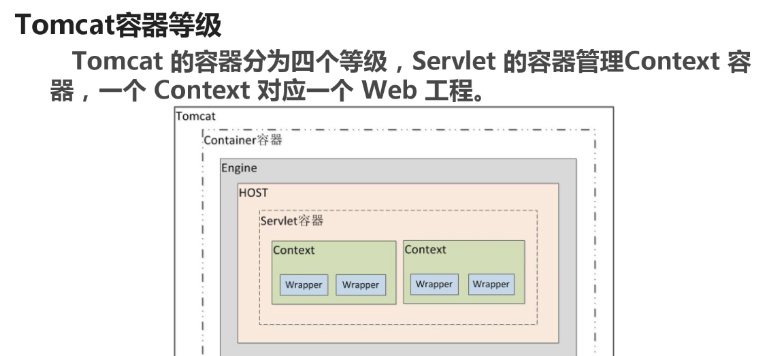
    <dispatcher>ERROR</dispatcher>

</filter-mapping>



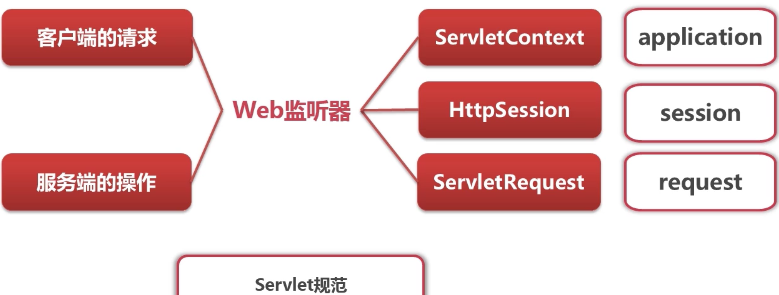
Servlet生命周期.png

思路:  
①首先判断Servlet实例是否存在，如果存在则跳入第③步。  
②初始化阶段，init初始化方法是在创建实例之后完成的。  
③响应阶段，响应客户端的请求，调用service()方法。由service()方法根据提交方式选择性执行doGet()方法或者doPost()方法。  
④终止阶段，调用Destroy()方法销毁，服务器关闭。



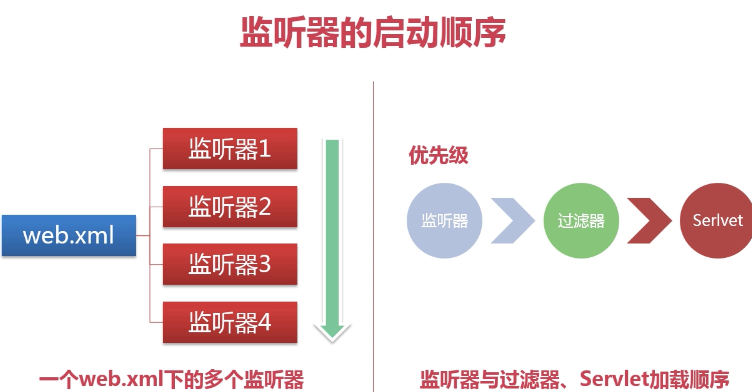
Tomcat容器等级.png

### Web监听器



web监听器.png

**Web监听器是有Servlet规范定义的一种特殊类**，**它可以监听客户端的请求和服务端的操作，同时它还可以监听，包括ServletContext、HttpSession、ServletRequest等对象。**



Web监听器的启动顺序.png

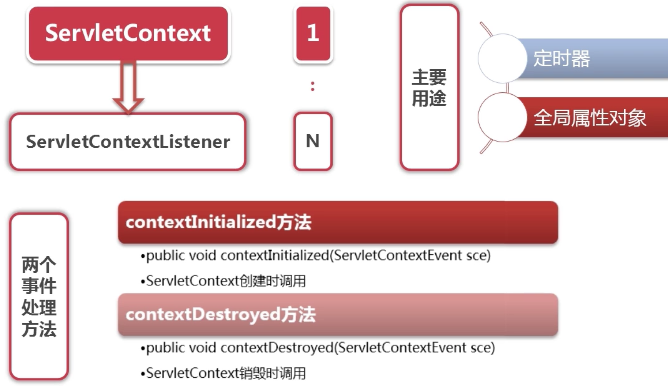
### **监听器的分类**

**HttpSession的事件监听器**



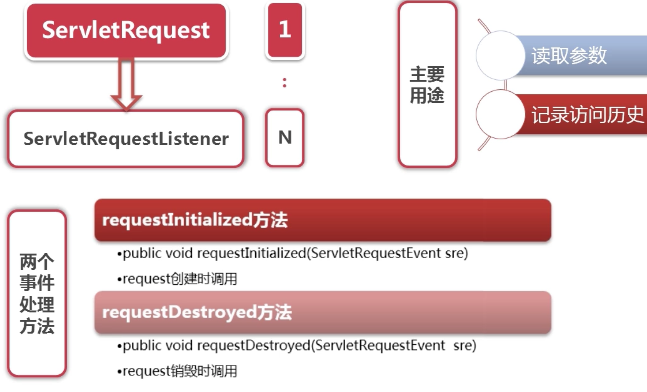
HttpSession的事件监听器.png

**ServletContext的事件监听器**



ServletContext的事件监听器.png

**ServletRequest的事件监听器**



ServletRequest的事件监听器.png

*注：****这三个监听事件，分别实现对应的接口****，也就是图示的1..N。既然实现了接口，即必然要实现其对应的方法，如图所示。主要用途也如图所示。*

**绑定到HttpSession域中的对象状态的事件监听器**



绑定到HttpSession域中的对象状态的事件监听器.png

*注：  
为什么要用session钝化？  
首先，正常情况下，我们的session是存放在服务器的session当中的，但是如果我们的用户很多的时候，仍然把session存放在服务器里，这时服务器的开销是很大的，会直接影响我们Web应用的使用。所以就需要使用到session钝化机制，****它的本质就是将服务器未使用到的session对象暂时序列化到系统文件当中，当要使用的时候再反序列化到服务器内存当中。***

在Servlet规范中定义了多种类型的监听器，它们用于监听的事件源分别为**SerlvetConext,HttpSession和ServletRequest**这三个域对象。

Servlet规范针对这三个对象上的操作，又把这多种类型的监听器划分为三种类型：

1>**监听三个域对象创建和销毁的事件监听器**

2>**监听域对象中属性的增加和删除的事件监听器**

3>**监听绑定到HttpSession域中的某个对象的状态的事件监听器。**

a、如三个域对象的创建与销毁方法签名：

ServletRequestListener有以下两个方法

void requestDestroyed(ServletRequestEvent sre)

void requestInitialized(ServletRequestEvent sre)

HttpSessionListener有以下两个方法

void sessionCreated(HttpSessionEvent se)

void sessionDestroyed(HttpSessionEvent se)

ServletContextListener

void contextDestroyed(ServletContextEvent sce)

void contextInitialized(ServletContextEvent sce)

b、三个类型对象域中属性增、删、改的监听器（3个）

ServletContextAttributeListener，

HttpSessionAttributeListener，

ServletRequestAttributeListener

c、感知型监听器（2个）：监听自己何时被帮到session上，何时解绑了；何时被钝化了，何时被活化了(序列化到某个存储设置中)。

注意：**这种监听器不需要注册**。某个javabean实现这些接口后就可以监听何时被绑定、解绑或被激活或钝化。

HttpSessionBindingListener：实现该接口的类，能**检测自己何时被Httpsession绑定，和解绑**

HttpSessionActivationListener：实现该接口的类(要求些javabean必须是实现了Serializable接口的)，**能监测自己何时随着HttpSession一起激活和钝化。**

三、Listenner的注册

Javaweb的这些特别的监听器注册比较简单，因为ServletRequest、HttpSession、ServletContext对象都是容器创建的，所以需要它们的监听器的话，只需要在应用的web-info的web.xml里配置即可：

<listener>

<listener-class>com.test.listener.MyHttpSessionAttributeListener</listener-class>

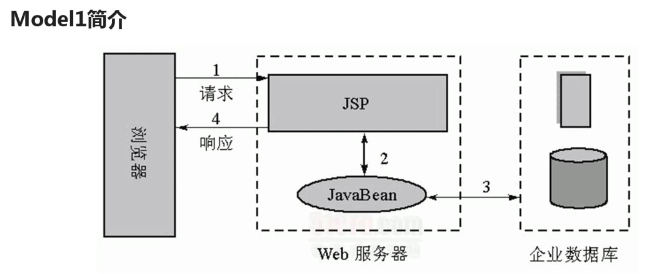
<listener-class>com.test.listener.MyServletContextAttributeListener</listener-class>

<!--  多个实现相同的Listenner接口，在执行时是按web.xml注册出现的来决定。 -->

</listener>

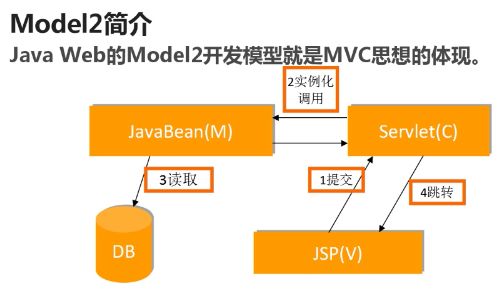
注意：感知型监听器HttpSessionBindingListener、HttpSessionActivationListener不需要注册。

### Model1和Model2



Model1模型.png

*注：  
Model1是最早的能体现分层思想的一种开发模式，简单来说Model1就是一种Jsp+JavaBean的一种开发模式。  
缺点：可维护性和可扩展性是比较差的。*



Model2模型.png

思路：  
①首先由Jsp页面向Servlet提交一个请求。  
②然后在我们的控制层Servlet实例化一个模型层的对象或者调用模型层的一些功能。  
③然后由模型层来访问读取我们的数据库层。  
④当得到读取到结果后，再将结果返回到我们的控制层。  
⑤控制层得到这个结果后，会根据这个结果给用户展示不同的Jsp页面。

**JSP 与 Servlet 的关系**

Tomcat 等 Web 容器最终会把 JSP转化为 Servlet

Jsp更擅长表现于页面显示, Servlet更擅长于逻辑控制

Servlet是利用 System.out.println()来输出 html 代码，由于包括大量的HTML标签、大量的静态文本及格式等，导致Servlet的开发效率低下

JSP通过在标准的HTML页面中嵌入Java代码，其静态的部分无须Java程序控制，Java 代码只控制那些动态生成的信息

最终 JSP 被容器解释为 Servlet，其中Html 代码也是用System.out.println()等拼接输出的

JSP 第一次访问的时候，要转化为 java 文件，然后编译为 class 文件，所以第一次访问 JSP 速度会比较慢，后面会快很多

**Servlet 生命周期**

主要是java.servlet.Servlet接口中的init() 、service() 、和destroy() 3个方法。

初始化阶段，web容器通过调用init()方法来初始化Servlet实例，在Servlet的整个生命周期类，init()方法只被调用一次

客户请求到来时，容器会开始一个新线程，并调用servlet的 service()方法，service() 方法根据请求的http方法来调用 doget() 或dopost()

终止阶段调用destroy()方法，销毁一些资源

GET 请求 vs POST 请求

GET用于信息获取，是安全的和幂等的，GET一般是对后台数据库的信息进行查询

POST表示可能修改变服务器上的资源的请求，一般是对后台数据库进行增、删、改的操作

GET请求的参数会跟在URL后进行传递，请求的数据会附在URL之后，以?分割URL和传输数据，参数之间以&相连，一般浏览器对 URL 的长度会有限制

POST请求，提交的数据则放置在是HTTP包的包体中，用类似Key-Value的格式发送一些数据，相对来说，GET请求会把请求的参数暴露在 URL 中，安全性比POST差一些

HTTP 请求的基本格式

<request line> 请求行

<headers> 请求头（参数头）

<blank line> 空白行

[\*][<request-body>] 请求实体（GET没有， POST有）