**第四章 请求和响应**

1. **容器提供的服务：**

通信支持：让servlet与Web服务器之间通话；

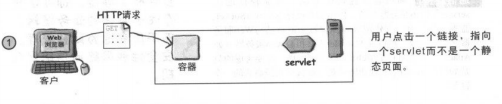
生命周期管理：负责Servlet的加载，实例化，调用init初始化，调用service方法，释放资源

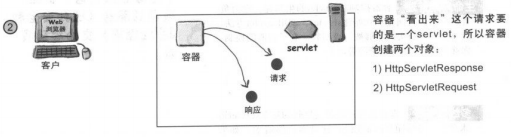
多线程支持：响应服务器的多个请求

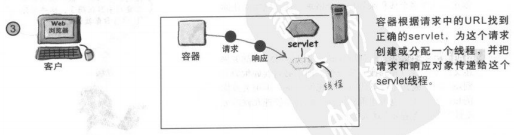
负责JSP代码转换：将JSP转换为Servlet

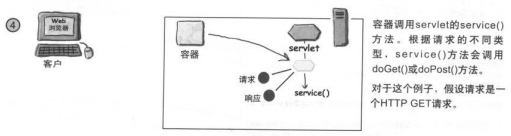
声明方式实现安全：用XML部署描述文件来配置安全性，不需要硬编码到Servlet中。

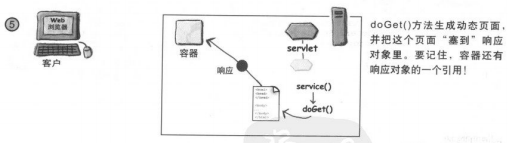
1. **容器处理请求的步骤：**

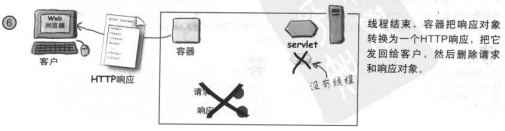




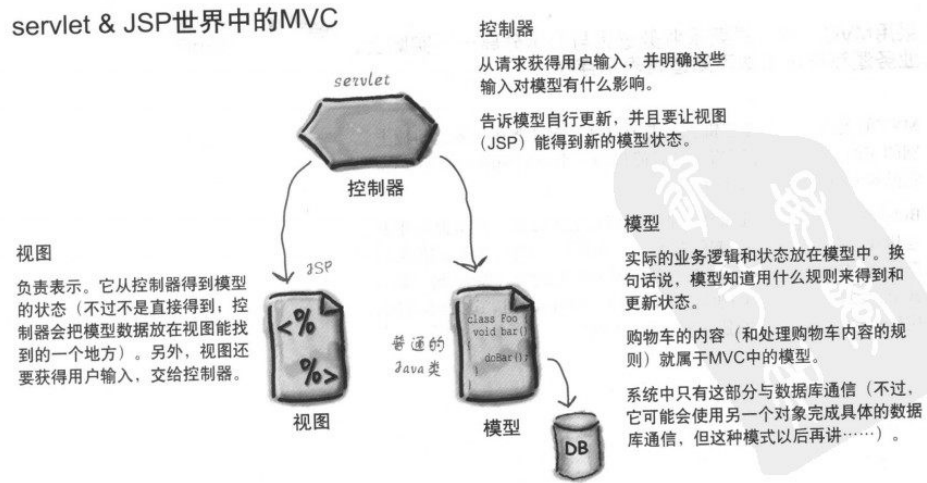






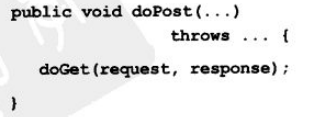


1. **JSP，Servlet中的MVC思想**



1. 每个容器中都维护一个Servlet的实例，但针对每一次请求，容器会为此请求创建一个单独的线程，访问servlet实例获得服务。
2. Http的幂等性：指一次或多次对一个资源的访问不会产生不同的副作用，如Get，Put是幂等的，而Post则不是，因为Post发出的请求可能要对服务器进行修改。
3. Http的请求方法Get，Post的区别：
4. 请求大小：Get请求参数放在URL地址的后面，大小有限制，但Post方法则将请求参数放到请求体中，大小没有限制；这样也比较安全；
5. 功能：Get用于从服务器中获取某个资源，Post的不但可以获取动态的资源，还可以更改服务器中的信息；

如果在HTML的表单中没有指定请求方法，则默认为Get，所以如果Servlet类中没有实现doGet方法，访问时可能出现错误。因此设计人员如果想同时支持这两种操作，则可以将Post操作委托给Get方法，如下所示：



1. Servlet中的重定向和请求分派：

重定向：服务器向用户返回一个状态为301的响应，包含一个URL，用户浏览器再向此URL发出请求，原来的请求参数已经丢失。且浏览器的地址栏也发生了改变，变为了新的URL。服务器的任务仅仅是返回一个URL。

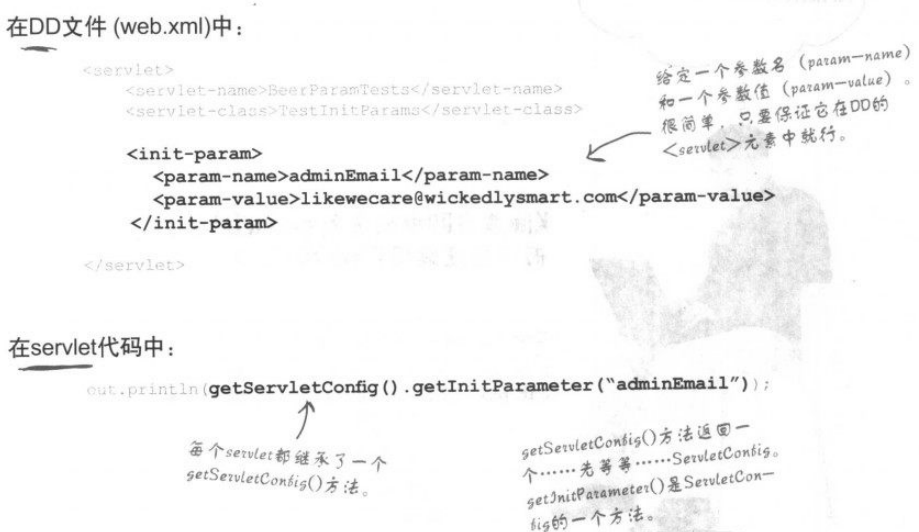


请求分派：由服务器完成，如果用户请求的资源不能直接获得，则它会将此请求转发到其他的JSP或Servlet来处理，并将结果返回给用户，用户不知道期间的变化，浏览器的地址栏显示的仍是原来的地址。需要注意的是我们不能在将响应已经返回给用户后再坐请求转发，此时会出现IIlegialStateException异常。



**第五章：属性和监听器**

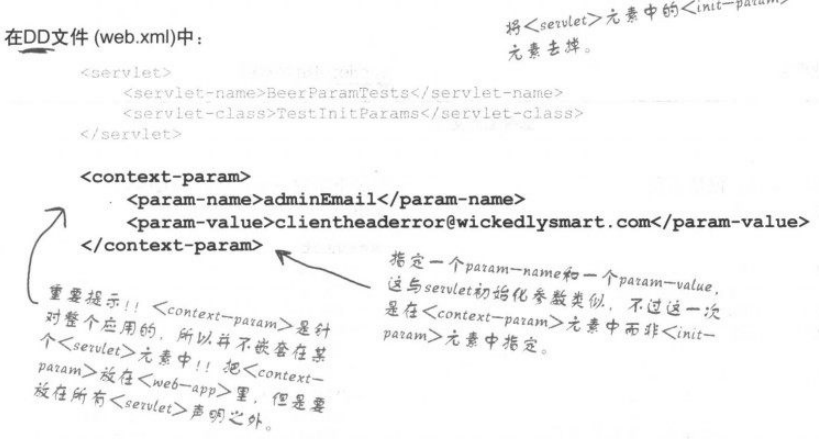
1．Servlet可以在部署文件web.xml中设置参数，在程序中通过servletConfig获取，不需要把一些变量配置硬编码到程序中。



注意DD中设置的变量只能在本Servlet中可用，一个Servlet实例有一个ServletConfig实例。如果需要在其他JSP页面中访问设置的变量，需要将此变量存储在request中(request.setAttribute)，通过请求分派，在其他的JSP中使用。

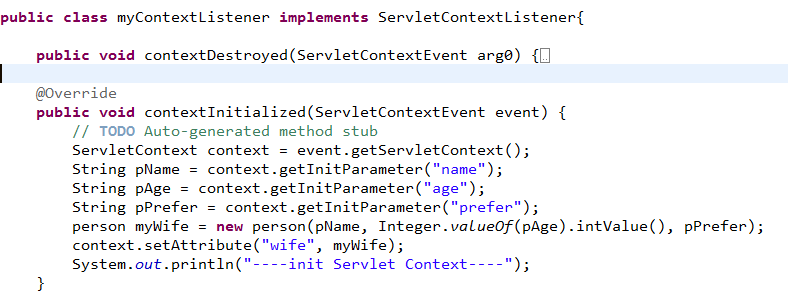
2. ServletContext—上下文变量

在DD中配置如下，它在整个Web应用中有一个实例，可以被所有的Servlet和JSP访问



3.我们可以定义一个监听器，当容器实例化一个ServletContext时，告知此监听器，完成一些初始化工作，如根据上下文变量创建一个全局的数据库连接对象，或者一些其他的资源。

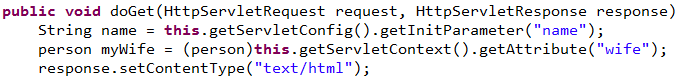
步骤：1）创建一个实现接口ServletContextListener的类：实现初始化和释放资源时调用的两个函数，如下所示。我们可以根据变量创建对象，并将对象存为context的属性。



2）在web.xml中配置节点Listener：

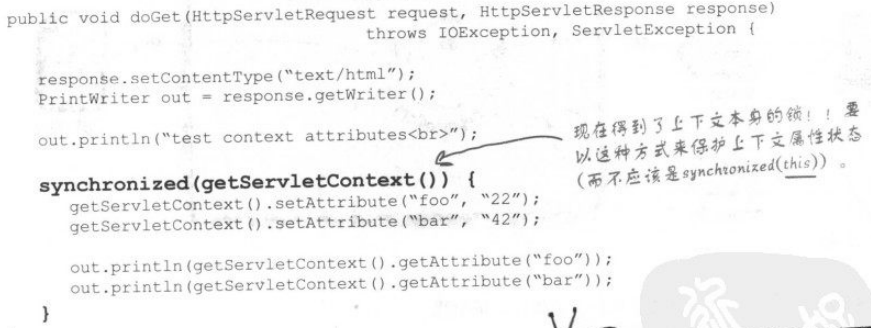


3）在所有的servlet或JSP中访问实例创建就的全局对象。



**注意：**当然我们不止可以监听上下文实例的创建，还可以监听上下文属性的添加、删除、更改，监听会话与请求等。可以参考帮助文档。

4. 属性主要有三个作用域：上下文，会话，请求。但这些属性并非是线程安全的，因为针对用户的每次请求，容器都会产生一个单独的线程为其服务，这样每个线程都有可能同时操作上下文或会话属性，导致有些请求可能获得脏数据。因此我们编程时为了做到线程安全，需要对访问上下文实例或会话实例的代码进行同步。



只有请求属性和局部变量时线程安全的。

**第六章：会话管理**

1. 客户浏览器与服务器建立连接，发出请求，得到响应，然后关闭连接，但服务器不会持久保存连接，也即服务器认不出来做第二个请求的客户与前一个请求的客户是否是同一个客户。因此如果需要标示一个客户，我们需要一个ID，此ID是当用户第一次向服务器发出请求时，容器产生一个唯一的会话ID返回给客户，在以后的交互中，客户需要将此ID作为请求的一部分发给容器，容器通过此ID来识别会话。

A:此会话ID可以通过使用Cooki来保存，且所有的任务包括生成会话ID，创建新的cookie对象，把会话ID放到cookie中，把cookie设置为响应的一部分等等工作将由容器负责，客户只需要告诉容器想创建或使用一个会话即可。



常用的session方法有getSession(Boolean), 传入false时如果会话未创建，则返回null.

Session.isNew() 判断一个会话是刚刚创建。



B. 当用户禁用cookie时，可以使用URL重写完成会话ID的传递。



由于会话ID是动态产生的，所以静态ＨＴＭＬ中不能硬编码URL重写，必须在响应HTML中动态生成URL。

注意：cookie不止可以传递会话ID，还可用于客户和服务器间交换少量的数据，如用户名和密码，且可以在客户端保存时间很长，像浏览器提供的保存密码相一致。使用时主要涉及三个实例：HttpServletRequest, HttpServletResponse, Cookie

2. 删除会话：如果服务器中维护的会话过多，那些已经被用户终止或意外终止的会话则仍然占用系统资源。容器需要删除那些无用的会话。

我们可以为会话设置一个最大超时时间，超过的话容器进行资源的回收。可以参考HttpSession提供的方法（

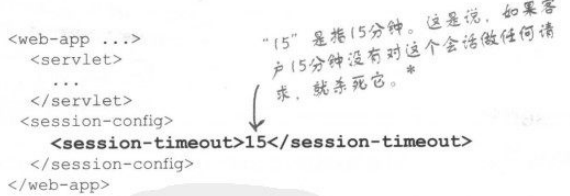
|  |  |
| --- | --- |
| long | [**getCreationTime**](mk:@MSITStore:E:\3--JavaEE\Jsp--Servlet\Java+Servlet帮助文档.chm::/jsdk23/javax/servlet/http/HttpSession.html#getCreationTime())() |
| void | [**setMaxInactiveInterval**](mk:@MSITStore:E:\3--JavaEE\Jsp--Servlet\Java+Servlet帮助文档.chm::/jsdk23/javax/servlet/http/HttpSession.html#setMaxInactiveInterval(int))**(int interval)** |
| long | [**getLastAccessedTime**](mk:@MSITStore:E:\3--JavaEE\Jsp--Servlet\Java+Servlet帮助文档.chm::/jsdk23/javax/servlet/http/HttpSession.html#getLastAccessedTime())**()** |
| void | [**invalidate**](mk:@MSITStore:E:\3--JavaEE\Jsp--Servlet\Java+Servlet帮助文档.chm::/jsdk23/javax/servlet/http/HttpSession.html#invalidate())**()  Invalidates this session then unbinds any objects bound to it.** |

）

会话死亡的三种情况：

超时；用户在会话对象上调用invalidate()； 程序崩溃

我们可以通过在DD中配置所有会话的超时时间；



3. 在分布式Web应用中，同一个servlet可能分布在不同的VM上，此时对于对象ServletContext, ServletConfig, HttpSession会如何处理？

HttpSession:会从一个VM移动到另一个VM中，每个Web应用的一个给定会话ID，只有一个httpSession对象，而不论应用分布在多少个VM上。

每一个VM有一个ServletContext， 每个VM上每个servlet有一个ServletConfig。

4. 会话常用的监听器：

1）HttpSessionListener: 监听会话的创建于销毁

2）HttpSessionAttributeListener: 跟踪向会话中增加，删除，替换属性

3）HttpSessionBindingListener：属性自身监听会话对它自身的一些操作

4）HttpSessionActivatinListener: HttpSessions迁移时可能用到的方法

