1. c2.a3.d4.d5.b6.b7.c8.d9.b10.b11.ac12.ae13.d14.ae15ab

1.javaEE container

**Java EE Containers**

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent Java EE architecture makes Java EE applications easy to write because business logic is organized into reusable components. In addition, the Java EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

**Container Services**

**Containers** are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container.

The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself. Container settings customize the underlying support provided by the Java EE server, including services such as security, transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity. Here are some of the highlights:

通常, 瘦客户端多层应用程序很难编写, 因为它们涉及许多复杂的代码行来处理事务和状态管理、多线程、资源池和其他复杂的低级细节。基于组件和独立于平台的 javaee 体系结构使 java ee 应用程序易于编写, 因为业务逻辑被组织成可重用的组件。此外, javaee 服务器以容器的形式为每个组件类型提供基础服务。因为您不必自己开发这些服务, 所以您可以自由地集中精力解决手头的业务问题。

集装箱服务

容器是组件和支持组件的低级平台特定功能之间的接口。在 web、企业 bean 或应用程序客户端组件可以执行之前, 必须将其组装到 Java EE 模块中并部署到其容器中。

程序集过程包括为 java ee 应用程序中的每个组件以及 java ee 应用程序本身指定容器设置。容器设置自定义 javaee 服务器提供的底层支持, 包括诸如安全性、事务管理、java 命名和目录接口 (JNDI) 查找和远程连接等服务。以下是一些亮点:

* The Java EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.The javaee 安全模型允许您配置 web 组件或企业 bean, 以便仅由授权用户访问系统资源
* **The Java EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.**
* JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access these services.
* The Java EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.The javaee 远程连接模型管理客户端和企业 bean 之间的低级通信。创建企业 bean 后, 客户端将对其调用方法, 就像在同一虚拟机中一样。

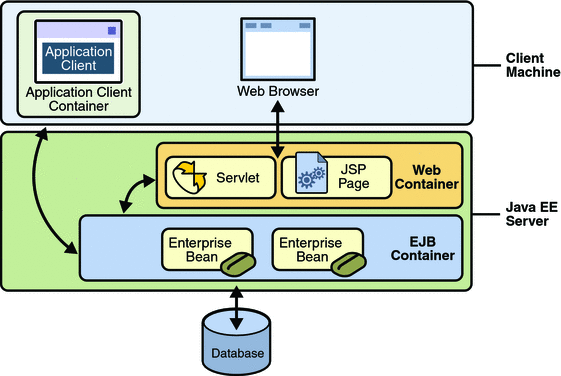
Because the Java EE architecture provides configurable services, application components within the same Java EE application can behave differently based on where they are deployed. For example, an enterprise bean can have security settings that allow it a certain level of access to database data in one production environment and another level of database access in another production environment.由于 javaee 体系结构提供了可配置的服务, 因此在同一 java ee 应用程序中的应用程序组件可以根据部署位置的不同而有所不同。例如, 企业 bean 可以具有安全设置, 使其能够在一个生产环境中对数据库数据进行一定程度的访问, 并允许在另一个生产环境中进行其他级别的数据库访问。

The container also manages nonconfigurable services such as enterprise bean and servlet life cycles, database connection resource pooling, data persistence, and access to the Java EE platform APIs (see [Java EE 5 APIs](https://docs.oracle.com/javaee/5/tutorial/doc/bnacj.html)).

**Container Types**

The deployment process installs Java EE application components in the Java EE containers as illustrated in [Figure 1-5](https://docs.oracle.com/javaee/5/tutorial/doc/bnabo.html" \l "bnabr).

**Figure 1-5 Java EE Server and Containers**



* **Java EE server**: The runtime portion of a Java EE product. **A Java EE server provides EJB and web containers.java ee的运行部分**
* **Enterprise JavaBeans (EJB) container**: Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
* **Web container**: Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.
* **Application client container**: Manages the execution of application client components. Application clients and their container run on the client.
* **Applet container**: Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.applet 容器: 管理小程序的执行。由在客户端上运行的 Web 浏览器和 Java 插件组成。

选择题2A

在Servlet中，利用RequestDispatcher对象，可以将请求转发给另外一个Servlet或JSP页面，甚至是HTML页面，来处理对请求的响应。

**一，RequestDispatcher接口方法简介**

     1，RequestDispatcher对象由Servlet容器来创建，封装一个由路径所标识的服务器资源。

     2，RequestDispatcher接口中定义了二种方法用于请求转发：

**forward（ServletRequest，ServletResponse）方法：**

                   将请求转发给服务器上另外一个Servlet，JSP页面，或者HTML文件      这个方法必须在响应被提交给客户端之前调用，否则抛出异常。

                  方法调用后在响应中的没有提交的内容被自动消除。

**include（ServletRequest，ServletResponse）方法 ：**

                 用于在响应中包含其他资源(Servlet，JSP页面或HTML文件)的内容。

                 即请求转发后，原先的Servlet还可以继续输出响应信息，转发到的Servlet对请求做出的响应将并入原先Servlet的响应对象中。

      3，forward方法和include方法的区别：

                  forward方法调用后在响应中的没有提交的内容被自动消除。   include方法使原先的Servlet和转发到的Servlet都可以输出响应信息。

**二，得到RequestDispatcher对象**

**三种方法可以得到RequestDispatcher对象：**

             1，利用ServletRequest接口中的getRequestDispatcher（String  path）方法。

             2，ServletContext接口中getNamedDispatcher（String  path）和getRequestDispatcher（String  path）方法。

**ServletRequest接口和ServletContext接口中getRequestDispatcher方法区别**：

              1，参数的区别

                      参数虽然都是资源路径名，ServletContext接口的中参数路径必须以“/”开始，是相对于当前Servlet上下文根，    ServletRequest接口中的参数路径不仅可以相对于当前Servlet上下文根，还可以相对与当前Servlet路径

             2，跨WEB应用程序访问资源

                      通过ServletContext.getContext（）方法获取另个WEB应用程序的上下文环境对象来    调用getRequestDispatcher（String  path）方法 将请求转发到另个WEB应用程序的资源。

                     还需要在当前WEB应用程序配置中设置<context>元素，指定crossContext属性值为true。

RequestDispatcher rd = request.getRequestDispatcher(“/seeclet/DataServlet”);

rd.include(request,response);

**三，ServletResqonse接口中的sendReadirect()方法和forward()方法的区别**

       二个方法都是用于请求转发的方法，转发给另外的资源为客户端服务。但二者有本质的区别

**sendReadirect()方法原理：**

              1，客户端发送请求，Servlet1做出处理。

              2，Servlet1调用sendReadirect()方法，将客户端的请求 **重新定位** 到Servlet2。

              3，客户端浏览器访问Servlet2.

              4，Servlet2对客户端浏览器做出响应。

**forward()方法原理：**

             1，客户端发送请求，Servlet1做出处理。

             2，Servlet1调用sendReadirect()方法，将请求转发给Servlet2来处理请求，为客户端服务。

             3，Servlet2对客户端浏览器做出响应。

**区别：**

**1，定位与转发**

                    sendReadirect()方法是重新定位到另外一个资源来处理请求，URL会重新定位，让客户端重新访问另外一个资源。    forward()方法是转发到另外一个资源来处理请求。URL不会变化。隐藏了处理对象的变化。

**2，处理请求的资源的范围**

                  sendReadirect()方法可以跨WEB应用程序和服务器重新定位资源来处理请求。    forward()方法只能在应用程序内部转发。

选择题3

Page指令向容器提供与当前页面相关的指令说明。可以在JSP页面的任何地方使用Page指令。按照惯例，通常写在JSP文件的开头。  
  
Page指令的基础语法：

<%@ page attribute="value" %>

|  |  |
| --- | --- |
| errorPage | 指定当JSP页面发生异常时需要转向的错误处理页面 |
| isErrorPage | 指定当前页面是否可以作为另一个JSP页面的错误处理页面 |
| session | 指定JSP页面是否使用session |

**session属性**

session属性指明JSP页面是否使用HTTP会话。若值为true，则意味着JSP页面可以访问内嵌的session对象，若值为false则不能访问。

**errorPage属性**

errorPage属性告诉JSP引擎如果当前页面运行出错需要显示的另一个页面。errorPage属性的值通常为URL。  
  
接下来的指令在抛出一个未捕获的异常时将会显示MyErrorPage.jsp页面：

<%@ page errorPage="MyErrorPage.jsp" %>

**isErrorPage属性**

isErrorPage属性指明当前页面是否是错误页面。isErrorPage属性的默认值是false。  
  
举例来说，handleError.jsp将isErrorPage输出设为了true，因为它是处理错误用的：

<%@ page isErrorPage="true" %>

选择题4

**JSP Session**

HTTP是无状态协议，这意味着每次客户端检索网页时，都要单独打开一个服务器连接，因此服务器不会记录下先前客户端请求的任何信息。

有三种方法来维持客户端与服务器的会话：

**Cookies**

网络服务器可以指定一个唯一的session ID作为cookie来代表每个客户端，用来识别这个客户端接下来的请求。

这可能不是一种有效的方式，因为很多时候浏览器并不一定支持cookie，所以我们不建议使用这种方法来维持会话。

**隐藏表单域**

一个网络服务器可以发送一个隐藏的HTML表单域和一个唯一的session ID，就像下面这样：

<input type="hidden" name="sessionid" value="12345">

这个条目意味着，当表单被提交时，指定的名称和值将会自动包含在GET或POST数据中。每当浏览器发送一个请求，session\_id的值就可以用来保存不同浏览器的轨迹。

这种方式可能是一种有效的方式，但点击<A HREF>标签中的超链接时不会产生表单提交事件，因此隐藏表单域也不支持通用会话跟踪。

**重写URL**

您可以在每个URL后面添加一些额外的数据来区分会话，服务器能够根据这些数据来关联session标识符。

举例来说，http://w3cschool.cc/file.htm;sessionid=12345， session标识符为sessionid=12345，服务器可以用这个数据来识别客户端。

相比而言，重写URL是更好的方式来，就算浏览器不支持cookies也能工作，但缺点是您必须为每个URL动态指定session ID，就算这是个简单的HTML页面。

**session对象**

除了以上几种方法外，JSP利用servlet提供的HttpSession接口来识别一个用户，存储这个用户的所有访问信息。

默认情况下，JSP允许会话跟踪，一个新的HttpSession对象将会自动地为新的客户端实例化。禁止会话跟踪需要显式地关掉它，通过将page指令中session属性值设为false来实现，就像下面这样：

<%@ page session="false" %>

JSP引擎将隐含的session对象暴露给开发者。由于提供了session对象，开发者就可以方便地存储或检索数据。

|  |  |
| --- | --- |
| 1 | **public Object getAttribute(String name)**  返回session对象中与指定名称绑定的对象，如果不存在则返回null |
| 2 | **public Enumeration getAttributeNames()**  返回session对象中所有的对象名称 |
| 3 | **public long getCreationTime()**  返回session对象被创建的时间， 以毫秒为单位，从1970年1月1号凌晨开始算起 |
| 4 | **public String getId()**  返回session对象的ID |
| 5 | **public long getLastAccessedTime()**  返回客户端最后访问的时间，以毫秒为单位，从1970年1月1号凌晨开始算起 |
| 6 | **public int getMaxInactiveInterval()**  返回最大时间间隔，以秒为单位，servlet 容器将会在这段时间内保持会话打开 |
| 7 | **public void invalidate()**  将session无效化，解绑任何与该session绑定的对象 |
| 8 | **public boolean isNew()**  返回是否为一个新的客户端，或者客户端是否拒绝加入session |
| 9 | **public void removeAttribute(String name)**  移除session中指定名称的对象 |
| 10 | **public void setAttribute(String name, Object value)**  使用指定的名称和值来产生一个对象并绑定到session中 |
| 11 | **public void setMaxInactiveInterval(int interval)**  用来指定时间，以秒为单位，servlet容器将会在这段时间内保持会话有效 |

%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ page import="java.io.\*,java.util.\*" %>

<%

// 获取session创建时间

Date createTime = new Date(session.getCreationTime());

// 获取最后访问页面的时间

Date lastAccessTime = new Date(session.getLastAccessedTime());

String title = "再次访问菜鸟教程实例";

Integer visitCount = new Integer(0);

String visitCountKey = new String("visitCount");

String userIDKey = new String("userID");

String userID = new String("ABCD");

// 检测网页是否由新的访问用户

if (session.isNew()){

title = "访问菜鸟教程实例";

session.setAttribute(userIDKey, userID);

session.setAttribute(visitCountKey, visitCount);

} else {

     visitCount = (Integer)session.getAttribute(visitCountKey);

     visitCount += 1;

     userID = (String)session.getAttribute(userIDKey);

     session.setAttribute(visitCountKey, visitCount);

}

HttpSession session = req.getSession();

PrintWriter out = resp.getWriter();

int count = 1;

Integer i = (Integer) session.getAttribute(COUNTER);

if(i != null){

count = i.intValue() +1;

}

session.setAttribute(COUNTER, new Integer(count));

%>

<html>

<head>

<title>Session 跟踪</title>

</head>

<body>

<h1>Session 跟踪</h1>

<table border="1" align="center">

<tr bgcolor="#949494">

<th>Session 信息</th>

<th>值</th>

</tr>

<tr>

<td>id</td>

<td><% out.print( session.getId()); %></td>

</tr>

<tr>

<td>创建时间</td>

<td><% out.print(createTime); %></td>

</tr>

<tr>

<td>最后访问时间</td>

<td><% out.print(lastAccessTime); %></td>

</tr>

<tr>

<td>用户 ID</td>

<td><% out.print(userID); %></td>

</tr>

<tr>

<td>访问次数</td>

<td><% out.print(visitCount); %></td>

</tr>

</table>

</body>

</html>

选择题5

选择题6

可维护性

Maintainability—"The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment

软件系统或组件可以轻松修改以纠正错误，提高性能或其他属性，或适应变化的环境

可管理性

Manageability—How efficiently and easily a software system can be monitored and maintained to keep the system performing, secure, and running smoothly.

 可以监控和维护软件系统的效率和方便程度，从而保持系统的正常运行，安全和正常运行。

 maintainability is when adding features, fixing bugs, or making enhancements can be easily done. Manageability on the other hand is the ability to monitor and keep a secure system when its up.

可维护性是在添加功能，修复错误或增强功能时可以轻松完成的。而另一方面，可管理性则是能够监视并保持安全的系统。

选择题7

在Servlet中懂得ServletContext HttpSession 以及HttpServletRequest之间的关系

② 懂得怎样使用它们

概念介绍：

1. [共同点]不管对象的作用域怎样，共享变量和获得变量的 方法都是一致的  
–setAttribute(“varName”,obj);  
–getAttribute（“varName”）。  
2. 变量的作用域  
ServletContext–范围最大。应用程序级别的，整个应用程序都能訪问  
HttpSession   – 次之，会话级别的，在当前的浏览器中都能訪问[不论是在同一浏览器开多少窗体，都能够訪问]。可是换个浏览器就不行了。就必须又一次创建session  
HttpServletRequest  –范围最小，请求级别，请求结束，变量的作用域也结束[也就是仅仅是一次訪问，訪问结束，这个也结束]



监听器就是一个实现特定接口的普通java程序，这个程序专门用于监听另一个java对象的方法调用或属性改变，当被监听对象发生上述事件后，监听器某个方法将立即被执行。   
servlet的监听器，我们可以继承他们完成各种各种的功能，对于那些框架中也是有各种可以使用的监听器，实质是不变的，就是进行了封装而已。   
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 ServletContext, 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。   
• 监听三个域对象创建和销毁的事件监听器   
• 监听域对象中属性的增加和删除的事件监听器   
• 监听绑定到 HttpSession域中的某个对象的状态的事件监听器。（查看API文档）

监听servletContext域对象创建和销毁   
ServletContextListener 接口用于监听 ServletContext对象的创建和销毁事件。   
当 ServletContext对象被创建时，激发contextInitialized (ServletContextEvent sce)方法   
当ServletContext 对象被销毁时，激发contextDestroyed(ServletContextEvent sce)方法。

监听HttpSession域对象创建和销毁   
HttpSessionListener接口用于监听HttpSession创建和销毁   
创建一个Session时，sessionCreated(HttpSessionEventse) 方法将会被调用。 销毁一个Session时，sessionDestroyed(HttpSessionEvent se) 方法将会被调用。Session域对象创建和销毁的时机创建：用户每一次访问时，服务器创建session   
销毁：如果用户的session30分钟没有使用，服务器就会销毁session，我们在web.xml里面也可以配置session失效时间

监听HttpRequest域对象创建和销毁   
ServletRequestListener 接口用于监听ServletRequest 对象的创建和销毁。   
Request 对象被创建时，监听器的requestInitialized方法将会被调用。   
Request对象被销毁时，监听器的requestDestroyed方法将会被调用。   
创建：用户每一次访问，都会创建一个reqeust   
销毁：当前访问结束，request对象就会销毁

监听三个域对象属性变化   
Servlet规范定义了监听ServletContext, HttpSession, HttpServletRequest 这三个对象中的属性变更信息事件的监听器。   
这三个监听器接口分别是ServletContextAttributeListener,HttpSessionAttributeListener ServletRequestAttributeListener   
这三个接口中都定义了三个方法来处理被监听对象中的属性的增加，删除和替换的事件，同一个事件在这三个接口中对应的方法名称完全相同，只是接受的参数类型不同   
attributeAdded 方法当监听器的域对象中的某个属性被增加时，web容器调用事件监听器的这个方法进行相应   
attributeRemoved 方法当监听器的域对象中的某个属性被删除时，web容器调用事件监听器的这个方法进行相应   
attributeReplaced 方法 当监听器的域对象中的某个属性被替换时，web容器调用事件监听器的这个方法进行相应   
public void attributeRmoved(ServletRequestAttributeEvent srae)   
public void attributeRemoved (HttpSessionBindingEvent hsbe)   
public void attributeRemoved(ServletContextAttributeEvent scae)

选择题8

选择题9

容器的配置：

Property

选择题10

Choosing between HttpSession and Stateful session bean

Both Servlet's HttpSession object and EJB's Stateful session bean are meant to maintain client state, so which one is a better option? let's look into the advantages and disadvantages of both the mechanisms :

1 . Stateful session bean

Advantages :

It supports transaction service ,security service, life cycle management, RMI, instance cache, thread safe etc. You need not write code for these services.

It can be used by both web based and non web based clients (like swing etc.) .

It can be used for multiple operations for a single http request.

Disadvantages :

Since it supports a number of services mentioned above it takes more time to process a request.

2. HttpSession object

Advantages:

It is a simple java object (perhaps a Hashtable) and takes very less time and resources to maintain a client state

Disadvantages:

It does not support the features discussed above

It cannot process multiple operations for a single http request.

So depending on your application's requirement you can choose the one best suited for you, if you want the bean only for client state management then HttpSession object gives better performance than Stateful session bean.

选择题11

**getParameter** 是用来接受用post和get方法传递过来的参数的.  
**getAttribute**必须先setAttribute.  
**（1）request.getParameter()**取得是通过容器的实现来取得通过类似post，get等方式传入的数据，request.setAttribute()和getAttribute()只是在web容器内部流转，仅仅是请求处理阶段。  
**（2）request.getParameter()**方法传递的数据，会从Web客户端传到Web服务器端，代表HTTP请求数据。request.getParameter()方法返回String类型的数据。  
**request.setAttribute()**和 **getAttribute()**方法传递的数据只会存在于Web容器内部  
还有一点就是，HttpServletRequest 类有 setAttribute() 方法，而没有setParameter() 方法。  
拿一个例子来说一下吧，假如两个WEB页面间为链接关系时，就是说要从1.jsp链接到2.jsp时，被链接的是2.jsp可以通过getParameter()方法来获得请求参数.  
假如1.jsp里有

**Html代码  [收藏代码](javascript:void())**

1. **<form** name="form1" method="post" action="2.jsp"**>**
2. 请输入用户姓名：**<input** type="text" name="username"**>**
3. **<input** type="submit" name="Submit" value="提交"**>**
4. **</form>**

的话在2.jsp中通过request.getParameter("username")方法来获得请求参数username:

**Html代码  [收藏代码](javascript:void())**

1. **<** % String username=request.getParameter("username"); %**>**

但是如果两个WEB间为转发关系时，转发目的WEB可以用getAttribute()方法来和转发源WEB共享request范围内的数据，也还是说一个例子吧。  
  
有1.jsp和2.jsp  
  
1.jsp希望向2.jsp传递当前的用户名字，如何传递这一数据呢？先在1.jsp中调用如下setAttribute()方法：

**Html代码  [收藏代码](javascript:void())**

1. **<**%
2. String username=request.getParameter("username");
3. request.setAttribute("username"，username);
4. %**>**
6. **<jsp:forward** page="2.jsp" **/>**

在2.jsp中通过getAttribute()方法获得用户名字:

**Html代码  [收藏代码](javascript:void())**

1. **<**% String username=(String)request.getAttribute("username"); %**>**

**（1）HttpServletRequest**类有setAttribute()方法，而没有setParameter()方法  
**（2）**当两个Web组件之间为链接关系时，被链接的组件通过getParameter()方法来获得请求参数，  
**（3）**当两个Web组件之间为转发关系时，转发目标组件通过getAttribute()方法来和转发源组件共享request范围内的数据。  
一般通过表单和链接传递的参数使用getParameter  
  
通过request.setAttribute("name","jerry")的方式赋值的使用request.getAttribute("name")  
  
这个问题主要是request和session的差别，request范围较小一些，只是一个请求，简单说就是你在页面上的一个操作， request.getParameter()就是从上一个页面中的url、form中获取参数，但如果一个request涉及多个类，后面还要取参数，可以用request.setAttribute()和request.getAttribute()，但是当结果输出之后，request就结束了。  
  
而session可以跨越很多页面，可以理解是客户端同一个IE窗口发出的多个请求。这之间都可以传递参数，比如很多网站的用户登录都用到了。  
  
一般可以用getParameter得到页面参数。。。字符串。。。  
  
getAttribute()可以得到对象。。。  
  
getParameter可以得到页面传来的参数如？id=123之类的。  
  
getAttribute()常用于servlet页面传递参数给jsp

request.getParameterValues(String name)是获得如checkbox类（名字相同，但值有多个）的数据。 接收数组变量，如checkobx类型  
request.getParameter（String name）是获得相应名的数据，如果有重复的名，则返回第一个的值. 接收一般变量，如text类型  
  
例：  
  
1. 三个名字相同的text标签  
  
<input type="text" name="test" value="1" />  
  
<input type="text" name="test" value="2" />  
  
<input type="text" name="test" value="3" />  
  
request.getParameterValues("test") ----------------- [1, 2, 3]  
  
request.getParameter("test") ------------------ 1

选择题12

javax.servlet.ServletContext接口  
  
一个servlet上下文是servlet引擎提供用来服务于Web应用的接口。Servlet上下文具有名字（它属于Web应用的名字）唯一映射到文件系统的一个目录。  
一个servlet可以通过ServletConfig对象的getServletContext()方法得到servlet上下文的引用，如果servlet直接或间接调用子类GenericServlet，则可以使用getServletContext()方法。  
Web应用中servlet可以使用servlet上下文得到：  
1.在调用期间保存和检索属性的功能，并与其他servlet共享这些属性。  
2.读取Web应用中文件内容和其他静态资源的功能。  
3.互相发送请求的方式。  
4.记录错误和信息化消息的功能。  
  
   
ServletContext接口中的方法         
Object getAttribute(String name)    返回servlet上下文中具有指定名字的对象，或使用已指定名捆绑一个对象。从Web应用的标准观点看，这样的对象是全局对象，因为它们可以被同一servlet在另一时刻访问。或上下文中任意其他servlet访问。         
void setAttribute(String name,Object obj)    设置servlet上下文中具有指定名字的对象。         
Enumeration getAttributeNames()    返回保存在servlet上下文中所有属性名字的枚举。         
ServletContext getContext(String uripath)    返回映射到另一URL的servlet上下文。在同一服务器中URL必须是以“/”开头的绝对路径。         
String getInitParameter(String name)    返回指定上下文范围的初始化参数值。此方法与ServletConfig方法名称不一样，后者只应用于已编码的指定servlet。此方法应用于上下文中所有的参数。         
Enumeration getInitParameterNames()    返回（可能为空）指定上下文范围的初始化参数值名字的枚举值。         
int getMajorVersion()    返回此上下文中支持servlet API级别的最大和最小版本号。         
int getMinorVersion()             
String getMimeType(String fileName)    返回指定文件名的MIME类型。典型情况是基于文件扩展名，而不是文件本身的内容（它可以不必存在）。如果MIME类型未知，可以返回null。         
RequestDispatcher getNameDispatcher(String name)    返回具有指定名字或路径的servlet或JSP的RequestDispatcher。如果不能创建RequestDispatch，返回null。如果指定路径，必须心“/”开头，并且是相对于servlet上下文的顶部。         
RequestDispatcher getNameDispatcher(String path)             
String getRealPath(String path)    给定一个URI，返回文件系统中URI对应的绝对路径。如果不能进行映射，返回null。         
URL getResource(String path)    返回相对于servlet上下文或读取URL的输入流的指定绝对路径相对应的URL，如果资源不存在则返回null。         
InputStream getResourceAsStream(String path)             
String getServerInfo()    返顺servlet引擎的名称和版本号。         
void log(String message)  
void log(String message,Throwable t)    将一个消息写入servlet注册，如果给出Throwable参数，则包含栈轨迹。         
void removeAttribute(String name)    从servlet上下文中删除指定属性。

[getServletContext()和getServletConfig()的意思](http://tag.csdn.net/Article/aacf9ff0-39af-4b2c-9f5c-97ea3b1ff8e4.html)

getServletConfig() 在servlet初始化时，容器传递进来一个ServletConfig对象并保存在servlet实例中，该对象允许访问两项内容：初始化参数和ServletContext对象，前者通常由容器在文件中指定，允许在运行时向sevrlet传递有关调度信息，比如说getServletConfig().getInitParameter("debug")后者为servlet提供有关容器的信息。

[getServletContext()和getServletConfig()的意思](http://tag.csdn.net/Article/c44889cb-2950-4155-841c-53a67863b36f.html)

getServletContext()和getServletConfig()的意思2007-07-09 11:10.getServletContext() 一个servlet可以使用getServletContext（）方法得到web应用的servletContext 即而使用getServletContext的一些方法来获得一些值 比如说getServletContext().getRealPath("/")来获得系统绝对路径 getServletContext().getResource("WEB-INF/config.xml")来获得xml文件的内容。

选择题13

编译器指示 <%@ 编译器指示 %>   
声明 <%! 声明 %>   
表达式 <%= 表达式 %>   
程序代码段/小型指令 <% 程序代码片段 %>   
注释 <%-- 注释 --%>   
例如<%="hello"%>就会在网页上对应的位置显示hello  
  
而<% 里面的是java代码，和普通java类的语法一样，而在两个符号外面的都作为html处理 %>  
  
<% for (int i=1; i<=4; i++) { %>   
<H<%=i%>>Hello</H<%=i%>>   
<% } %>  
相当于[html代码](https://www.baidu.com/s?wd=html%E4%BB%A3%E7%A0%81&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y4uj7Buyfsn16kP1KBPvFW0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3En1csnjDdn1cz" \t "_blank)：  
<H1>Hello</H1><H2>Hello</H2><H3>Hello</H3><H4>Hello</H4>

<%=表达式%>是输出表达式的，用来页面上显示变量的值的

选择题14

JSP 隐式对象是Web容器加载的一组类的实例。它是可以直接在JSP页面使用的对象。分为4个主要类别：

    · **输入和输出对象**：控制页面的输入和输出（request、response、out）。

    · **作用域通信对象**：检索与JSP页面的Servlet相关的信息（session、application、pageContext）。

    ·**Servlet对象：**提供有关页面环境的信息（page、config）。

    · **错误对象：**处理页面中的错误（exception）。

**1、输入和输出对象**

**1）request对象**

        request隐式对象表示客户端的请求，包含了所有的请求信息。

        · String getParameter(String name)：根据页面表单组件名称获取请求页面提交的数据。

        · String getParameterValues(String name)：获取页面请求中一个表单组件对应多个值时的用户的请求数据（例如复选框）。

**2）response对象**

        response隐式对象处理JSP生成的响应， 然后将响应结果发送给客户端。

        · void setContentType(String name)：设置作为响应生成的内容的类型和字符编码。

        · void sendRedirect(string name)：发送一个响应给浏览器，指示其应请求另一个URL（重定向到另外一个URL，会丢失数据，跳转后不再执行sendRedirect方法下面的代码）。

**3）out对象**

        out对象表示输出流，此输出流将作为请求的响应发送到客户端，常用的方法有print()、println()和write()。

**2、作用域通信对象**

    JSP作用域通信对象从小到大有pageContext对象、session对象、application对象。

**1）pageContext对象**

    pageContext对象使用户可以访问当前页面作用域中定义的所有隐式对象。pageContext对象最常用的方法有：

    · void setAttribute(String  name, Object value)：以名称/值的方式，将一个对象的值存放到pageContext中（存的值的类型为Object）。

    · void getAttribute(String name)：根据名称去获取pageContext存放对象的值（获取的值的类型为Object）。

**2）session对象**

    session对象表示用户的会话状况，用此项机制可以轻易识别每一个用户，能保存和跟踪用户的会话状态。session对象最常用的方法有：

    · void setAttribute(String  name, Object value)：以名称/值的方式，将一个对象的值存放到session中（存的值的类型为Object）。

    · void getAttribute(String name)：根据名称去获取session存放对象的值（获取的值的类型为Object）。

**3)application对象**

    application对象作用于整个应用程序，所有的客户端窗口都可以共享该对象，从服务器开始就存在，直到服务器关闭为止。application对象最常用的方法有：

    · void setAttribute(String  name, Object value)：以名称/值的方式，将一个对象的值存放到application中（存的值的类型为Object）。

    · void getAttribute(String name)：根据名称去获取application存放对象的值（获取的值的类型为Object）。

**3、Servlet对象**

**1）page对象**

    page对象提供对网页上定义的所有对象的访问。page对象表示页面本身，它是java.lang.Object类的一个实例。

**2）config对象**

    config对象存储Servlet的一些初始信息。config对象是javax.servlet.ServletConfig接口的一个实例，ServletConfig接口提供方法以检索Servlet初始化参数。config对象表示编译JSP页面的Servlet初始化数据的配置。

**4、错误对象exception**

    exception对象处理JSP页面中的错误。printStackTrace()方法用于显示异常的堆栈跟踪。exception.getClass().getName()获取异常信息。

选择题15

static variable也称为class variable，因为它属于类所有。static variable也是多线程编程时产生问题的主要来源。

non-static variable 称为 instance variable，实例变量

local variable，局部变量，只在某个域中有效

静态工具类，是指不用实例化就可使用的工具类， 就其方法及方法内的局部变量来说，本身就是线程安全， 要注意的就是万一工具类里的静态变量是需要线程安全的， 就要考虑了，因为静态变量是共用的； 还有一种就是方法传入的参数变量是否需要线程安全，但是一般这种情况都是在调用方法前就进行线程安全了，所以很少用考虑

|  |  |  |
| --- | --- | --- |
| **Variable/Attribute scope** | **Thread Safe in Multi-Threaded Model** | **Thread Safe in Single-Threaded Model** |
| Local Variables | Yes | Yes |
| Instance Variables | No | Yes |
| Class Variables | No | No |
| Request Attributes | Yes | Yes |
| Session Attributes | No | No |
| Context Attributes | No | No |

註：  
Instance Variables 只在 Single-Threaded Model thread-safe，原因在於 Multi-Threaded Model 讓每個 request 在同一 servlet 實體叫起一個 service() thread 同時跑，而 Single-Threaded Model 是叫起不同的 servlet 實體來跑  
Class Variables 一定不是 thread-safe，因為在 Single-Threaded Model，雖然是叫起不同的 servlet 實體，但是更改的還是同一個值，所以會有 concurrency 問題  
Request Attribute 一定是 Thread Safe，因為他的 scope 就是在那 method 裡，不可能在中間被別的 method 裡被更動  
Session 一定不是 Thread Safe，原因在於使用者可能另開視窗，造成多個視窗同一 session 的問題(同browser)  
Context Attributes 一定不是 Thread Safe，因為它就是那個 Web App 的一個共同記憶體，都可能會被更改