Servlet 第一天　2007 年 6 月 21 日

一、简述

1、Servlet，服务器端的小程序，它是相对于 Applet 而言的，Applet 是客户端小程序。

Servlet，是接受来自网络的请求（form 表单，以及其他的请求），并对不同请求作出不同的响应

Servlet，是基于 Http 协议的，是运行在 web 服务器中的程序。这里要提出一个容器的概念。

servlet 是运行在 web 容器中，在后面会详细地讲解，这个 web 容器可以控制 Servlet 对象的生命周期，控制请求由 Servlet 对象处理。

2、web 服务器，这里的服务器不是硬件概念，而是软件，常用的 web 服务器有 Tomcat，Jboss 等，我们所用到的 Tomcat 是一个开源的服务器，

Tomcat 是一个用 java 语言编写的 web 服务器，所以需要有相应的 java 运行环境，也就是 JVM，还要配置 tomcat 的具体路径。

二、Tomcat 的配置

JAVA_HOME=/XXX/XXX/（JDK 路径，bin 的上一层目录）
CATALINA_HOME=/XXXX/XXX（tomcat 的绝对路径 windows 中 X:\xxx\xxx）

在启动 Tomcat 时，是运行 Tomcat 的 bin 目录下的 startup.sh（windows 中使用 startup.bat）
判断 Tomcat 是否启动成功，可以在浏览器的地址栏中使用 http://localhost:8080/或 http://127.0.0.1:8080/可以访问到 tomcat 的主页就是启动成功了。
要想停止 tomcat 服务器要使用 shutdown.sh（windows 中使用 shutdown.bat），如果直接关闭启动窗口，就会造成 8080 端口占用错误，这时可以在使用 shutdown.sh 关闭一下服务器。

startup.sh，shutdown.sh，shutdown.bat，startup.bat 这些文件其实是一些脚本文件用来执行大量的命令，也就是大量 java 命令。

tomcat 的默认监听端口是 8080 端口，当接受到每一个连接请求，就会为其分配一个线程。

tomcat 可以识别的资源只有在 webapps 文件夹下，webapps 也就是 web 应用文件夹，webapps 下的文件夹这些文件夹中存放的就是 web 应用，web 应用是有格式规范的，每个 web 应用的文件夹下都要有 WEB-INF 文件夹，WEB-INF 文件夹下有 classes，和 lib 文件夹，以及一个 web.xml 文件，一些使用到的类文件放在 classes 中，一些使用到的相应的 jar 文件。

注意：使用完有限的资源要进行释放。
tomcat 中配置了 root 缺省应用，也就是在不指定的情况下会默认访问这个应用。

web.xml 文件的写法

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
```

"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

</web-app>

在 tomcat 服务器中，访问应用下的资源可以在端口号后架上 web 应用文件夹得名字就可以看到资源 http://localhost:8080/xxxxx/xxxx.html，静态页面只能放在 web 应用的文件夹下，不能够放在 WEB-INF 文件夹下，WEB-INF 文件夹中的资源是受保护的，不能够通过网络访问到。

三、Servlet 基础

Servlet，可以实现动态的页面，可以针对不同的请求作出不同的响应，可以实现页面的流转，Servlet 可以充当 MVC 模式中的 Ctrl 模块，他可以控制信息的流向。
web 服务器会 web 应用在 WEB-INF 文件夹下的 classes 文件夹搜索要加载的 class 文件，所以我们写的 class 文件要放在 web 应用中的 WEB-INF 文件夹下的 classes 文件夹下。

设置 servlet 的类和访问的方式

web.xml 文件的配置，一个 web.xml 中可以配置多个 Servlet
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
<servlet>
<servlet-name>servlet 的名字</servlet-name>
<servlet-class>servlet 类全名</servlet-class>
</servlet>
<servlet>
<servlet-name>servlet 的名字 1</servlet-name>
<servlet-class>servlet 类全名 1</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>servlet 的名字（要和 servlet 标签中的相同）</servlet-name>
<url-pattern>指定 servlet 相对于应用目录的路径</url-pattern>
</servlet-mapping>
servlet-mapping>
<servlet-name>servlet 的名字 1</servlet-name>
<url-pattern>指定 servlet 相对于应用目录的路径</url-pattern>
</servlet-mapping>
</web-app>

catalina.sh run 带控制台启动 tomcat 服务器。

四、Servlet 的调用过程

1，用户通过浏览器向 web 服务器发送请求
http://serverip:port/apppath
2，服务器为用户定位资源
静态资源：/a.html /a/b.html （这里的路径是针对 web 应用文件夹目录）读文件并把内容发送到客户端
动态资源：解析 web.xml 定位 Servlet 类的名字
装载类（WEB-INF/classes|WEB-INF/lib/*.jar）
创建该对象的实例
Servlet ser=(Servlet)(Class.forName("servle 的类名")).newInstance();
//我们写的 Servlet 一定要实现 Servlet 接口或者继承实现了 Servlet 接口的类
ser.service(request,response);

<servlet-mapping>
<servlet-name>servlet 的名字（要和 servlet 标签中的相同）</servlet-name>
<url-pattern>指定 servlet 相对于应用目录的路径</url-pattern>
</servlet-mapping>
url-parttern 的配置，这个 url 就是 Servlet 的虚拟路径，可以使用相对路径或者是绝对路径。
/xxx/xxx（绝对路径） ,xxx（相对路径），尽量使用绝对路径。
访问 servlet 的方法
http://serverip:port/应用文件夹名/url-pattern

http 的请求
get 请求，post 请求。
get 请求在发出请求时会把参数写在地址栏上，而 post 请求不会把要发送的参数显示在地址栏上。

<form method="get" action="应用名/url-pattern">
……
</form>

我们可以通过 ServletRequest 对象的方法来获取请求中传送的参数。
getParameter(String name)方法，可以获得 form 表单中指定名字的参数，多参数同名时，只取一个。
getParameterNames()，可以获得一个迭代器 Enumeration，通过这个迭代器，来获得 form 表单中参数的名字。
getParameterValues(String name)获得指定的所有同名参数的值。

Servlet 第二天 2007 年 6 月 22 日

一、复习
servlet 接口的实现类中的 service()方法，在继承 HttpServlet 类时，如果没有覆盖父类的 service()方法，那么父类的 service()方法会根据请求类型不同的会分别调用覆盖的 doGet()，doPost()方法，如果响应两种请求的动作相同，那么可以直接覆盖 service()方法。如果覆盖了 doGet()，doPost()方法之一，那么就会只对一种请求作出相应。在浏览

器的地址栏操作按回车键，或者是热连接，都是 get 请求，form 的 method 属性如果不指定，默认为 get 请求。

我们可以通过 ServletRequest 对象或 HttpServletRequest 对象的方法来获取请求中传送的参数。
getParameter(String name)方法，可以获得 form 表单中指定名字的参数，多参数同名时，只取一个。
getParameterNames()，可以获得一个迭代器 Enumeration，通过这个迭代器，来获得 form 表单中参数的名字。
getParameterValues(String name)获得指定的所有同名参数的值。

get 请求，会将参数显示在浏览器的地址栏上，其显示格式，在地址之后会以问号开始，以'&'分隔参数，可以通过 HttpServletRequest 对象的 getQueryString()方法来获得 get 请求的参数值。

ServletRequest 对象的 getInputStream()方法可以获得一个由 Socket 得来的输入流，可以使用这个流来实现文件的上传。getReader()方法可以直接获取 post 请求的参数。

注意：getParameter()与 getReader()方法不能同时调用 ServletContext 对象是 Servlet 的上下文对象，这个对象是在服务器启动时创建的，他可以看作是一个应用的对象，他可以看作是包含 Servlet，管理 Servlet 的对象。

二、servlet 的生命周期

遵守 servlet 规范的类，就可以通过服务器产生对象（反射机制），并处理请求。

servlet 接口中的方法
所有的 servlet 都必须实现 javax.servlet.Servlet 接口

```
public class TestServlet implements Servlet {
    ServletConfig config;
    public void init(ServletConfig config) throws ServletException {
        this.config=config;
        //这个 ServletConfig 对象是由服务器生成，也就是有系统提供的，通过他可以获得启动信息。ServletConfig 对象和 Servlet 是一一对应的。
        //这个方法是在 Servlet 创建后调用的。如果要是用到 ServletConfig 对象是一定要为对象赋值。
    }

    public ServletConfig getServletConfig() {
        return this.config;
    }

    public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException {
        ...;//这个方法是用来处理请求的核心方法
    }

    public String getServletInfo() {
        return "....";//这个是用来写 Servlet 信息的，用来写作者，以及版本信息
    }
```

```
    public void destroy() {
        ...;//这个方法是用来销毁 Servlet 对象的
    }

}
```

HttpServlet 和 GenericServlet 都实现了 Servlet 接口。
HttpServlet 中的 service(HttpServletRequest request，HttpServletResponse response)方法是通过 service(ServletRequest request,ServletResponse response)方法的调用来实现对请求的处理。

1、Servlet 的生命周期分为四个阶段
（1）创建 Servlet 对象，通过服务器反射机制创建 Servlet 对象，第一次请求时才会创建。（默认）
（2）调用 Servlet 对象的 init()方法，初始化 Servlet 的信息，init()方法只会在创建后被调用一次
（3）响应请求，调用 service()或者是 doGet()，doPost()方法来处理请求，这些方法是运行的在多线程状态下的。
（4）在长时间没有被调用或者是服务器关闭时，会调用 destroy()方法来销毁 Servlet 对象。

2、servlet 创建时机：
（1）第一个请求到来的时候创建
（2）在 web.xml 中<servlet>标签下添加<load-on-startup>2</load-on-startup>，则在服务器启动的时候就创建
标签体是正数的时候，启动服务器，即创建 servlet
并且当多个<servlet>标签都定义了这个时，数字小的先加载

3、初始化参数
可以通过<init-param>标签来配置初始化参数，可以用 ServletConfig 对象的
getInitParameter(String name)方法来得到参数。

```
<init-param>
<param-name>...</param-name>
<param-value>...</param-value>
</init-param>
```

三、多线程下的操作
多线程下所操作的变量，如果操作的是一个变量，且兼有读写操作，就要考虑加上同步，但同步不能乱加，否则会造成死锁问题。
init()和 destroy()方法都是运行在单线程下的。

四、分布式模型
把系统部署在不同的地址空间执行
在有限的硬件条件下，服务更多的用户
在 Servlet 中可以访问，JDBC，RMI（远程方法调用），以及跨语言平台的组件等资源。
在 Servlet 中是用 JDBC 很容易，也就是在 Servlet 中调用 JDBC 中的方法，就可以实现对数据库的访问

五、有参和无参的 init 方法。

```
public void init(ServletConfig config)throws ServletException
{
        this.config = config;
        init();
}

public void init()throws ServletException
{
   .....//覆盖了的无参的 init()方法，会在 Servlet 创建，调用有参的 init 方法时
        也会被调用。
}
```

先覆盖 init（），有父类调 init（config）
init（）方法可抛异常，初始化成功才进入 service（）方法中

六、war 文件
扩展名为“.war”的文件放在 tomcat 的 webapps 下，当服务器启动的时候，自动把文件解压开
注意：文件名字，就是将来访问时应用的名字

Servlet 第三天　　2007 年 6 月 25 日

一、连接池
在应用的 META-INF 文件夹下 context.xml 文件中

```
<Context>
    <Resource
        name="jdbc/oracle"  配置 JDNI 的名字
        type="javax.sql.DataSource"  绑定资源的类型
        password="sd0605"
        driverClassName="oracle.jdbc.driver.OracleDriver"  驱动名
        maxIdle="1"最大连接数
        maxWait="-1"等待时间,配置为-1 就是无限等待,只到有空闲连接为止
        username="sd0605"
        url="jdbc:oracle:thin:@192.168.0.39:1521:TARENADB"
        maxActive="3"  最大活动连接数/>
</Context>
```

以下是从连接池去取数据库连接的代码

```
public static  Connection getConnection(String JNDIName){
```

```
        Connection conn = null;
        try{
            Context initCtx = new InitialContext();
            Context envCtx = (Context) initCtx.lookup("java:comp/env"); //这个是在 tomcat 下默认绑定 Context 的
JNDIname            DataSource ds = (DataSource) envCtx.lookup(JNDIName);          conn = ds.getConnection();
        } catch (NamingException ne){
            ne.printStackTrace();
        } catch (SQLException se){
            se.printStackTrace();
        }
        return conn;
}
```

二、Java EE 编程分层

表现层，业务层，数据层。

表现层，也就算用来显示数据，接受数据的。JSP，Servlet

业务层，是处理核心业务的程序 EJB，JDBC（Hibernate）

数据层，也就是数据库，用来存放数据。Oracle，SQLServer

MVC 框架

Model，模型层，这一层一般是进行数据库访问，并且封装对象，这一层中也存放在访问数据库取出信息封装成对象的类，也就是实体类的信息，可以使用 JDBC 或者 Hibernate 实现这一层的功能。

Ctrl，控制层，用来相应请求和调用写好的相应的访问数据库的方法，这一层是用来控制请求的响应的，现在我们是使用 Servlet 来实现这一层，不过一般是会用开源的 MVC 框架来实现这层，例如 struts，或者是 Spring 的 MVC 框架。

View，表现曾，他只用来显示数据和收集必要数据，收集数据的一般是 form 表单，不过要保证数据的正确性要是用 JavaScript 验证信息，以后我们会学到的 JSP（java server page）就是用来表现、显示数据的。

三、Servlet 的控制流转

ServletContext，Servlet 上下文对象，在每个 Servlet 中都会有一个 ServletContext 的引用，这个 ServletContext 是一个全局的对象，每个应用中只有一个 ServletContext 对象。

HttpServlet 中的 getServletContext()方法，获得 ServletContext 对象。

ServletContext 类的 getRequestDispatcher(String path)方法获得一个 RequestDispatcher 对象，

并且跳转到指定的 Servlet，getRequestDispatcher(String path)方法中的参数就是 path，就是指定跳转的 Servlet 的 url-pattern。

RequestDispatcher 类的 forward(ServletRequest request, ServletResponse response) 方法，可以把请求对象转发给其他的 Servlet。

include 用在 v 层 forward 用在 c 层

```
ServletRequest.getRequestDispatcher();   //relative
ServletContext.getRequestDispatcher();   //absolute
```

一个 Servlet 对应一个 Config，不能互相读取

要想让所有的 Serlvet 都能读到参数：配置 Context 初始化参数，或者配置 JNDI Naming 的初始化参数

(1)配置 Context 容器的初始化参数

```
<context-param>
    <param-name>email</param-name>
    <param-vlaue>liyan@tarena.com.cn</param-vlaue>
</context-param>
```

在 servlet 的使用

```
getSevletContext().getInitParameter("email");
```

(2)配置 JNDI Naming 的初始化参数

```
<env-entry>
    <env-entry-name>email<env-entry-name>
    <env-entry-type>java.lang.String<env-entry-type>
    <env-entry-value>liyan@tarena.com.cn<env-entry-value>
</env-entry>
```

使用，得到初始化参数

```
Context ctx = new InitialContext();
String email = (String)ctx.lookup("java:comp/env/email");
```

四、会话

会话是可以保存状态的

Session（会话）和 Cookie（会话跟踪机制）

Session 对象用来解决客户端发送多个请求时来用户请求信息的存储问题，但是他和 ServletRequest 对象是不同的，他会在有需要时创建，但是他的生命周期会比请求对象要长。Session 对象的生命周期也是有限制的，如果长时间的没有访问，就会销毁掉 Session 对象，可以通过 Session 对象的 setAttribute(String name, Object o) 和 getAttribute(String name)来存取数据信息。Session 是用户级的对象。

```
public void service(ServletRequest request,ServletResponse response){

    String user = request.getParameter("user");
        String pass = request.getParameter("pass");
    HttpSession session = request.getSession(true);//使用请求对象来创建 Session
        session.setAttribute("username", user);
        session.setAttribute("passwd", pass);
}
```

getSession(true)就表示如果 Session 不存在就创建一个新的 Session，并把 Session 的标识 SessionID 写到 Cookie 中，如果存在就是用这个 Session。getSession(false)就是在 Session 不存在时不会创建新 Session 而是返回 null。如果使用 getSession()方法，就等同于 getSession(true)。

注意：ServletRequest 对象适用于传输大量的数据，因为其生命周期比较短，可以有效的节省内存资源。
大数据量的传输或保存不适合使用 Session 空间。

Cookie，是记录用户的 Session 信息，也可以记录用户的请求信息，也就是 SessionID，来分辨哪一个用户是否登陆过。在每次登陆时，还会将 Cookie 发送回服务器端，Cookie 是用来跟踪 Session 的。

```
public void service(ServletRequest request，ServletResponse response){

    String user = request.getParameter("user");
        String pass = request.getParameter("pass");
    Cookie userCookie = new Cookie("user", user);
        userCookie.setMaxAge(60 * 60 * 24 * 365);//设置 Cookie 的最大有效期,秒为单位
        Cookie passCookie = new Cookie("pass", pass);
        passCookie.setMaxAge(60 * 60 * 24 * 365);
        response.addCookie(userCookie);
        response.addCookie(passCookie);}
```

Servlet 第四天　　2007 年 6 月 26 日

一、复习
1、连接池
2、MVC
3、cookie

二、会话 session
Session 是基于 Cookie 来跟踪的，即：没有 Cookies 的支持，Session 是不能运行起来的。

Session 对象用来解决客户端发送多个请求时来用户请求信息的存储问题，但是他和 ServletRequest 对象是不同的，他会在有需要时创建，但是他的生命周期会比请求对象要长。Session 对象的生命周期也是有限制的，如果长时间的没有访问，就会销毁掉 Session 对象，可以通过 Session 对象的 setAttribute(String name, Object o) 和 getAttribute(String name)来存取数据信息。Session 是用户级的对象。

Session 是存在于服务器内存中的，用于存用户多个请求的信息的；同时也要求客户端发送个 Session 的标志：SessionID (地址栏或封装在请求的 Header 中)。

getSession(true)就表示如果 Session 不存在就创建一个新的 Session，并把 Session 的标识 SessionID 写到 Cookie 中，如果存在就是用这个 Session。getSession(false)就是在 Session 不存在时不会创建新 Session 而是返回 null。如果使用 getSession()方法，就等同于 getSession(true)。

注意：ServletRequest 对象适用于传输大量的数据，因为其生命周期比较短，可以有效的节省内存资源。
大数据量的传输或保存不适合使用 Session 空间。
用户身份认证登录时，创建 session
访问资源页面时，先判断 session 是否存在
退出时清除 session：session. invalidate()，接着可以将用户引导到登录页面

IE 中一个窗口代表一个会话，Mozilla firefox 不一样
多个窗口可通过 Cookies 来识别 Seesion。

Sssion 第二种跟踪机制：URLRewriting
Response.sendRediret(Response.encodeRedirectURL(/serv-app/student/ctrl“))
把在地址栏后加上 SessionID 地址地址参数
out.println(”<a href=”+response.encodeURL(url)”
<form actioin=”response.encodeURL(url)”>
Forward 也是 encodeURL
特殊一个：Response.sendRedirect(response.encodeRedirectURL(url));
每一个 URL 都要加上 sessionID，但它不能跨越静态页面。
所以一般默认针对 Cookies 可用编写程序。

1、session.invalidate()　--- session 被立即销毁
　session.setMaxInactiveInterval(int interval)　--- 设置最大的超时时间，以秒为单位
2、会话的空间不是系统自动创建的，是程序创建的
　request.getSession(false);　判断请求中是否存在 session，以确保服务中资源的保护

三、重定向
产生新的请求，其中不包含原请求的 session 信息，只包含 cookie 信息

四、Session 跟踪机制
1、cookie 机制
2、URL 回写机制：把 sessionid 绑定在地址栏中
　response.sendRedirect(response.encodeRedirectURL(url));

五、Servlet 过滤器（Filter）
应用：Sessioin logging encoding
过滤器是用于过滤 Servlet 的请求和响应，过滤器是存在于请求和被请求资源之间的。
过滤器就像当于一个中间件，请求要经过过滤器，然后过滤器才去掉用 Servlet，Servlet 的响应也会被过滤器截获并作相应的处理。

Filter 是一个接口，要写一个自己的 Filter 就只能实现 Filter 接口。
Filter 也有自己的生命周期，他的生命周期和 Servlet 比较相似，也是会先调用 init()方法，然后再调用核心的处理过滤的方法 doFilter()，这个方法中可定义了过滤规则，然后是 destory()方法销毁 Filter 对象。

dofilter(ServletRequest request,ServletResponse response,FilterChain chain)
这个是过滤的核心方法，FilterChain 的方法 doFilter(ServletRequest request, ServletResponse response)也就是用过滤后的请求调用资源的方法，如果不写这个方法，也就算不会去调用相应的资源。

Filter 的配置

Filter 的配置和 Servlet 相似。

```
<filter>
    <filter-name>SessionFilter</filter-name>
    <filter-class>alan.filter.SessionFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>SessionFilter</filter-name>
    <url-pattern>/protected/*</url-pattern>
    <!--这里的 url-pattern 就是要过滤的 Servlet 的 url-pattern-->
    <dispatcher>request</dispatcher>
    <dispatcher>forward</dispatcher>
    <dispatcher>include</dispatcher>
    <!--上面的三个是过滤的范围-->
</filter-mapping>
```

CharArrayWriter 类，是一个将数据保存在字符数组中的输出流，我们可以使用它来构造一个 PrintWriter 对象，也就实现了向内存输出。
CharArrayWriter 类的 toString()和 toCharArray()方法就可以取得写入内存中的数据。
注意：CharArrayWriter 类是一个不会真正输出的类，他的 write()方法只会将内容写入字符数组，而且这个字符数组是会自动增长的。

六、Servlet 的 url-pattern

url-pattern 可以使用以下三种方式
1，确切路径匹配，也就是给出确定的路径 xxx/xxxx
2，模糊路径匹配，也就是指给出一部分路径，xxxx/*，他会匹配确定路径，也就是 xxxx/a 或者是 xxxx/b 都是可以匹配的
3，扩展名匹配，也就是会匹配扩展名，只要是扩展名相同就匹配，xxx.xxx *.xx

注意：扩展名匹配和确切路径匹配不能放在一起使用，也就是不能写成 xxxx/xxxx/xxx.xx，但是可以用 *.xxx。

ServletRequest 对象的三个返回路径的方法
getContextPath()获得应用的路径，用动态获取应用路径
getServletPath()获得 Servlet 路径，也就是 form 中的 action，如果使用确切路径那么就会是这个 Servlet 配置的 url-pattern。
getPathInfo()使用模糊路径匹配时会返回匹配模糊部分。

注意：在 html 的 form 表单的 action 中，如果使用了扩展名匹配，一定要写明/xxxxx/xxx.xx，不要写成/xxxx/*.xx，在 form 的 action 中要尽量使用绝对路径，也就是要用 应用名/xxx.xx 或者应用名/xxx。

SingleThreadModel 接口
1) 如果希望禁止多线程访问，可以让 Servlet 使用 SingleThreadModel 接口:
   public class YourServlet extends HttpServlet implements SingleThreadModel{ ... }
2) 使用此接口，系统将保证不会存在多个请求线程同时访问 Servlet 的单个实例。但是仍然需要同步对存储在

Servlet 外部的类变量或共享字段的访问。
3) 如 Servlet 频繁被访问，则 Servlet 访问的同步将严重影响性能(延时)。

ServletRequest 对象的生命周期就是在 service()方法中，除了 forward(..,..)方法将这个请求对象转发给其他的 Servlet。

Servlet 第五天　　2007 年 6 月 27 日

一、监听器
以下是 3 个 Listener 接口。
ServletRequestListener
HttpSessionListener
ServletContextListener
这三个监听器接口，分别监听 Servlet 中 3 种比较中要的对象的创建和销毁。这三个接口中分别有监听该对象创建和销毁事件的方法，服务器本身就是事件源。

listener 的配置

```
<listener>
    <listener-class>alan.servlet.listener.AlanContextListener</listener-class>
        <!--listener-class 也就是实现 Listener 接口的类-->
</listener>
```

Servlet 中的重要对象（只针对应用）

	数量	生命周期	是否线程安全	方法	
ServletContext	1	（天）全局的 只有在应用关闭时才销毁	不安全 需要加同步访问	setAttribute(String,Object) Object getAttribute(String) void removeAttibute(String)	全局属性 任何 Session 都可以取到
HttpSession	和用户的数量相同	（分/小时）局部的 只在有效时间内存在 synchronized(session){.....}	不安全 可不加同步	setAttribute(String,Object) Object getAttribute(String) void removeAttibute(String)	存储用户级 的属性
ServletRequest (HttpServletRequest)	多个	（秒）局部的 只在 servic()和 doGet() doPost()中存在	线程安全	setAttribute(String,Object) Object getAttribute(String) void removeAttibute(String)	可以传递 大量信息 只使用一次

二、作用
1、主要对 ServletContext、HttpSession、ServletRequest 等对象行为的监听
2、必须要实现一个或多个 listener interface

3、有两类监听器：
声明周期监听器　ServletContextListener　HttpSessionListener　ServletRequestListener　（2.4 版本之后）
对象状态监听器　ServletContextAttributeListener　HttpSessionAttributeListener
ervletRequestAttributeListener
4、ServletContextListener 中不能取得请求中的参数

Servlet 的基础概念

1. Servlet 是什么?

答：1) 模块化的程序，运行在服务器端，增强了请求/响应导向服务;

2) 应用示例：

a. 访问远端对象;

b. 跟踪大量信息;

c. 多用户协作

2. HTTP 和 Servlets

答：1) Servlet 是 HTTP 协议中作为 CGI 的一个替代品;

2) HttpServlet 类用于开发 HTTP 为基础的 Servlet

3. HttpServlet

答：1) 继承抽象类 javax.servlet.GenericServlet，实现接口 java.io.Serializable;

2) 用以开发 Http 协议为基础的 Servlet

4. 服务方法

答：1) 每当服务接收到对 Servlet 的请求时，服务器就会产生一个新线程，并调用 Service。service 方法检查 HTTP 请求类型(GET、POST、PUT、

DELETE 等),并相应地调用 doGet、doPost、doPut、doDelete 等。

2) doGet/doPost 方法接收 HttpServletRequest 和 HttpServletResponse 对象。

3) 99%的时间里，只需注意 GET 和/或 POST 请求;

4) 没有任何 doHead 方法。

5. 返回响应

答：1) PrintWriter out = response.getWriter // 用于返回文本数据给客户端

2) ServletOutputStream out = response.getOutputStream // 用于返回二进制数据给客户端

6. 支持 Servlet 的 Web 服务器

答：1) J2EE 应用服务器包括：Web Container 和 EJB Container;

2) Web Container 的 Servlet Engine 提供对 Servlet 的运行支持;

用 Servlet 处理表单数据及 Servlet 的生命周期(上)

用 Servlet 处理表单数据

1. FORM 元素的属性

答：1) ACTION: 用来指定要处理 FORM 数据的 Servlet 的 URL，也可以指定 FORM 数据将要发送到的电子邮件;

2) METHOD: 指定数据传送给 HTTP 服务器的方法;

3) ENCTYPE: 指定数据在传输之前进行编码的方式, 例 multipart/FORM-data 编码将每个字段作为 MIME 可兼容的文档的单独部分传输。

2. 解析请求

答：1) 对于所有的请求：

a. getParameterNames: 以 Enumeration 形式获取表单中清单, 每一项都可以转换成 String;

b. getParameter: 返回表单中参数名(区分大小写)对应的值(没有这样的参数，返回 null;没有任何值，返回空 String);

c. getParametervalues: 返回表单中参数名(区分大小写)对应的字符串数组(没有这样的参数，返回 null;只有一个值，返回值为单一元素组);

Servlet 的生命周期

1. Servlet 的生命周期

答：1) 通过 web Container 装载(J2EE 的组件都是被动地装载入 Container)并实例化 Servlet 对象;

2) 调用 init()方法(在整个生命周期中只被调用一次);

3) 调用 service()方法(在整个生命周期中可被调用多次);

4) 调用 destroy()方法(在整个生命周期中只被调用一次);

2. init 方法

答：1) 当首次创建 Servlet 时就会调用 init 方法, 而不是每个用户请求都会调用该方法。

2) 除非被 destroy 方法移除，否则不能被重载;

3) init 方法一结束，servlet 即可接受客户端请求;

3. init 方法实例

答：1) 在编写接受 ServletConfig 作为参数的 init 方法时，应该总是在首行调用 super.init;

2) init 方法接受 ServletConfig 作为参数, 用以下方法获得参数值：

a. getInitParameter: 返回指定参数名称对应的值，如果参数不存在，返回 null;

b. getInitParameterNames: 返回指定参数名称对应的值枚举，如果参数不存在，返回的空枚举;

Servlet 的生命周期(下)

3. service 方法

答：1) 每当服务器接收到对 Servlet 的请求时，服务器就会产生一个新线程, 并调用 service。service 方法检查 HTTP 请求类型，请相应地调用 doGet、doPost、doPut、doDelete。

2) 被 container 调用去响应(ServletResponse)来自客户端的请求(ServletRequest);

4. Servlets 的多线程安全

答：1) 多线程占用资源少，处理速度快，提高了效率。

2) 一些编码建议：

a. 对变量和方法定义适当的访问方式, 例如单纯取值操作不会有多线程安全问题;

b. 同步化所有访问重要数据的实例变量;

c. 创建访问类变量的访问方法。

5. SingleThreadModel 接口

答：1) 如果希望禁止多线程访问，可以让 Servlet 使用 SingleThreadModel 接口:

```
public class YourServlet extends HttpServlet implements SingleThreadModel{
...
}
```

2) 使用此接口，系统将保证不会存在多个请求线程同时访问 Servlet 的单个实例。但是仍然需要同步对存储在 Servlet 外部的类变量或共享字段的访问。

3) 如 Servlet 频繁被访问，则 Servlet 访问的同步将严重影响性能(延时)。

6. destroy 方法

答：1) 服务器决定删除已经加载的 Servlet 实例之前将调用 Servlet 的 destroy 方法;

2) 该方法允许 Servlet:

a. 关闭数据库连接;

b. 中止后台线程;

c. 将 Cookie 程序清单或访问计数写到磁盘以及执行其他类似的收尾工作。

7. 在 Servlet 终止时处理 Service 线程

答：1) 在 destroy()方法中：如有服务(通过一个同步化的实例方法取得当前线程数大于 0)，则置关闭状态为 true(通过一个同步化的实例方法实现)。然后循环等待服务线程数为 0.

2) 在 Service()方法中: 如见关闭状态为 true，便不执行具体逻辑方法，直接退出。

资源访问

1. 分布式 JAVA 技术

答：1) JDBC;

a. 实现了 Data 和 Client 的分开;

b. 通过简单的配置可以适用不同种类的数据库。

2) RMI(RMI 使用的协议为 Internet Inter ORB Protocol);

3) CORBA(核心技术为 ORB：相应的你的请求转为另一个物理地址另一个不同语言对象的请求。纯 Java 的情况下根本不用 CORBA);

2. 转发结果至可视页面

答：1) 用 JavaBean(用来装载一组值，遵从一定协议的 class)封装结果;

2) 每个 JVM 中的每一个应用程序里都存在一个上下文;

3) servletContext 在 servletConfig 的对象中;

4) ServletContext.getRequestDispatcher(String path)：返回一个 RequestDispatcher

5) 通过 RequestDispatcher 的 forward()或 include()方法传送请求。

3. 转发请求至新的资源

答：1) request dispatcher 的二种传送请求方式

a. Forward: 将请求从一个 servlet 传到服务器上的其他资源(servlet、JSP、HTML);

b. Include: 包括静态或动态内容;

2) 获得 request dispatcher 的二种方式：

a. ServletRequest.getRequestDispatcher()　　// 相对路径

b. ServletContext.getRequestDispatcher()　　// 绝对路径

3) 四种资源范围

a. javax.servlet.ServletContext: 整个应用程序范围内;

b. javax.servlet.http.HttpSession: 会话期间;

c. javax.servlet.ServletRequest: 一个请求期间;

d. javax.servlet.jsp.PageContext: 一个 JSP 页面;

过滤器

1. 什么是过滤器?

答：与 Servlet 相似，过滤器是一些 Web 应用程序组件，可以绑定到一个 Web 应用程序档案中。但是与其他 Web 应用程序组件不同的是，过滤器是"链"在容器的处理过程中的。这就意味着它们会在 servlet 处理器之前访问一个进入的请求，并且在外发的响应信息返回到客户前访问这些响应信息。这种访问使得过滤器可以检查并修改请求和响应的内容。

2. 过滤器可以用于：

答：1) 为一个 Web 应用程序的新功能建立原型(可被添加到 Web 应用程序中或者从 Web 应用程序中删除而不需重写基层应用程序代码);

2) 向过去的代码中添加新功能。

3. 过滤器放在容器结构什么位置?

答：过滤器放在 Web 资源之前，可以在请求抵达它所应用的 Web 资源(可以是一个 servlet、一个 JSP 页面，甚至是一个 HTML 页面这样的静态内容)之前截获进入的请求，并且在它返回到客户之前截获输出请求。

4. 过滤器的存活周期

答：过滤器有四个阶段(与 servlet 类似)：

1) 实例化;

2) 初始化(调用 init()方法);

3) 过滤(调用 doFilter()方法);

4) 销毁(调用 destroy()方法);

5. 过滤器类和接口

答：所有的过滤器都必须实现 javax.servlet.Filter 接口：

1) 容器调用 init()方法初始化过滤器实例：

```
public void init(FilterConfig config) throws ServletException
```

2) doFilter()方法包含过滤器逻辑：

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException
```

3) destroy()方法由容器在销毁过滤器实例之前调用：

```
public void destroy();
```

4) FilterChain 的 doFilter()方法之后的代码构成了后期处理过滤器调用。

6. 配置过滤器

答：使用<filter>和<filter-mapping>元素来配置：

```
<filter>
   <filter-name>XSLTFilter</filter-name>                    //过滤器名
   <filter-class>filters.SmartXSLFilter</filter-class>      //具体过滤器类
   <init-param>                                             //初始化参数
      <param-name>xsltfile</param-name>
      <param-value>/xsl/stockquotes.xsl</param-value>
   </init-param>
</filter>
<filter-mapping>                          //将过滤器应用于 Web 应用程序中的每个 Web 资源
   <filter-name>Logger</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Web 应用程序生命周期事件及监听器(Servlet V2.3 版本以后新增功能)

1. 什么是事件监听器?

答：1) 支持 ServletContext、HttpSession(since v2.3)及 ServletRequest(since v2.4)中状态改变的事件通知;

2) 实现了一个或多个 servlet 事件监听器接口的类型;

3) 控制 ServletContext、HttpSession(since v2.3)及 ServletRequest(since v2.4)中的生命周期;

2. Servlet Context 事件监听器

答：1) 对于应用程序而言在 JVM 层别管理资源或保存状态

2) 有二种类型的事件监听器：

a. ServletContextListener(以下是该监听器的方法)

contextDestroyed(ServletContextEvent sce)

contextInitialized(ServletContextEvent sce)

b. ServletContextAttributeListener(以下是该监听器的方法)

attributeAdded(ServletContextAttributeEvent scab)

attributeRemoved(ServletContextAttributeEvent scab)

attributeReplaced(ServletContextAttributeEvent scab)

3. HTTP Session 事件监听器

答：1) 管理从同一个客户端或用户向一个 Web 应用程序发出的一系列请求相关的状态或资源;

2) 有二种类型的事件监听器:

a. HttpSessionListener(以下是该监听器的方法)

sessionCreated(HttpSessionEvent se)

sessionDestroyed(HttpSessionEvent se)

b. HttpSessionAttributeListener(以下是该监听器的方法)

attributeAdded(HttpSessionBindingEvent se)

attributeRemoved(HttpSessionBindingEvent se)

attributeReplaced(HttpSessionBindingEvent se)

4. Servlet Requst 事件监听器

答：1) 管理整个 request 生命周期的状态

2) 有二种类型的事件监听器

a. ServletRequestListener(以下是该监听器的方法)

requestDestroyed(ServletRequestEvent sre)

requestInitialized(ServletRequestEvent sre)

b. ServletRequestAttributeListener(以下是该监听器的方法)

attributeAdded(ServletRequestAttributeEvent srae)

attributeRemoved(ServletRequestAttributeEvent srae)

attributeReplaced(ServletRequestAttributeEvent srae)

5. 监听器类的规定

答：1) 必须在部署描述符中配置实现类;