1.Servlet是相对于applet的，服务器端的小程序。

2.**Servlet的特点**

移植性好，本身是一个Java类，具有跨平台性；

Java EE平台支持的全部Java API都可用于Servlet；

安全性提高，服务器崩溃的可能性减小；

多个Servlet可以组织在一起，输出可由组生成，有助于代码复用；

可以与服务器中的其它组件交互。

3.GET和POST的区别：

|  |  |  |
| --- | --- | --- |
| 比较内容 | GET | POST |
| **是否在浏览器地址栏显示？** | **显示** | **不显示，保密性好** |
| **数据放置的位置** | **作为请求资源的一部分放置在请求行中** | **放置在请求内容(即请求体)中** |
| **传输数据量有无限制？** | **一般在1KB以下** | **无限制** |

GET方式的请求参数查看：在浏览器地址栏

POST方式的请求参数查看：在开发者模式：

4.调试使用Servlet的完整步骤：

(1)继承HttpServlet抽象类；

(2)覆盖HttpServlet的部分方法(如：doGet()、doPost())；

(3)获取Http请求信息；

(4)生成Http响应信息；

(5)配置映射路径；

(6)触发Servlet执行；

(7)跟踪断点调试。

5.在Eclipse中创建Servlet类，有两种方式：

A.创建Java类，通过继承HttpServlet抽象类；

B.向导模式，一步一步创建出来。

6.配置Servlet的方式有两种：

A.传统的Web.xml的方式；

B.新式的注解的方式。

7.传统的Web.xml的方式配置步骤：

A.在web.xml的配置文件中，先声明Servlet；

例一：

<servlet>

<servlet-name>ServletDemo</servlet-name> ----定义Servlet的名称，以便映射引用

<servlet-class>com.ddb.javaweb.servlet.ServletDemo</servlet-class> -----指定Servlet的类

<load-on-startup>1</load-on-startup> ----指定随容器启动而启动

</servlet>

例二：

<servlet>

<servlet-name>dbinit</servlet-name>

<jsp-file>/WEB-INF/initDemo.jsp</jsp-file> ----指定的是jsp文件

<init-param> ----配置Servlet的初始化参数

<param-name>driver</param-name>

<param-value>org.gjt.mm.mysql.Driver</param-value>

</init-param>

<init-param>

<param-name>url</param-name>

<param-value>jdbc:mysql://localhost:3306/abcd</param-value>

</init-param>

</servlet>

B.然后配置Servlet的映射路径URL

例一：

<servlet-mapping>

<servlet-name>ServletDemo</servlet-name>

<url-pattern>/servlet/ServletDemo</url-pattern>

</servlet-mapping>

例二：

<servlet-mapping>

<servlet-name>dbinit</servlet-name>

<url-pattern>/config.abcd</url-pattern>

</servlet-mapping>

8.注解配置步骤：

**注解使用的位置是：在Servlet类的上面。**

依据Eclipse向导，会自动生成相应的注解配置，例如：

@WebServlet(

value = { "/servlet/InitParameter", "/servlet/abc", "/servlet/cde" },

/\*urlPatterns = { "/servlet/InitParameter", "/servlet/abc", "/servlet/cde" },\*/

initParams = {

@WebInitParam(name = "institute", value = "职业培训学院"),

@WebInitParam(name = "major", value = "Java技术"),

@WebInitParam(name = "count", value = "10000")},

loadOnStartup=1)

若想知道@WebServlet注解支持哪些属性，有两种方法：

A.查看servlet-api-3.1的在线手册---<http://tomcat.apache.org/tomcat-8.0-doc/servletapi/>

B.利用Eclipse中的JD插件，在Eclipse源码中，按住ctrl键，同时将鼠标移动到@WebServlet上，单击，打开反编译源码，查看拥有的属性。

9.注解配置与XML的配置方式比较：----不做考试要求

A.两者基本上完全一样效果；

B.在注解配置中修改配置，Tomcat可以捕捉并自动加载，XML的配置修改必须重启Tomcat；

C.对于经常变动的参数，建议使用XML的配置。

10.配置servlet的注意事项：

A.同一个Servlet不要同时存在XML配置和注解配置，

若要同时存在则让URL不同，URL相同会导致tomcat服务器启动失败。

B.同一个Servlet可以配置多个URL的链接．

11.表单Form的action的URL为Servlet的链接时，可能会有404错误。通常在Servlet的链接前加：

<%=request.getContextPath()%> 下面的例子，可以参考：

action='<%=request.getContextPath()%>/servlet/MyLoginServlet2' 或

action='/JavaWebCore/servlet/MyLoginServlet2'

12.Servlet的生命周期：包括以下4个阶段：

加载和实例化

初始化：调用init()方法

请求处理：调用service()方法

服务终止：调用destroy()方法

13.Servlet容器负责加载和实例化，是否在容器启动时自动加载Servlet，这由在web.xml中设置的<load-on-startup>属性决定：

14.对于每一个Servlet实例来说，只初始化一次。GenericServlet提供了两种形式的init()方法**：**

public void init(ServletConfig config) throws ServletException {

this.config = config;

init();

}

public void init() throws ServletException {

}

对于用户自己的Servlet，可以重写init方法，通常使用带参数的来获取一些配置信息。

15.Servlet实例化后**接收客户端请求、作出响应，都是通过调用service()方法来实现的**。由于Servlet**采用多线程机制来提供服务，因此，该方法被同时、多次地调用。**每一个请求都调用自己的service()方法，但要注意线程安全。

  用户在**实现具体的Servlet时，一般不重载service()方法，**web容器在调用service()时，会根据请求方式的不同自动调用doGet()、doPost()、doPut()、doDelete()中的一种或几种，因此，**只要重载对应的doXxx()即可。**

16.服务器通过调用destroy方法释放Servlet运行时所占用的资源，web容器有可能崩溃或者暴力终止，destroy()方法不一定总被执行。

17.Servlet**在初始化时，web容器会从web.xml提取初始化参数和Servlet名称生成ServletConfig对象**，它还会创建ServletContext对象(运行时环境的信息)并存储到ServletConfig中。

初始化参数，可以是存在XML里面的配置，也可以是存在注解里面的配置。例如：

ServletConfig config=getServletConfig();

  String myInstitute=config.getInitParameter(“institute”);

  String mymajor=config.getInitParameter(“major”);

18.Servlet-API：

public class InitServletDemo extends HttpServlet

public abstract class HttpServlet extends GenericServlet

public abstract class GenericServlet implements Servlet, ServletConfig, Serializable

19.Tomcat的乱码问题：----根本原因，编码和解码方式不一致！Tomcat默认的字符集为iso-8859-1

A.页面内容乱码；

解决方法：让**响应对象在输出内容前，调用下列方法之一**

A1.：**setCharacterEncoding(“字符集”)**

例如：response.setCharacterEncoding(“UTF-8”);

A2.**setContentType(“文本类型;charset=字符集”)**

例如：response. setContentType(“text/html;charset=UTF-8”);

B.页面参数乱码。

解决方法：传递中文参数乱码的解决办法

B1.在获取请求对象内容前，**调用setCharacterEncoding(“字符集”)**

request.setCharacterEncoding(“UTF-8”);

B2.**创建新的字符串**，用tomcat的编码方式get，用新的解码方式解码。

String info = request.getParameter("information");

String newinfo=**new** String(info.getBytes("iso8859\_1"),"UTF-8");

20.Servlet的通信：

A.Servlet与浏览器之间的通信

B.Servlet之间以及Servlet与其它web组件之间的通信

21.Servlet与浏览器之间的通信：

**A.向浏览器发送错误消息**

HttpServletResponse定义了如下方法**：**

**void sendError(int sc)**

**void sendError(int sc,String msg)**

(其中：sc为出错状态码，msg为错误描述字符串)

**B.浏览器重定向**

当原URL永久移动(状态码为301)或临时移动(状态码为302)时，浏览器要定位到新URL。

有两种方法实现：

B1.利用HttpServletResponse的setStatus()和setHeader()

void setStatus(int sc)

void setHeader(String name,String value)

B2.利用HttpServletResponse的sendRedirect ()

void sendRedirect(String location)

相当于在地址栏中重新输入一个新URL，这个URL指向的位置没有限制。

22.Servlet之间以及Servlet与其它web组件之间的通信

Servlet之间的通信是通过“请求分派”(request dispatch)来实现的，这一过程包含两个步骤：

A.获得即将分派请求的web组件引用

ServletContext 接口中的RequestDispatcher getRequestDispatcher(String path)：

参数必须以“/”开始，表示相对于当前上下文根的路径

ServletContext 接口中的RequestDispatcher getNamedDispatcher(String name)：

参数是web.xml中servlet的命名

ServletRequest 接口中的RequestDispatcher getRequestDispatcher(String path)：

参数可以“/”开始，表示相对于当前上下文根的路径；

不以“/”开始，表示相对于当前Servlet的路径

B.分派请求----调用RequestDispatcher对象的两个

**void include(ServletRequest request,  ServletResponse response)：**

将请求转发给其它servlet,被调用servlet对请求作出响应将并入原先的响应对象中

**void forward(ServletRequest request,  ServletResponse response)：**

将请求转发给其它servlet,被调用servlet对请求作出响应，原先Servlet的执行被中止

23.forward与include方法的比较：

|  |  |  |
| --- | --- | --- |
| **比较内容** | **forward()** | **Include()** |
| 使用场合不同 | 将控制权转移到其它组件 | 须由另一组件执行部分处理，一旦执行完毕，当前组件将收回控制权 |
| 输出结果不同 | 在请求转发之前，web组件不应使用输出流向客户端发送消息 | 所有web组件共享同一个输出流与客户机进行通信 |
| 相同点 | 在地址栏中只显示原URL，不显示新组件的URL | |

forward与sendRedirect方法的比较：

|  |  |  |
| --- | --- | --- |
| **比较内容** | **forward()** | **sendRedirect()** |
| 执行方式不同 | 在web容器中运行 | 需要往返客户机 |
| 执行速度不同 | 快 | 慢 |
| 重定向的位置范围不同 | 只限制在同一个应用程序范围内。只显示原URL，不显示新组件的URL | 可重定向到任何URL上，显示的是新URL |

24.Servlet的线程安全问题：

Web容器**采用多线程模式**运行，它**为并发的每一个访问请求都准备了一个独立的线程来响应**，这种模式提高了访问性能，但**也可能带来线程的安全问题**。当多个请求访问一个Servlet实例时，就可能**对类的成员变量的修改带来问题**。

解决办法：可选用下列方法之一

A.将类的实例变量改为局部变量，局部变量是安全的；

B.将确实需要共享的资源，放在synchronized块中或将方法定义为synchronized类型，但这样会影响程序执行效率。