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。