注册Servlet组件有两种方法：

1.实现ServletContextListener接口,在contextInitialized方法中完成注册.

2.在jar文件中放入实现ServletContainerInitializer接口的初始化器

servlet 容器对于每一个 Servlet 声明必须且只能产生一

个实例。不过，如果 Servlet 实现了 SingleThreadModel 接口，servlet 容器可以选择实例化多个实例以便处

理高负荷请求或者串行化请求到一个特定实例。

多线程问题：

Servlet 容器可以并发的发送多个请求到 Servlet 的 service 方法。为了处理这些请求，Servlet 开发者必须为

service 方法的多线程并发处理做好充足的准备。一个替代的方案是开发人员实现 SingleThreadModel 接口，

由容器保证一个 service 方法在同一个时间点仅被一个请求线程调用，但是此方案是不推荐的。Servlet 容器

可以通过串行化访问 Servlet 的请求，或者维护一个 Servlet 实例池完成该需求。如果 Web 应用中的 Servlet

被标注为分布式的，容器应该为每一个分布式应用程序的 JVM 维护一个 Servlet 实例池。

对于那些没有实现 SingleThreadModel 接口的 Servlet，但是它的 service 方法（或者是那些 HttpServlet 中通

过 service 方法分派的 doGet、doPost 等分派方法）是通过 synchronized 关键词定义的，Servlet 容器不能使

用实例池方案，并且只能使用序列化请求进行处理。强烈推荐开发人员不要去通过 service 方法（或者那些

由 Service 分派的方法），因为这将严重影响性能。

该规范还允许的 cookies 是 HttpOnly cookie。HttpOnly cookie 暗示客户端它们不会暴

露给客户端脚本代码（它没有被过滤掉，除非客户端知道如何查找此属性）。使用 HttpOnly cookie 有助于减

少某些类型的跨站点脚本攻击。

getResource和getResourceAsStream方法需要一个以“/”开头的String字符串作为参数，给定的资源路径是相对于上下文的根，或者相对于web应用的WEB-INF/lib目录下的JAR文件中的META-INF/resources目录。这两个方法首先根据请求的资源查找web应用上下文的根，然后查找所有WEB-INF/lib目录下的JAR文件。查找WEB-INF/lib目录中JAR文件的顺序是不确定的。这种层次结构的文件可以存在于服务器的文件系统，Web应用的归档文件，远程服务器，或在其他位置。

Srevlet3 容器启动时会自动扫描当前服务中ServletContainerInitializer的实现类，并调用其onStartup方法，其参数Set<Class<?>> c，可通过在实现类上声明注解javax.servlet.annotation.HandlesTypes(xxx.class)注解自动注入，@HandlesTypes会自动扫描项目中所有的xxx.class的实现类，并将其全部注入Set。

[Create a new source folder](http://help.eclipse.org/juno/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-wizard-source-folder.htm) with the location src/main/resources, then create your META-INF/services folder in there and drop in your FQCN file. This should copy them into the jar file automatically. So you'll have:

Project| src
|| main
|| java
||[your source code]|| resources
|| META-INF
|| services
||[your service files]

ServletContainerInitializer类通过jar services API查找。对于每一个应用，应用启动时，由容器创建一个ServletContainerInitializer实例。框架提供的ServletContainerInitializer实现必须绑定在jar包的META-INF/services目录中的一个叫做javax.servlet.ServletContainerInitializer的文件，根据jar services API，指定ServletContainerInitializer的实现。

除ServletContainerInitializer外，我们还有一个注解—HandlesTypes。在ServletContainerInitializer 实现上的HandlesTypes注解用于表示感兴趣的一些类，它们可能指定了HandlesTypes的value中的注解（类型、方法或自动级别的注解），或者是其类型的超类继承/实现了这些类之一。无论是否设置了metadata-complete，HandlesTypes注解将应用。

当检测一个应用的类看是否它们匹配ServletContainerInitializer的HandlesTypes指定的条件时，如果应用的一个或多个可选的JAR包缺失，容器可能遇到类装载问题。由于容器不能决定是否这些类型的类装载失败将阻止应用正常工作，它必须忽略它们，同时也提供一个将记录它们的配置选项。

如果ServletContainerInitializer实现没有@HandlesTypes注解，或如果没有匹配任何指定的HandlesType，那么它会为每个应用使用null 值的集合调用一次。这将允许initializer基于应用中可用的资源决定是否需要初始化Servlet/Filter。

在任何Servlet Listener的事件被触发之前，当应用正在启动时，ServletContainerInitializer的onStartup方法将被调用。

ServletContainerInitializer’s的onStartup得到一个类的Set，其或者继承/实现initializer表示感兴趣的类，或者它是使用指定在@HandlesTypes注解中的任意类注解的。

JAX-WS运行时实现通常不是绑定到每个war包。其实现将绑定一个ServletContainerInitializer的实现（如下所示）且容器将查找使用的services API（绑定在jar包中的META-INF/services目录中的一个叫做javax.servlet.ServletContainerInitializer的文件，它将指出如下所示的JAXWSServletContainerInitializer）。

@HandlesTypes(WebService.class)

JAXWSServletContainerInitializer implements ServletContainerInitializer {

public void onStartup(Set<Class<?>> c, ServletContext ctx) throws ServletException {

// 在此，使用JAX-WS 特定的代码来初始化运行库和设置mapping等。

ServletRegistration reg = ctx.addServlet("JAXWSServlet", "com.sun.webservice.JAXWSServlet");

reg.addServletMapping("/foo");

}

}

框架的jar包也可能被绑定到war报目录中的WEB-INF/lib目录。如果ServletContainerInitializer被绑定到应用的WEB-INF/lib目录内的一个JAR包中，它的onStartup方法在绑定到的应用启动期间将被仅调用一次。如果，相反，ServletContainerInitialzer被绑定到WEB-INF/lib目录外的一个JAR包中，但仍能被运行时的服务提供商查找机制发现时，每次启动应用时，它的onStartup方法将被调用。

----------编写servlet可插拨的插件-----------------

编写一个类继承自 HttpServlet，将该类打成 JAR 包，并且在 JAR 包的 META-INF 目录下放置一个 web-fragment.xml 文件，该文件中声明了相应的 Servlet 配置。web-fragment.xml 文件示例如下：

从上面的示例可以看出，web-fragment.xml 与 web.xml 除了在头部声明的 XSD 引用不同之外，其主体配置与 web.xml 是完全一致的。

由于一个 Web 应用中可以出现多个 web-fragment.xml 声明文件，加上一个 web.xml 文件，加载顺序问题便成了不得不面对的问题。Servlet 规范的专家组在设计的时候已经考虑到了这个问题，并定义了加载顺序的规则。

web-fragment.xml 包含了两个可选的顶层标签，<name> 和 <ordering>，如果希望为当前的文件指定明确的加载顺序，通常需要使用这两个标签，<name> 主要用于标识当前的文件，而 <ordering> 则用于指定先后顺序。一个简单的示例如下：

如上所示， <name> 标签的取值通常是被其它 web-fragment.xml 文件在定义先后顺序时引用的，在当前文件中一般用不着，它起着标识当前文件的作用。

在 <ordering> 标签内部，我们可以定义当前 web-fragment.xml 文件与其他文件的相对位置关系，这主要通过 <ordering> 的 <after> 和 <before> 子标签来实现的。在这两个子标签内部可以通过 <name> 标签来指定相对应的文件。比如：

以上片段则表示当前文件必须在 FragmentB 和 FragmentC 之后解析。<before> 的使用于此相同，它所表示的是当前文件必须早于 <before> 标签里所列出的 web-fragment.xml 文件。

除了将所比较的文件通过 <name> 在 <after> 和 <begin> 中列出之外，Servlet 还提供了一个简化的标签 <others/>。它表示除了当前文件之外的其他所有的 web-fragment.xml 文件。该标签的优先级要低于使用 <name> 明确指定的相对位置关系。