servlet 规范

前言

本文内容为Java Servlet规范v2.2。

可以从下面网址下载到响应的Java Servlet API的Javadoc文档（v2.2）及其参考实现：

http://java.sun.com/products/servlet/index.html

上面提供的参考实现提供了一个规范实现的参考标准。如果规范、API和参考实现三者存在不一致的情况，则Servlet规范2.2 > Java Servlet API Reference 2.2 > 参考实现。

0.1. 谁应当阅读本规范

本文档的目标读者为：

☐ 遵循规范提供Servlet引擎的Web服务器或应用服务器提供商

☐ Web应用开发工具提供商

☐ 需要了解servlet运行机制，以编写复杂的servlet应用的开发人员

说明：

本规范不是关于servlet的用户手册。

0.2. API参考

Java Servlet API Reference v2.2 提供了组成Servlet API的所有接口、类、例外（Exception）、方法的说明。本文档中对各函数的参数签名做了简要说明。请参考API参考文档了解详细说明。

0.3. 其他的Java规范

在本规范中，将多处参考如下Java API规范：

☐ Java2 Platform Enterprise Edition v1.2（J2EE）

☐ JavaServer Pages v1.1（JSP）

☐ JavaNaming and Direcotry Interface（JNDI）

上述规范可以从J2EE网站上找到：

http://java.sun.com/j2ee

0.4. 其他重要的参考资料

在Servlet API和Servlet应用的开发实现中，还参考和遵循了如下互联网规范：

☐ RFC 1945 Hypertext Transfer Protocol (HTTP/1.0)

☐ RFC 2045 MIME Part One: Format of Internet message Bodies

☐ RFC 2046 MIME Part Two: Media Types

☐ RFC 2047 MIME Part Three: Message Header Extensions for non-ASCII text

☐ RFC 2048 MIME Part Four: Registration Procedures

☐ RFC 2049 MIME Part Five: Conformance Criteria and Examples

☐ RFC 2109 HTTP State Management Mechanism

☐ RFC 2145 Use and Interpretation of HTTP Version Numbers

☐ RFC 2324 Hypertext Coffee Pot Control Protocol (HTCPCP/1.0)

☐ RFC 2616 Hypertext Transfer Protocol (HTTP/1.1)

☐ RFC 2617 HTTP Authentication: Basic and Digest Authentication

上述RFC文档可以下面的网站上找到：

http://www.rfc-editor.org/

W3c协会（http://www.w3.org）是HTTP相关信息的权威发布机构

在本规范中，部署描述符使用了XML（Extensible Markup Language）。在如下网站可以找到关于XML的更多信息：

http://java.sum.com

http://www.xml.org

0.5. 提供反馈

Java社区的成功有赖于您的积极参与。我们欢迎您就本规范提出任何方面的反馈意见，请将您的意见email到：

servletapi-feedback@eng.sun.com

由于会收到大量的反馈信息，我们的工程师无法对邮件一一进行回复。但我们将安排一个专门小组，对所有的反馈信息进行认真地阅读、评估和存档。

0.6. 鸣谢

感谢Anselm Baird-Smith, Elias Bayeh, Vince Bonfanti, Larry Cable, Robert Clark, Daniel Coward, Satish Dharmaraj, Jim Driscoll, Shel Finkelstein, Mark Hapner, Jason Hunter, Rod McChesney, Stefano Mazzocchi, Craig McClanahan, Adam Messinger, Ron Monzillo, Vivek Nagar, Kevin Osborn, Bob Pasker, Eduardo Pelegri-Lopart, Harish Prabandham, Bill Shannon, Jon S. Stevens, James Todd, Spike Washburn, and Alan Williamson为本规范的改进和发展作出了巨大贡献。

感谢Connie Weiss, Jeff Jackson和Mala Chandra支持和推动servlet的发展提供了非凡的管理和帮助。

本规范是一项持续的、广泛的努力的成果，包含了来自Sun及其合作伙伴的大量贡献，尤其是如下这些公司和小组，对Servlet规范的发展作出了巨大的贡献：Apache Developer Community, Art Technology Group, BEA Weblogic, Clear Ink, IBM, Gefion Software, Live Software, Netscape Communications, New Atlanta Communications和Oracle。

规范的检查和修订过程同样是非常有价值的。我们的合作伙伴和公众提供了很多反馈意见来帮助我们定义和改进规范。

再次，谨对所有提供反馈的人和机构致意诚挚的感谢。

1. 概述

1.1. 什么是Servlet

Servlet是受容器管理的web组件，它能动态地生成内容。Servlet是一段小程序，被编译成平台无关、架构中立的的字节码之后，可以被Web服务器器动态地加载和运行。Servlet通过容器实现的请求-相应（request-response）方式与Web浏览器进行交互，这种请求-相应模式是基于超文本传输协议（HTTP）的。

1.2. 什么是Servlet容器

Servlet容器和Web服务器或者应用服务器一起提供网络服务，能解析MIME编码的请求，能生成MIME编码格式的相应。容器还负责容纳servlet，并对其生命周期进行管理。

Servlet容器可以内置在Web服务器中，也可以通过web服务器的扩展API作为附加组件安装。Servlet容器同样可以作为具体Web服务功能的应用服务器的内置模块或者附加组件。

所有的Servlet容器必须支持HTTP协议，并可以支持其他基于请求－相应模式的协议，例如HTTPS。Servlet容器至少需要支持HTTP 1.0版本，并强烈建议同时支持HTTP 1.1版本。

Servlet容易可以对servlet的运行环境设置安全限制。在J2SE1.2或者J2EE 1.2环境下，这些限制条件应当使用Java2平台所提供的授权框架来实现。例如，high end application servers 会限制某些操作，例如创建Thread对象，来保证容器的其他组件不会收到负面影响。

1.3. 一个例子

客户端程序，如Web浏览器，使用HTTP请求来访问Web服务器。请求首先被Web服务器处理，并被转交给Servlet容器。Servlet根据内部配置决定调用哪一个servlet，并在调用时将代表request和response的对象传递给它。Servlet容器可以与Web服务器运行在同一个进程中，同一个主机的不同进程中，或者运行在不同的主机中。

Servlet通过request对象知道谁是远程对象，哪些HTML表单参数作为request的一部分被发送，以及其他相关的数据。

Servlet可以执行程序设定的各种逻辑，生成发还给客户端的数据，并通过response对象将这些数据发还给客户端。

一旦servlet完成对request的处理，servlet容器需要保证response内容被正确刷新，并将控制返还给Web服务器。

1.4. Servlet和其他技术的比较

从功能性的角度，Servlet介于CGI程序和私有服务器扩展（例如Netscape服务器API－NSAPI，Apache模块）之间。

相对于其他的服务器扩展机制，Servlet具备如下优点：

☐ 由于使用了不同的进程模型，其速度远远超过CGI脚本

☐ 使用标准API，这些API得到大量Web服务器的支持

☐ 拥有Java编程预言的所有有点，包括易于开发，平台无关性

☐ 可以使用Java平台所提供的大量API

1.5. Servlet和J2EE的关系

Servlet API是Java 2 平台企业版 1.2版本所需要的API。J2EE规范描述了对servlet和servlet容易的附加要求。Servlet应当

被部署到容器中，而容器和servlet则都运行在J2EE环境中。

1.6. 可分布的Servlet容器

在这个版本的规范中，增加了一个特性：将一个Web应用标记为可分布。这个标记允许servlet容器提供商将一个Web应用的servlet部署在多个Java虚拟机中，而这些虚拟机可以运行在同一台主机上，也可以运行在不同的主机上。一个被标记为可分布的应用必须遵循一些限制条件，使得支持分布式应用的容器能实现集群、失效转移等特性。

高性能的环境支持可扩展性、集群、失效转移（J2EE兼容）。所有需要在高性能的环境下运行的Web应用，应当设计实现成为可发布的Web应用，这使得应用可以最大程度地利用服务器所提供的特性。如果一个不可分布的应用部署在这样一个服务器上，则不能充分利用服务器提供的特性。

1.7. 自2.1版本之后的变动

自2.1版正式发布之后，对本规范的主要变动如下：

☐ Web应用的概念的介绍

☐ web application archive files的介绍

☐ Response buffering（响应缓存）的介绍

☐ 可分布servlet的介绍

☐ 增加通过名称获取RequestDispatcher的功能

☐ 增加通过相对路径获取RequestDispatcher的功能

☐ 改进国际化

☐ 对于分布式servlet引擎语义的一些澄清

对API做了如下变动：

☐ ServletConfig接口添加了getServletName方法，用于获取在系统中表示本servlet的名称

☐ ServletContext接口添加getInitParameter，getInitParameterNames这两个方法，使初始化参数能在应用层面被设置，并被改应用的所有servlet所共享

☐ ServletRequest接口添加getLocale方法，帮助决定客户端当前在哪个locale

☐ ServletRequest接口添加isSecure方法，用于标识request是否通过安全的方式进行传输，例如使用HTTPS协议

☐ 修改了UnavailableException类的构造函数。因为现有的构造器方面参数签名容易被开发人员混淆。修改之后的构造器使用了更简单的参数签名。

☐ HttpServletRequest接口添加getHeaders方法，用于获取在request中，所有用某个名称标识的头信息

☐ HttpServletRequest添加isUserInRole、getUserPrinciple两个方法，使servlet可以使用基于抽象角色的认证

☐ HttpServletResponse接口添加addHeader、addIntHeader、addDateHeader三个方法，允许使用同一个名字，创建多个头信息

☐ HttpSession接口添加getAttribute、getAttributeNames、setAttribute、removeAttribute四个方法，以改进API的命名规范，相应地，getValue,、getValueNames,、setValue、removeValue这四个方法被废弃。

此外，还增加了大量概念的说明和澄清。

2. 术语

在本规范中，将大量使用这些术语。

2.1. 基本术语

2.1.1. 统一资源定位符

统一资源定位符（URL）是一段简洁的字符串，用以标识在网络上的某个资源。当通过URL访问资源时，可能对该资源进行不同的操作处理。URL是通用资源标识（URI）的一种形式，通常使用如下格式：

<protocol>//<servername>/<resource>

基于本规范的目的，我们主要关心基于HTTP协议的URL，其格式如下所示：

http[s]://<servername>[:port]/<url-path>[?<query-string>]

样例：

http://java.sun.com/products/servlet/index.html

https://javashop.sun.com/purchase

在基于HTTP的URL中，斜杠'/'是保留字符，用于划分URL的url-path部分分层结构的路径。服务器负责决定分层结构的含义。url-path和某个文件系统路径的分层结构之后，不要求存在对应关系。

2.1.2. Servlet定义

一个servlet的定义是将一个唯一的名称与一个全限定格式的类名进行关联，该类必须实现Servlet接口。Servlet定义中还可以设定一组初始化参数。

2.1.3. Servlet映射

一个servlet映射是在一个serlvet容器中，将一个servlet定义与一个URL路径格式进行关联。所有符合指定格式的请求都将被关联的servlet处理。

2.1.4. Web应用

Web应用是一个集合，它包括servlet，JSP页面，HTML文档，一级其他web资源，包括图片文件、压缩档案等。一个Web应用可以被打包成一个存档文件，或者放在一个开放的目录结构中。

所有兼容的servlet容器都必须能接受Web应用，并能将Web应用的内容部署到运行环境中。这意味着容器应当既可以通过Web应用存档文件运行一个应用，也可以将web应用的内容移动到容器指定的特定位置，而后运行。

2.1.5. Web Application Archive（Web应用存档）

Web应用存档是一个单独的文件，它包含了web应用的所有组件。这个存档文件可使用标准的JAR工具来创建，可以对Web应用的部分或所有组件进行签名。

Web应用存档文件使用.war后缀名。使用这个新的后缀而不使用.jar的原因是：jar文件用于包含一组class文件，并可以通过存放在classpath下，或通过GUI双击来启动一个应用程序。而Web应用存档文件的内容不适用于这样的情况，故此应当使用一个新的后缀名。

2.2. 角色

基于servlet的应用的开发、部署、运行过程往往由不同的人员负责，这些人员需要进行不同的活动，并承担不同的职责。实际情况下，一个小组可能会承担多个角色的职责，也可能每个角色都由一个单独的小组负责。

2.2.1. 应用开发员

应用开发人员是Web应用的生产者，其生产成果是一批servlet classes，jsp页面，html页面，及其相关的类库和其他文件（如图片文件，压缩存档文件等）。应用开发人员一般是应用领域专家，需要了解servlet环境及其编程相关的知识，例如并发访问等，并据此开发web应用。

2.2.2. 应用装配员

应用装配员的职责是将开发员交付的成果组装为一个可部署单元。其工作的输入是开发院所提供的java classes，JSP页面，HTML页面，以及web应用所许需的类库和其他文件。其工作的输出则是Web应用存档文件，或者是保存了Web应用的一个目录结构

2.2.3. 部署员

部署人员的职责是将一个或多个，web应用存档文件或者web应用目录，部署到指定的运营环境下。运营环境往往包括指定的servlet容器和web服务器。部署人员必须解决开发人员所声明的所有外部依赖需求。部署人员通常使用servlet容器提供的工具来完成这些工作。

部署人员是某个运营环境的专家。例如，部署人员需要负责将开发人员定义的安全角色映射成为运营环境下已经存在的用户组或者帐号。

2.2.4. 系统管理员

系统管理员负责对servlet容器和web服务器进行配置和管理。系统管理员同时需要监视web应用的运行是否健康。

本规范没有定义系统管理员的具体工作内容contracts for system management and administrator. 系统管理员通常使用容器提供商和主机制造商所提供的运行监控和管理工具来进行他们的工作。

2.2.5. Servlet容器提供商

Servlet容器提供商负责提供运行环境－servlet容器，并可能同时提供web服务器。其中一般包含一个web应用，作为部署web应用的工具。

Servlet容器提供商主要在HTTP层面商进行编程。本规范没有定义web服务器和servlet容器之间的接口，因此servlet容器提供商可以根据自己的需求决定两者的边界及其实现。

2.3. 安全术语

2.3.1. Principal（参与者）

一个principal是指可能被身份认证协议认证所的对象实体。一个principal由principal名称来标识，并使用认证数据进行认证。Principal名称、认证数据的内容和格式，取决于所选择的身份认证协议。

2.3.2. Security Policy Domain（安全策略域）

Security policy domain是一个scope，其中，安全服务管理员定义了安全策略，并对其中的活动强制应用了这些安全策略。Security policy domain又经常被成为realm（领域）。

2.3.3. Security Technology Domain（安全技术域）

Security technology domain是一个scope，其中，相同的安全机制，例如Kerboros，被用于执行安全策略。在一个security technology domain中，可以同时存在多个security policy domain.

2.3.4. 角色（Role）

在一个应用中，角色作为一个抽象概念被开发人员定义，而部署人员则需要在一个security policy domain中将它们映射成为具体的用用户或者用户组。

3. Servlet接口

Servlet接口是Servlet API的核心抽象，所有的servlet都需要实现这个接口。可以直接实现servlet接口，而更常见的方式则是通过扩展/继承一个实现了servlet接口的类来实现这一点。在API中提供两个实现了servlet接口的类：GenericServlet和HttpServlet。大部分情况下，开发人员通过继承HttpServlet来实现自己的servlet。

3.1. 处理请求的方法（Request Handling Methods）

Servlet接口定义了一个名为service的方法来处理客户端的请求。每当servlet容器将request路由到一个servlet实例时，这个方法都会被调用。多个request线程可以同时调用并执行同一个service方法。

3.1.1. HTTP专用的请求处理方法（HTTP Specific Request Handling Methods）

HttpServlet抽象子类添加了几个附加的方法，service方法会根据request自动调用这些附加的方法。这些方法是：

☐ doGet方法，用于处理HTTP GET请求

☐ doPost方法，用于处理HTTP POST请求

☐ doPut方法，用于处理HTTP PUT请求

☐ doDelete方法，用于处理HTTP DELETE请求

☐ doHead方法，用于处理HTTP HEAD请求

☐ doOptions方法，用于处理HTTP OPTIONS请求

☐ doTrace方法，用于处理HTTP TRACE请求

在开发基于HTTP的servlet的时候，开发人员一般只涉及doGet和doPost方法。其他的方法可视为高级方法，为熟悉HTTP编程的程序员准备。

Servlet开发人员可以通过实现doPut和doDelete方法来支持HTTP/1.1客户端。HttpServlet的doHead方法会执行doGet方法，但是只返回doGet方法生产的头信息给客户端。doOptions方法自动检测servlet支持哪些HTTP方法，并将此信息发送给客户端。doTrace方法则将返回一个response，其中包含了trace 请求中的所有头信息。

由于HTTP/1.0没有定义PUT, DELETE, OPTIONS 和TRACE方法，因此，只支持HTTP/1.0的容器，只会使用servlet的doGet, doHead和doPost。

3.1.2. 有条件GET的支持（Conditional GET Support）

HttpServlet接口定义了getLastModified方法用于支持有条件get操作。有条件get操作是指客户端通过HTTP GET方法请求一个资源的时候，在头信息里设置只有当被请求的资源在指定时间之后被修改过，才返回响应的body部分。

Servlets that implement the doGet method and that provide content that does not necessarily change from request to request should implement this method to aid in efficient utilization of network resources.

3.2. 实例数量

缺省情况下，在容器中，每个servlet定义（servlet definition）只有一个实例。

在servlet实现了SingleThreadModel接口的情况下，容器将创建多个实例，这样容器能够处理高负载请求，并同时保证request排队访问单个servlet实例。

对于一个标记为可分布的应用，对于容器所使用的每一个虚拟机，都将创建为一个servlet定义的创建一个实例。如果servlet实现了SingleThreadModel方法，则对容器使用的每个虚拟机，都可以创建多个实例。

3.2.1. 关于单线程模式（SingleThreadModel）的说明

SingleThreadModel接口用于保证在同一时刻，只有一个线程，访问一个servlet实例的service方法。需要重点说明的是，上述保证只限于对servlet实例的访问。对于能被多个servlet实例访问的对象，例如HttpSession的实例对象，还是能够被多个servlet实例同时访问的，不管servlet是否实现SingleThreadModel接口。

3.3. Servlet生命周期

Servlet生命周期管理包括：如何被载入（load），如何实例化，如何初始化，如何处理客户端情况，以及如何被销毁。这个生命周期，在API中，通过Servlet接口（javax.servlet.Servlet）的init，service和destroy方法得以体现。所有的servlet都必须实现这些接口。可以直接实现servlet接口，也可以通过继承GenericServlet或HttpServlet抽象类来实现。

3.3.1. 载入和实例化

Servlet容器负责servlet的载入和实例化。这项工作可以在servlet引擎启动的时候进行，也可以延迟到当容器检测到需要servlet来处理某个请求的时候进行。

首先，servlet容器得能找到servlet对应的class文件。容器可以根据需要选择使用java类装载器（class loader）从本地文件系统、远程文件系统或者网络服务中加载这个类。

在这个类被加载之后，容器就创建该类的一个实例。

需要重点说明的是，在servlet容器中，一个servlet类可能会有多个实例。例如，多个servlet定义使用同一个类，但定义了不同的初始化参数。另外，如果servlet实现了SingleThreadModel接口，容器会创建一个实例池。

3.3.2. 初始化

在servlet对象载入和实例化之后，容器必须在servlet处理客户端请求之前对它进行初始化。在初始化过程中，servlet可以读取持久化的配置数据，初始化昂贵（costly）的资源，例如jdbc数据库连接，并执行其他一次性的操作。容器通过调用Servlet接口的init来初始化servlet，对于每个servlet定义，容器将创建一个唯一的、实现了ServletConfig接口的对象，并将其作为参数传递给init方法。通过这个配置信息对象，servlet可以name-value的方式获取初始化参数。这个配置信息对象还包含了一个实现了ServletContext接口的对象，描述了servlet的运行环境信息。关于ServletContext接口的详细信息请看第4章。

3.3.2.1. 初始化过程出错

在初始化过程中，servlet实例可以通过抛出UnavailableException或者ServletException异常来声明初始化失败。这种情况下，容器必须释放servlet实例，不能将其作为一个活动的服务。由于初始化没有成功，此时destroy方法不会被调用。

在初始化失败的servlet实例被释放之后，容器可以在任何时候实例化并初始化新的servlet。唯一的例外是在servlet初始化过程中抛出的UnavailableException中定义了最短失效时间，在这段时间内，不能创建新的实例。

3.3.2.2. Tool Considerations

用工具加载并分析（introspect）一个web应用的时候，它可以加载并分析（introspect）web应用的成员类，这会触发静态初始化方法的执行。出于这个特性的考虑，在Servlet接口的init方法被调用之前，开发人员不能认为servlet已经在活动在容器的运行环境内。例如，当servlet的静态初始化方法被调用的时候，不应当去创建到数据库或者EJB容器的连接

3.3.3. 请求处理

Servlet被正确初始化之后，容器就能用它来处理请求。请求被封装在ServletRequest类型的对象中，响应信息被封装在ServletResponse类型的对象中，这两个对象以参数的形式传给Servlet接口的service方法。处理HTTP请求时，容器必须通过实现HttpServletRequest和HttpServletResponse接口来提供请求和响应对象。

需要说明的是，servlet实例被创建成为服务之后，可能在整个生命周期不需要响应任何请求。

3.3.3.1. 多线程问题

在处理客户端情况过程中，容器有可能并发访问servlet的service方法，以处理来自客户端的并发访问。开发人员必须注意这一点，保证servlet在并发情况下正确运行。

开发人员可以通过让servlet实现SingleThreadModel接口来避免这种缺省行为。通过实现这个接口可以保证同一时刻只允许一个请求线程调用service方法。Servlet容器可以多种方式实现这个要求，可以让访问一个servlet的请求排队，也可以提供servlet实例池。如果servlet属于可分布应用，容器可以在应用所分布的每个虚拟机中保持一个servlet实例池。

如果service方法使用了synchronized修饰符（或者是HttpServlet的doGet、doPost等通过service方法分发调用的方法），servlet容器将保证对该方法的排队访问，并且不能为其创建实例池。我们强烈建议不要同步service方法和HttpServlet的doGet、doPost这些服务方法。

3.3.3.2. 处理请求过程中的异常

在处理请求过程中，servlet可以抛出ServletException或者UnavailableException异常。ServletException表明处理情况过程

中发生了某种错误，容器需要采取适当的方式来清理请求。UnavailableException异常则表示servlet临时性或者永久性不能响应请求。

如果servlet抛出了声明为永久性失效的UnavailableException，servlet容器必须移除这个servlet服务，调用其destroy方法，并释放servlet实例。

如果是临时性失效，在指定的最短失效期内，容器不能把请求路由到这个servlet。容器必须向被拒绝的请求返回SERVICE_UNAVAILABLE（503）响应，并在用Retry-After后信息标明何时再次生效。容器也可以选择不区分临时性失效和永久性失效，将所有UnavailableException当作永久性失效，然后移除servlet服务。

3.3.3.3. 线程安全

请求和响应对象是不保证线程安全的。这意味着它们只应当在请求处理线程范围内被使用。请求对象和响应对象的引用不应当被传递给在其他线程中失效的对象，否则会产生无法预料的结果。

3.3.4. 服务终止

Servlet容器不需要在任何时刻都保持servlet处于被加载的状态。Servlet实例的活动时间可以只有几毫秒，可以和引擎的生命周期一样长（几天，几个月，或者几年），也可以介于两者之间。

当servlet容器检测到应当将一个servlet服务移除的时候（例如容器需要保留内存，或者容器被停止），必须允许servlet释放资源，并保存持久化状态。容器通过调用Servlet接口的destroy方法实现这一点。

在servlet容器调用destroy方法之前，必须先确保所有在servlet方法中运行的线程或者停止，或者超出服务器定义的运行限制时间。

Servlet实例的destroy方法一旦被调用，容器就不能再将请求路由到这个servlet实例。如果容器需要重新提供这个服务，必须使用新的servlet实例。

Destroy方法完成执行后，容器必须释放servlet实例，以便进行垃圾回收。