译者前言：

近来在整理有关 Servlet 资料时发现，在网上竟然找不到一份中文的 Java Servlet API 的说明文档，而在有一本有关 JSP 的书后面附的 Java Servlet API 说明竟然不全，而这份文档的 2.1a 版在 1998 年的 11 月份就已定稿。所以我决定翻译一份中文的文档（其中一些与技术关系不大的部分已被略去），有兴趣的读者可以从 http://java.sun.com/products/servlet/2.1/servletspec-2.1.zip 下载原文阅读。

Java Servlet API 说明文档（2.1a 版）

1998 年 11 月

绪言

这是一份关于 2.1 版 Java Servlet API 的说明文档，作为对这本文档的补充，你可以到 http://java.sun.com/products/servlet/index.html 下面下载 Javadoc 格式的文档。

谁需要读这份文档

这份文档描述了 Java Servlet API 的最新版本 2.1 版。所以，这本书对于 Servlet 的开发者及 servlet 引擎的开发者同样适用。

Java Servlet API 的组成

Java Servlet API 由两个软件包组成：一个是对应 HTTP 的软件包，另一个是不对应 HTTP 的通用的软件包。这两个软件包的同时存在使得 Java Servlet API 能够适应将来的其他请求-响应的协议。

这份文档以及刚才提及的 Javadoc 格式的文档都描述了这两个软件包，Javadoc 格式的文档还描述了你应该如何使用这两个软件包中的所有方法。

有关规范

你也许对下面的这些 Internet 规范感兴趣，这些规范将直接影响到 Servlet API 的发展和执行。你可以从 http://info.internet.isi.edu/7c/in-notes/rfc/.cache 找到下面提到的所有这些 RFC 规范。

RFC 1738 统一资源定位器(URL)

RFC 1808 相关统一资源定位器

RFC 1945 超文本传输协议--HTTP/1.0

RFC 2045 多用途 Internet 邮件扩展(多用途网际邮件扩充协议(MIME))第一部分:Internet 信息体格式

RFC 2046 多用途 Internet 邮件扩展(多用途网际邮件扩充协议(MIME))第二部分:媒体类型

RFC 2047 多用途网际邮件扩充协议(MIME)(多用途 Internet 邮件扩展)第三部分:信息标题扩展用于非 ASCII 文本

RFC 2048 多用途 Internet 邮件扩展(多用途网际邮件扩充协议(MIME))第四部分: 注册步骤

RFC 2049 多用途 Internet 邮件扩展(多用途网际邮件扩充协议(MIME))第五部分:一致性标准和例子

RFC 2068 超文本传输协议 -- HTTP/1.1

RFC 2069 一个扩展 HTTP:摘要访问鉴定

RFC 2109 HTTP 状态管理机制

RFC 2145 HTTP 版本号的使用和解释

RFC 2324 超文本 Coffee Pot 控制协议 (HTCPCP/1.0)

万维网协会（http://www.w3.org）管理着这些协议的规范和执行。

有关 Java Servlets

JavaTM servlets 是一个不受平台约束的 Java 小程序，它可以被用来通过多种方法扩充一个 Web 服务器的功能。你可以把 Servlet 理解成 Server 上的 applets，它被编译成字节码，这样它就可以被动态地载入并用效地扩展主机的处理能力。

Servlet 与 applets 不同的地方是，它不运行在 Web 浏览器或其他图形化的用户界面上。Servlet 通过 servlet 引擎运行在 Web 服务器中，以执行请求和响应，请求、响应的典型范例是 HTTP 协议。

一个客户端程序，可以是一个 Web 浏览器，或者是非其他的可以连接上 Internet 的程序，它会访问 Web 服务器并发出请求。这个请求被运行在 Web 服务器上的 Servlet 引擎处理，并返回响应到 Servlet。Servlet 通过 HTTP 将这个响应转发到客户端。

在功能上，Servlet 与 CGI、NSAPI 有点类似，但是，与他们不同的是：Servlet 具有平台无关性。

Java Servlet 概论

Servlet 与其他普通的 server 扩展机制有以下进步：

因为它采用了不同的进程处理模式，所以它比 CGI 更快。

它使用了许多 Web 服务器都支持的标准的 API。

它继承了 Java 的所有优势，包括易升级以及平台无关性。

它可以调用 Java 所提供的大量的 API 的功能模块。

这份文档说明了 Java Servlet API 的类和接口的方法。有关更多的信息，请参看下面的 API 说明。

Servlet 的生命周期

一个 Java servlet 具有一个生命周期，这个生命周期定义了一个 Servlet 如何被载入并被初始化，如何接收请求并作出对请求的响应，如何被从服务中清除。Servlet 的生命周期被 javax.servlet.Servlet 这个接口所定义。

所有的 Java Servlet 都会直接地或间接地执行 javax.servlet.Servlet 接口，这样它才能在一个 Servlet 引擎中运行。Servlet 引擎是 Web 服务器按照 Java Servlet API 定制的扩展。Servlet 引擎提供网络服务，能够理解 MIME 请求，并提供一个运行 Servlet 的容器。

javax.servlet.Servlet 接口定义了在 Servlet 的生命周期中特定时间以及特定顺序被调用的方法。

Servlet 的解析和载入

Servlet 引擎解析并载入一个 Servlet，这个过程可以发生在引擎启动时，需要一个 Servlet 去响应请求时，以及在此之间的任何时候。

Servlet 引擎利用 Java 类载入工具载入一个 Servlet，Servlet 引擎可以从一个本地的文件系统、一个远程的文件系统以及网络载入 Servlet。

Servlet 的初始化

Servlet 引擎载入 Servlet 后，Servlet 引擎必须对 Servlet 进行初始化，在这一过程中，你可以读取一些固定存储的数据、初始化 JDBC 的连接以及建立与其他资源的连接。

在初始化过程中，javax.servlet.Servlet 接口的 init()方法提供了 Servlet 的初始化信息。这样，Servlet 可以对自己进行配置。

init()方法获得了一个 Servlet 配置对象（ServletConfig）。这个对象在 Servlet 引擎中执行，并允许 Servlet 通过它获处相关参数。这个对象使得 Servlet 能够访问 ServletContext 对象。

Servlet 处理请求

Servlet 被初始化之后，它已经可以处理来自客户端的请求，每一个来自客户端的请求都被描述成一个 ServletRequest 对象，Servlet 的响应被描述成一个 ServletResponse 对象。

当客户端发出请求时，Servlet 引擎传递给 Servlet 一个 ServletRequest 对象和一个 ServletResponse 对象，这两个对象作为参数传递到 service()方法中。

Servlet 也可以执行 ServletRequest 接口和 ServletResponse 接口。ServletRequest 接口使得 Servlet 有权使用客户端发出的请求。Servlet 可以通过 ServletInputStream 对象读取请求信息。

ServletResponse 接口允许 Servlet 建立响应头和状态代码。通过执行这个接口，Servlet 有权使用 ServletOutputStream 类来向客户端返回数据。

多线程和映射

在多线程的环境下，Servlet 必须能处理许多同时发生的请求。例外的情况是这个 Servlet 执行了 SingleThreadModel 接口，如果是那样的话，Servlet 只能同时处理一个请求。

Servlet 依照 Servlet 引擎的映射来响应客户端的请求。一个映射对包括一个 Servlet 实例以及一个 Servlet 返回数据的 URL，例如：HelloServlet with /hello/index.html。

然而，一个映射可能是由一个 URL 和许多 Servlet 实例组成，例如：一个分布式的 Servlet 引擎可能运行在不止一个的服务器中，这样的话，每一个服务器中都可能有一个 Servlet 实例，以平衡进程的载入。作为一个 Servlet 的开发者，你不能假定一个 Servlet 只有一个实例。

Servlet 的卸载

Servlet 引擎并不必需保证一个 Servlet 在任何时候或在服务开启的任何时候都被载入。Servlet 引擎可以自由的在任何时候使用或清除一个 Servlet。因此，我们不能依赖一个类或实例来存储重要的信息。

当 Servlet 引擎决定卸载一个 Servlet 时（例如，如果这个引擎被关闭或者需要让资源），这个引擎必须允许 Servlet 释放正在使用的资源并存储有关资料。为了完成以上工作，引擎会调用 Servlet 的 destroy()方法。

在卸载一个 Servlet 之前，Servlet 引擎必须等待所有的 service()方法完成或超时结束（Servlet 引擎会对超时作出定义）。当一个 Servlet 被卸载时，引擎将不能给 Servlet 发送任何请求。引擎必须释放 Servlet 并完成无用存储单元的收集

Servlet 映射技术

作为一个 Servlet 引擎的开发者，你必须对于如何映射客户端的请求到 Servlet 有大量的适应性。这份说明文档不规定映射如何发生。但是，你必须能够自由地运用下面的所有技术：

映射一个 Servlet 到一个 URL

例如，你可以指定一个特殊的 Servlet 它仅被来自/feedback/index.html 的请求调用。

映射一个 Servlet 到以一个指定的目录名开始的所有 URL

例如，你可以映射一个 Servlet 到/catalog，这样来自/catalog/、 /catalog/garden 和 /catalog/housewares/index.html 的请求都会被映射到这个 Servlet。但是来自/catalogtwo 或 /catalog.html 的请求没被映射。

映射一个 Servlet 到所有以一个特定的字段结尾的所有 URL

例如，你可以映射一个来自于所有以 in.thtml 结尾的请求到一个特定的 Servlet。

映射一个 Servlet 到一个特殊的 URL /servlet/servlet_name。

例如，如果你建立了一个名叫 listattributes 的 Servlet，你可以通过使用 /servlet/listattributes 来访问这个 Servlet。

通过类名调用 Servlet

例如，如果 Servlet 引擎接收了来自/servlet/com.foo.servlet.MailServlet 的请求，Servlet 引擎会载入这个 com.foo.servlet.MailServlet 类，建立实例，并通过这个 Servlet 来处理请求。

Servlet 环境

ServletContext 接口定义了一个 Servlet 环境对象，这个对象定义了一个在 Servlet 引擎上的 Servlet 的视图。通过使用这个对象，Servlet 可以记录事件、得到资源并得到来自 Servlet 引擎的类（例如 RequestDispatcher 对象）。一个 Servlet 只能运行在一个 Servlet 环境中，但是不同的 Servlet 可以在 Servlet 引擎上有不同的视图。

如果 Servlet 引擎支持虚拟主机，每个虚拟主机有一个 Servlet 环境。一个 Servlet 环境不能在虚拟主机之间共享。

Servlet 引擎能够允许一个 Servlet 环境有它自己的活动范围。

例如，一个 Servlet 环境是属于 bank 应用的，它将被映射到/bank 目录下。在这种情况下，一个对 getContext 方法的调用会返回/bank 的 Servlet 环境。

HTTP 会话

HTTP 是一个没有状态的协议。要建立一个有效的 Web 服务应用，你必须能够识别一个连续的来自远端的客户机的唯一的请求。随着时间的过去，发展了许多会话跟踪的技术，但是使用起来都比较麻烦。

Java Servlet API 提供了一个简单的接口，通过这个接口，Servlet 引擎可以有效地跟踪用户的会话。

建立 Session

因为 HTTP 是一个请求-响应协议，一个会话在客户机加入之前会被认为是一个新的会话。加入的意思是返回会话跟踪信息到服务器中，指出会话已被建立。在客户端加入之前，我们不能判断下一个客户端请求是目前会话的一部分。

在下面的情况下，Session 会被认为是新的 Session。

客户端的 Session 在此之前还不知道

客户端选择不加入 Session，例如，如果客户端拒绝接收来自服务器的 cookie

作为一个 Servlet 的开发者，你必须决定你的 Web 应用是否处理客户机不加入或不能加入 Session。服务器会在 Web 服务器或 Servlet 规定的时间内维持一个 Session 对象。当

Session 终止时，服务器会释放 Session 对象以及所有绑定在 Session 上的对象。

绑定对象到 Session 中

如果有助于你处理应用的数据需求，你也许需要绑定对象到 Session 中，你可以通过一个唯一的名字绑定任何的对象到 Session 中，这时，你需要使用 HttpSession 对象。任何绑定到 Session 上的对象都可以被处理同一会话的 Servlet 调用。

有些对象可能需要你知道什么时候会被放置到 Session 中或从 Session 中移开。你可以通过使用 HttpSessionBindingListener 接口获得这些信息。当你的应用存储数据到 Session 中，或从 Session 中清除数据，Servlet 都会通过 HttpSessionBindingListener 检查什么类被绑定或被取消绑定。这个接口的方法会通报被绑定或被取消绑定的对象。

2 。API 对象的说明

这一部分包含了对 Java Servlet API 的全部类和接口的详细说明。这个说明与 Javadoc API 差不多，但是这份文档提供了更多的信息。

API 包含了两个软件包，十二个接口和九个类。

软件包：javax.servlet

所包含的接口： RequestDispatcher； Servlet； ServletConfig； ServletContext； ServletRequest；ServletResponse；SingleThreadModel。

所包含的类：GenericServlet；ServletInputStream；ServletOutputStream；ServletException；UnavailableException。

一、RequestDispatcher 接口：

定义：

public interface RequestDispatcher;

定义一个对象，从客户端接收请求，然后将它发给服务器的可用资源（例如 Servlet、CGI、HTML 文件、JSP 文件）。Servlet 引擎创建 request dispatcher 对象，用于封装由一个特定的 URL 定义的服务器资源。

这个接口是专用于封装 Servlet 的，但是一个 Servlet 引擎可以创建 request dispatcher 对象用于封装任何类型的资源。

request dispatcher 对象是由 Servlet 引擎建立的，而不是由 Servlet 开发者建立的。

方法

1、forward

public void forward(ServletRequest request, ServletReponse response)
    throws ServletException, IOException;

被用来从这个 Servlet 向其它服务器资源传递请求。当一个 Servlet 对响应作了初步的处理，并要求其它的对象对此作出响应时，可以使用这个方法。

当 request 对象被传递到目标对象时，请求的 URL 路径和其他路径参数会被调整为反映目标对象的目标 URL 路径。

如果已经通过响应返回了一个 ServletOutputStream 对象或 PrintWriter 对象，这个方法将不能使用，否则，这个方法会抛出一个 IllegalStateException。

2、include

public void include(ServletRequest request, ServletResponse response)
throws ServletException, IOException

用来包括发送给其他服务器资源的响应的内容。本质上来说，这个方法反映了服务器端的内容。

请求对象传到目标对象后会反映调用请求的请求 URL 路径和路径信息。这个响应对象只能调用这个 Servlet 的 ServletOutputStream 对象和 PrintWriter 对象。

一个调用 include 的 Servlet 不能设置头域，如果这个 Servlet 调用了必须设置头域的方法（例如 cookie），这个方法将不能保证正常使用。作为一个 Servlet 开发者，你必须妥善地解决那些可能直接存储头域的方法。例如，即使你使用会话跟踪，为了保证 session 的正常工作，你必须在一个调用 include 的 Servlet 之外开始你的 session

二、Servlet 接口。

定义

public interface Servlet

这个接口定义了一个 Servlet：一个在 Web 服务器上继承了这个功能的 Java 类。

方法

1、init

public void init(ServletConfig config) throws ServletException;

Servlet 引擎会在 Servlet 实例化之后，置入服务之前精确地调用 init 方法。在调用 service 方法之前，init 方法必须成功退出。

如果 init 方法抛出一个 ServletException，你不能将这个 Servlet 置入服务中，如果 init 方法在超时范围内没完成，我们也可以假定这个 Servlet 是不具备功能的，也不能置入服务中。

2、service

public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException;

Servlet 引擎调用这个方法以允许 Servlet 响应请求。这个方法在 Servlet 未成功初始化之前无法调用。在 Servlet 被初始化之前，Servlet 引擎能够封锁未决的请求。

在一个 Servlet 对象被卸载后，直到一个新的 ServeIt 被初始化，Servlet 引擎不能调用这个方法

3、destroy

public void destroy();

当一个 Servlet 被从服务中去除时，Servlet 引擎调用这个方法。在这个对象的 service 方法所有线程未全部退出或者没被引擎认为发生超时操作时，destroy 方法不能被调用。

4、getServletConfig

public ServletConfig getServletConfig();

返回一个 ServletConfig 对象，作为一个 Servlet 的开发者，你应该通过 init 方法存储 ServletConfig 对象以便这个方法能返回这个对象。为了你的便利，GenericServlet 在执行这个接口时，已经这样做了。

5、getServletInfo

public String getServletInfo();

允许 Servlet 向主机的 Servlet 运行者提供有关它本身的信息。返回的字符串应该是纯文本格式而不应有任何标志（例如 HTML，XML 等）。

三、ServletConfig 接口

定义

public interface ServletConfig

这个接口定义了一个对象，通过这个对象，Servlet 引擎配置一个 Servlet 并且允许 Servlet 获得一个有关它的 ServletContext 接口的说明。每一个 ServletConfig 对象对应着一个唯一的 Servlet。

方法

1、getInitParameter

public String getInitParameter(String name);

这个方法返回一个包含 Servlet 指定的初始化参数的 String。如果这个参数不存在，返加空值。

2、getInitParameterNames

public Enumeration getInitParameterNames();

这个方法返回一个列表 String 对象，该对象包括 Servlet 的所有初始化参数名。如果 Servlet 没有初始化参数，getInitParameterNames 返回一个空的列表。

3、getServletContext

public ServletContext getServletContext();

返回这个 Servlet 的 ServletContext 对象。

四、ServletContext 接口

定义

public interface ServletContext

定义了一个 Servlet 的环境对象，通过这个对象，Servlet 引擎向 Servlet 提供环境信息。

一个 Servlet 的环境对象必须至少与它所驻留的主机是一一对应的。在一个处理多个虚拟主机的 Servlet 引擎中（例如，使用了 HTTP1.1 的主机头域），每一个虚拟主机必须被视为一个单独的环境。此外，Servlet 引擎还可以创建对应于一组 Servlet 的环境对象。

方法

1、getAttribute

public Object getAttribute(String name);

返回 Servlet 环境对象中指定的属性对象。如果该属性对象不存在，返回空值。这个方法允许访问有关这个 Servlet 引擎的在该接口的其他方法中尚未提供的附加信息。

2、getAttributeNames

public Enumeration getAttributeNames();

返回一个 Servlet 环境对象中可用的属性名的列表。

3、getContext

public ServletContext getContext(String uripath);

返回一个 Servlet 环境对象，这个对象包括了特定 URI 路径的 Servlets 和资源，如果该路径不存在，则返回一个空值。URI 路径格式是/dir/dir/filename.ext。

为了安全，如果通过这个方法访问一个受限制的 Servlet 的环境对象，会返回一个空值。

4、getMajorVersion

public int getMajorVersion();

返回 Servlet 引擎支持的 Servlet API 的主版本号。例如对于 2.1 版，这个方法会返回一个整数 2。

5、getMinorVersion

public int getMinorVersion();

返回 Servlet 引擎支持的 Servlet API 的次版本号。例如对于 2.1 版，这个方法会返回一个整数 2。

6、getMimeType

public String getMimeType(String file);

返回指定文件的 MIME 类型，如果这种 MIME 类型未知，则返回一个空值。MIME 类型是由 Servlet 引擎的配置决定的。

7、getRealPath

public String getRealPath(String path);

一个符合 URL 路径格式的指定的虚拟路径的格式是：/dir/dir/filename.ext。用这个方法，可以返回与一个符合该格式的虚拟路径相对应的真实路径的 String。这个真实路径的格式应该适合于运行这个 Servlet 引擎的计算机（包括其相应的路径解析器）。

不管是什么原因，如果这一从虚拟路径转换成实际路径的过程不能执行，该方法将会返回一个空值。

8、getResource

public URL getResource(String uripath);

返回一个 URL 对象，该对象反映位于给定的 URL 地址（格式：/dir/dir/filename.ext）的 Servlet 环境对象已知的资源。无论 URLStreamHandlers 对于访问给定的环境是不是必须的，Servlet 引擎都必须执行。如果给定的路径的 Servlet 环境没有已知的资源，该方法会返回一个空值。

这个方法和 java.lang.Class 的 getResource 方法不完全相同。java.lang.Class 的 getResource 方法通过装载类来寻找资源。而这个方法允许服务器产生环境变量给任何资源的任何 Servlet，而不必依赖于装载类、特定区域等等。

9、getResourceAsStream

public InputStream getResourceAsStream(String uripath);

返回一个 InputStream 对象，该对象引用指定的 URL 的 Servlet 环境对象的内容。如果没找到 Servlet 环境变量，就会返回空值，URL 路径应该具有这种格式：/dir/dir/filename.ext。

这个方法是一个通过 getResource 方法获得 URL 对象的方便的途径。请注意，当你使用这个方法时，meta-information（例如内容长度、内容类型）会丢失。

10、getRequestDispatcher

public RequestDispatcher getRequestDispatcher(String uripath);

如果这个指定的路径下能够找到活动的资源(例如一个 Servlet，JSP 页面，CGI 等等)就返回一个特定 URL 的 RequestDispatcher 对象，否则，就返回一个空值，Servlet 引擎负责用一个 request dispatcher 对象封装目标路径。这个 request dispatcher 对象可以用来完全请求的传送。

11、getServerInfo

public String getServerInfo();

返回一个 String 对象，该对象至少包括 Servlet 引擎的名字和版本号。

12、log

public void log(String msg);

public void log(String msg, Throwable t);

public void log(Exception exception, String msg); // 这种用法将被取消

写指定的信息到一个 Servlet 环境对象的 log 文件中。被写入的 log 文件由 Servlet 引擎

指定，但是通常这是一个事件 log。当这个方法被一个异常调用时，log 中将包括堆栈跟踪。

13、setAttribute

public void setAttribute(String name, Object o);

给予 Servlet 环境对象中你所指定的对象一个名称。

14、removeAttribute

public void removeAttribute(String name);

从指定的 Servlet 环境对象中删除一个属性。

注：以下几个方法将被取消

15、getServlet

public Servlet getServlet(String name) throws ServletException;

最初用来返回一个指定名称的 Servlet，如果没找到就返回一个空值。如果这个 Servlet 能够返回，这就意味着它已经被初始化，而且已经可以接受 service 请求。这是一个危险的方法。当调用这个方法时，可能并不知道 Servlet 的状态，这就可能导致有关服务器状态的问题。而允许一个 Servlet 访问其他 Servlet 的这个方法也同样的危险。

现在这个方法返回一个空值，为了保持和以前版本的兼容性，现在这个方法还没有被取消。在以后的 API 版本中，该方法将被取消。

16、getServletNames

public Enumeration getServletNames();

最初用来返回一个 String 对象的列表，该列表表示了在这个 Servlet 环境下所有已知的 Servlet 对象名。这个列表总是包含这个 Servlet 自身。

基于与上一个方法同样的理由，这也是一个危险的方法。

现在这个方法返回一个空的列表。为了保持和以前版本的兼容性，现在这个方法还没有被取消。在以后的 API 版本中，该方法将被取消。

17、getServlets

public Enumeration getServlets();

最初用来返回在这个 Servelet 环境下所有已知的 Servlet 对象的列表。这个列表总是包含这个 Servlet 自身。

基于与 getServlet 方法同样的理由，这也是一个危险的方法。

现在这个方法返回一个空的列表。为了保持和以前版本的兼容性，现在这个方法还没有被取消。在以后的 API 版本中，该方法将被取消。

五、ServletRequest 接口

定义

public interface ServletRequest

定义一个 Servlet 引擎产生的对象，通过这个对象，Servlet 可以获得客户端请求的数据。这个对象通过读取请求体的数据提供包括参数的名称、值和属性以及输入流的所有数据。

方法

1、getAttribute

public Object getAttribute(String name);

返回请求中指定属性的值，如果这个属性不存在，就返回一个空值。这个方法允许访问一些不提供给这个接口中其他方法的请求信息以及其他 Servlet 放置在这个请求对象内的数据。

2、getAttributeNames

public Enumeration getAttributeNames();

返回包含在这个请求中的所有属性名的列表。

3、getCharacterEncoding

public String getCharacterEncoding();

返回请求中输入内容的字符编码类型，如果没有定义字符编码类型就返回空值。

4、getContentLength

public int getContentLength();

请求内容的长度，如果长度未知就返回-1。

5、getContentType

public String getContentType();

返回请求数据体的 MIME 类型，如果类型未知返回空值。

6、getInputStream

public ServletInputStream getInputStream() throws IOException;

返回一个输入流用来从请求体读取二进制数据。如果在此之前已经通过 getReader 方法获得了要读取的结果，这个方法会抛出一个 IllegalStateException。

7、getParameter

public String getParameter(String name);

以一个 String 返回指定的参数的值，如果这个参数不存在返回空值。例如，在一个 HTTP Servlet 中，这个方法会返回一个指定的查询语句产生的参数的值或一个被提交的表单中的参数值。如果一个参数名对应着几个参数值，这个方法只能返回通过 getParameterValues 方法返回的数组中的第一个值。因此，如果这个参数有（或者可能有）多个值，你只能使用 getParameterValues 方法。

8、getParameterNames

public Enumeration getParameterNames();

返回所有参数名的 String 对象列表，如果没有输入参数，该方法返回一个空值。

9、getParameterValues

public String[] getParameterValues(String name);

通过一个 String 对象的数组返回指定参数的值，如果这个参数不存在，该方法返回一个空值。

10、getProtocol

public String getProtocol();

返回这个请求所用的协议，其形式是协议/主版本号.次版本号。例如对于一个 HTTP1.0 的请求，该方法返回 HTTP/1.0。

11、getReader

public BufferedReader getReader() throws IOException;

这个方法返回一个 buffered reader 用来读取请求体的实体，其编码方式依照请求数据的编码方式。如果这个请求的输入流已经被 getInputStream 调用获得，这个方法会抛出一个 IllegalStateException。

12、getRemoteAddr

public String getRemoteAddr();

返回发送请求者的 IP 地址。

13、getRemoteHost

public String getRemoteHost();

返回发送请求者的主机名称。如果引擎不能或者选择不解析主机名（为了改善性能），这个方法会直接返回 IP 地址。

14、getScheme

public String getScheme();

返回请求所使用的 URL 的模式。例如，对于一个 HTTP 请求，这个模式就是 http。

15、getServerName

public String getServerName();

返回接收请求的服务器的主机名。

16、getServerPort

public int getServerPort();

返回接收请求的端口号。

17、setAttribute

public void setAttribute(String name, Object object);

这个方法在请求中添加一个属性，这个属性可以被其他可以访问这个请求对象的对象（例如一个嵌套的 Servlet）使用。

注：以下方法将被取消

getRealPath

public String getRealPath(String path);

返回与虚拟路径相对应的真实路径，如果因为某种原因，这一过程不能进行，该方法将返回一个空值。

这个方法和 ServletContext 接口中的 getRealPath 方法重复。在 2.1 版中，ServletContext 接口将阐明一个 Servlet 所能用的所有的路径的映射。该方法执行的结果将会与 ServletContext 中 getRealPath 方法的结果完全一样。

六、ServletResponse 接口

定义

public interface ServletResponse

定义一个 Servlet 引擎产生的对象，通过这个对象，Servlet 对客户端的请求作出响应。这个响应应该是一个 MIME 实体，可能是一个 HTML 页、图象数据或其他 MIME 的格式。

方法

1、getCharacterEncoding

public String getCharacterEncoding();

返回 MIME 实体的字符编码。这个字符编码可以是指定的类型，也可以是与请求头域所反映的客户端所能接受的字符编码最匹配的类型。在 HTTP 协议中，这个信息被通过 Accept-Charset 传送到 Servlet 引擎。

有关字符编码和 MIME 的更多信息请参看 RFC 2047。

2、getOutputStream

public ServletOutputStream getOutputStream() throws IOException;

返回一个记录二进制的响应数据的输出流。

如果这个响应对象已经调用 getWriter，将会抛出 IllegalStateException。

3、getWriter

public PrintWriter getWriter throws IOException;

这个方法返回一个 PringWriter 对象用来记录格式化的响应实体。如果要反映使用的字符编码，必须修改响应的 MIME 类型。在调用这个方法之前，必须设定响应的 content 类型。

如果没有提供这样的编码类型，会抛出一个 UnsupportedEncodingException，如果这个响应对象已调用 getOutputStream，会抛出一个 getOutputStream。

4、setContentLength

public void setContentLength(int length);

设置响应的内容的长度，这个方法会覆盖以前对内容长度的设定。

为了保证成功地设定响应头的内容长度，在响应被提交到输出流之前必须调用这个方法。

5、setContentType

public void setContentType(String type);

这个方法用来设定响应的 content 类型。这个类型以后可能会在另外的一些情况下被隐式地修改，这里所说的另外的情况可能当服务器发现有必要的情况下对 MIME 的字符设置。

为了保证成功地设定响应头的 content 类型，在响应被提交到输出流之前必须调用这个方法。

七、SingleThreadModel 接口

定义

public interface SingleThreadModel;

这是一个空接口，它指定了系统如何处理对同一个 Servlet 的调用。如果一个 Servlet 被这个接口指定，那么在这个 Servlet 中的 service 方法中将不会有两个线程被同时执行。

Servlet 可以通过维持一个各自独立的 Servlet 实例池，或者通过只让 Servlet 的 service 中只有一个线程的方法来实现这个保证。

八、GenericServlet 类

public abstract class GenericServlet implements Servlet,

ServletConfig, Serializable;

这个类的存在使得编写 Servlet 更加方便。它提供了一个简单的方案，这个方案用来执行有关 Servlet 生命周期的方法以及在初始化时对 ServletConfig 对象和 ServletContext 对象进行说明。

方法

1、destroy

public void destroy();

在这里 destroy 方法不做任何其他的工作。

2、getInitParameter

public String getInitParameter(String name);

这是一个简便的途径，它将会调用 ServletConfig 对象的同名的方法。

3、getInitParameterNames

public Enumeration getInitParameterNames();

这是一个简便的途径，它将会调用 ServletConfig 对象的同名的方法。

4、getServletConfig

public ServletConfig getServletConfig();

返回一个通过这个类的 init 方法产生的 ServletConfig 对象的说明。

5、getServletContext

public ServletContext getServletContext();

这是一个简便的途径，它将会调用 ServletConfig 对象的同名的方法。

6、getServletInfo

public String getServletInfo();

返回一个反映 Servlet 版本的 String。

7、init

public void init() throws ServletException;

public void init(ServletConfig config) throws ServletException;

init(ServletConfig config)方法是一个对这个 Servlet 的生命周期进行初始化的简便的途径。

init()方法是用来让你对 GenericServlet 类进行扩充的，使用这个方法时，你不需要存储 config 对象，也不需要调用 super.init(config)。

init(ServletConfig config)方法会存储 config 对象然后调用 init()。如果你重载了这个方法，你必须调用 super.init(config)，这样 GenericServlet 类的其他方法才能正常工作。

8、log

public void log(String msg);

public void log(String msg, Throwable cause);

通过 Servlet content 对象将 Servlet 的类名和给定的信息写入 log 文件中。

9、service

public abstract void service(ServletRequest request, ServletResponse response) throws ServletException, IOException;

这是一个抽象的方法，当你扩展这个类时，为了执行网络请求，你必须执行它。

九、ServletInputStream 类

定义

public abstract class ServletInputStream extends InputStream

这个类定义了一个用来读取客户端的请求信息的输入流。这是一个 Servlet 引擎提供的抽象类。一个 Servlet 通过使用 ServletRequest 接口获得了对一个 ServletInputStream 对象的说明。

这个类的子类必须提供一个从 InputStream 接口读取有关信息的方法。

方法

1、readLine

public int readLine(byte[] b, int off, int len) throws IOException;

从输入流的指定的偏移量开始将指定长度的字节读入到指定的数组中。如果该行所有请求的内容都已被读取，这个读取的过程将结束。如果是遇到了新的一行，新的一行的首个字符也将被读入到数组中。

十、ServletOutputStream 类

定义

public abstract class ServletOutputStream extends OutputStream

这是一个由 Servlet 引擎使用的抽象类。Servlet 通过使用 ServletResponse 接口的使用获得了对一个这种类型的对象的说明。利用这个输出流可以将数据返回到客户端。

这个类的子类必须提供一个向 OutputStream 接口写入有关信息的方法。

在这个接口中，当一个刷新或关闭的方法被调用时。所有数据缓冲区的信息将会被发送到客户端，也就是说响应被提交了。请注意，关闭这种类型的对象时不一定要关闭隐含的 socket 流。

方法

1、print

public void print(String s) throws IOException;
public void print(boolean b) throws IOException;
public void print(char c) throws IOException;
public void print(int i) throws IOException;
public void print(long l) throws IOException;
public void print(float f) throws IOException;
public void print(double d) throws IOException;
输出变量到输出流中
2、println
public void println() throws IOException;
public void println(String s) throws IOException;
public void println(boolean b) throws IOException;
public void println(char c) throws IOException;
public void println(int i) throws IOException;
public void println(long l) throws IOException;
public void println(float f) throws IOException;
public void println(double d) throws IOException;
输出变量到输出流中，并增加一个回车换行符

十一、ServletException 类
定义
public class ServletException extends Exception
当 Servlet 遇到问题时抛出的一个异常。
构造函数
public ServletException();
public ServletException(String message);
public ServletException(String message, Throwable cause);
public ServletException(Throwable cause);
构造一个新的 ServletException，如果这个构造函数包括一个 Throwable 参数，这个 Throwable 对象将被作为可能抛出这个异常的原因。
方法
1、getRootCause
public Throwable getRootCause();
如果配置了抛出这个异常的原因，这个方法将返回这个原因，否则返回一个空值。

十二、UnavailableException 类
定义
public class UnavailableException extends ServletException
不论一个 Servlet 是永久地还是临时地无效，都会抛出这个异常。Servlet 会记录这个异常以及 Servlet 引擎所要采取的相应措施。
临时的无效是指 Servlet 在某一时间由于一个临时的问题而不能处理请求。例如，在另一个不同的应用层的服务（可能是数据库）无法使用。这个问题可能会自行纠正或者需要采取其他的纠正措施。
永久的无效是指除非管理员采取措施，这个 Servlet 将不能处理客户端的请求。例如，

这个 Servlet 配置信息丢失或 Servlet 的状态被破坏。

Servlet 引擎可以安全地处理包括永久无效在内的这两种异常，但是对临时无效的正常处理可以使得 Servlet 引擎更健壮。特别的，这时对 Servlet 的请求只是被阻止（或者是被延期）一段时间，这显然要比在 service 自己重新启动前完全拒绝请求更为科学。

构造函数

public UnavailableException(Servlet servlet, String message);

public UnavailableException(int seconds, Servlet servlet,

String message);

构造一个包含指定的描述信息的新的异常。如果这个构造函数有一个关于秒数的参数，这将给出 Servlet 发生临时无效后，能够重新处理请求的估计时间。如果不包含这个参数，这意味着这个 Servlet 永久无效。

方法

1、getServlet

public Servlet getServlet();

返回报告无效的 Servlet。这被 Servlet 引擎用来识别受到影响的 Servlet。

2、getUnavailableSeconds

public int getUnavailableSeconds();

返回 Servlet 预期的无效时间，如果这个 Servlet 是永久无效，返回-1。

3、isPermanent

public boolean isPermanent();

如果这个 Servlet 永久无效，返回布尔值 true，指示必须采取一些管理行动以使得这个 Servlet 可用。

3 。软件包

javax.servlet.http

所包含的接口：HttpServletRequest；HttpServletResponse；HttpSession；HttpSessionBindingListener；HttpSessionContext。

所包含的类：Cookie；HttpServlet；HttpSessionBindingEvent；HttpUtils。

一、HttpServletRequest 接口

定义\

public interface HttpServletRequest extends ServletRequest;

用来处理一个对 Servlet 的 HTTP 格式的请求信息。

方法

1、getAuthType

public String getAuthType();

返回这个请求的身份验证模式。

2、getCookies

public Cookie[] getCookies();

返回一个数组，该数组包含这个请求中当前的所有 cookie。如果这个请求中没有 cookie，返回一个空数组。

3、getDateHeader

public long getDateHeader(String name);

返回指定的请求头域的值，这个值被转换成一个反映自 1970-1-1 日（GMT）以来的精确到毫秒的长整数。

如果头域不能转换，抛出一个 IllegalArgumentException。如果这个请求头域不存在，这个方法返回-1。

4、getHeader

public String getHeader(String name);

返回一个请求头域的值。（译者注：与上一个方法不同的是，该方法返回一个字符串）

如果这个请求头域不存在，这个方法返回-1。

5、getHeaderNames

public Enumeration getHeaderNames();

该方法返回一个 String 对象的列表，该列表反映请求的所有头域名。

有的引擎可能不允许通过这种方法访问头域，在这种情况下，这个方法返回一个空的列表。

6、getIntHeader

public int getIntHeader(String name);

返回指定的请求头域的值，这个值被转换成一个整数。

如果头域不能转换，抛出一个 IllegalArgumentException。如果这个请求头域不存在，这个方法返回-1。

7、getMethod

public String getMethod();

返回这个请求使用的 HTTP 方法（例如：GET、POST、PUT）

8、getPathInfo

public String getPathInfo();

这个方法返回在这个请求的 URL 的 Servlet 路径之后的请求 URL 的额外的路径信息。如果这个请求 URL 包括一个查询字符串，在返回值内将不包括这个查询字符串。这个路径在返回之前必须经过 URL 解码。如果在这个请求的 URL 的 Servlet 路径之后没有路径信息。这个方法返回空值。

9、getPathTranslated

public String getPathTranslated();

这个方法获得这个请求的 URL 的 Servlet 路径之后的额外的路径信息，并将它转换成一个真实的路径。在进行转换前，这个请求的 URL 必须经过 URL 解码。如果在这个 URL 的 Servlet 路径之后没有附加路径信息。这个方法返回空值。

10、getQueryString

public String getQueryString();

返回这个请求 URL 所包含的查询字符串。一个查询字串符在一个 URL 中由一个“？”引出。如果没有查询字符串，这个方法返回空值。

11、getRemoteUser

public String getRemoteUser

返回作了请求的用户名，这个信息用来作 HTTP 用户论证。

如果在请求中没有用户名信息，这个方法返回空值。

12、getRequestedSessionId

public String getRequestedSessionId();

返回这个请求相应的 session id。如果由于某种原因客户端提供的 session id 是无效的，这个 session id 将与在当前 session 中的 session id 不同，与此同时，将建立一个新的

session。

如果这个请求没与一个 session 关联，这个方法返回空值。

13、getRequestURI

public String getRequestURI();

从 HTTP 请求的第一行返回请求的 URL 中定义被请求的资源的部分。如果有一个查询字符串存在，这个查询字符串将不包括在返回值当中。例如，一个请求通过 /catalog/books?id=1 这样的 URL 路径访问，这个方法将返回/catalog/books。这个方法的返回值包括了 Servlet 路径和路径信息。

如果这个 URL 路径中的的一部分经过了 URL 编码，这个方法的返回值在返回之前必须经过解码。

14、getServletPath

public String getServletPath();

这个方法返回请求 URL 反映调用 Servlet 的部分。例如，一个 Servlet 被映射到 /catalog/summer 这个 URL 路径，而一个请求使用了/catalog/summer/casual 这样的路径。所谓的反映调用 Servlet 的部分就是指/catalog/summer。

如果这个 Servlet 不是通过路径匹配来调用。这个方法将返回一个空值。

15、getSession

public HttpSession getSession();

public HttpSession getSession(boolean create);

返回与这个请求关联的当前的有效的 session。如果调用这个方法时没带参数，那么在没有 session 与这个请求关联的情况下，将会新建一个 session。如果调用这个方法时带入了一个布尔型的参数，只有当这个参数为真时，session 才会被建立。

为了确保 session 能够被完全维持。Servlet 开发者必须在响应被提交之前调用该方法。

如果带入的参数为假，而且没有 session 与这个请求关联。这个方法会返回空值。

16、isRequestedSessionIdValid

public boolean isRequestedSessionIdValid();

这个方法检查与此请求关联的 session 当前是不是有效。如果当前请求中使用的 session 无效，它将不能通过 getSession 方法返回。

17、isRequestedSessionIdFromCookie

public boolean isRequestedSessionIdFromCookie();

如果这个请求的 session id 是通过客户端的一个 cookie 提供的，该方法返回真，否则返回假。

18、isRequestedSessionIdFromURL

public boolean isRequestedSessionIdFromURL();

如果这个请求的 session id 是通过客户端的 URL 的一部分提供的，该方法返回真，否则返回假。请注意此方法与 isRequestedSessionIdFromUrl 在 URL 的拼写上不同。

以下方法将被取消\

19、isRequestedSessionIdFromUrl

public boolean isRequestedSessionIdFromUrl();

该方法被 isRequestedSessionIdFromURL 代替。

二、HttpServletResponse 接口

定义\

public interface HttpServletResponse extends ServletResponse

描述一个返回到客户端的 HTTP 回应。这个接口允许 Servlet 程序员利用 HTTP 协议规定的头信息。

成员变量

public static final int SC_CONTINUE = 100;

public static final int SC_SWITCHING_PROTOCOLS = 101;

public static final int SC_OK = 200;

```
public static final int SC_CREATED = 201;

public static final int SC_ACCEPTED = 202;

public static final int SC_NON_AUTHORITATIVE_INFORMATION = 203;

public static final int SC_NO_CONTENT = 204;

public static final int SC_RESET_CONTENT = 205;

public static final int SC_PARTIAL_CONTENT = 206;

public static final int SC_MULTIPLE_CHOICES = 300;

public static final int SC_MOVED_PERMANENTLY = 301;

public static final int SC_MOVED_TEMPORARILY = 302;

public static final int SC_SEE_OTHER = 303;

public static final int SC_NOT_MODIFIED = 304;

public static final int SC_USE_PROXY = 305;

public static final int SC_BAD_REQUEST = 400;

public static final int SC_UNAUTHORIZED = 401;

public static final int SC_PAYMENT_REQUIRED = 402;

public static final int SC_FORBIDDEN = 403;

public static final int SC_NOT_FOUND = 404;

public static final int SC_METHOD_NOT_ALLOWED = 405;

public static final int SC_NOT_ACCEPTABLE = 406;

public static final int SC_PROXY_AUTHENTICATION_REQUIRED = 407;

public static final int SC_REQUEST_TIMEOUT = 408;

public static final int SC_CONFLICT = 409;

public static final int SC_GONE = 410;

public static final int SC_LENGTH_REQUIRED = 411;

public static final int SC_PRECONDITION_FAILED = 412;

public static final int SC_REQUEST_ENTITY_TOO_LARGE = 413;
```

public static final int SC_REQUEST_URI_TOO_LONG = 414;

public static final int SC_UNSUPPORTED_MEDIA_TYPE = 415;

public static final int SC_INTERNAL_SERVER_ERROR = 500;

public static final int SC_NOT_IMPLEMENTED = 501;

public static final int SC_BAD_GATEWAY = 502;

public static final int SC_SERVICE_UNAVAILABLE = 503;

public static final int SC_GATEWAY_TIMEOUT = 504;

public static final int SC_HTTP_VERSION_NOT_SUPPORTED = 505;

以上 HTTP 产状态码是由 HTTP/1.1 定义的。

方法

1、addCookie

public void addCookie(Cookie cookie);

在响应中增加一个指定的 cookie。可多次调用该方法以定义多个 cookie。为了设置适当的头域，该方法应该在响应被提交之前调用。

2、containsHeader

public boolean containsHeader(String name);

检查是否设置了指定的响应头。

3、encodeRedirectURL

public String encodeRedirectURL(String url);

对 sendRedirect 方法使用的指定 URL 进行编码。如果不需要编码，就直接返回这个 URL。之所以提供这个附加的编码方法，是因为在 redirect 的情况下，决定是否对 URL 进行编码的规则和一般情况有所不同。所给的 URL 必须是一个绝对 URL。相对 URL 不能被接收，会抛出一个 IllegalArgumentException。

所有提供给 sendRedirect 方法的 URL 都应通过这个方法运行，这样才能确保会话跟踪能够在所有浏览器中正常运行。

4、encodeURL

public String encodeURL(String url);

对包含 session ID 的 URL 进行编码。如果不需要编码，就直接返回这个 URL。Servlet 引擎必须提供 URL 编码方法，因为在有些情况下，我们将不得不重写 URL，例如，在响应对应的请求中包含一个有效的 session，但是这个 session 不能被非 URL 的（例如 cookie）的手段来维持。

所有提供给 Servlet 的 URL 都应通过这个方法运行，这样才能确保会话跟踪能够在所有浏览器中正常运行。

5、sendError

public void sendError(int statusCode) throws IOException;

public void sendError(int statusCode, String message) throws

IOException;

用给定的状态码发给客户端一个错误响应。如果提供了一个 message 参数，这将作为响应体的一部分被发出，否则，服务器会返回错误代码所对应的标准信息。

调用这个方法后，响应立即被提交。在调用这个方法后，Servlet 不会再有更多的输出。

6、sendRedirect

public void sendRedirect(String location) throws IOException;

使用给定的路径，给客户端发出一个临时转向的响应（SC_MOVED_TEMPORARILY）。给定的路径必须是绝对 URL。相对 URL 将不能被接收，会抛出一个 IllegalArgumentException。

这个方法必须在响应被提交之前调用。调用这个方法后，响应立即被提交。在调用这个方法后，Servlet 不会再有更多的输出。

7、setDateHeader

public void setDateHeader(String name, long date);

用一个给定的名称和日期值设置响应头，这里的日期值应该是反映自 1970-1-1 日（GMT）以来的精确到毫秒的长整数。如果响应头已经被设置，新的值将覆盖当前的值。

8、setHeader

public void setHeader(String name, String value);

用一个给定的名称和域设置响应头。如果响应头已经被设置，新的值将覆盖当前的值。

9、setIntHeader

public void setIntHeader(String name, int value);

用一个给定的名称和整形值设置响应头。如果响应头已经被设置，新的值将覆盖当前的值。

10、setStatus

public void setStatus(int statusCode);

这个方法设置了响应的状态码，如果状态码已经被设置，新的值将覆盖当前的值。

以下的几个方法将被取消\

11、encodeRedirectUrl

public String encodeRedirectUrl(String url);

该方法被 encodeRedirectURL 取代。

12、encodeUrl

public String encodeUrl(String url);

该方法被 encodeURL 取代。

13、setStatus

public void setStatus(int statusCode, String message);

这个方法设置了响应的状态码，如果状态码已经被设置，新的值将覆盖当前的值。如果提供了一个 message，它也将会被作为响应体的一部分被发送。

三、HttpSession 接口

定义\

public interface HttpSession

这个接口被 Servlet 引擎用来实现在 HTTP 客户端和 HTTP 会话两者的关联。这种关联可能在多外连接和请求中持续一段给定的时间。session 用来在无状态的 HTTP 协议下越过

多个请求页面来维持状态和识别用户。

一个 session 可以通过 cookie 或重写 URL 来维持。

方法

1、getCreationTime

public long getCreationTime();

返回建立 session 的时间，这个时间表示为自 1970-1-1 日（GMT）以来的毫秒数。

2、getId

public String getId();

返回分配给这个 session 的标识符。一个 HTTP session 的标识符是一个由服务器来建立和维持的唯一的字符串。

3、getLastAccessedTime

public long getLastAccessedTime();

返回客户端最后一次发出与这个 session 有关的请求的时间，如果这个 session 是新建立的，返回-1。这个时间表示为自 1970-1-1 日（GMT）以来的毫秒数。

4、getMaxInactiveInterval

public int getMaxInactiveInterval();

返加一个秒数，这个秒数表示客户端在不发出请求时，session 被 Servlet 引擎维持的最长时间。在这个时间之后，Servlet 引擎可能被 Servlet 引擎终止。如果这个 session 不会被终止，这个方法返回-1。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

5、getValue

public Object getValue(String name);

返回一个以给定的名字绑定到 session 上的对象。如果不存在这样的绑定，返回空值。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

6、getValueNames

public String[] getValueNames();

以一个数组返回绑定到 session 上的所有数据的名称。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

7、invalidate

public void invalidate();

这个方法会终止这个 session。所有绑定在这个 session 上的数据都会被清除。并通过 HttpSessionBindingListener 接口的 valueUnbound 方法发出通告。

8、isNew

public boolean isNew();

返回一个布尔值以判断这个 session 是不是新的。如果一个 session 已经被服务器建立但是还没有收到相应的客户端的请求，这个 session 将被认为是新的。这意味着，这个客户端还没有加入会话或没有被会话公认。在他发出下一个请求时还不能返回适当的 session 认证信息。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

9、putValue

public void putValue(String name, Object value);

以给定的名字，绑定给定的对象到 session 中。已存在的同名的绑定会被重置。这时会调用 HttpSessionBindingListener 接口的 valueBound 方法。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

10、removeValue

public void removeValue(String name);

取消给定名字的对象在 session 上的绑定。如果未找到给定名字的绑定的对象，这个方法什么出不做。 这时会调用 HttpSessionBindingListener 接口的 valueUnbound 方法。

当 session 无效后再调用这个方法会抛出一个 IllegalStateException。

11、setMaxInactiveInterval

public int setMaxInactiveInterval(int interval);

设置一个秒数，这个秒数表示客户端在不发出请求时，session 被 Servlet 引擎维持的最

长时间。

以下这个方法将被取消\

12、getSessionContext

public HttpSessionContext getSessionContext();

返回 session 在其中得以保持的环境变量。这个方法和其他所有 HttpSessionContext 的方法一样被取消了。

四、HttpSessionBindingListener 接口

定义\

public interface HttpSessionBindingListener

这个对象被加入到 HTTP 的 session 中，执行这个接口会通告有没有什么对象被绑定到这个 HTTP session 中或被从这个 HTTP session 中取消绑定。

方法

1、valueBound

public void valueBound(HttpSessionBindingEvent event);

当一个对象被绑定到 session 中，调用此方法。HttpSession.putValue 方法被调用时，Servlet 引擎应该调用此方法。

2、valueUnbound

public void valueUnbound(HttpSessionBindingEvent event);

当一个对象被从 session 中取消绑定，调用此方法。HttpSession.removeValue 方法被调用时，Servlet 引擎应该调用此方法。

五、HttpSessionContext 接口

定义\

此接口将被取消\

public interface HttpSessionContext

这个对象是与一组 HTTP session 关联的单一的实体。

这个接口由于安全的原因被取消，它出现在目前的版本中仅仅是为了兼容性的原因。这个接口的方法将模拟以前的版本的定义返回相应的值。

方法

1、getSession

public HttpSession getSession(String sessionId);

当初用来返回与这个 session id 相关的 session。现在返回空值。

2、getIds

public Enumeration getIds();

当初用来返回这个环境下所有 session id 的列表。现在返回空的列表。

六、Cookie 类\

定义\

public class Cookie implements Cloneable

这个类描述了一个 cookie，有关 cookie 的定义你可以参照 Netscape Communications Corporation 的说明，也可以参照 RFC 2109。

构造函数

public Cookie(String name, String value);

用一个 name-value 对定义一个 cookie。这个 name 必须能被 HTTP/1.1 所接受。

以字符$开头的 name 被 RFC 2109 保留。

给定的 name 如果不能被 HTTP/1.1 所接受，该方法抛出一个 IllegalArgumentException。

方法

1、getComment

public String getComment();

返回描述这个 cookie 目的的说明，如果未定义这个说明，返回空值。

2、getDomain

public String getDomain();

返回这个 cookie 可以出现的区域，如果未定义区域，返回空值。

3、getMaxAge

public int getMaxAge();

这个方法返回这个 cookie 指定的最长存活时期。如果未定义这个最长存活时期，该方法返回-1。

4、getName

public String getName();

该方法返回 cookie 名。

5、getPath

public String getPath();

返回这个 cookie 有效的所有 URL 路径的前缀，如果未定义，返回空值。

6、getSecure

public boolean getSecure();

如果这个 cookie 只通过安全通道传输返回真，否则返回假。

7、getValue

public String getValue();

该方法返回 cookie 的值。

8、getVersion

public int getVersion();

返回 cookie 的版本。版本 1 由 RFC 2109 解释。版本 0 由 Netscape Communications Corporation 的说明解释。新构造的 cookie 默认使用版本 0。

9、setComment

public void setComment(String purpose);

如果一个用户将这个 cookie 提交给另一个用户，必须通过这个说明描述这个 cookie 的目的。版本 0 不支持这个属性。

10、setDomain

public void setDomain(String pattern);

这个方法设置 cookie 的有效域的属性。这个属性指定了 cookie 可以出现的区域。一个有效域以一个点开头（.foo.com），这意味着在指定的域名解析系统的区域中（可能是 www.foo.com 但不是 a.b.foo.com）的主机可以看到这个 cookie。默认情况是，cookie 只能返回保存它的主机。

11、setMaxAge

public void setMaxAge(int expiry);

这个方法设定这个 cookie 的最长存活时期。在该存活时期之后，cookie 会被终目。负数表示这个 cookie 不会生效，0 将从客户端删除这个 cookie。

12、setPath

public void setPath(String uri);

这个方法设置 cookie 的路径属性。客户端只能向以这个给定的路径 String 开头的路径返回 cookie。

13、setSecure

public void setSecure(boolean flag);

指出这个 cookie 只能通过安全通道（例如 HTTPS）发送。只有当产生这个 cookie 的服务器使用安全协议发送这个 cookie 值时才能这样设置。

14、setValue

public void setValue(String newValue);

设置这个 cookie 的值，对于二进制数据采用 BASE64 编码。

版本 0 不能使用空格、{}、()、=、，、、“”、/、?、@、：以及；。

15、setVersion

public void setVersion(int v);

设置 cookie 的版本号

七、HttpServlet 类\

定义\

public class HttpServlet extends GenericServlet implements

Serializable

这是一个抽象类，用来简化 HTTP Servlet 写作的过程。它是 GenericServlet 类的扩充，提供了一个处理 HTTP 协议的框架。

在这个类中的 service 方法支持例如 GET、POST 这样的标准的 HTTP 方法。这一支持过程是通过分配他们到适当的方法（例如 doGet、doPost）来实现的。

方法

1、doDelete

protected void doDelete(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP DELETE 操作。这个操作允许客户端请求从服务器上删除 URL。这一操作可能有负面影响，对此用户就负起责任。

这一方法的默认执行结果是返回一个 HTTP BAD_REQUEST 错误。当你要处理 DELETE 请求时，你必须重载这一方法。

2、doGet

protected void doGet(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP GET 操作。这个操作允许客户端简单地从一个 HTTP 服务器“获得”资源。对这个方法的重载将自动地支持 HEAD 方法。

GET 操作应该是安全而且没有负面影响的。这个操作也应该可以安全地重复。

这一方法的默认执行结果是返回一个 HTTP BAD_REQUEST 错误。

3、doHead

protected void doHead(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP HEAD 操作。默认的情况是，这个操作会按照一个无条件的 GET 方法来执行，该操作不向客户端返回任何数据，而仅仅是返回包含内容长度的头信息。

与 GET 操作一样，这个操作应该是安全而且没有负面影响的。这个操作也应该可以安全地重复。

这个方法的默认执行结果是自动处理 HTTP HEAD 操作，这个方法不需要被一个子类执行。

4、doOptions

protected void doOptions(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP OPTION 操作。这个操作自动地决定支持哪一种 HTTP 方法。例如，一个 Servlet 写了一个 HttpServlet 的子类并重载了 doGet 方法，doOption 会返回下面的头：

Allow: GET,HEAD,TRACE,OPTIONS

你一般不需要重载这个方法。

5、doPost

protected void doPost(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP POST 操作。这个操作包含请求体的数据，Servlet 应该按照他行事。

这个操作可能有负面影响。例如更新存储的数据或在线购物。

这一方法的默认执行结果是返回一个 HTTP BAD_REQUEST 错误。当你要处理 POST 操作时，你必须在 HttpServlet 的子类中重载这一方法。

6、doPut

protected void doPut(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP PUT 操作。这个操作类似于通过 FTP 发送文件。

这个操作可能有负面影响。例如更新存储的数据或在线购物。

这一方法的默认执行结果是返回一个 HTTP BAD_REQUEST 错误。当你要处理 PUT 操作时，你必须在 HttpServlet 的子类中重载这一方法。

7、doTrace

protected void doTrace(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

被这个类的 service 方法调用，用来处理一个 HTTP TRACE 操作。这个操作的默认执行结果是产生一个响应，这个响应包含一个反映 trace 请求中发送的所有头域的信息。

当你开发 Servlet 时，在多数情况下你需要重载这个方法。

8、getLastModified

protected long getLastModified(HttpServletRequest request);

返回这个请求实体的最后修改时间。为了支持 GET 操作，你必须重载这一方法，以精确地反映最后修改的时间。这将有助于浏览器和代理服务器减少装载服务器和网络资源，从而更加有效地工作。返回的数值是自 1970-1-1 日（GMT）以来的毫秒数。

默认的执行结果是返回一个负数，这标志着最后修改时间未知，它也不能被一个有条件的 GET 操作使用。

9、service

protected void service(HttpServletRequest request,

HttpServletResponse response) throws ServletException,

IOException;

public void service(ServletRequest request, ServletResponse response)

throws ServletException, IOException;

这是一个 Servlet 的 HTTP-specific 方案，它分配请求到这个类的支持这个请求的其他方法。

当你开发 Servlet 时，在多数情况下你不必重载这个方法。

八、HttpSessionBindingEvent 类\

定义\

public class HttpSessionBindingEvent extends EventObject

这个事件是在监听到 HttpSession 发生绑定和取消绑定的情况时连通 HttpSessionBindingListener 的。这可能是一个 session 被终止或被认定无效的结果。

事件源是 HttpSession.putValue 或 HttpSession.removeValue。

构造函数

public HttpSessionBindingEvent(HttpSession session, String name);

通过引起这个事件的 Session 和发生绑定或取消绑定的对象名构造一个新的 HttpSessionBindingEvent。

方法

1、getName

public String getName();

返回发生绑定和取消绑定的对象的名字。

2、getSession

public HttpSession getSession();

返回发生绑定和取消绑定的 session 的名字。

九、HttpUtils 类\

定义\

public class HttpUtils

收集 HTTP Servlet 使用的静态的有效的方法。

方法

1、getRequestURL

public static StringBuffer getRequestURL(HttpServletRequest

request);

在服务器上重建客户端用来建立请求的 URL。这个方法反映了不同的协议（例如 http 和 https）和端口，但不包含查询字符串。

这个方法返回一个 StringBuffer 而不是一个 String，这样 URL 可以被 Servlet 开发者有效地修改。

2、parsePostData

public static Hashtable parsePostData(int len,

ServletInputstream in);

解析一个包含 MIME 类型 application/x-www-form-urlencoded 的数据的流，并创建一个具有关键值-数据对的 hash table。这里的关键值是字符串，数据是该字符串所对应的值的列表。一个关键值可以在 POST 的数据中出现一次或多次。这个关键值每出现一次，它的相应的值就被加入到 hash table 中的字符串所对应的值的列表中。

从 POST 数据读出的数据将经过 URL 解码，+将被转换为空格以十六进制传送的数据（例如%xx）将被转换成字符。

当 POST 数据无效时，该方法抛出一个 IllegalArgumentException。

3、parseQueryString

public static Hashtable parseQueryString(String s);

解析一个查询字符串，并创建一个具有关键值-数据对的 hash table。这里的数据是该字

符串所对应的值的列表。一个关键值可以出现一次或多次。这个关键值每出现一次，它的相应的值就被加入到 hash table 中的字符串所对应的值的列表中。

从查询字符串读出的数据将经过 URL 解码，+将被转换为空格以十六进制传送的数据（例如%xx）将被转换成字符。

当查询字符串无效时，该方法抛出一个 IllegalArgumentException。

4 。术语表

bytecode

字节码：由 Java 编译器和 Java 解释程序生成的机器代码。

cookie

由 Web 服务器建立的数据，该数据存储在用户的计算机上，提供了一个 Web 站点跟踪用户的参数并存储在用户自己硬盘上的方法。

HTTP

超文本传输协议。一个请求响应协议用来连接 WWW 服务器向客户端浏览器传输 HTML 页面。

输入流对象

一个对象，由 ServletInputStream 类定义，被 Servlet 用来从客户端读取请求。

映射

由 Servlet 实例和 Servlet 返回数据的 URL 组成的一对，例如，HelloServlet 和 /hello/index.html。

输出流对象

一个对象，由 ServletOutputStream class 类定义，被 Servlet 用来向客户端返回数据。

request dispatcher object

由 RequestDispatcher 接口定义的一个对象，用来从客户端接收请求，并将其发送到 Web 服务器上可用的其他资源（例如 Servlet、CGI、HTML 文件或 JSP 文件）。

sandboxed servlet

在一个安全性约束下运行的 Servlet。

servlet

一个小的，具有平台无关性的，没有图形用户界面的 Java 程序。它可以在许多方面扩充 Web 服务的功能。

servlet configuration object

ServletConfig 接口定义的一个对象，用来配置一个 Servlet。

servlet context object

ServletContext 接口定义的一个对象。给予 Servlet 有关 Servlet 引擎的信息。

servlet 引擎

由 Web 服务器提供商制作的一个环境，可以允许 Servlet 在具体的 Web 服务器上运行。

servlet 请求对象

由 ServletRequest 接口定义的一个对象，允许 Servlet 获得用关客户端请求的数据。

servlet response object

由 ServletResponse 接口定义的一个对象，允许 Servlet 作出响应。

servlet runner

Java Servlet Developer's Kit (JSDK)中的 sun.servlet.http.HttpServer 过程，它使得 Servlet 得以运行。

会话跟踪

在一个 Web 应用程序中，识别一个从同一个客户端发出的连续的唯一的请求的能力。

SSL

加密套接字协议层。一个安全协议，用来在 Iternet 上的客户端浏览器和服务器交换密钥和加密数据。

URI

统一资源标识。定义一个 Internet 地址，它是一个 URL 的超集。

URL

统一资源路径。这个地址定义了到达一个 WWW 上的文件的路线，通常由协议前缀、域名、目录名和文件名组成。