

Servlet

本文资料大多来自网上，但内容十分丰富，是学习servlet相当全面且有用的资料，希望对大家有所帮助。

若有疑问<ymkyve501@gmail.com>

目录

Servlet	1
什么是servlet	1
Servlet相关名称	1
Servlet作用	1
Servlet流程	2
正确认识Servlet	4
Servlet树	4
Servlet生命周期	5
Servlet与URL匹配	12
servlet转发	13
理解	14

什么是servlet

Servlet是一种服务器端的Java应用程序，具有独立于平台和协议的特性,可以生成动态的Web页面。它担当客户请求（Web浏览器或其他HTTP客户程序）与服务器响应（HTTP服务器上的数据库或应用程序）的中间层。**Servlet**是位于Web 服务器内部的服务器端的Java应用程序，与传统的从命令行启动的Java应用程序不同，**Servlet**由Web服务器进行加载，该Web服务器必须包含 支持Servlet的Java虚拟机。

Servlet相关名称

Web服务器

Web服务器，即Web Server，是Web（互联网）上的一台或多台机器。这些机器上部署了系统软件以及需要在网络中共享的信息。客户端可以通过HTTP协议访问部署到服务器上的资源。

应用服务器

应用服务器，即Application Server，是网络上的一台或多台机器。这些机器上部署了系统软件用以为客户端及其它的服务器比如Web服务器提供数据服务、业务处理服务等。常见的应用服务器包括文件服务器、打印服务器以及运行着EJB组件的EJB服务器等。

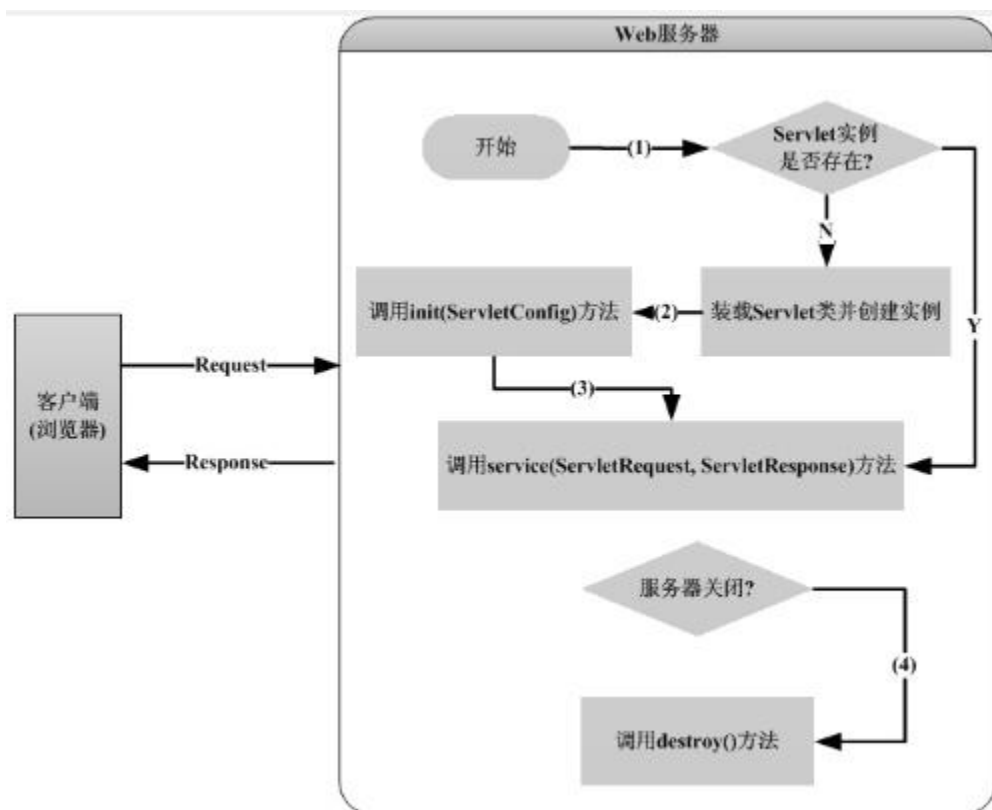
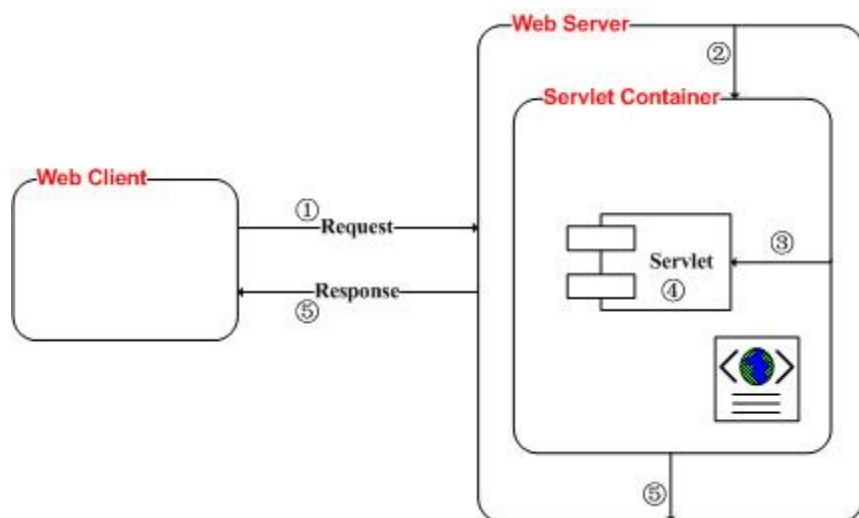
Servlet容器

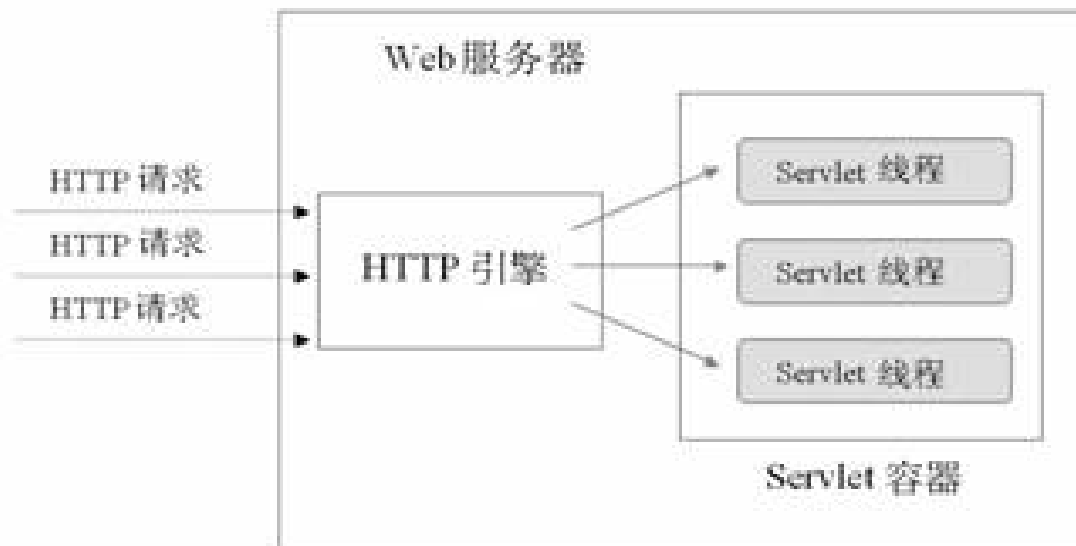
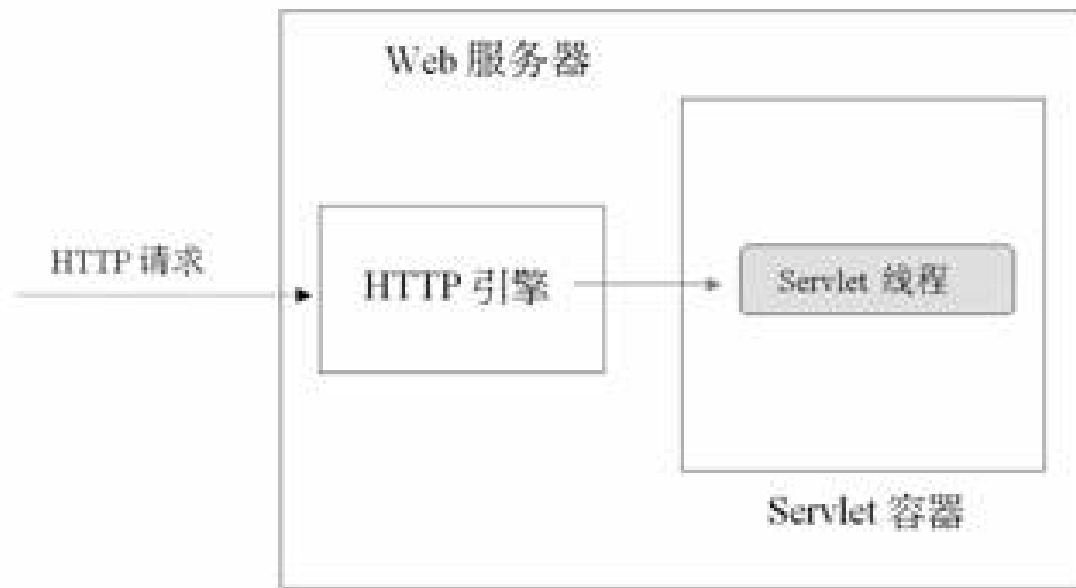
Servlet容器，即Servlet Container，是Web服务器或应用服务器的组成部分，它可以为Servlet提供请求和响应的服务，同时也负责管理Servlet的生命周期。Servlet容器也可以为Servlet提供其它的服务，比如资源服务、安全服务、线程的服务等。Servlet容器即可以集成到Web服务器或应用服务器中，也可以作为独立的程序嵌入到Web服务器或应用服务器中。

Servlet作用

Servlet规范通过规定一个编程的框架来达到扩展服务器功能的目的，采用请求-响应模式提供基于Web的服务。当客户机发送请求至服务器时，服务器将请求信息转发给Servlet，Servlet处理请求并生成响应内容并将其传给Web服务器，然后再由Web服务器将响应返回给客户端。

Servlet流程





说明:

- 客户端（通常为Web浏览器）向Web服务器发送一个基于HTTP协议的请求。
- Web服务器接收到该请求，并将请求交给Servlet容器处理。
- Servlet容器根据Servlet的配置来查找或创建Servlet的实例，并执行该Servlet，Servlet容器必须把客户端请求和响应封装成Servlet规范中规定的请求和响应对象传给Servlet。

- Servlet可以使用请求对象获取客户端的信息，比如IP地址、请求的参数等，以及执行特定的业务逻辑。Servlet可以使用响应对象向客户端发送业务数据及业务执行的结果。
- Servlet处理完该请求后，Servlet容器要保证Servlet的响应内容能够发送到客户端去（flush），最后返回到Web服务器。

正确认识Servlet

- Servlet不可以独立运行。
Servlet必须运行在Servlet容器中，由容器调用它的使用寿命方法。因此，Servlet不需要main方法。
- Servlet不是线程。
Servlet就是一个普通的Java对象，它实现了Servlet接口，并没有继承于Thread，因此Servlet不是线程。
当服务器接收到客户端请求时，它会在一个单独的线程中来执行Servlet的方法。

Servlet树



Servlet 实现的do方法



Servlet生命周期

Servlet的生命周期可以分为四个阶段，即装载类及创建实例阶段、初始化阶段、服务阶段和实例销毁阶段。下面针对每个阶段的编程任务及注意事项进行详细的说明。

(1) 装载类及创建实例

客户端向Web服务器发送一个请求，请求的协议及路径必须遵守如下的格式：

<http://serverip:port/application-path/resource-path>

其中，serverip为Web服务器的IP地址，也可以是域名，比如：192.168.0.1、202.196.152.115、

localhost、www.sina.com.cn等。port为Web服务器的服务端口，如果是80端口可以不写。

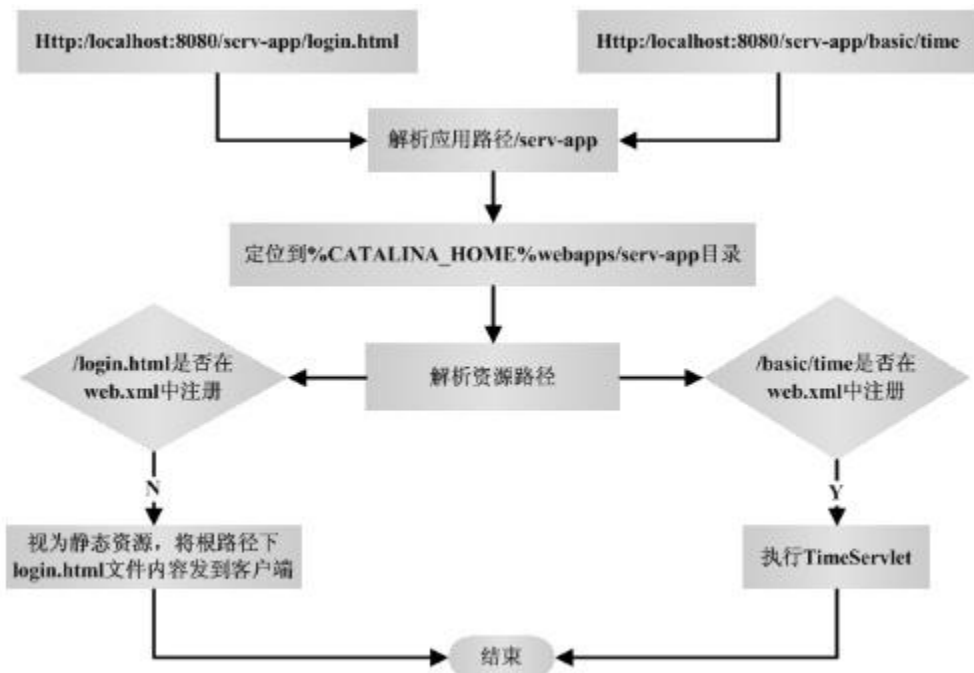
application-path为服务器中发布的某个应用的路径，如果为缺省应用（比如tomcat的ROOT）可以为

空。resource-path为客户端要访问的服务器中的资源的路径。比如：

<http://localhost:8080/serv-app/login.html> 表示通过8080端口访问本地机器上名字为路径为 serv-app中/login.html对应的资源。

<http://localhost:8080/serv-app/basic/time> 表示通过8080端口访问本地机器上路径为serv-app的应用中/basic/time对应的资源。

那么Web服务器是如何解释该请求的路径，以及将资源发送给客户端呢？在前面的“建立并发布一个Web应用”部分，我们说过Web服务器会将应用的路径/serv-app映射到磁盘的某个特定的目录结构，本例中为tomcat服务器中webapps目录下的serv-app。/login.html和/basic/time为该应用下的资源的路径，该路径同应用路径一样为“虚拟的”路径，由服务器把它映射为系统的具体文件或程序，具体流程如下图所示：



JavaEE Web规范规定了服务器搜索Servlet类的路径为应用目录结构中WEB-INF/classes目录及WEB-INF/lib下的所有jar文件。因此需要将TimeServlet按照如下的目录结构放到WEB-INF/classes中：

WEB-INF/classes/com/allanlxf/servlet/basic/TimeServlet.class

该Servlet部署描述如下：

```
<servlet>
    <servlet-name>TimeServlet</servlet-name>
    <servlet-class>com.allanlxf.servlet.basic.
TimeServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>TimeServlet</servlet-name>
    <url-pattern>/basic/time</url-pattern>
</servlet-mapping>
```

I. 何时创建Servlet实例？

在默认情况下Servlet实例是在第一个请求到来的时候创建，以后复用。如果有的Servlet需要复杂的操作需要载初始化时完成，比如打开文件、初始化网络连接等，可以通知服务器在启动的时候创建该Servlet的实例。具体配置如下：

```
<servlet>
    <servlet-name>TimeServlet</servlet-name>
    <servlet-class>com.allanlxf.servlet.basic.TimeServlet</servlet-class>

    <load-on-startup>1</load-on-startup>

</servlet>
```

其中<load-on-startup>标记的值必须为数值类型，表示Servlet的装载顺序，取值及含义如下：

正数或零：该Servlet必须在应用启动时装载，容器必须保证数值小的Servlet先装载，如果多个Servlet的<load-on-startup>取值相同，由容器决定它们的装载顺序。

负数或没有指定<load-on-startup>：由容器来决定装载的时机，通常为第一个请求到来时。

(2) 初始化

一旦Servlet实例被创建，Web服务器会自动调用init (ServletConfig config)方法来初始化该Servlet。其中方法参数config中包含了Servlet的配置信息，比如初始化参数，该对象由服务器创建。

I. 如何配置Servlet的初始化参数？

在web.xml中该Servlet的定义标记中，比如：

```
<servlet>
    <servlet-name>TimeServlet</servlet-name>
    <servlet-class>com.allanlxf.servlet.basic.
TimeServlet</servlet-class>
    <init-param>
        <param-name>user</param-name>
        <param-value>allanlxf</param-value>
    </init-param>
    <init-param>
        <param-name>blog</param-name>
        <param-value>http://allanlxf.blog.sohu.com</param-value>
    </init-param>
</servlet>
```

配置了两个初始化参数user和blog它们的值分别为allanlxf和<http://allanlxf.blog.sohu.com>，这样以后要修改用户名和博客的地址不需要修改Servlet代码，只需修改配置文件即可。

II. 如何读取Servlet的初始化参数？

ServletConfig中定义了如下的方法用来读取初始化参数的信息：

```
public String getInitParameter(String name)
```

参数：初始化参数的名称。

返回：初始化参数的值，如果没有配置，返回null。

比如：getInitParameter(“user”) 返回 allanlxf

getInitParameter(“blog”) 返回 http://allanlxf.blog.sohu.com

```
public java.util.Enumeration getInitParameterNames()
```

返回：该Servlet所配置的所有初始化参数名称的枚举。

III. init(ServletConfig)方法执行次数

在Servlet的生命周期中，该方法执行一次。

IV. init(ServletConfig)方法与线程

该方法执行在单线程的环境下，因此开发者不用考虑线程安全的问题。

V. init(ServletConfig)方法与异常

该方法在执行过程中可以抛出ServletException来通知Web服务器Servlet实例初始化失败。一旦ServletException抛出，Web服务器不会将客户端请求交给该Servlet实例来处理。

理，而是报告初始化失败异常信息给客户端，该Servlet实例将被从内存中销毁。如果在新的请求，Web服务器会创建新的Servlet实例，并执行新实例的初始化操作。

VI. 配置初始化参数VS覆盖init(ServletConfig)方法

配置初始化参数与覆盖init(ServletConfig)方法并没有必然的联系，这是很多初学者容易搞混的地方。配置初始化参数的目的是为了编写“通用”的Servlet，即通过改变初始化参数的值来改变Servlet的功能，而不必修改Servlet的源代码。覆盖init(ServletConfig)方法的原因是某些Servlet为客户提供服务需要执行一次性的操作，比如申请资源、打开文件、建立网络连接等，这些操作要么比较耗时，要么这些资源是提供服务的必要条件。

(3) 服务

一旦Servlet实例成功创建及初始化，该Servlet实例就可以被服务器用来服务于客户端的请求并生成响应。在服务阶段Web服务器会调用该实例的service(ServletRequest request, ServletResponse response)方法，request对象和response对象有服务器创建并传给Servlet实例。request对象封装了客户端发往服务器端的信息，response对象封装了服务器发往客户端的信息。

I. service()方法的职责

service()方法为Servlet的核心方法，客户端的业务逻辑应该在该方法内执行，典型的服务方法的开发流程为：

解析客户端请求-> 执行业务逻辑-> 输出响应页面到客户端

II. service()方法与线程

为了提高效率，Servlet规范要求一个Servlet实例必须能够同时服务于多个客户端请求，即service()方法运行在多线程的环境下，Servlet开发者必须保证该方法的线程安全性。

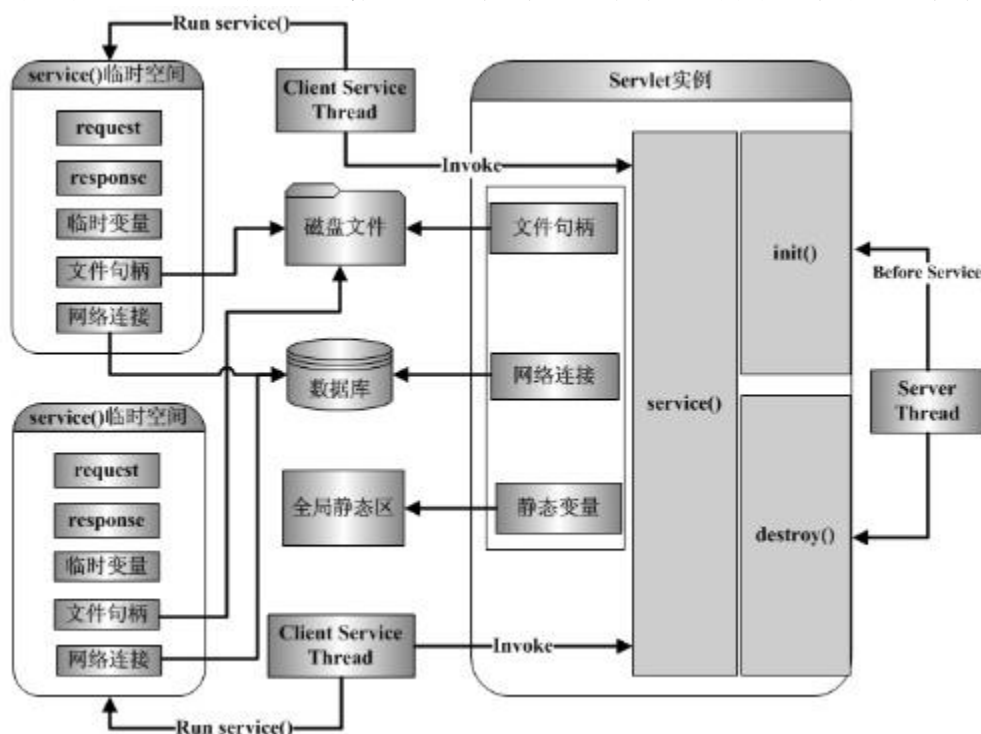
III. service()方法与异常

service()方法在运行的过程中可以抛出ServletException和IOException。其中ServletException可以在处理客户端请求的过程中抛出，比如请求的资源不可用、数据库不可用等。一旦该异常抛出，容器必须回收请求对象，并报告客户端该异常信息。IOException表示输入输出的错误，编程者不必关心该异常，直接由容器报告给客户端即可。

IV. 编写线程安全的资源

由于Servlet实例的service()方法在同一时刻会运行到多线程的环境下，因此，编写Servlet不得不考虑的因素就是线程安全的问题，这也是编写Servlet最容易出错的地方。

下面对Servlet的方法和线程之间的关系以及编程的原则进行详细的说明。



编程注意事项说明：

1) 当Server Thread线程执行Servlet实例的init()方法时，所有的Client Service Thread线程都不能执行该实例的service()方法，更没有线程能够执行该实例的destroy()方法，因此Servlet的init()方法是工作在单线程的环境下，开发者不必考虑任何线程安全的问题。

2) 当服务器接收到来自客户端的多个请求时，服务器会在单独的Client Service Thread线程中执行Servlet实例的service()方法服务于每个客户端。此时会有多个线程同时执行同一个Servlet实例的service()方法，因此必须考虑线程安全的问题。

3) 请大家注意，虽然service()方法运行在多线程的环境下，并不一定要同步该方法。而是要看这个方法在执行过程中访问的资源类型及对资源的访问方式。分析如下：

i. 如果service()方法没有访问Servlet的成员变量也没有访问全局的资源比如静态变量、文件、数据库连接等，而是只使用了当前线程自己的资源，比如非指向全局资源的临时变量、request和response对象等。该方法本身就是线程安全的，不必进行任何的同步控制。

ii. 如果service()方法访问了Servlet的成员变量，但是对该变量的操作是只读操作，该方法本身就是线程安全的，不必进行任何的同步控制。

iii. 如果service()方法访问了Servlet的成员变量，并且对该变量的操作既有读又有写，通常需要加上同步控制语句。

iv. 如果`service()`方法访问了全局的静态变量，如果同一时刻系统中也可能有其它线程访问该静态变量，如果既有读也有写的操作，通常需要加上同步控制语句。

v. 如果`service()`方法访问了全局的资源，比如文件、数据库连接等，通常需要加上同步控制语句。

V. 关于SingleThreadModel

在默认的情况下，Web服务器会为`web.xml`中每个`<servlet>`标签声明的Servlet创建Servlet**唯一**一个的实例，运行是会将该实例交给多个线程处理并发的客户端请求。因此Servlet的开发者必须保证Servlet的线程安全性。

Servlet规范中也规定了一个SingleThreadModel接口，该接口为标记型接口，没有任何方法，目的在于告诉容器该类型的Servlet的工作方式。只要Servlet类实现了该接口，Web服务器必须保证该类型的Servlet的实例在同一时刻只能服务于某一个请求，即`service()`方法不在并发的线程中。

注意，该运行方式只保证了Servlet实例的成员属性工作在单线程的环境下，但被Servlet访问的其它资源，比如HttpSession、文件、网络连接等也有可能同时被其它的Servlet实例访问，因此该运行方式并不能彻底解决线程并发的问题，建议开发者慎重使用。

(4) 销毁

当Web服务器认为Servlet实例没有存在的必要了，比如应用重新装载，或服务器关闭，以及Servlet很长时间都没有被访问过。服务器可以从内存中销毁（也叫卸载）该实例。Web服务器必须保证在卸载Servlet实例之前调用该实例的`destroy()`方法，以便回收Servlet申请的资源或进行其它的重要的处理。

I. `destroy()`与`service()`

Web服务器必须保证调用`destroy()`方法之前，让所有正在运行在该实例的`service()`方法中的线程退出或者等待这些线程一段时间。一旦`destroy()`方法已经执行，Web服务器将拒绝所有的新到来的对该Servlet实例的请求，`destroy()`方法退出，该Servlet实例即可以被垃圾回收。

编写一个Servlet，该Servlet记录实例创建以来所有访问过该实例的客户端的IP地址到服务器的某个日志文件中。该日志文件的路径必可以在部署Servlet的时候由部署者指定。

Servlet与URL匹配

为了让客户端访问服务器中的Servlet，部署者需要为每个Servlet配置一个访问路径，该路径有如下的三种写法：

(1) 确切路径匹配

以“/” 开始，后面跟一个具体的路径名称，也可以包含子路径。比如： /time、/basic/time、/basic/time/http都属确切的路径匹配。在该匹配模式下，客户端只能通过这一唯一的路径来访问该Servlet实例。

(2) 模糊路径匹配

以“/” 开始，以“/*” 结束，中间可以包含子路径。比如： /*、/basic/*、/user/management/* 都属于模糊路径匹配。在该匹配模式下，客户端可以通过一组相关的路径来访问该Servlet的实例，即可以通过URL来传递附加信息。

(3) 扩展名匹配

以“*. ” 开始，以任意其它的字符结束。比如： *.do、*.action、*.ctrl等都属于扩展名的匹配。在该匹配模式下，客户端可以通过一组相关的路径来访问该Servlet的实例，即可以通过URL来传递附加信息。

(4) 缺省的Servlet

配置成“/” 的Servlet为该应用的缺省的Servlet，Web服务器会将所有的无法识别的客户端请求交给缺省的Servlet来处理。

匹配优先级别

在一个Web应用中会同时发布多个Servlet，不可避免的会出现多个Servlet都可以服务于某一个请求的情况。比如系统中发布了三个Servlet，它们的匹配路径分别为： /*、 *.do 及 /basic，如果客户端的请求路径位： /basic，那么 /basic以及 /* 都可以服务于该请求。因此规范中规定了Web服务器匹配Servlet的顺序规则，具体顺序规定如下：

1. 寻找确切的路径匹配的Servlet。
2. 如果没有确切的路径匹配，按照模糊的路径匹配，如果有多个路径存在，取固定部分路径最长的Servlet。
3. 寻找扩展名的匹配。
4. 如果上述规则都无法匹配到Servlet，系统会将请求交给缺省的Servlet处理。
5. 如果没有缺省的Servlet，报告错误信息给调用者。

假如系统中有如下的Servlet的匹配模式：

Servlet 的<url-pattern>信息	Servlet 名字
/foo/bar/*	Servlet1
/baz/*	Servlet2
/catalog	Servlet3
*.bop	Servlet4
/	Default

下面分别用不同的路径访问该应用，匹配结果如下：

客户端请求的路径	被调用的 Servlet
/foo/bar/index.html	Servlet1
/foo/bar/index.bop	Servlet1
/baz	Servlet2
/baz/index.html	Servlet2
/catalog	Servlet3
/catalog/index.html	Default
/catalog/search.bop	Servlet4
/index.bop	Servlet4

servlet转发

Servlet可以将发送给自己的请求转发给另一个URL地址，这个URL地址可以是html、jsp、servlet或其他的http地址。Servlet中的请求转发有三种形式，主要是通过HttpServletRequest对象和HttpServletResponse对象实现。

Java代码

```
public void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    request.getRequestDispatcher("/url").include(request, response);
    request.getRequestDispatcher("/url").forward(request, response);
    request.sendRedirect("/url");
}
```

三种方式的区别是：

include：将url中的内容包含到当前的 servlet输出当中，在服务器端完成，一般叫做包含；

forward：将当前的Request和 response对象交给指定的url处理，一般称作转发请求；在服务器端完成，客户端

地址栏看到的路径还是当前请求的servlet路径；

sendRedirect：指示客户端重新发起 一次请求，第二次请求指向其参数的url，一般称作重定向；客户端的地址栏将

改变为url值，是由客户端发起的第二次请求。

include与forward的区别：

include是把别人包含进 来，forward则是丢掉自己

include是把另一个 servlet/jsp处理过后的内容拿过来与本身的servlet的内容一起输出；

forward是把请求的内容转发到另一个 servlet/jsp中。在forward之前，在原始的servlet中不能对输出做flush。最终只有被被forward去的servlet的内 容被输出。

在原始servlet中设置的响应状态码和响应头会被include的servlet忽略。include的servlet在处理的时候只会保留进入include的servlet之后的状态码和状态头。
对于forward，在两个servlet设置的状态码和状态头都会被用到。

使用include转发时，response的编码格式不是UTF-8，所以请求时会出现乱码现象，因此需要显示的response的编码修改为UTF-8/GBK/GB2312。

不管是include、forward还是 sendRedirect, 在该行代码执行之后的语句仍然被执行，只是输出到response的内容没有作用了。

总结：

sendRedirect与include、forward的区别在于是不是同一个Request，sendRedirect会有两次交互；

include与forward的区别在于输出的内容，include包含两者的结果，而forward只有forward一个，以及响应头。

理解

1. Servlet基于应答请求模式

Servlet的核心内容就是要对客户端发来的请求进行处理，然后将处理的结果写入到响应中发送到客户端。请求与响应在实现上是由两个具体的类来完成的。

- 请求:HttpServletRequest
- 响应:HttpServletResponse

2. Servlet生命周期

创建：什么时候创建servlet实例，

由<load-on-startup>1</load-on-startup>决定

正数依数字大小在容器起来时依次初始化

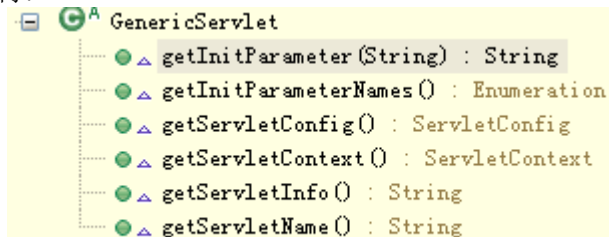
负数或不写则为首次调用时初始化

初始化：init方法，只执行一次 init做什么：提供服务需要执行一次性的操作

初始化参数：

```
<init-param>
    • <param-name></param-name>
    • <param-value></param-value>
</init-param>
```

取得：



```
GenericServlet
- getInitParameter(String) : String
- getInitParameterNames() : Enumeration
- getServletConfig() : ServletConfig
- getServletContext() : ServletContext
- getServletInfo() : String
- getServletName() : String
```

服务：service方法（线程安全）

销毁: `destroy` 只执行一次

3. 转发功能

```
//站外重定向http://127.0.0.1:8080/test.dispatcher  
response.sendRedirect("http://news.163.com/");
```

```
//站内转发 转发到 TimeHttpServlet  
RequestDispatcher dispatcher = request.getRequestDispatcher("/test.time");  
dispatcher.forward(request, response);
```

4. 上下文

`ServletConfig`

`ServletContext (config.getServletContext)`

参考文档:

`Servlert.ppt`

<http://allanlxf.blog.sohu.com/31156030.html>系列

<http://zjc85878482.javaeye.com/blog/547083>跳转