Servlet

Servlet 是什么?

Java Servlet 是运行在 Web 服务器或应用服务器上的程序，它是作为来自 Web 浏览器或其他 HTTP 客户端的请求和 HTTP 服务器上的数据库或应用程序之间的中间层。

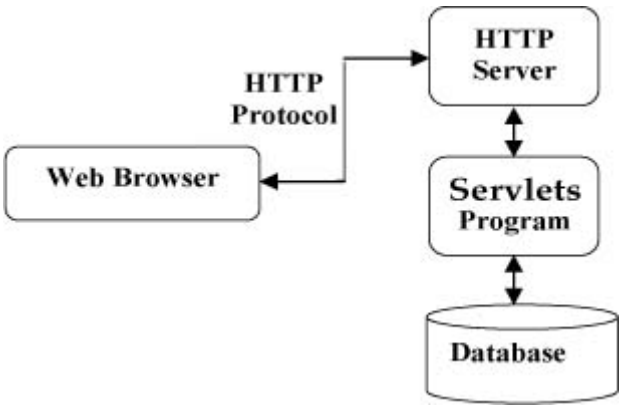
使用 Servlet，您可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。

Java Servlet 通常情况下与使用 CGI（Common Gateway Interface，公共网关接口）实现的程序可以达到异曲同工的效果。但是相比于 CGI，Servlet 有以下几点优势：

- 性能明显更好。
- Servlet 在 Web 服务器的地址空间内执行。这样它就没有必要再创建一个单独的进程来处理每个客户端请求。
- Servlet 是独立于平台的，因为它们是用 Java 编写的。
- 服务器上的 Java 安全管理器执行了一系列限制，以保护服务器计算机上的资源。因此，Servlet 是可信的。
- Java 类库的全部功能对 Servlet 来说都是可用的。它可以通过 sockets 和 RMI 机制与 applets、数据库或其他软件进行交互。

Servlet 架构

下图显示了 Servlet 在 Web 应用程序中的位置。



Servlet 任务

Servlet 执行以下主要任务：

- 读取客户端（浏览器）发送的显式的数据。这包括网页上的 HTML 表单，或者也可以是来自 applet 或自定义的 HTTP 客户端程序的表单。
- 读取客户端（浏览器）发送的隐式的 HTTP 请求数据。这包括 cookies、媒体类型和浏览器能理解的压缩格式等等。
- 处理数据并生成结果。这个过程可能需要访问数据库，执行 RMI 或 CORBA 调用，调用 Web 服务，或者直接计算得出对应的响应。
- 发送显式的数据（即文档）到客户端（浏览器）。该文档的格式可以是多种多样的，包括文本文件（HTML 或 XML）、二进制文件（GIF 图像）、Excel 等。
- 发送隐式的 HTTP 响应到客户端（浏览器）。这包括告诉浏览器或其他客户端被返回的文档类型（例如 HTML），设置 cookies 和缓存参数，以及其他类似的任务。

Servlet 包

Java Servlet 是运行在带有支持 Java Servlet 规范的解释器的 web 服务器上的 Java 类。

Servlet 可以使用 **javax.servlet** 和 **javax.servlet.http** 包创建，它是 Java 企业版的标准组成部分，Java 企业版是支持大型开发项目的 Java 类库的扩展版本。

这些类实现 Java Servlet 和 JSP 规范。在写本教程的时候，二者相应的版本分别是 Java Servlet 2.5 和 JSP 2.1。

Java Servlet 就像任何其他的 Java 类一样已经被创建和编译。在您安装 Servlet 包并把它们添加到您的计算机上的 Classpath 类路径中之后，您就可以通过 JDK 的 Java 编译器或任何其他编译器来编译 Servlet。

Servlet生命周期

Servlet 生命周期可被定义为从创建直到毁灭的整个过程。以下是 Servlet 遵循的过程：

- Servlet 初始化后调用 **init ()** 方法。
- Servlet 调用 **service()** 方法来处理客户端的请求。
- Servlet 销毁前调用 **destroy()** 方法。
- 最后，Servlet 是由 JVM 的垃圾回收器进行垃圾回收的。

现在让我们详细讨论生命周期的方法。

init() 方法

init 方法被设计成只调用一次。它在第一次创建 Servlet 时被调用，在后续每次用户请求时不再调用。因此，它是用于一次性初始化，就像 Applet 的 init 方法一样。

Servlet 创建于用户第一次调用对应于该 Servlet 的 URL 时，但是您也可以指定 Servlet 在服务器第一次启动时被加载。

当用户调用一个 Servlet 时，就会创建一个 Servlet 实例，每一个用户请求都会产生一个新的线程，适当的时候移交给 doGet 或 doPost 方法。init() 方法简单地创建或加载一些数据，这些数据将被用于 Servlet 的整个生命周期。

init 方法的定义如下：

```
@Override
    public void init() throws ServletException {
        System.out.println("servlet初始化");
    }
```

第一次访问servlet：

再次访问：

说明：servlet的init()只在访问servlet第一次被初始化，只要服务器没有关闭，则只会初始化一次。

如果在web.xml配置了 `<load-on-startup>1</load-on-startup>` 则该servlet在服务器启动时就被加载。

service() 方法

service() 方法是执行实际任务的主要方法。Servlet 容器（即 Web 服务器）调用 service() 方法来处理来自客户端（浏览器）的请求，并把格式化的响应写回给客户端。

每次服务器接收到一个 Servlet 请求时，服务器会产生一个新的线程并调用服务。service() 方法检查 HTTP 请求类型（GET、POST、PUT、DELETE 等），并在适当的时候调用 doGet、doPost、doPut，doDelete 等方法。

下面是该方法的特征：

```
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        resp.getWriter().write("servlet life");
        System.out.println("servlet life");
    }
```

service() 方法由容器调用，service 方法在适当的时候调用 doGet、doPost、doPut、doDelete 等方法。所以，您不用对 service() 方法做任何动作，您只需要根据来自客户端的请求类型来重写 doGet() 或 doPost() 即可。

doGet() 和 doPost() 方法是每次服务请求中最常用的方法。下面是这两种方法的特征。

doGet() 方法

GET 请求来自于一个 URL 的正常请求，或者来自于一个未指定 METHOD 的 HTML 表单，它由 doGet() 方法处理。

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

简单登陆：

```
    <form action="servletmethod" method="get">
        用户名：<input type="text" name="uname" ><br>
        密码：<input type="text" name="password"><br>
        <input type="submit" value="登陆">
    </form>
```

点击登陆后搜索框：

即：对用户是可见的，不安全，当同时有doGet方法和service方法时，只调用service方法。

doPost() 方法

POST 请求来自于一个特别指定了 METHOD 为 POST 的 HTML 表单，它由 doPost() 方法处理。

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet 代码
}
```

```
    <form action="servletmethod" method="post">
        用户名：<input type="text" name="uname" ><br>
        密码：<input type="text" name="password"><br>
        <input type="submit" value="登陆">
    </form>
```

使用post请求：

对用户不可现，更加安全。

destroy() 方法

destroy() 方法只会被调用一次，在 Servlet 生命周期结束时被调用。destroy() 方法可以让您的 Servlet 关闭数据库连接、停止后台线程、把 Cookie 列表或点击计数器写入到磁盘，并执行其他类似的清理活动。

在调用 destroy() 方法之后，servlet 对象被标记为垃圾回收。destroy 方法定义如下所示：

```
    @Override
    public void destroy() {
        System.out.println("servlet 销毁");
    }
```

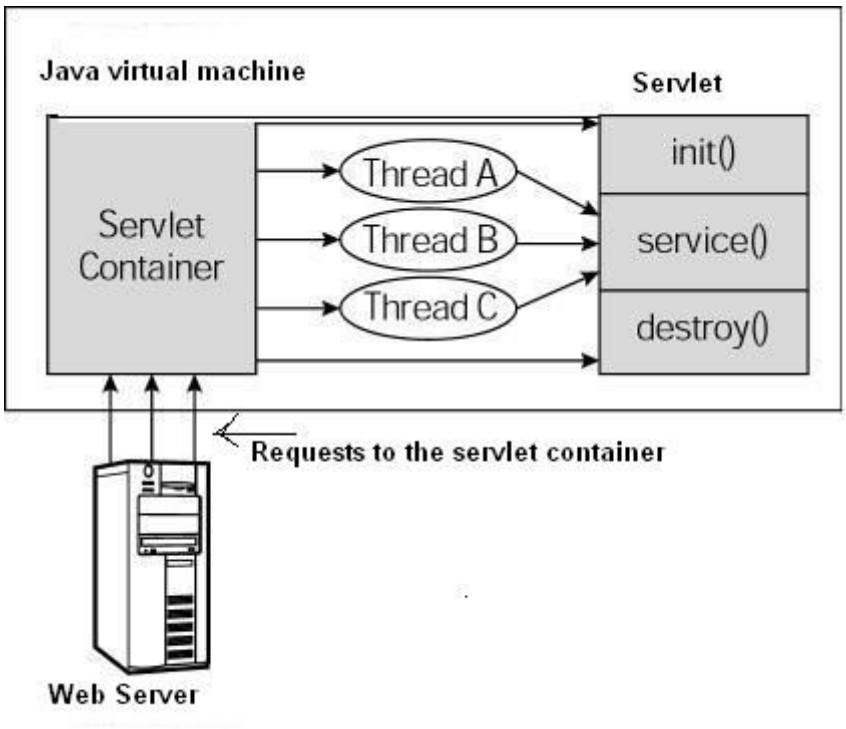
当停止tomcat服务器时：

说明servlet在服务器关闭时被销毁。

架构图

下图显示了一个典型的 Servlet 生命周期方案。

- 第一个到达服务器的 HTTP 请求被委派到 Servlet 容器。
- Servlet 容器在调用 service() 方法之前加载 Servlet。
- 然后 Servlet 容器处理由多个线程产生的多个请求，每个线程执行一个单一的 Servlet 实例的 service() 方法。



注意

如果在复写的service方法中调用了父类的service方法（super.service(arg0,arg1)）,则service方法处理完后，会再次根据请求方式响应的doGet和doPost方法执行。所以，一般情况下，我们是不在复写的service中调用父类打的service方法的，避免出现405错误。

Servlet常见错误

404错误：资源未找到

原因1：在请求地址中的servlet的别名书写错误。

原因2：虚拟项目名拼写错误。

500错误：内部服务器错误

错误1：java.lang.ClassNotFoundException: com.youxin.servlet.ServletMethod

解决：

在web.xml中校验servlet类的全限路径是否拼写错误

错误2：因为service方法体的代码执行导致错误

解决：

根据错误提示对service方法体中的代码进行错误比较

405错误：请求方式不支持

原因：请求方式和servlet中的方法不匹配造成的

解决：尽量使用service方法进行请求处理，并且不要在service方法中调用父类的service方法。

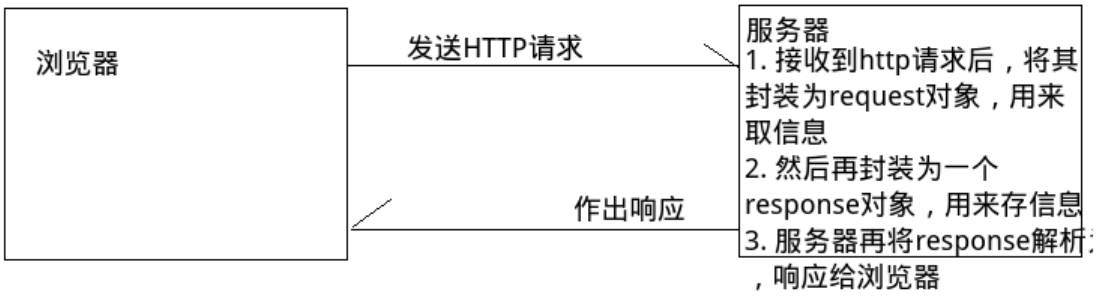
Request和Response对象

概述：

我们在创建Servlet时会覆盖service()方法，或doGet()/doPost(),这些方法都有两个参数，一个为代表请求的request和代表响应的response。

service方法中的request的类型是ServletRequest，而doGet/doPost方法的request的类型是HttpServletRequest，HttpServletRequest是ServletRequest的子接口，功能和方法更加强大。

service方法中的response的类型是ServletResponse，而doGet/doPost方法的response的类型是HttpServletResponse，HttpServletResponse是ServletResponse的子接口，功能和方法更加强大。



###Request对象

request获取请求头

```
/*
* request对象
*   作用：request对象封存了当前请求的所有请求信息
*   使用：
*       获取请求头数据
*           req.getMethod();
*           req.getRequestURL();
*           req.getRequestURI();
*           req.getScheme();
*       获取请求行数据
*           req.getHeader("Accept-Language");
*       获取用户数据
*           req.getParameter()
*           req.getParameterValues("键名")；//返回同键不同值的请求数据（多选），返回的数组
*           req.getParameterNames();//返回所有用户请求数据的枚举集合
*   注意：
*       request对象由tomcat服务器创建，并作为实参传递给处理请求的servlet的service方法
* */
public class RequestServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //获取请求头数据
        //获取请求方式
        String method = req.getMethod();
        System.out.println(method);
        //获取请求URL
        StringBuffer requestURL = req.getRequestURL();
        System.out.println(requestURL);
        //获取URI
```

```
        String requestURI = req.getRequestURI();
        System.out.println(requestURI);
        //获取协议
        String scheme = req.getScheme();
        System.out.println(scheme);
    }
}
```

输出:

request获取请求行

```
//获取请求行数据
        //获取指定的请求行信息
        String value = req.getHeader("Accept-Language");
        System.out.println(value);
        //获取所有的请求行的键的枚举
        Enumeration headerNames = req.getHeaderNames();
        while (headerNames.hasMoreElements()){
            String header = req.getHeader(headerNames.nextElement().toString());
            System.out.print("请求头的键："+headerNames.nextElement()+",");
            System.out.println("请求头的值："+header);
        }
```

结果:

```
        //不管是get还是post都用req.getParameter()获取用户属性值
        //获取用户信息
        String uname = req.getParameter("uname");
        String password = req.getParameter("password");
        System.out.println("uname:"+uname+",password:"+password);
```

getparameter不能获取同键不同值的对象

如果出现同键不同值的情况则需要用 `req.getParameterValues()` 获取

如:

```
        爱好：<br>
        <input type="checkbox" name="fav" value="1">唱歌
        <input type="checkbox" name="fav" value="2">跳舞
        <input type="checkbox" name="fav" value="3">打游戏
```

获取:

```
String[] favs = req.getParameterValues("fav");
if (favs != null){
    for (String s: favs
         ) {
        System.out.println(s);
    }
}
```

Response对象：

问题：在使用Requset对象获取了请求数据并进行处理后，处理的结果如何显示到浏览器中呢?

解决：使用Reponse对象

解释：服务器在调用指定的servlet进行请求处理的时候，会给servlet的方法传递两个实参request和response。其中requset中封存了请求相关的请求数据，而response则是用来进行相应的一个对象。

```
/*
*
* Response
* 作用：响应数据到浏览器的一个对象
* 使用：设置响应头
*           setHeader()//在响应头中添加响应信息，但同键会覆盖
*           addHeader()//在响应头中添加信息，但不会覆盖
*       设置响应状态：sendError();//自定义响应状态码
*       设置响应体
*           resp.getWriter().write()
*       设置响应编码格式：
*       resp.setContentType("text/html;charset=utf-8")
* */
public class ResponseServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        //获取请求信息
        //获取请求头
        //获取请求行
        //获取用户数据
        //处理请求

        //响应处理结果
        //设置响应头
        resp.setHeader("mouse", "leigod");
        resp.addHeader("key", "leigod");

        //设置编码格式
//            resp.setCharacterEncoding("utf-8");
        resp.setContentType("text/html;charset=utf-8");

        //设置响应状态码
//            resp.sendError(404,"sorry");
//            resp.sendError(405,"this method is not supported");

        //设置响应实体
            resp.getWriter().write("<a href='http://www.baidu.com'>百度一下</a>");
```

```
    }
}
```

####请求转发和重定向的区别

1.请求转发是服务器内部跳转，地址栏不会发生改变

重定向地址栏会发生改变。

2.请求转发，只有一次请求，一次响应.

重定向，有两次请求，两次响应。

3.请求转发存在request域，可以共享数据.

重定向不存在request域。

4.请求转发只能在服务器的内部跳转，简单说，只能访问本站内资源。

重定向可以访问站外资源，也可以访问站内资源.

5.请求转发是由request 发起的 . request.getRequestDispatcher().forward()

重定向是由response 发起的 response.sendRedirect();

6.请求转发与重定向时路径写法不一样.

重定向要跳转的路径是浏览器在次发起的，是浏览器端路径:写法: /工程名/资源

请求转发是服务器内部跳转，这时它与浏览器无关 写法:/资源

1. 请求包含

RequestDispatcher.include()方法用于将RequestDispatcher对象封装的资源内容作为当前响应内容的一部分包含进来，从而实现可编程的服务器端包含功能

被包含的Servlet程序不能改变响应消息的状态码和响应头，如果它里面存在这样的语句，这些语句的执行结果将被忽略.include在程序执行上效果类似forward,但是使用forward只有一个程序可以生成响应，include可以由多个程序一同生成响应 ----- 常用来页面布局

New两个html页面分别为 i1.html&i2.html

i1内容1111111111

i2内容22222222222

New一个Servlet get方法下

request.getRequestDispatcher(“/i1.html”).include(request,response);

request.getRequestDispatcher(“/i2.html”).include(request,response);

最后在页面输出的内容为11111111122222222222是两个页面的所用内容.