Servlet 教程 Servlet 环境设置

Servlet 简介

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。

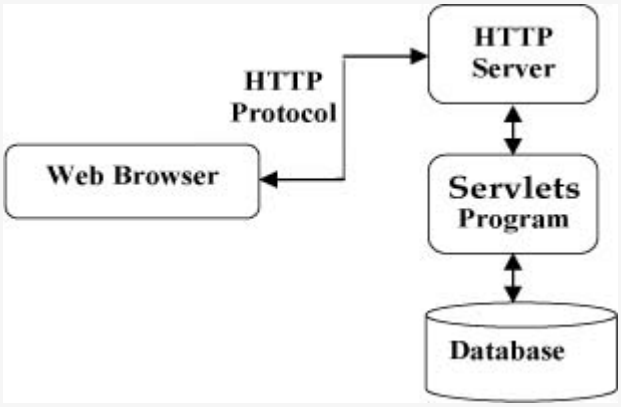
使用 Servlet，您可以收集来自网页表单的用户输入，呈现来自数据库或者其他源的记录，还可以动态创建网页。

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。

下一步呢？

接下来，本教程会带你一步一步地设置您的 Servlet 环境，以便开始后续的 Servlet 使用。因此，请系紧您的安全带，随我们一起开始 Servlet 的学习之旅吧！相信您会很喜欢这个教程的。

← Servlet 教程 Servlet 环境设置 →

2 篇笔记

写笔记

Servlet 创有三种方式。

1、实现 Servlet 接口

因为是实现 Servlet 接口，所以我们需要实现接口里的方法。

下面我们也说明了 Servlet 的执行过程，也就是 Servlet 的生命周期。

```
//Servlet的生命周期:从Servlet被创建到Servlet被销毁的过程
//一次创建，到处服务
//一个Servlet只会有一个对象，服务所有的请求
/*
 * 1.实例化（使用构造方法创建对象）
 * 2.初始化  执行init方法
 * 3.服务     执行service方法
 * 4.销毁     执行destroy方法
 */
public class ServletDemo1 implements Servlet {

    //public ServletDemo1(){}

     //生命周期方法:当Servlet第一次被创建对象时执行该方法,该方法在整个生命周期中只执行一次
    public void init(ServletConfig arg0) throws ServletException {
                System.out.println("=======init=========");
```

```
        }

    //生命周期方法:对客户端响应的方法,该方法会被执行多次，每次请求该servlet都会执行该方法
    public void service(ServletRequest arg0, ServletResponse arg1)
            throws ServletException, IOException {
        System.out.println("hehe");

    }

    //生命周期方法:当Servlet被销毁时执行该方法
    public void destroy() {
        System.out.println("******destroy**********");
    }
//当停止tomcat时也就销毁的servlet。
    public ServletConfig getServletConfig() {

        return null;
    }

    public String getServletInfo() {

        return null;
    }
}
```

2、继承 GenericServlet 类

它实现了 Servlet 接口除了 service 的方法，不过这种方法我们极少用。

```
public class ServletDemo2 extends GenericServlet {

    @Override
    public void service(ServletRequest arg0, ServletResponse arg1)
            throws ServletException, IOException {
        System.out.println("heihei");

    }
}
```

3、继承 HttpServlet 方法

```
public class ServletDemo3 extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        System.out.println("haha");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
```

```
            throws ServletException, IOException {
        System.out.println("ee");
        doGet(req,resp);
    }

}
```

创建 Servlet 的第三种方法，也是我们经常用的方法。

这里只简单讲 Servlet 的三种创建方式，关于更详细的应用我们后面再说。

南离 10个月前 (05-29)

关于 HttpServlet、GenericServlet 和 Servlet 的关系

对于一个 Servlet 类，我们日常最常用的方法是继承自 HttpServlet 类，提供了 Http 相关的方 HttpServlet 扩展了 GenericServlet 类，而 GenericServlet 类又实现了 Servlet 类和 ServletConfig 类。

Servlet

Servlet 类提供了五个方法，其中三个生命周期方法和两个普通方法，关于 Servlet 类的方法，不再赘述，我主要补充一下另外两个类的实现思路。

GenericServlet

GenericServlet 是一个抽象类，实现了 Servlet 接口，并且对其中的 init() 和 destroy() 和 service() 提供了默认实现。在 GenericServlet 中，主要完成了以下任务：

- 将 init() 中的 ServletConfig 赋给一个类级变量，可以由 getServletConfig 获得；
- 为 Servlet 所有方法提供默认实现；
- 可以直接调用 ServletConfig 中的方法；

基本的结构如下：

```
abstract class GenericServlet implements Servlet,ServletConfig{

   //GenericServlet通过将ServletConfig赋给类级变量
   private trServletConfig servletConfig;

   public void init(ServletConfig servletConfig) throws ServletException {

      this.servletConfig=servletConfig;

      /*自定义init()的原因是：如果子类要初始化必须覆盖父类的init() 而使它无效 这样
       this.servletConfig=servletConfig不起作用 这样就会导致空指针异常 这样如果子类要初始化，
       可以直接覆盖不带参数的init()方法 */
      this.init();
   }

   //自定义的init()方法，可以由子类覆盖
   //init()不是生命周期方法
   public void init(){

   }
```

```
    //实现service()空方法，并且声明为抽象方法，强制子类必须实现service()方法
    public abstract void service(ServletRequest request,ServletResponse response)
      throws ServletException,java.io.IOException{
    }

    //实现空的destroy方法
    public void destroy(){ }
}
```

以上就是 GenericServlet 的大致实现思想，可以看到如果继承这个类的话，我们必须重写 service() 方法来对处理请求。

HttpServlet

HttpServlet 也是一个抽象类，它进一步继承并封装了 GenericServlet，使得使用更加简单方便，由于是扩展了 Http 的内容，所以还需要使用 HttpServletRequest 和 HttpServletResponse，这两个类分别是 ServletRequest 和 ServletResponse 的子类。代码如下：

```
abstract class HttpServlet extends GenericServlet{

    //HttpServlet中的service()
    protected void service(HttpServletRequest httpServletRequest,
                           HttpServletResponse httpServletResponse){
         //该方法通过httpServletRequest.getMethod()判断请求类型调用doGet() doPost()
    }

    //必须实现父类的service()方法
    public void service(ServletRequest servletRequest,ServletResponse servletResponse){
       HttpServletRequest request;
       HttpServletResponse response;
       try{
          request=(HttpServletRequest)servletRequest;
          response=(HttpServletResponse)servletResponse;
       }catch(ClassCastException){
          throw new ServletException("non-http request or response");
       }
       //调用service()方法
       this.service(request,response);
    }
}
```

我们可以看到，HttpServlet 中对原始的 Servlet 中的方法都进行了默认的操作，不需要显式的销毁初始化以及 service()，在 HttpServlet 中，自定义了一个新的 service() 方法，其中通过 getMethod() 方法判断请求的类型，从而调用 doGet() 或者 doPost() 处理 get,post 请求，使用者只需要继承 HttpServlet，然后重写 doPost() 或者 doGet() 方法处理请求即可。

我们一般都使用继承 HttpServlet 的方式来定义一个 servlet。

612星球的一只天才猪 3个月前 (12-17)