一、什么 Servlet？

servlet 是运行在 Web 服务器中的小型 Java 程序（即：服务器端的小应用程序）。servlet 通常通过 HTTP（超文本传输协议）接收和响应来自 Web 客户端的请求。

servlet 是一个 Java 类，但是它不用 new 就能运行。

1.1 编写一个 servlet 程序：

注：需要引入 tomcat 中的 Tomcat 9.0\lib\servet-api.jar 和 jsp-api.jar

1.写一个 java 类，实现 servlet 接口

注：在 service 中使用 System.out.print()会输出到控制端。

```
public class ServletDemo1 implements Servlet{


    //接收用户请求，并做出响应
    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        res.getWriter().write("hello ServletDemo1");

    }
```

2.修改 web.xml 文件，给 servlet 提供一个可访问的 URI 地址

（WEB-INF 中的类不能直接访问，需要在 web.xml 中配置）

```
<!-- 创建一个servlet实例 -->
<servlet>
    <servlet-name>servletDemo1</servlet-name>
    <servlet-class>com.itcast.servlet.ServletDemo1</servlet-class>
</servlet>

<!-- 给servlet提供(映射)一个可访问的URI地址 -->
<servlet-mapping>
    <servlet-name>servletDemo1</servlet-name>
    <url-pattern>/demo1</url-pattern>
</servlet-mapping>
```

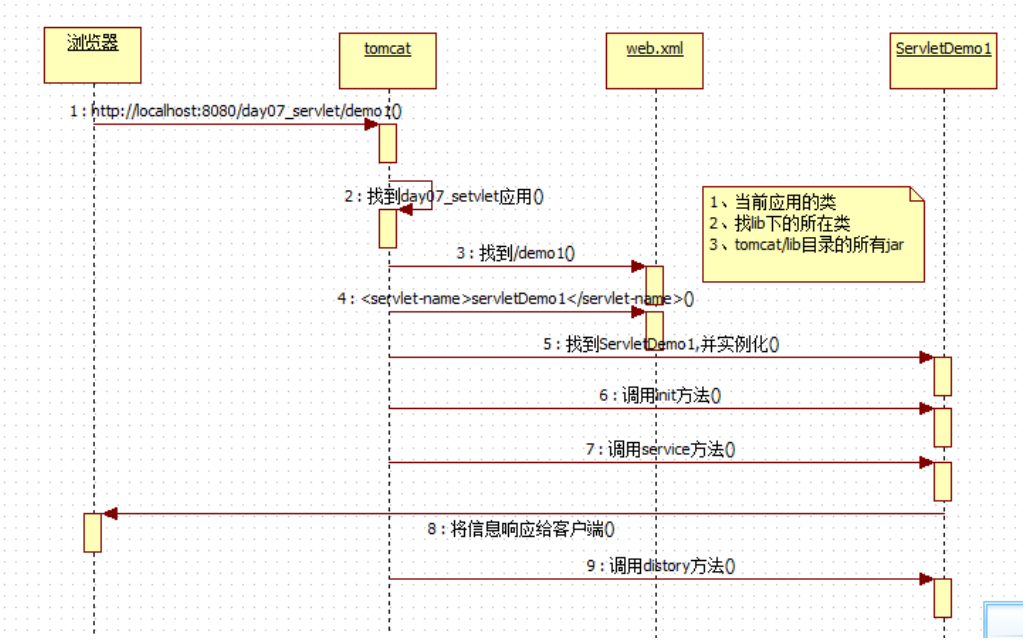
servlet 类所在的路径，包路径

外部访问的名称，注意加“/”

3.部署应用到 tomcat 服务器

4.测试：http://locahost:8080/day08_servlet/demo1

二、执行过程



三、Servlet 生命周期（重要）

实例化-->初始化-->服务->销毁

- 出生：（实例化-->初始化）第一次访问 Servlet 就出生（默认情况下）
- 活着：（服务）应用活着，servlet 就活着
- 死亡：（销毁）应用卸载了 servlet 就销毁。

区别服务器停止与服务卸载，Servlet 创建后一直存在，它会为每个请求创建一个线程。可以卸载服务：

- 停止服务器，此时服务肯定卸载了；
- 通过 tomcat 进行卸载，进入 tomcat 的默认主页，在右侧点击，Manager App，按要求操作即可。

```
//默认当第一次请求时，创建servet实例。应用存在实例就存在；当应用被卸载了，实例就销毁了。单实例
public class ServletDemo1 implements Servlet{
    //生命周期的方法：实例化对象
    //第一次被访问时调用
    public ServletDemo1(){
        System.out.println("********ServletDemo1被调用了********");
    }
    //生命周期的方法：初始化方法
    //第一次被访问时调用
    public void init(ServletConfig config) throws ServletException {
        System.out.println("********init被调用了********");
    }

    //生命周期的方法：服务方法
    //接收用户请求，并做出响应
    //每次请求都被调用
    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        System.out.println("********service被调用了********");
        //res.getWriter().write("hello ServletDemo1");
    }
    //生命周期的方法：销毁
    //当应用卸载时调用
    public void destroy() {
        System.out.println("********destroy被调用了********");
    }
```

小知识：

如何让 servlet 在服务器启动时就创建（默认是在第一次请求时创建）。

```
<servlet>
  <servlet-name>servletDemo2</servlet-name>
  <servlet-class>cn.itcast.servlet.ServletDemo2</servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>
```

使当前Servlet在服务器启动时创建

注：数值“2”越大越先启动。

四、Servlet 的三种创建方式

以下三种方式创建的本质都是继承和实现 Servlet 接口，Servlet 接口中有五个方法，tomcat 使用该接口创建 servlet 对象，并调用接口中的五个方法进行后续操作。如：

Servlet servlet = new ServletDemo2();(ServletDemo2 的类信息从配置文件 web.xml 中获取，通过反射创建)；

servlet.init(传入 ServletConfig);

servlet.service(传入 ServletRequest 和 ServletResponse);

4.1 实现 javax.servlet.Servlet 接口（参见：编写一个 servlet 程序：）

实现 Servlet 接口时，必须实现接口中的五个方法。

4.2 继承 javax.servet.GenericServlet 类(适配器模式)

GenericServlet 类是抽象类，实现了 Servlet 接口，对部分方法进行了重写并通过适配器方式增加了部分方法，当用户继承该抽象类时，就不用实现所有的方法，只需要必须实现 service 方法。

```
public class ServletDemo2 extends GenericServlet{

    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        System.out.println("servletDemo2");
    }

}
```

适配器在此的理解：Servlet 类中没有 init()方法，因此需要将该方法适配为 Servlet 类可以调用的方式，即在 init(ServletConfig config)方法中调用。

增加 init()的好处是子类继承中不用调用 super.init(config)方法，因为父类中有 this.config=config 语句，因此子类继承覆盖时，需要调用 super.init(config)方法，否则不能为 this.config 设置值。

```
@Override
public void init(ServletConfig config) throws ServletException {
    this.config = config;
    this.init();
}

public void init() throws ServletException {

}
```

4.3 继承 javax.servlet.http.HttpServlet 类（模板方法设计模式）

1.开发中常用方式

```java
//不要重写父类的service方法。
public class ServletDemo3 extends HttpServlet{
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        System.out.println("servletDemo3--get请求方法");
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

    }
```

2.理解为什么 HttpServlet 的子类要重写 doPost 和 doGet 方法

我们首先查看源码：

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
{
        String protocol = req.getProtocol();
        String msg = lStrings.getString("http.method_get_not_supported");
        if (protocol.endsWith("1.1")) {
            resp.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, msg);
        } else {
            resp.sendError(HttpServletResponse.SC_BAD_REQUEST, msg);
        }
}
```

如果我们没有重写 doGet 方法，会调用默认的 doGet 方法，该方法将获取 http 协议，如果是以“1.1”结尾，则会抛出“405，http.method_get_not_supported”异常。

3.理解 HttpServletRequest 和 HttpServletResponse

HttpServletRequest 和 HttpServletResponse 是接口，分别继承了 ServletRequest 和 ServletResponse 接口。HttpServletRequest 和 HttpServletResponse 由 tomcat 服务器供应商实现并提供，表示使用 http 协议的请求和响应。

Servlet 接口中的 service(ServletRequest req, ServletResponse res)，在 HttpServlet 中将 ServletRequest 强制转为 HttpServletRequest 类型，然后获取其参数进行操作。

4.理解模板方法设计模式

定义一个操作中的算法的骨架，而将步骤延迟到子类中。模板方法使得子类可以不改变一个算法的结构即可重定义算法的某些特定步骤。

```
        protected void service(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
            String method = req.getMethod();
            if (method.equals(METHOD_GET)) {
               doGet(req, resp);
            } else if (method.equals(METHOD_HEAD)) {
               doHead(req, resp);
            } else if (method.equals(METHOD_POST)) {
               doPost(req, resp);
            } else if (method.equals(METHOD_PUT)) {
               doPut(req, resp);
            } else if (method.equals(METHOD_DELETE)) {
               doDelete(req, resp);
```

```
        }
}
```
在 service 中，从 HttpServletRequest 中获去请求方式，根据请求方式调用不同的方法。

Servlet --> GenericServlet --> HttpServlet　(继承 HttpServlet)
曾祖父　　爷爷　　爸爸　　孙子

小技巧：使生成的 servlet 更清新一些
找到：MyEclipse\Common\plugins 目录
把 com.genuitec.eclipse.wizards_9.0.0.me201108091322.jar 复制到上面目录

可以使用 myEclipse 直接创建 Servlet 对象，并进行配置。

五、Servet 映射细节：

5.1 servlet 映射配置

servlet 映射细节 1：

```
  <!-- 配置多个映射路径 -->
<servlet-mapping>
  <servlet-name>ServletDmo4</servlet-name>
  <url-pattern>/ServletDmo4</url-pattern>
</servlet-mapping>

  <servlet-mapping>
  <servlet-name>ServletDmo4</servlet-name>
  <url-pattern>/ServletDmo44</url-pattern>
</servlet-mapping>
    <servlet-mapping>
  <servlet-name>ServletDmo4</servlet-name>
  <url-pattern>/admin/ServletDmo44</url-pattern>
</servlet-mapping>
```

servet 映射细节 2: 通配符* 代表任意字符串
url-pattern: *.do　以.do 结尾的字符串的请求都可以访问 注：不要加/
url-pattern: /*　任意字符串都可以访问
url-pattern：　/action/* 以/action 开头的请求都可以访问
匹配规则：
优先级：从高到低
绝对匹配-->　/开头匹配 --> 扩展名方式匹配
如果 url-pattern 的值是/，表示执行默认映射。所有资源都是 servlet

5.2 servlet 默认映射

在 tomcat 的配置文件 web.xml 中对“/”和“*.jsp”进行了配置。

```
<servlet>
    <servlet-name>default</servlet-name>
    <servlet-class>org.apache.catalina.servlets.DefaultServlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
        <servlet-name>default</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

<servlet>
    <servlet-name>jsp</servlet-name>
    <servlet-class>org.apache.jasper.servlet.JspServlet</servlet-class>
</servlet>
<!-- The mappings for the JSP servlet -->
<servlet-mapping>
    <servlet-name>jsp</servlet-name>
    <url-pattern>*.jsp</url-pattern>
    <url-pattern>*.jspx</url-pattern>
</servlet-mapping>
```

当用户使用 http://localhost:8080/index.html 访问服务器时，会调用 DefaultServlet 类，因此访问的所有服务器资源都是 Servlet 类，都经过了 Servlet 类的处理。

六、Servlet 的线程安全

Servlet 是单对象多线程，对象在第一次访问时创建（或启动服务器时），为每次访问创建一个线程。因此，定义在 Servlet 类中的变量是全局变量，需要注意线程安全问题。

解决线程安全问题的最佳办法，不要写全局变量，而写局部变量。

七、 Servlet 获取配置信息

ServletConfig 的使用

作用 1：可以获取 servlet 配置信息

方式 1：

```
private ServletConfig config;
//使用初始化方法回复到ServletConfig对象，此对象由服务器创建
public void init(ServletConfig config) throws ServletException {
    this.config = config;
}

public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    /*String encoding = "UTF-8";
    System.out.println(encoding);*/
    String value = config.getInitParameter("encoding");//根据配置文件中的名，得到值
    System.out.println(value);
```

方式 2：

```
//通过使用继承父类的方法得到 ServletConfig对象
String value = this.getServletConfig().getInitParameter("encoding");
System.out.println(value);
```

方式 3：

```
String value = this.getInitParameter("encoding");
System.out.println(value);
```

作用 2：可以获得 ServletContext 对象

八、 ServletContext（重要）

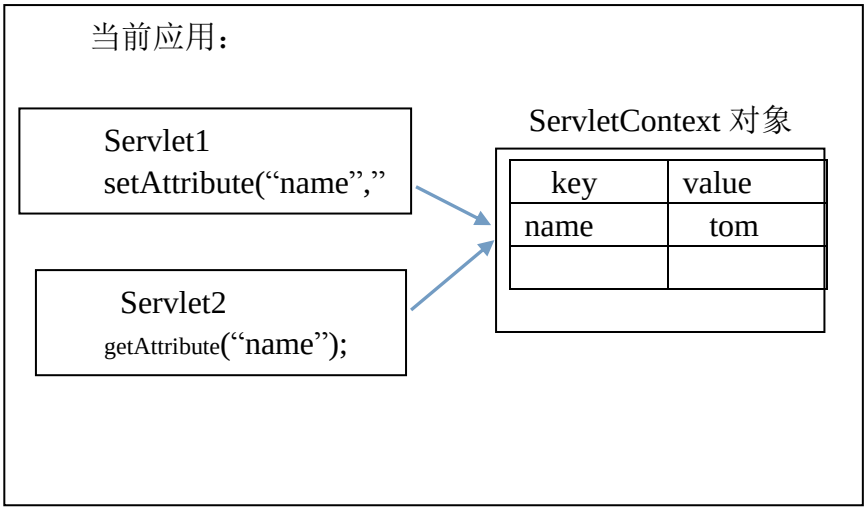
ServletContext: 代表的是整个应用。一个应用只有一个 ServletContext 对象。单实例。

作用：

域对象：在一定范围内（当前应用），使多个 Servlet 共享数据。

常用方法：

void setAttribute(String name,object value);//向 ServletContext 对象的 map 中添加数据
Object getAttribute(String name);//从 ServletContext 对象的 map 中取数据
void rmoveAttribute(String name);//根据 name 去移除数据



获取全局配置信息：

修改 web.xml 文件：

```
<!-- 配置当前应用的全局信息 -->
<context-param>
  <param-name>encoding</param-name>
  <param-value>UTF-8</param-value>
</context-param>
```

String	**getInitParameter**(String name) //根据配置文件中的 key 得到 value

```
//获取全局配置信息
String encoding = sc.getInitParameter("encoding");
System.out.println(encoding);
```

获取资源路径：

String　getRealPath(String path);//根据资源名称得到资源的绝对路径.
可以得到当前应用任何位置的任何资源。

```
private void test3() throws IOException, FileNotFoundException {
    ServletContext sc = this.getServletContext();
    //得到a.properties
    String path = sc.getRealPath("/WEB-INF/classes/com/itcast/servletContext/c.properties");//要以/开头，
    Properties prop = new Properties();
    prop.load(new FileInputStream(path));
    System.out.println(prop.getProperty("key"));
}
//得到b.properties
private void test2() throws IOException, FileNotFoundException {
    ServletContext sc = this.getServletContext();
    //得到a.properties
    String path = sc.getRealPath("/WEB-INF/classes/b.properties");//要以/开头，/代表的是当前应用的名称
    Properties prop = new Properties();
    prop.load(new FileInputStream(path));
    System.out.println(prop.getProperty("key"));
}
//获取a.properties文件的数据
private void test1() throws IOException, FileNotFoundException {
    ServletContext sc = this.getServletContext();
    //得到a.properties
    String path = sc.getRealPath("/WEB-INF/a.properties");//要以/开头，/代表的是当前应用的名称
    Properties prop = new Properties();
```

实现 Servlet 的转发。

```
//实现请求转发,目标是跳转到ServletContextDemo1
ServletContext sc = this.getServletContext();
RequestDispatcher rd = sc.getRequestDispatcher("/ServletContextDemo1");
rd.forward(request, response);//将请求信息向下传递
```

RequestDispatcher　getRequestDispatcher(String path) ;//参数表示要跳转到哪去