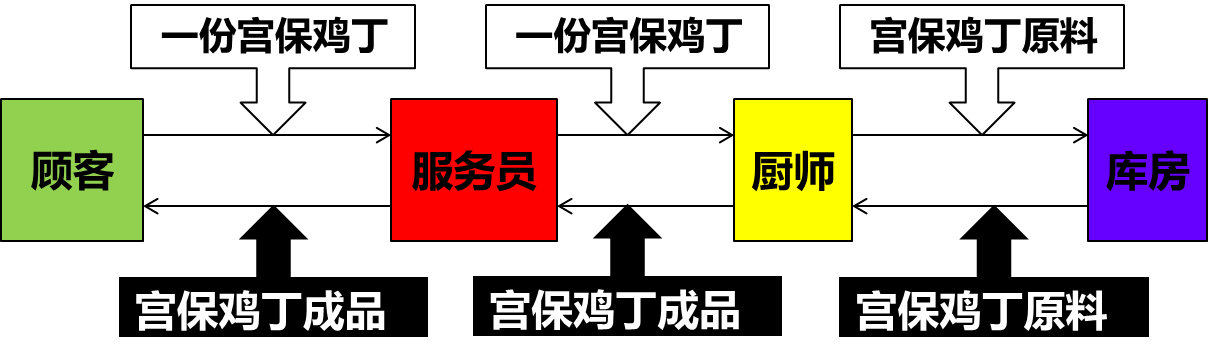
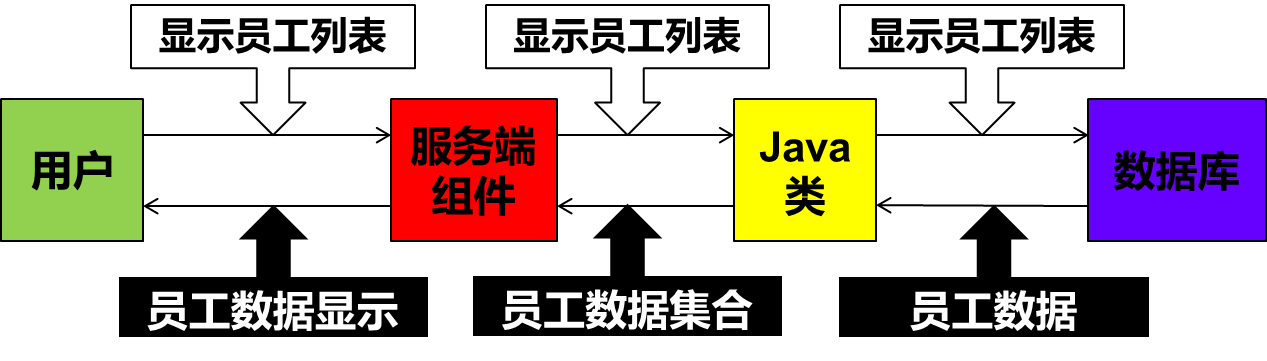
# Servlet

1. Why？我们为什么需要Servlet？
   1. Web应用基本运行模式

①生活中的例子



②Web应用运行模式



* 1. 通过网页驱动服务器端的Java程序。在网页上显示Java程序返回的数据。



1. What？什么是Servlet？

●如果把Web应用比作一个餐厅，Servlet就是餐厅中的服务员——负责接待顾客、上菜、结账。



●从广义上来讲，Servlet规范是Sun公司制定的一套技术标准，包含与Web应用相关的一系列接口，是Web应用实现方式的宏观解决方案。而具体的Servlet容器负责提供标准的实现。

●从狭义上来讲，Servlet指的是javax.servlet.Servlet接口及其子接口，也可以指实现了Servlet接口的实现类。

●Servlet作为服务器端的一个组件，它的本意是“服务器端的小程序”。Servlet的实例对象由Servlet容器负责创建；Servlet的方法由容器在特定情况下调用；Servlet容器会在Web应用卸载时销毁Servlet对象的实例。

1. How？如何使用Servlet？
   1. 操作步骤

①搭建Web开发环境

②创建动态Web工程

③创建javax.servlet.Servlet接口的实现类：com.atguigu.servlet.MyFirstServlet

④在service(ServletRequest, ServletResponse)方法中编写如下代码，输出响应信息：

|  |
| --- |
| @Override  **public** **void** service(ServletRequest req, ServletResponse res)  **throws** ServletException, IOException {  //1.编写输出语句，证明当前方法被调用  System.*out*.println("Servlet worked...");  //2.通过PrintWriter对象向浏览器端发送响应信息  PrintWriter writer = res.getWriter();  writer.write("Servlet response");  writer.close();  } |

⑤在web.xml配置文件中注册MyFirstServlet

|  |
| --- |
| <!-- 声明一个Servlet -->  <servlet>  <!-- 为Servlet指定一个名称，以便于引用 -->  <servlet-name>MyFirstServlet</servlet-name>  <!-- 注册Servlet实现类的全类名 -->  <servlet-class>com.atguigu.servlet.MyFirstServlet</servlet-class>  </servlet>  <!-- 建立一个从虚拟路径到Servlet之间的映射关系 -->  <servlet-mapping>  <!-- 引用Servlet名称 -->  <servlet-name>MyFirstServlet</servlet-name>  <!-- 映射到Servlet的虚拟路径：“/MyFirstServlet” -->  <url-pattern>/MyFirstServlet</url-pattern>  </servlet-mapping> |

⑥在WebContent目录下创建index.html

⑦在index.html中加入超链接

<a href=*"MyFirstServlet"*>To Servlet</a>

⑧点击超链接测试Servlet

* 1. 运行分析
* index.html



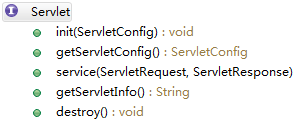
* web.xml



1. Servlet技术体系（了解）
   1. Servlet



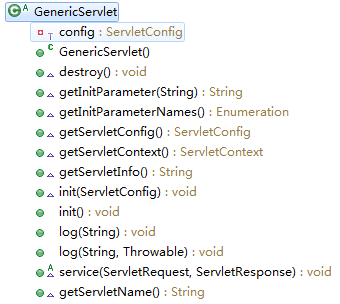
①Servlet接口



②GenericServlet抽象类



●对Servlet功能进行了封装和完善，将service(ServletRequest req, ServletResponse res)保留为抽象方法，让使用者仅关心业务实现即可。

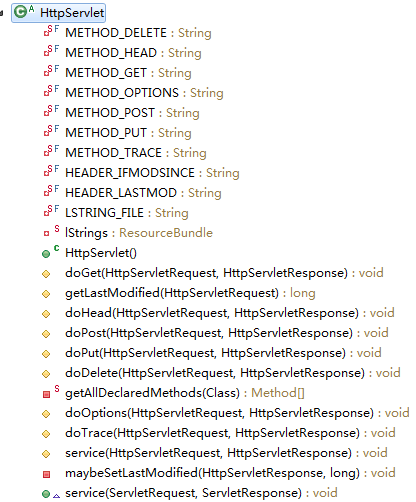


③HttpServlet抽象类



●对GenericServlet进行了进一步的封装和扩展，更贴近HTTP协议下的应用程序编写，在service(ServletRequest req, ServletResponse res)方法中，根据不同HTTP请求类型调用专门的方法进行处理。

●今后在实际使用中继承HttpServlet抽象类创建自己的Servlet实现类即可。重写doGet(HttpServletRequest req, HttpServletResponse resp)和doPost(HttpServletRequest req, HttpServletResponse resp)方法实现请求处理，不再需要重写service(ServletRequest req, ServletResponse res)方法了。



* 1. ServletConfig接口：封装了Servlet配置信息
  2. ServletContext接口：封装了当前Web应用上下文信息
  3. HttpServletRequest接口：封装了HTTP请求信息，ServletRequest的子接口
  4. HttpServletResponse接口：封装了HTTP响应信息，ServletResponse的子接口

1. Servlet生命周期
   1. 应用程序中的对象不仅在空间上有层次结构的关系，在时间上也会因为处于程序运行过程中的不同阶段而表现出不同状态和不同行为——这就是对象的生命周期。
   2. Servlet对象是Servlet容器创建的，生命周期方法都是由容器调用的。这一点和我们之前所编写的代码有很大不同。在今后的学习中我们会看到，越来越多的对象交给容器或框架来创建，越来越多的方法由容器或框架来调用，开发人员要尽可能多的将精力放在业务逻辑的实现上。
   3. Servlet生命周期的主要过程

①Servlet对象的创建

默认情况下，Servlet容器第一次收到HTTP请求时创建对应Servlet对象。容器之所以能做到这一点是由于我们在注册Servlet时提供了全类名，容器使用反射技术创建了Servlet的对象。

②Servlet对象初始化

●Servlet容器创建Servlet对象之后，会调用init(ServletConfig config)方法，对其进行初始化。在javax.servlet.Servlet接口中，public void init(ServletConfig config)方法要求容器将ServletConfig的实例对象传入，这也是我们获取ServletConfig的实例对象的根本方法。

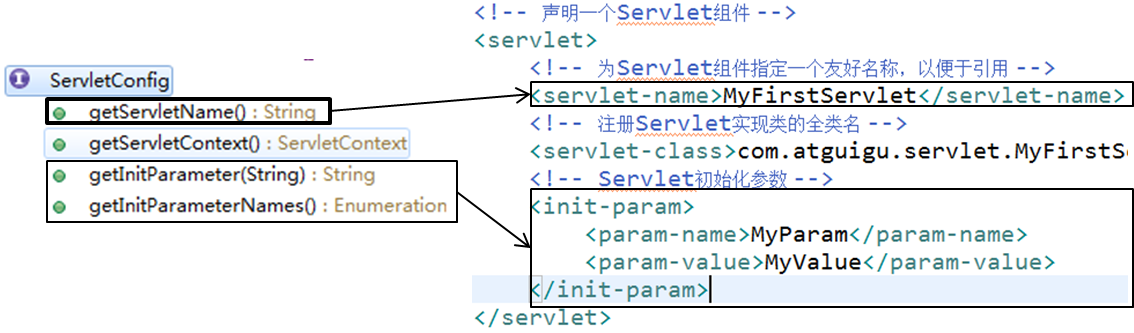
③处理请求

●在javax.servlet.Servlet接口中，定义了service(ServletRequest req, ServletResponse res)方法处理HTTP请求，同时要求容器将ServletRequest对象和ServletResponse对象传入。

④Servlet对象销毁

●服务器重启或服务器停止执行时会销毁Servlet对象，而销毁之前为了执行一些诸如释放缓存、关闭连接、保存数据等操作，所以设计了public void destroy()方法。

1. ServletConfig接口



* 1. ServletConfig接口封装了Servlet配置信息，这一点从接口的名称上就能够看出来。但同时，代表当前Web应用的ServletContext对象也封装到了ServletConfig对象中，使ServletConfig对象成为了获取ServletContext对象的一座桥梁。
  2. ServletConfig对象的主要功能

①获取Servlet名称

②获取Servlet初始化参数

③获取ServletContext对象

1. ServletContext接口



* 1. Web容器在启动时，它会为每个Web应用程序都创建一个对应的ServletContext对象，它代表当前Web应用——作用很像餐厅的经理。



* 1. 由于一个Web应用程序中的所有Servlet都共享同一个ServletContext对象，所以ServletContext对象也被称为 application 对象（Web应用程序对象）。
  2. 在应用程序中能够获取运行环境或容器信息的对象通常称之为“上下文对象”。
  3. ServletContext对象的主要功能

①获取WEB应用程序的初始化参数

●设置Web应用初始化参数的方式是在web.xml的根标签下加入如下代码

|  |
| --- |
| <!-- Web应用初始化参数 -->  <context-param>  <param-name>ParamName</param-name>  <param-value>ParamValue</param-value>  </context-param> |

●获取Web应用初始化参数

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //1.获取ServletContext对象  ServletContext context = **this**.getServletContext();  //2.获取Web应用初始化参数  String paramValue = context.getInitParameter("ParamName");  System.*out*.println("paramValue="+paramValue);  } |

②获取虚拟路径所映射的本地路径

●虚拟路径：浏览器访问Web应用中资源时所使用的路径。

●本地路径：资源在文件系统中的实际保存路径。

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  //1.获取ServletContext对象  ServletContext context = **this**.getServletContext();  //2.获取index.html的本地路径  //index.html的虚拟路径是“/index.html”,其中“/”表示当前Web应用的根目录，  //即WebContent目录  String realPath = context.getRealPath("/index.html");  //realPath=D:\DevWorkSpace\MyWorkSpace\.metadata\.plugins\  //org.eclipse.wst.server.core\tmp0\wtpwebapps\MyServlet\index.html  System.*out*.println("realPath="+realPath);  } |

③application域范围的属性

1. HttpServletRequest接口

①该接口是ServletRequest接口的子接口，封装了HTTP请求的相关信息，由Servlet容器创建其实现类对象并传入service(ServletRequest req, ServletResponse res)方法中。以下我们所说的HttpServletRequest对象指的是容器提供的HttpServletRequest实现类对象。

②HttpServletRequest对象的主要功能有

●获取请求参数。

●在请求域中保存数据。

●将请求转发给另外一个URL地址，参见[请求的转发与重定向]。

●获取项目虚拟路径。

1. HttpServletResponse接口

①该接口是ServletResponse接口的子接口，封装了HTTP响应的相关信息，由Servlet容器创建其实现类对象并传入service(ServletRequest req, ServletResponse res)方法中。以下我们所说的HttpServletResponse对象指的是容器提供的HttpServletResponse实现类对象。

②主要功能

●使用PrintWriter对象向浏览器输出数据

●实现请求重定向，参见[请求的转发与重定向]。

1. 请求的转发与重定向
   1. 请求的转发与重定向是Servlet控制页面跳转的主要方法，在Web应用中使用非常广泛。
   2. 请求的转发

①Servlet接收到浏览器端请求后，进行一定的处理，先不进行响应，而是在服务器端内部“转发”给其他Servlet程序继续处理。在这种情况下浏览器端只发出了一次请求，浏览器地址栏不会发生变化，用户也感知不到请求被转发了。

②转发请求的Servlet和目标Servlet共享同一个request对象。

③实现转发的API

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request,  HttpServletResponse response) **throws** ServletException, IOException {  //1.使用RequestDispatcher对象封装目标资源的虚拟路径  RequestDispatcher dispatcher = request  .getRequestDispatcher("/index.html");  //2.调用RequestDispatcher对象的forward()方法“前往”目标资源  //[注意：传入的参数必须是传递给当前Servlet的service方法的  //那两个ServletRequest和ServletResponse对象]  dispatcher.forward(request, response);  } |

* 1. 请求的重定向

①Servlet接收到浏览器端请求并处理完成后，给浏览器端一个特殊的响应，这个特殊的响应要求浏览器去请求一个新的资源，整个过程中浏览器端会发出两次请求，且浏览器地址栏会改变为新资源的地址。

②重定向的情况下，原Servlet和目标资源之间就不能共享请求域数据了。

③实现重定向的API

|  |
| --- |
| **protected** **void** doGet(HttpServletRequest request,  HttpServletResponse response) **throws** ServletException, IOException {  //1.调用HttpServletResponse对象的sendRedirect()方法  //2.传入的参数是目标资源的虚拟路径  response.sendRedirect("index.html");  } |

* 1. 对比请求的转发和重定向

|  |  |  |
| --- | --- | --- |
|  | 转发 | 重定向 |
| 浏览器地址栏 | 不改变 | 改变 |
| 发送请求次数 | 1 | 2 |
| 能否共享request对象数据 | 能 | 否 |
| 目标资源：WEB-INF下的资源 | 能访问 | 不能访问 |