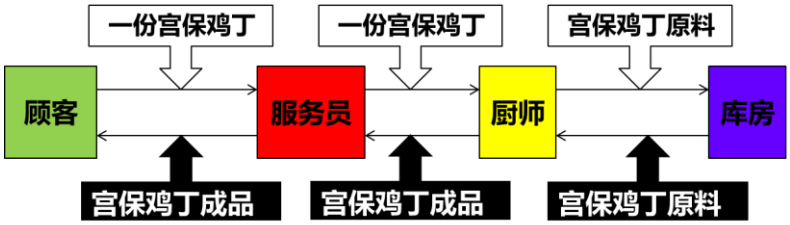
Servlet

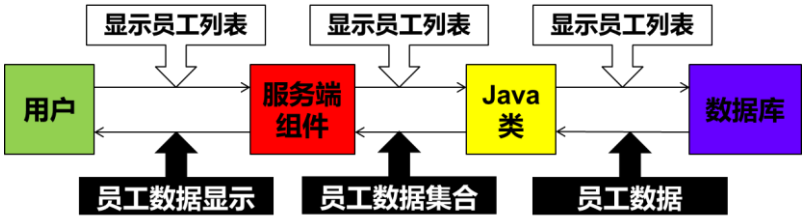
1 Why？我们为什么需要 Servlet？

1.1 Web 应用基本运行模式

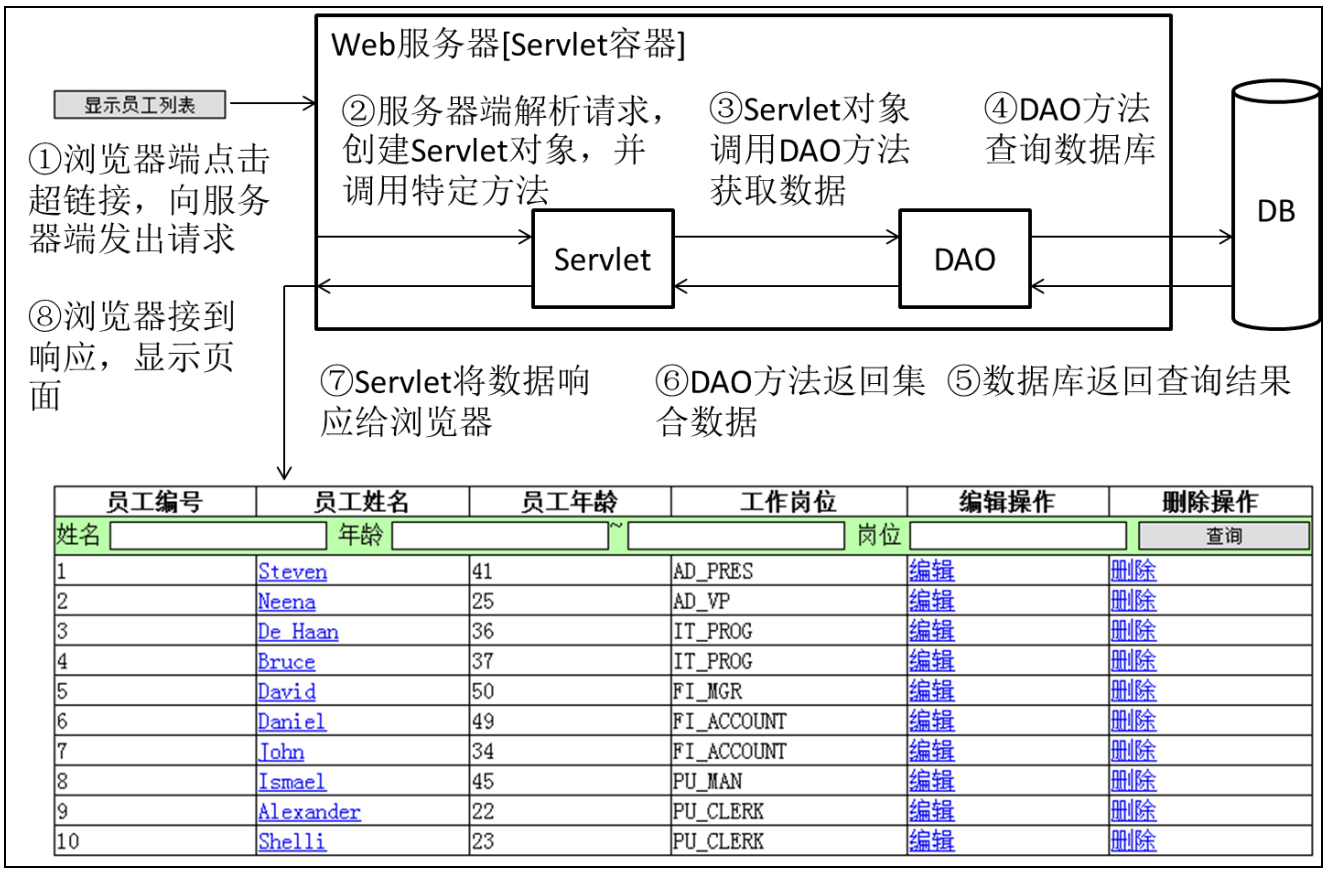
①生活中的例子



②Web 应用运行模式



1.2 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。



员工编号	员工姓名	员工年龄	工作岗位	编辑操作	删除操作
姓名		年龄	~	岗位	查询
1	Steven	41	AD_PRES	编辑	删除
2	Neena	25	AD_VP	编辑	删除
3	De Haan	36	IT_PROG	编辑	删除
4	Bruce	37	IT_PROG	编辑	删除
5	David	50	FI_MGR	编辑	删除
6	Daniel	49	FI_ACCOUNT	编辑	删除
7	John	34	FI_ACCOUNT	编辑	删除
8	Ismael	45	PU_MAN	编辑	删除
9	Alexander	22	PU_CLERK	编辑	删除
10	Shelli	23	PU_CLERK	编辑	删除

2 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。

3 How？如何使用 Servlet？

3.1 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组件 -->
<servlet>
    <!-- 为Servlet组件指定一个友好名称，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组件之间的映射关系 -->
<servlet-mapping>
    <!-- 引用Servlet组件名称 -->
    <servlet-name>MyFirstServlet</servlet-name>
    <!-- 映射到Servlet的虚拟路径：“/MyFirstServlet” -->
    <url-pattern>/MyFirstServlet</url-pattern>
</servlet-mapping>
```

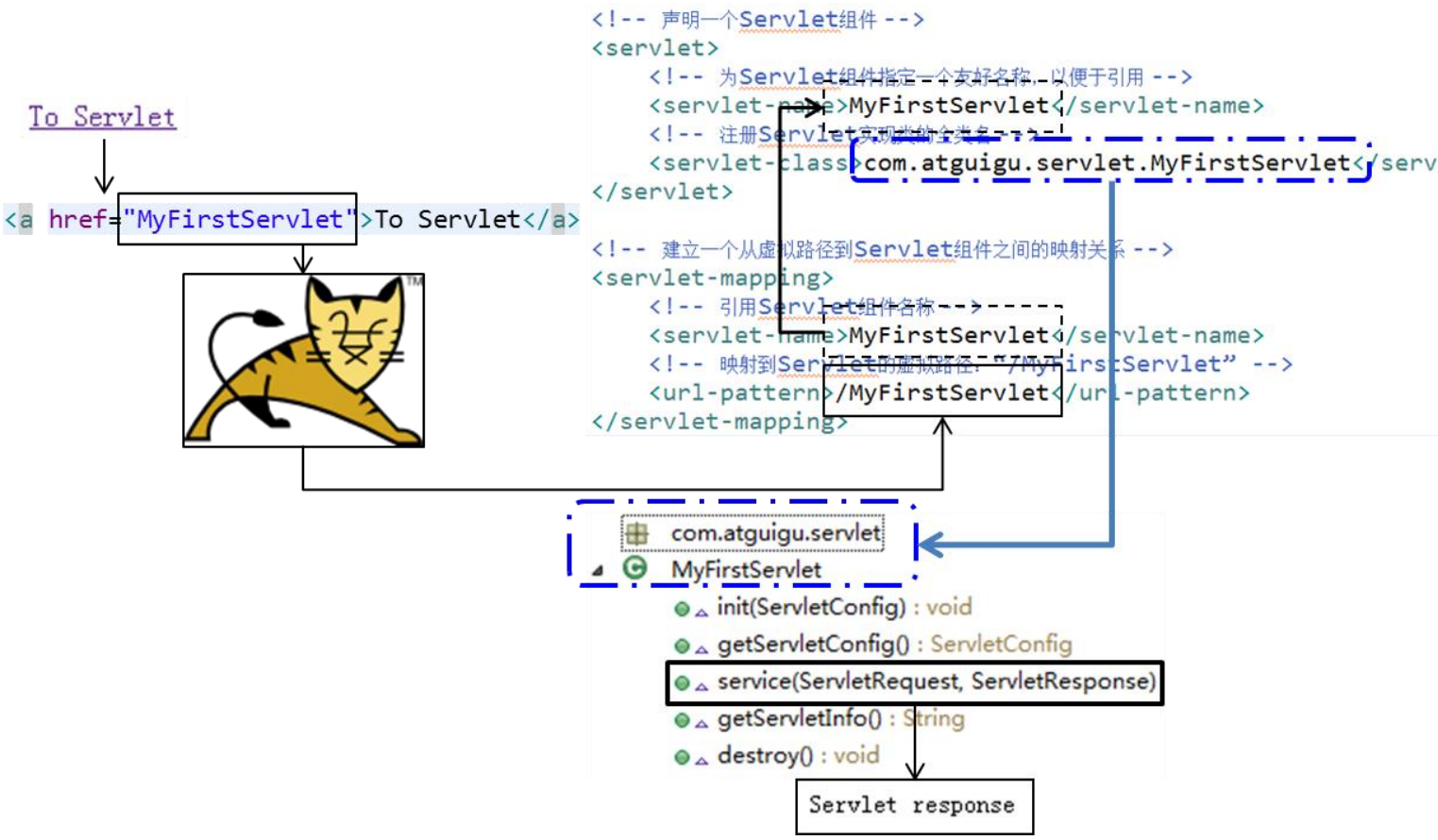
⑥在 WebContent 目录下创建 index.html

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

3.2 运行分析



4 Servlet 技术体系

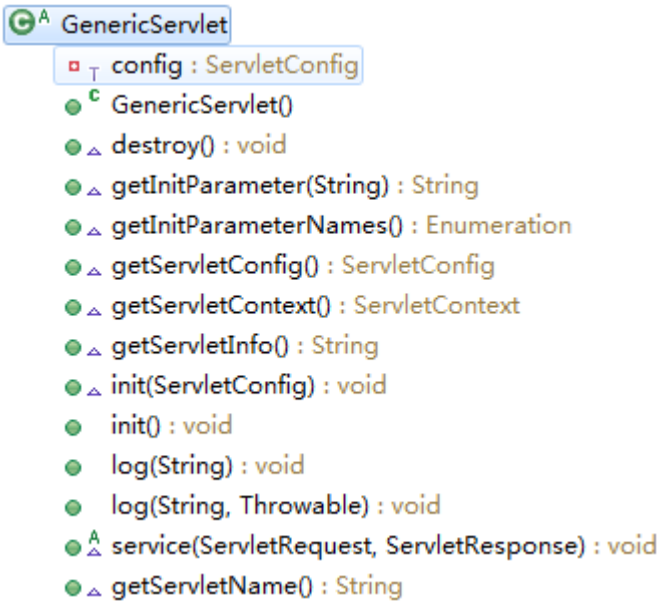
4.1 Servlet

①Servlet 接口

②GenericServlet 抽象类

```
public abstract class GenericServlet
    implements Servlet, ServletConfig, java.io.Serializable
```

●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。



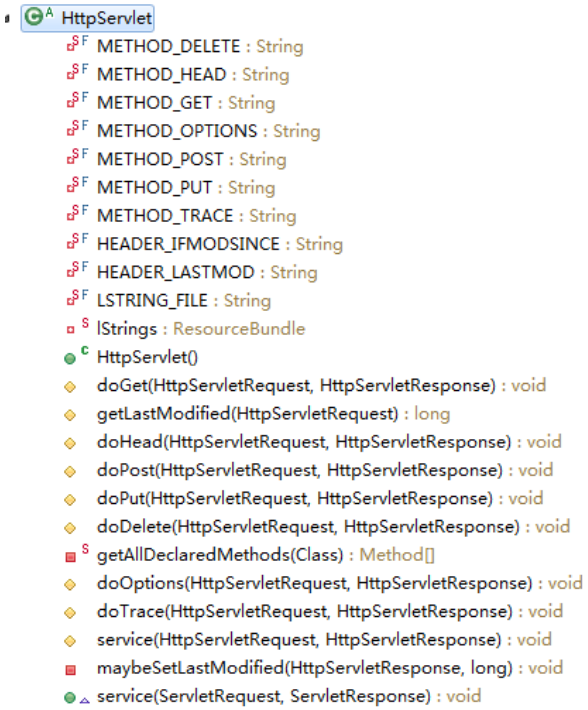
③HttpServlet 抽象类

```
public abstract class HttpServlet extends GenericServlet
    implements java.io.Serializable {
```

●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。



4.2 ServletConfig 接口：封装了 Servlet 配置信息

4.3 ServletContext 接口：封装了当前 Web 应用上下文信息

4.4 HttpServletRequest 接口：封装了 HTTP 请求信息，ServletRequest 的子接口

4.5 HttpServletResponse 接口：封装了 HTTP 响应信息，ServletResponse 的子接口

5 Servlet 生命周期

5.1 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。

5.2 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。

5.3 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。

[参考阅读]
●这是依赖注入思想的一种实现，所谓依赖注入指的是程序中需要的对象不是由开发人员自己创建或获取，而是由“环境”——也就是容器主动“注入”到我们的程序中。这样做的好处是屏蔽复杂对象的创建细节或简化开发过程。这一思想在很多框架中都得到了体现，特别是 Spring 的 IOC 容器就是依赖注入思想的典型实现。

●为了简化开发，GenericServlet 抽象类中实现了 init(ServletConfig config)方法，将 init(ServletConfig config)方法获取到的 ServletConfig 对象赋值给了成员变量 ServletConfig config，目的是使其它方法可以共享这个对象。这时有一个问题：如果子类重写了这个 init(ServletConfig config)方法，有可能会导致成员变量 config 对象赋值失败。所以 GenericServlet 抽象类另外提供了一个无参的 public void init()方法，并在 init(ServletConfig config)方法中调用，作为子类进行初始化操作时重写使用。而这个无参的 init()方法之所以没有设计成抽象方法，是为了避免子类继承时强制实现这个方法带来的麻烦，使用者可以根据需要选择是否要覆盖这个方法。

③处理请求

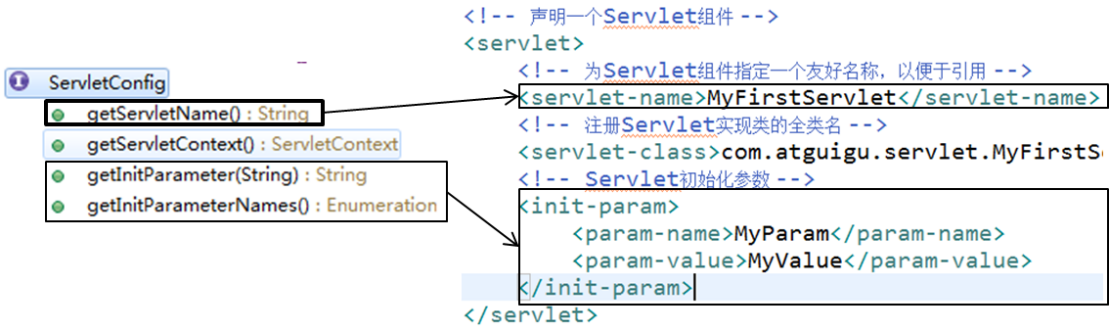
●在 javax.servlet.Servlet 接口中，定义了 service(ServletRequest req, ServletResponse res)方法处理 HTTP 请求，同时要求容器将 ServletRequest 对象和 ServletResponse 对象传入。

●在 HttpServlet 抽象类中，service(ServletRequest req, ServletResponse res)方法将 ServletRequest 对象和 ServletResponse 对象强转为了 HttpServletRequest、HttpServletResponse 子类对象，这样更适合于 HTTP 请求的处理，所以在 doGet()和 doPost()方法中使用的就是 HttpServletRequest、HttpServletResponse 的实现类对象了。

④Servlet 对象销毁

●Web 应用卸载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。

6 ServletConfig 接口



6.1 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。

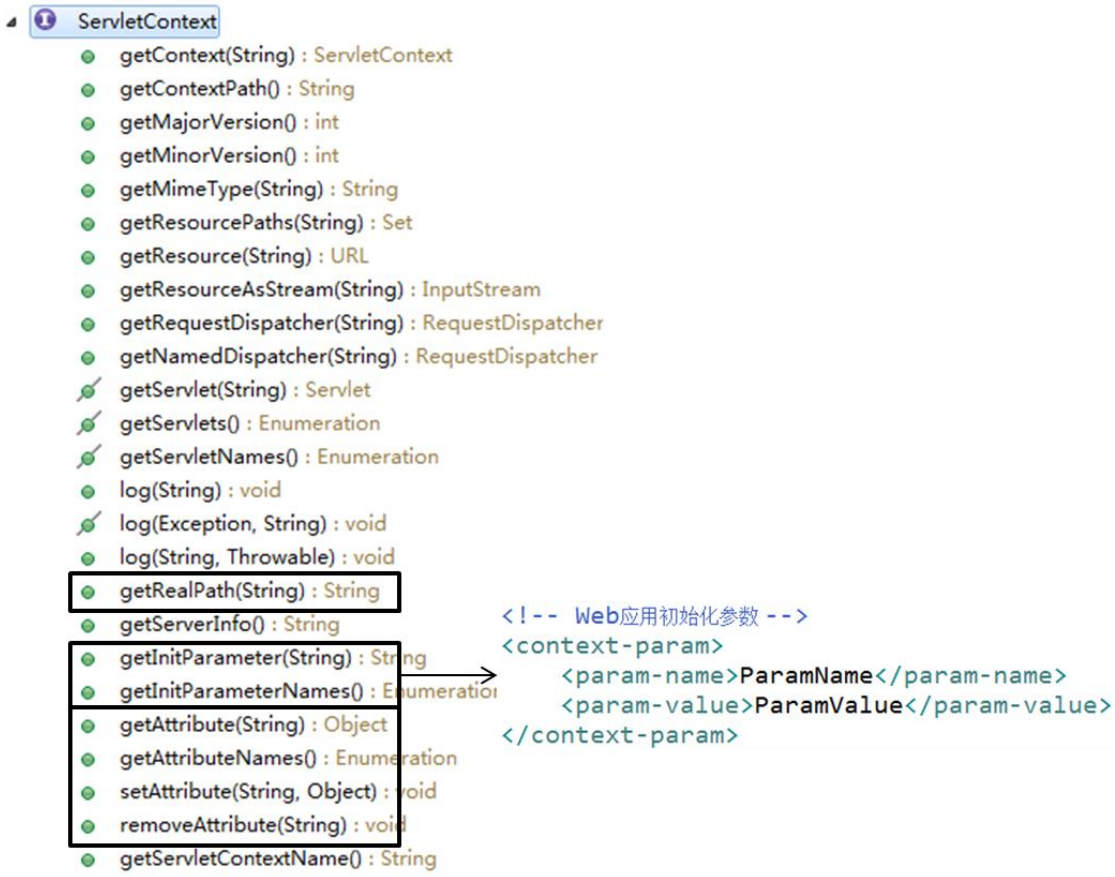
6.2 ServletConfig 对象的主要功能

①获取 Servlet 友好名称

②获取 Servlet 初始化参数

③获取 ServletContext 对象

7 ServletContext 接口



7.1 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。

7.2 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）。

7.3 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”。

7.4 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

7.5 ServletContext 对象与当前 Web 应用一致，即 Web 应用加载时创建，Web 应用卸载时销毁。

8 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

[1]什么是请求参数？就是浏览器向服务器端提交的数据

[2]浏览器端如何发送请求参数

○附着在 URL 地址后面

```
http://localhost:8989/MyServlet/MyHttpServlet?userId=20
```

○表单提交

```
<form action="MyHttpServlet" method="post">

        你喜欢的足球队<br /><br />

        巴西<input type="checkbox" name="soccerTeam" value="Brazil" />
        德国<input type="checkbox" name="soccerTeam" value="German" />
        荷兰<input type="checkbox" name="soccerTeam" value="Holland" />
        中国<input type="checkbox" name="soccerTeam" value="China" />
        法国<input type="checkbox" name="soccerTeam" value="French" />
        意大利<input type="checkbox" name="soccerTeam" value="Italy" />

        <br /><br />

        <input type="submit" value="提交" />

    </form>
```

[3]使用 HttpServletRequest 对象获取请求参数

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //一个name对应一个值
        String userId = request.getParameter("userId");
        System.out.println("userId="+userId);
    }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //一个name对应一组值
        String[] soccerTeams = request.getParameterValues("soccerTeam");
        for(int i = 0; i < soccerTeams.length; i++){
            System.out.println("team "+i+"="+soccerTeams[i]);
        }
    }
```

[说明：其实在实际使用中，很少将数据转发给一个 Servlet，更多的是将数据转发给一个 JSP 在页面上显示——而 JSP 之所以能够接收到数据是因为 JSP 本质上也是一个 Servlet。这是我们在后面要学习的内容。]

●在请求域中保存数据：将数据保存在请求域中，可以以转发的方式发送给其他 Servlet。

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        //1.将数据保存到request对象中
```

```
            request.setAttribute("myName", "FengJie");
            //2.转发到接收数据的Servlet
            request.getRequestDispatcher("/ReceiveServlet")
                    .forward(request, response);
        }
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
            //从request对象属性域中获取数据
            String myName = (String) request.getAttribute("myName");
            System.out.println("myName="+myName);
        }
```

●将请求转发给另外一个 URL 地址，参见[请求的转发与重定向]。

●获取与 HTTP 请求相关的信息，在学习 HTTP 协议时讨论。

9 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

```
        //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();
```

●实现请求重定向，参见[请求的转发与重定向]。

10 请求的转发与重定向

10.1 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。

10.2 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[这么说似乎不是很准确，因为如果我们分别输出转发前和转发后 request 对象的 hashCode 值会发现其实它们并不相等——这说明它们严格的说并不是同一个对象——但至少它们之间是可以共享请求域数据的]。

③用现实生活中的例子来说明，请求的转发很像这样一种情景：顾客在餐厅点菜，告诉了服务器张曼玉，张曼玉告诉了厨师，厨师完成之后将菜品交给了服务员林青霞，林青霞再交给顾客。在这个过程中顾客只发出了一次请求，但这个请求是先后由两个服务员执行的。

④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);
    }
```

10.3 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。这时浏览器会自动去访问这个新的资源的地址。整个过程无需用户操作的参与，但浏览器端发出的是两次请求，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请求的转发是在服务器内部进行的，所以目标资源只能是当前 Web 应用内的其他组件；而重定向是让浏览器请求新的资源，所以没有这个限制。

④用现实生活中的例子来说明：有些餐厅点餐的方式是顾客先在第一个窗口付款，凭小票到下一个窗口取餐，在这个过程中顾客发出了两次请求。

⑤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");
    }
```

11 作业

11.1 Servlet 的 HelloWorld

要求：创建一个 Java 类，实现 Servlet 接口，然后从浏览器端访问这个 Servlet。

11.2 HttpServlet

要求：使用 Eclipse 自动创建一个 Servlet，使用表单向这个 Servlet 发送数据，在 Servlet 中打印获取到的数据。

11.3 测试 Servlet 的生命周期方法，体会生命周期的含义，以及 Servlet 的工作过程。

11.4 Servlet 初始化参数

要求：在 web.xml 中创建 Servlet 的初始化参数，在 Servlet 中获取并打印。

11.5 Web 应用初始化参数

要求：在 web.xml 中创建整个 Web 应用的初始化参数，在 Servlet 中获取并打印。