1. Servlet 概述

1.1. Servlet 过程

运行在 Servlet 容器中的 java 类，能处理 Web 客户的 Http 请求，并产生 Http 响应

1. 请求处理和响应过程

- 1）接收 `Http` 请求
- 2）取得请求信息，包括**请求头和参数数据**
- 3）调用其它 `Java` 类方法完成具体的业务功能
- 4）实现到其它 `Web` 组件的跳转（**重定向或请求转发**）
- 5）生成 `Http` 响应（包括 `HTML` 或非 `HTML`）

2. 容器

- `Servlet` 运行在服务端，由 `Servlet` 容器管理

1.2. Servlet 体系结构

Servlet 是使用 Servlet API 以及相关类和方法的 Java 程序

Servlet API 包含两个软件包

- `javax.servlet` 包
- `javax.servlet.http` 包

Servlet 接口

- 提供与 `Servlet` 生命周期相关的方法，如 `init()、service()、destroy()` 方法
- 直接继承需要重写用到的方法
- `GenericServlet` 类：继承了 `Servlet` 接口的所有方法
- `HttpServlet` 类
    - 继承了 `GenericServlet` 类并是Http 协议
    - 用于在 Web 中处理 Http 请求的 Servlet

1）Servlet 生命周期

- 七种状态
    - 创建、初始化、服务可用、服务不可用、处理请求、终止服务、销毁
- 四个阶段
    - 1）加载和实例化
    - 2）初始化
    - 3）处理请求

- 4）销毁

2）加载和实例化

- `Servlet` 的创建指**加载和实例化**两个过程
- `Servlet` 容器在以下时刻加载和实例化一个 Servlet
  - 服务器运行中，客户机**首次**向 Servlet 发出请求时
  - 重新装入 Servlet 时，服务器重新启动、Servlet 被修改等
  - 在为 Servlet 配置了自动装入选项（`load-on-startup`）时，服务器在启动时会自动装入

3）初始化

- `Servlet` 实例化后，`Servlet` 容器将调用 `Servlet` 的 `init(ServletConfig config)` 方法来对 `Servlet` 实例进行初始化；
- 如果初始化没有问题，`Servlet` 在 `Web` 容器中会处于服务可用状态；如果初始化失败，`Servlet` 容器会从运行环境中清除掉该实例；
- 当 `Servlet` 运行出现异常时，`Servlet` 容器会使该实例变为服务不可用状态。`Web` 程序维护人员可以设置 `Servlet`，使其成为服务不可用状态，或者从服务不可用状态恢复成服务可用状态。

4）处理请求

- 服务器接收到客户端请求,会为该请求创建一个 `Request` 对象和一个 `Response` 对象并调用 `service()` 方法，`service()` 方法再调用其他方法来处理请求；
- 在 Servlet 生命周期中，`service()` 方法可能被多次调用。当多个客户端同时访问某个 Servlet 的 `service()` 方法时，服务器会为**每个请求创建一个线程**，这样可以并行处理多个请求，减少请求处理的等待时间，提高服务器的响应速度。但同时也要注意对同一对象的并发访问问题。

5）销毁

当 Servlet 容器需要终止 `Servlet`（比如Web服务器即将被关掉或需要出让资源），它会先调用 `Servlet` 的 `destroy()` 方法使其释放正在使用的资源。在 `Servlet` 容器调用 `destroy()` 方法之前，必须让当前正在执行 `service()` 方法的任何线程完成执行,或者超过了服务器定义的时间限制。在 `destroy()` 方法完成后，`Servlet` 容器必须释放 `Servlet` 实例以便被垃圾回收。

2. 创建 Servlet

2.1. 声明配置

方式

- `@WebServlet` 注解
- `Web.xml` 文件

常用属性

- `@WebServlet`
    - `name` ： Servlet 的名字
    - `urlPatterns`： url 匹配模式，映射地址
    - `value` ： 等价于 `urlPatterns`
    - `loadOnStartup`： 0 表示启动时加载，当值 ≥ 0 时，值越小优先级越高
    - `asyncSupported`： 表示是否支持异步操作，默认为 `False`
- `Web.xml`
    - `<servlet-name>`： 与 `Servlet` 类名相同，要求一个 `web.xml` 文件内名字唯一
    - `<load-on-startup>`
    - `<servlet-name>`：
    - `<url-pattern>` ： 指定 `URL` 匹配模式，等价于`urlPatterns`、 `value`

3. Servlet 的应用

3.1. 数据处理

1. **超链接**

- 为`Get` 请求，调用`doGet()`，在 `url` 中 `?`后面即参数 `&` 为关联符
- 当超链接请求到达 `Servlet` 时，包含数据请求将被容器转化为 `HttpServletRequest` 对象
- `HttpServletRequest` 处理方法
    - `public String getParamter(String name)`
    - `public String [] getParamterValues(String name)`
    - `public Enumeration getParamterNames()`

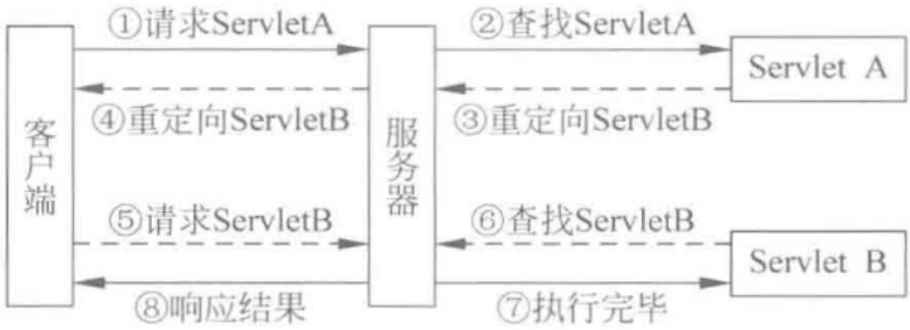
2. **表单**

- 为 `Post` 请求，调用 `doPost()`
- `method` 设置请求类型

3.2. 重定向和请求转发

1. **重定向**

- 指原请求地址重新定位到新的地址，原有的 `request` 请求失效，客户端地址变为新的地址
    - `public void sendRedirect(String location)`
    - `response.sendRedirect(String location)`
        - `location` 指定重定向的 `url`，可以是相对也可以是绝对



2. **请求转发**

- 将请求转发到其它地址，转发过程使用 `request` 请求，浏览器地址不变
- 请求转发发生在服务器内部，对客户端是透明的，服务器只能从应用内部查找相应的转发资源，不能转发到其他应用的资源
- 使用的是 `RequestDispatcher` 中的 `forward` 方法实现
    - `forward()` 方法：转发请求给其他资源
    - `include()` 方法：将其他资源并入到当前请求中
        - `RequestDispatcher dis = request.getRequestDispatcher(String path)`
            - `path` ：`/Servlet`
            - `path` 指定转发的 `url` ，只能是相对路径
        - `dispatcher.forward(ServletRequest req, ServletResponse resp)`
            - 请求皆为 `HttpServlet` 对象
- 请求转发中的相对路径 `/` 表示的是当前应用程序的根目录，重定向表示的是 `Web` 站点的根目录

3. **区别**

- 转发只能将请求转发给同一个 Web 应用中的组件;**重定向不仅可以重定向到当前应用程序中的其他资源**，还可以重定向到同一个站点上的其他应用程序中的资源,或者重定向到其他站点的资源;
- 重定向的访问过程结束后,浏览器地址栏中显示的 URL 会发生改变，由初始的URL地址变成重定向的目标URL ;而请求转发过程结束后,浏览器地址栏保持初始的 URL 地址不变;
- 重定向对浏览器的请求直接作出响应,响应的结果就是告诉浏览器去重新发出对另外一个 URL 的访问请求;请求转发在服务器端内部将请求转发给另外一个资源,浏览器只知道发出了请求并得到了响应结果,并不知道在服务器程序内部发生了转发行为;
- **请求转发调用者与被调用者之间共享相同的 `request` 对象和 `response` 对象**,它们属于同一个访问请求和响应过程; 而重定向调用者与被调用者使用各自的 `request` 对象和 `response` 对象,它们属于两个独立的访问请求和响应过程。