JavaWeb之Servlet

> ## 本节目标
>
> 1.掌握Servlet编程的核心类 2.掌握Servlet的什么周期

1. Servlet

1.1 Servlet

1.1.1 API核心包

Servlet API有以下4个Java包：

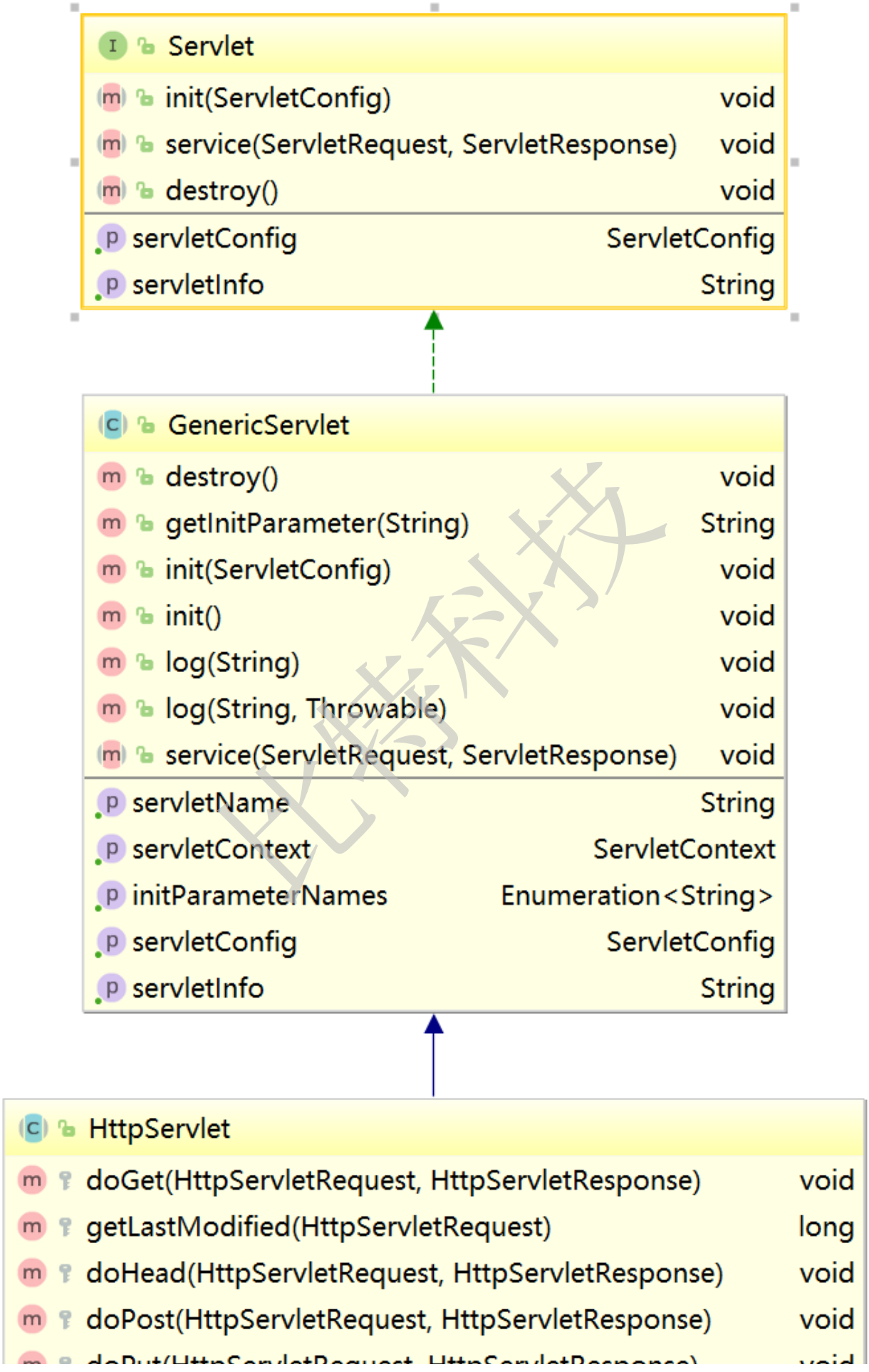
- javax.servlet：其中包含定义Servlet和Servlet容器之间的锲约类和接口
- javax.servlet.http：其中包含定义HTTP Servlet和Servlet容器之间的契约类和接口
- javax.servlet.annotation：其中包含标注Servlet，Filter，Listener的标注，它还为被标注元件定义元数据

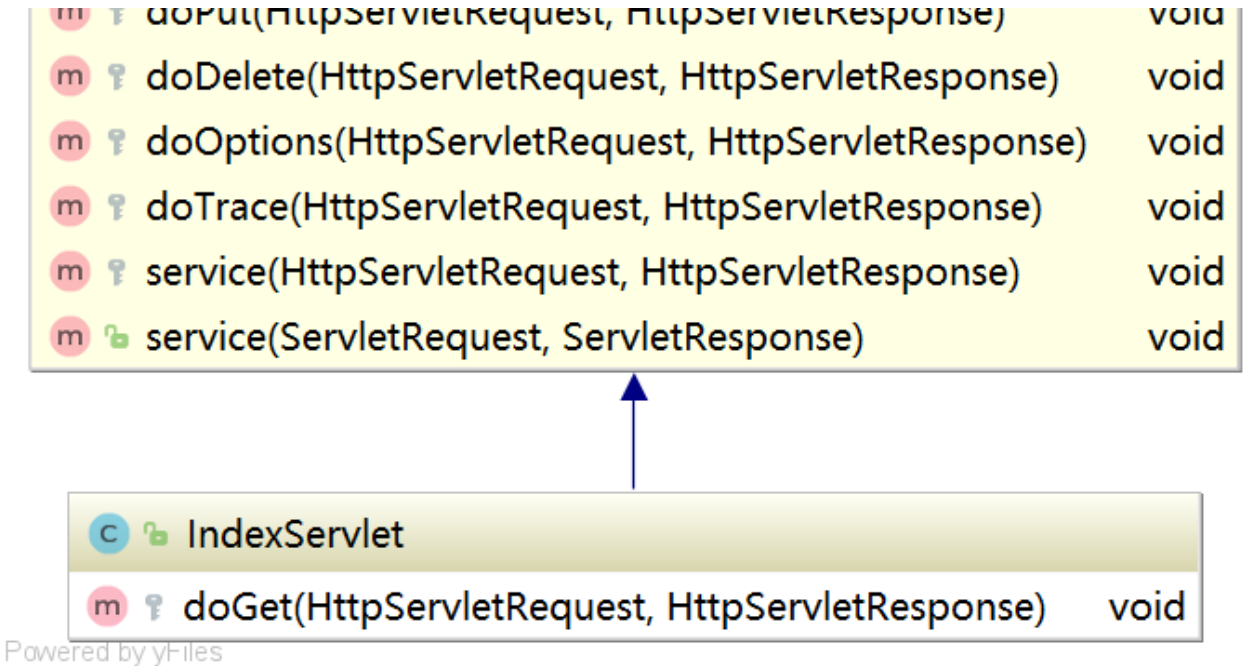
> 备注：JavaWeb中主要关注javax.servlet和javax.servlet.http的成员。

1.1.2 Servlet实现

Servlet技术的核心是Servlet，它是所有Servlet类必须之间或者间接实现的一个接口。在编写实现Servlet的Servlet类时，直接实现它。在扩展实现这个接口的类时，间接实现它。

下面图展示了用户自定义的Servlet类于Servlet接口的扩展关系：

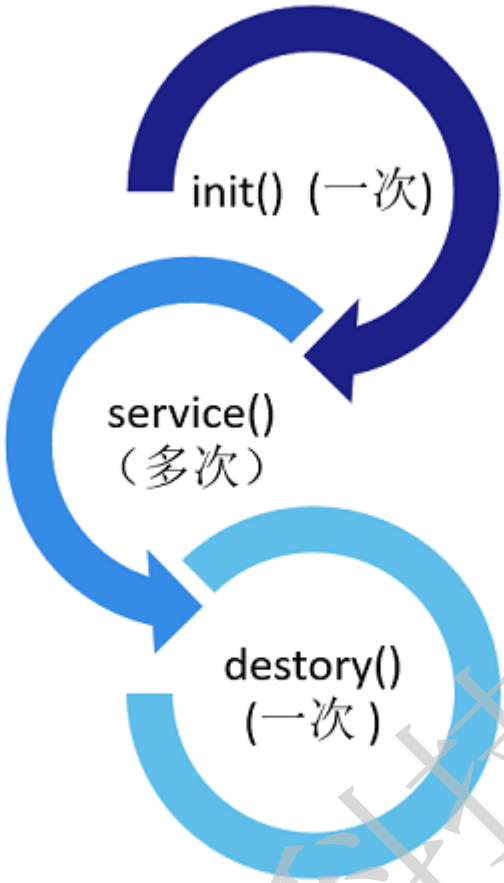

Servlet
init(ServletConfig) void
service(ServletRequest, ServletResponse) void
destroy() void
servletConfig ServletConfig
servletInfo String
GenericServlet
destroy() void
getInitParameter(String) String
init(ServletConfig) void
init() void
log(String) void
log(String, Throwable) void
service(ServletRequest, ServletResponse) void
servletName String
servletContext ServletContext
initParameterNames Enumeration<String>
servletConfig ServletConfig
servletInfo String
HttpServlet
doGet(HttpServletRequest, HttpServletResponse) void
getLastModified(HttpServletRequest) long
doHead(HttpServletRequest, HttpServletResponse) void
doPost(HttpServletRequest, HttpServletResponse) void



`HttpServlet`是适用于Web站点的`Servlet`类，其中`IndexServlet`是用户自定义的`Servlet`类。

1.1.3 Servlet方法

`init,service,destroy`是Servlet的生命周期方法。Servlet容器回根据以下规则调用这3个方法：



Servlet 生命周期方法调用

- init ： 当该Servlet第一被请求时，Servlet容器回调用该方法，后续请求不会再被调用。可以利用该方法进行相应的初始化功能。
- service：每当请求Servlet时，Servlet容器就会调用这个方法。
- destory：当要销毁Servlet时，Servlet容器就会调用该方法。通常再卸载程序，或者关闭Servlet容器就回发生调用销毁方法，一般会在这里写一下清除工作的代码。

注意Servlet的线程安全性，Servlet实例会被一个应用程序中的所有用户共享，因此不建议使用类级别变量，除非它们时只读的或支持并发修改。

下面案例通过使用Servlet的属性，展示并发访问时引起的数据异常问题：

```
public class ChanceServlet extends HttpServlet {

    private int chanceNumber;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        chanceNumber = Integer.parseInt(config.getInitParameter("chanceNumber"));
    }

    @Override
```

```
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        writer.append("<html>")
                .append("<head>")
                .append("<meta charset=\"UTF-8\">")
                .append("</head>")
                .append("<body>");
        String name = req.getParameter("name");
        if (number > 0) {
            number = number - 1;
            writer.append("<h1>")
                .append(name)
                .append("剩余")
                .append(String.valueOf(number))
                .append("次机会")
                .append("</h1>");
        } else {
            writer.append("<h1>")
                .append(name)
                .append("没机会了")
                .append("</h1>");
        }
        writer.append("</body>").append("</html>");
    }


    @Override
    public void destroy() {
        super.destroy();
        System.out.println("destroy");
    }
}
```

```
<servlet>
    <servlet-name>ChanceServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.sevlet.ChanceServlet</servlet-class>
    <!--初始化参数-->
    <init-param>
        <param-name>chanceNumber</param-name>
        <param-value>20</param-value>
    </init-param>
</servlet>
```

> 疑问：如何解决此问题？ 稍后ServletContext见。

1.2 ServletConfig

当Servlet容器初始化Servlet时，Servlet容器会给Servlet的init方法传入一个ServletConfig。ServletConfig封装可以通过部署描述符传给Servlet的配置信息。这样传入的每一条信息就叫做初始化参数，一个初始化参数有key和value组成。

```
<init-param>
    <param-name>logFileName</param-name>
    <param-value>filter.logging</param-value>
</init-param>
```

为了从Servlet内部获取到初始化参数的值，要在Servlet容器传给Servlet的init方法的ServletConfig中调用getInitParameter方法即可。

```
@Override
public void init(ServletConfig config) throws ServletException {
    super.init(config);
    String name = config.getInitParameter("name");
}
```

1.3 ServletContext

ServletContext表示Servlet应用程序。**每个Web应用程序只有一个上下文**，在将一个应用程序同时部署到多个容器的分布式环境中，每台Java虚拟机上的Web应用都会有一个ServletContext对象。

通过ServletConfig中调用getServletContext方法，可以获得ServletContext。

有了ServletContext，就可以**共享**从应用程序中的所有资料处访问到的信息，并且可以**动态注册Web对象**。共享信息可以将对象保存在ServletContext中的一个内部Map中，保存在ServletContext中的对象被称作属性。

> 备注：查看源码，以及基于ServletContext的方式应用程序共享数据案例

下面案例通过ServletContext共享数据，解决共享变量带来的多用户并发请求的数据访问错误问题。

```
public class ChanceServlet extends HttpServlet {

    private int chanceNumber;

    private ServletContext context;

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        chanceNumber = Integer.parseInt(config.getInitParameter("chanceNumber"));
        context = config.getServletContext();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        writer.append("<html>")
                .append("<head>")
                .append("<meta charset=\"UTF-8\">")
                .append("</head>")
                .append("<body>");
        String name = req.getParameter("name");
```

```
        Integer number = (Integer) context.getAttribute(name);
        if (number == null) {
            number = chanceNumber;
            context.setAttribute(name, number);
        }
        if (number > 0) {
            number = number - 1;
            context.setAttribute(name, number);
            writer.append("<h1>")
                .append(name)
                .append("剩余")
                .append(String.valueOf(number))
                .append("次机会")
                .append("</h1>");
        } else {
            writer.append("<h1>")
                .append(name)
                .append("没机会了")
                .append("</h1>");
        }
        writer.append("</body>").append("</html>");
    }


    @Override
    public void destroy() {
        super.destroy();
        System.out.println("destroy");
    }
}
```

```
<servlet>
    <servlet-name>ChanceServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.sevlet.ChanceServlet</servlet-class>
    <!--初始化参数-->
    <init-param>
        <param-name>chanceNumber</param-name>
        <param-value>20</param-value>
    </init-param>
</servlet>
```


1.4 HttpServlet

大多数应用程序都是要于HTTP结合起来使用。这意味着可以利用HTTP提供的特性。`javax.servlet.http`包时Servlet API中的第二个包，其中包含了用于编写Servlet应用程序的类和接口，并且许多类型都覆写了`javax.servlet`中的类型。

HttpServlet类覆盖了javax.servlet.GenericServlet类。使用HttpServlet时，还要借助分别代表Servlet请求和Servlet响应的HttpServletRequest和HttpServletResponse对象。

HttpServlet中的Service方法会检验用来发送请求的HTTP方法（通过调用request.getMethod），并调用以下方法之一：

- doGet (常用)
- doPost（常用）
- doHead
- doPut
- doTrace
- doOptions
- doDelete

这7中方法，每一种方法都表示一个HTTP方法，doGet和doPost是最常用的。因此，不再需要覆盖Service方法了，只要覆盖doGet或者doPost即可。

1.5 功能特性

1.5.1 表单提交

一个Web应用程序中几乎总是包含一个或者多个HTML表单，供用户输入值。你可以轻松的将一个HTML表达从一个Servlet发送到浏览器。当用户提交表单时，在表单元素中输入的值就会被当做请求参数发送到服务器。

- 表单视图-实现Servlet

```
public class FormServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        writer.append("<html>")
                .append("<head>")
                .append("<meta charset='UTF-8'>")
                .append("<title>Form</title>")
                .append("</head>")
                .append("<body>")
                .append("<form method='POST' action='/form'>")
                .append("请输出姓名: ")
                .append("<input name='name' type='text' value=''/>")
                .append("<input type='submit' value='提交>")
                .append("</form>")
                .append("</body>")
                .append("</html>");
    }
}
```

- 配置Servlet

```
<servlet>
    <servlet-name>FormServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.web.FormServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>FormServlet</servlet-name>
    <url-pattern>/form</url-pattern>
</servlet-mapping>
```

- 提交表单-处理逻辑

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
    String name = req.getParameter("name");
    resp.setContentType("text/html; charset=UTF-8");
    PrintWriter writer = resp.getWriter();
    writer.append("<html>")
        .append("<head>")
        .append("<meta charset='UTF-8'>")
        .append("<title>Form</title>")
        .append("</head>")
        .append("<body>")
        .append("<h1>")
        .append("欢迎, ")
        .append(name)
        .append("</h1>")
        .append("</body>")
        .append("</html>");
}
```

1.7.2 数据查询

在Web应用中检索也是一个很常见的操作，下面通过一个案例来演示如果使用查询。

- 实现查询处理Servlet，准备静态数据
- 配置Servlet

```
// 接口: /query?city=西安
public class QueryServlet extends HttpServlet {
    private Map<String, String> cityMap = new HashMap<String, String>();
    private Map<String, List<String>> scenicSpot = new HashMap<String, List<String>>();

    @Override
    public void init() throws ServletException {
        super.init();
        List<String> xian = new ArrayList<String>();
        xian.add("华清池");
        xian.add("兵马俑");
        xian.add("大雁塔");
        scenicSpot.put("xian", xian);
        cityMap.put("xian", "西安");
```

```
        List<String> baoJi = new ArrayList<String>();
        baoJi.add("太白山");
        baoJi.add("法门寺");
        baoJi.add("关山牧场");
        scenicSpot.put("baoJi", baoJi);
        cityMap.put("baoJi", "宝鸡");

        List<String> xianyang = new ArrayList<String>();
        xianyang.add("乾陵");
        xianyang.add("袁家村");
        scenicSpot.put("xianyang", xianyang);
        cityMap.put("xianyang", "咸阳");
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //处理参数
        String city = req.getParameter("city");
        //准备数据
        List<ScenicsDto> scenicsDtoArrayList = new ArrayList<ScenicsDto>();
        if (city == null || city.length() == 0) {
            //全部
            for (Map.Entry<String, List<String>> entry : scenicSpot.entrySet()) {
                String cityKey = entry.getKey();
                List<String> scenics = entry.getValue();
                for (String item : scenics) {
                    ScenicsDto scenicsDto = new ScenicsDto();
                    scenicsDto.setCity(cityMap.get(cityKey));
                    scenicsDto.setName(item);
                    scenicsDtoArrayList.add(scenicsDto);
                }
            }
        } else {
            //单个城市
            List<String> scenics = scenicSpot.get(city);
            if (scenics == null) {
                scenics = new ArrayList<String>();
            }
            for (String item : scenics) {
                ScenicsDto scenicsDto = new ScenicsDto();
                scenicsDto.setCity(cityMap.get(city));
                scenicsDto.setName(item);
                scenicsDtoArrayList.add(scenicsDto);
            }
        }

        //响应数据
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();

        writer.append("<!DOCTYPE html>\n" +
```

```
                "<html lang=\"en\">\n" +
                "<head>\n" +
                "    <meta charset=\"UTF-8\">\n" +
                "    <title>景点</title>\n" +
                "</head>\n" +
                "<body>\n" +
                "<table>\n" +
                "    <thead>\n" +
                "    <tr>\n" +
                "        <td>编号</td>\n" +
                "        <td>所在城市</td>\n" +
                "        <td>景点名称</td>\n" +
                "    </tr>\n" +
                "    </thead>\n" +
                "    <tbody>")
        ;

        int id = 1;
        for (ScenicsDto dto : scenicsDtoArrayList) {
            writer.append("<tr>")
                    .append("<td>").append(String.valueOf(id)).append("</td>")
                    .append("<td>").append(dto.city).append("</td>")
                    .append("<td>").append(dto.name).append("</td>")
                    .append("</tr>");
            id = id + 1;
        }
        writer.append("    </tbody>\n" +
                "</table>\n" +
                "</body>\n" +
                "</html>");

    }
    public static class ScenicsDto {
        private String city;
        private String name;
        //省略getter, setter方法
    }
}
```

```
<servlet>
    <servlet-name>QueryServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.sevlet.QueryServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>QueryServlet</servlet-name>
    <url-pattern>/query</url-pattern>
</servlet-mapping>
```

1.7.3 文件上传

在Web应用中处理表单提交外，还有一个种特殊的提交内容就是文件，比如：上传图片，音频，视频。

- 静态页面，一个类型为file的文本框，一个提交按钮，进行文件上传

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>上传的案例</title>
</head>
<body>
<h1>上传图片</h1>
<form method="post" action="/upload" enctype="multipart/form-data">
    <input type="file" name="filename">
    <input type="submit" value="上传文件">
</form>
</body>
</html>
```

- 实现上传文件Servlet，接收文件并且存储

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;

public class UploadFileServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        //获取上传的文件
        Part part = req.getPart("filename");
        InputStream is = part.getInputStream();
        //获取上传文件的路径
        String appUploadPath = req.getServletContext().getRealPath("/upload");
        File file = new File(appUploadPath, part.getSubmittedFileName());
        if (!file.getParentFile().exists()) {
            file.getParentFile().mkdirs();
        }
        OutputStream out = new FileOutputStream(file);
        byte[] buff = new byte[1024];
        int len = -1;
        while ((len = is.read(buff)) != -1) {
            out.write(buff, 0, len);
        }
        out.close();
```

```
            is.close();
            resp.setContentType("text/html; charset=UTF-8");
            PrintWriter writer = resp.getWriter();
            writer.append("<html>")
                    .append("<head>")
                    .append("<meta charset='UTF-8'>")
                    .append("<title>File</title>")
                    .append("</head>")
                    .append("<body>")
                    .append("<a href='")
                    .append("/upload/").append(part.getSubmittedFileName())
                    .append("'>")
                    .append("上传的文件")
                    .append("</a>")
                    .append("</body>")
                    .append("</html>");
        }
    }
```

- 配置Servlet

```
<servlet>
    <servlet-name>UploadFileServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.web.UploadFileServlet</servlet-class>
    <multipart-config>
        <!-- 表示在Part调用write方法时， 要将已上传的文件保存到磁盘中的位置。 此处绝对路径 -->
        <location>D:/upload</location>
        <!-- 上传文件的最大容量， 默认值为-1，表示没有限制。 大于指定值的文件将会遭到拒绝 单位：字节
-->
        <max-file-size>1048576</max-file-size>
        <!-- 表示多部分HTTP请求允许的最大容量， 默认值为-1， 表示没有限制 单位：字节-->
        <max-request-size>5242880</max-request-size>
        <!--上传文件超出这个容量界限时，会被写入磁盘 单位：字节-->
        <file-size-threshold>5242880</file-size-threshold>
    </multipart-config>
</servlet>
<servlet-mapping>
    <servlet-name>UploadFileServlet</servlet-name>
    <url-pattern>/upload</url-pattern>
</servlet-mapping>
```

2. 会话管理

由于HTTP的无状态性，使得会话管理或者会话跟踪成为Web应用开发一个无法避免的问题。默认情况一下，一个WEB服务器是无法区分一个HTTP请求是否为第一次访问。

比如：一个电商网站应用要求用户登录后才能添加商品到购物车，查看订单，交易支付， 因此当用户输入了相应的用户名和密码后， 应用不应该再次提示需要用户登录， 应用必须记住哪些用户已经登录。 换句话说， 应用必须能管理用户的会话。

本节内容将讲述不同的会话管理技术。

2.1 URL重写

URL重写是一种会话跟踪技术，它将一个或者多个token添加到URL的查询字符串汇总，每个token通常为`key=value`形式，如下：

```
url?key-1=value-1&key-2=value-2&key-3=value-3...&key-n=value-n
```

案例：省，市，区三级联动，URL重写查询

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class TokenServlet extends HttpServlet {

    private Map<String, List<String>> cityMap = new HashMap<>();

    private Map<String, List<String>> countryMap = new HashMap<>();

    @Override
    public void init() throws ServletException {

        List<String> shannxi = new ArrayList<>();
        shannxi.add("西安市");
        shannxi.add("宝鸡市");
        shannxi.add("铜川市");
        shannxi.add("咸阳市");
        cityMap.put("陕西省", shannxi);


        List<String> xian = new ArrayList<>();
        xian.add("临潼区");
        xian.add("灞桥区");
        xian.add("长安区");
        countryMap.put("西安市", xian);


    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        String pro = req.getParameter("pro");
```

```
        String city = req.getParameter("city");
        if (pro == null && city == null) {
            writer.write("没有数据");
        } else {
            if (pro == null) {
                writer.write("pro 参数不能为空");
            } else {
                if (city == null) {
                    List<String> cityList = cityMap.get(pro);
                    StringBuilder sb = new StringBuilder();
                    for (String c : cityList) {
                        sb.append("<a href='/token")
                                .append("?")
                                .append("pro=").append(pro)
                                .append("&")
                                .append("city=").append(c)
                                .append(" '>")
                                .append(pro)
                                .append(",")
                                .append(c)
                                .append("</a>")
                                .append("</br>");

                    }
                    writer.write(sb.toString());
                } else {
                    List<String> cityList = cityMap.get(pro);
                    if (cityList.contains(city)) {
                        StringBuilder sb = new StringBuilder();
                        for (String country : countryMap.get(city)) {
                            sb.append("<a href='/token")
                                    .append("?")
                                    .append("pro=").append(pro)
                                    .append("&")
                                    .append("city=").append(city)
                                    .append(" '>")
                                    .append(pro)
                                    .append(",")
                                    .append(city)
                                    .append(",")
                                    .append(country)
                                    .append("</a>")
                                    .append("</br>");

                        }
                        writer.write(sb.toString());
                    } else {
                        writer.write(city + " 不属于 " + pro);
                    }
                }
            }
        }
    }
```

```
}
```

```
<servlet>
    <servlet-name>TokenServlet</servlet-name>
    <servlet-class>com.bittech.web.feature.TokenServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>TokenServlet</servlet-name>
    <url-pattern>/token</url-pattern>
</servlet-mapping>
```

URL重写适合于token无须在太多URL间传递的情 况下， 然而它有如下限制：

- URL在某些浏览器上最大长度为2000字符；
- 若要传递值到下一个资源， 需要将值插入到链接 中， 换句话说， 静态页面很难传值；
- URL重写需要在服务端上完成， 所有的链接都必须带值， 因此当一个页面存在很多链接时， 处理过程会是一个不小的挑战；
- 特殊字符不容易处理， 例如空格、 与和问号等必须用base64编 码；
- 所有的信息都是可见的， 安全性有要求的情况下不合适。

因为存在如上限制， URL重写仅适合于信息仅在少 量页面间传递， 且信息本身不敏感。

2.2 隐藏域

使用隐藏域来保持状态类似于URL重写技术，但不是将值附加到URL上，而是反到HTML表单的隐藏域中。当表单提交时，隐藏域的值也同时提交到服务器端。隐藏域技术仅当网页有表单时有效。该技术相对于URL重写的优势在于：没有字符数限制，同时无需额外的编码。但是该技术同URL重写一样，不适合跨越多个界面。

```
<!-- 表单隐藏域使用hidden属性 -->
<input type="text" hidden="hidden">
```

案例：用户信息列表，用户信息更新

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;

public class HiddenServlet extends HttpServlet {

    protected static Map<String, Person> personMap = new HashMap<>();

    @Override
    public void init() throws ServletException {
        super.init();

        Person java = new Person("1001", "Java", 20);
```

```
        Person php = new Person("1002", "PHP", 18);
        Person c = new Person("1003", "C", 30);

        personMap.put("1001", java);
        personMap.put("1002", php);
        personMap.put("1003", c);

    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        writer.append("<!DOCTYPE html>\n" +
                "<html lang=\"en\">\n" +
                "<head>\n" +
                "    <meta charset=\"UTF-8\">\n" +
                "    <title>用户信息</title>\n" +
                "</head>\n" +
                "<body>\n" +
                "<h1>用户信息列表</h1>\n" +
                "<table>\n" +
                "    <thead>\n" +
                "    <tr>\n" +
                "        <td>编号</td>\n" +
                "        <td>姓名</td>\n" +
                "        <td>年龄</td>\n" +
                "    </tr>\n" +
                "    </thead>\n" +
                "    <tbody>");

        for (Map.Entry<String, Person> entry : personMap.entrySet()) {
            Person person = entry.getValue();
            writer.append("<tr>")
                    .append("<td>")
                    .append("<a href='/person?id=").append(person.id).append("'>")
                    .append(String.valueOf(person.id))
                    .append("</a>")
                    .append("</td>")
                    .append("<td>").append(person.name).append("</td>")
                    .append("<td>").append(String.valueOf(person.age)).append("</td>")
                    .append("</tr>");
        }

        writer.append("    </tbody>\n" +
                "</table>\n" +
                "</body>\n" +
                "</html>");
    }

    public static class Person {
        private String id;
```

```
        private String name;
        private int age;
        public Person(String id, String name, int age) {
            this.id = id;
            this.name = name;
            this.age = age;
        }
        public String getId() {
            return id;
        }
        public void setId(String id) {
            this.id = id;
        }
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public int getAge() {
            return age;
        }
        public void setAge(int age) {
            this.age = age;
        }
    }
}
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class PersonServlet extends HiddenServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        String id = req.getParameter("id");
        if (id == null) {
            writer.append("参数不能为空");
        } else {
            Person person = personMap.get(id);
            writer.append("<!DOCTYPE html>\n" +
                    "<html lang=\"en\">\n" +
                    "<head>\n" +
                    "    <meta charset=\"UTF-8\">\n" +
                    "    <title>修改人员信息</title>\n" +
                    "</head>\n" +
                    "<body>\n" +
```

```
                    "<h1>修改人员信息</h1>\n" +
                    "<form method=\"post\" action=\"/person\">\n" +
                    "    <input type=\"text\" name=\"id\" hidden=\"hidden\" value=\"" +
person.getId() + "\"/>\n" +
                    "    姓名: <input type=\"text\" name=\"name\" value=\"" +
person.getName() + "\" placeholder=\"请输入姓名\">\n" +
                    "    年龄:<input type=\"text\" name=\"age\" value=\"" +
person.getAge() + "\" placeholder=\"请输入年龄\">\n" +
                    "    <input type=\"submit\" value=\"保存更新\">\n" +
                    "</form>\n" +
                    "</body>\n" +
                    "</html>");
        }

    }


    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/html; charset=UTF-8");
        PrintWriter writer = resp.getWriter();
        String id = req.getParameter("id");
        if (id == null) {
            //新建用户
            writer.append("暂时不支持");
        } else {
            //修改用户
            String name = req.getParameter("name");
            String ageStr = req.getParameter("age");
            Person person = personMap.get(id);
            person.setAge(Integer.parseInt(ageStr));
            person.setName(name);
            writer.append("<a href='/hidden'>").append("回到列表").append("</a>");
        }
    }
}
```

```
<!-- 第一个Servlet，完成数据列表展示 -->
<servlet>
    <servlet-name>HiddenServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.sevlet.HiddenServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HiddenServlet</servlet-name>
    <url-pattern>/hidden</url-pattern>
</servlet-mapping>

<!-- 第一个Servlet，完成单条数据展示和修改 -->
<servlet>
    <servlet-name>PersonServlet</servlet-name>
    <servlet-class>com.bittech.javaweb.sevlet.PersonServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>PersonServlet</servlet-name>
    <url-pattern>/person</url-pattern>
</servlet-mapping>
```

2.3. Cookies

URL重写和隐藏域仅适合保存无需跨越太多页面的信息。如果需要在多个页面之间传递信息，则以上两种技术实现成本高昂，因此不得不在每个页面都进行相应的处理。Cookies技术可以实现着一点。

Cookies是一个很少的信息片段，可自动地在浏览器和Web服务器间交互，因此Cookies可存储在多个页面间传递信息。Cookie作为HTTP header的一部分，其传输由HTTP协议控制，此外，可以控制Cookies的有效时间。浏览器通常支持每个网站高达20个Cookies。

> 问题：用户可以通过改变浏览器设置来拒绝接受Cookies或者人为的清除Cookies。

下面是Cookies使用的代码片段：

- 生成Cookies

```
//生成Cookies
//创建Cookie
Cookie cookie = new Cookie("city", "xian");
//HTTP响应时会将Cookie信息返回给浏览器
resp.addCookie(cookie);
```

- 获取Cookies

```
//获取Cookies
Cookie[] cookies = req.getCookies();
Cookie cityCookie = null;
if (cookies != null) {
    for (Cookie c : cookies) {
        if (c.getName().equals("city")) {
            cityCookie = c;
        }
    }
}
String cityName = cityCookie != null ? cityCookie.getName() : "";
```

- 删除Cookies

```
//删除Cookie
Cookie rmCookie = new Cookie("city", "");
//存活时间0秒，即设置cookie立即过期，也就是删除
rmCookie.setMaxAge(0);
resp.addCookie(rmCookie);
```

> 备注：根据不同的城市，记录Cookie，展示特色信息示例

2.4 HttpSession

在所有的会话跟踪技术中，HttpSession对象是最强大和最通用的。一个用户可以有且最多有一个HttpSession，并且不会被其它用户访问到。

HttpSession对象是在用户第一次访问网站的时候自动创建。

- 创建HttpSession的代码示例如下：

```
//返回当前的HttpSession，如果没有，创建一个
HttpSession httpSession = req.getSession();
```

```
//返回当前的HttpSession，如果没有，返回null
HttpSession httpSession = req.getSession(false);
```

```
//返回当前的HttpSession，如果没有，创建一个
HttpSession httpSession = req.getSession(true);
```

- HttpSession赋值代码示例如下：

```
//setAttribute方法中第一个参数是String类型作为key，第二个参数是Object类型作为value
httpSession.setAttribute("city","xian");
httpSession.setAttribute("location","LinTong");
```

HttpSession存储数据的要求：

- 不同于URL重新、 隐藏域或cookie， 放入 到HttpSession 的值， 是存储在内存中的， 因此，不要往HttpSession放入太多对象或大对象。 尽管现代的 Servlet容器在内存不够用的时候会将保存在 HttpSessions 的对象转储到二级存储上， 但这样有性能问题， 因此小心存储。
- 放到HttpSession的值不限于String类型， 可以是任意实现 `java.io.Serializable` 的java对象，因为Servlet容器认为必要时会将这些对象放入文件或数据库中，尤其在内存不够用的时候，当然也可以将不支持序列化的对象放入HttpSession，但是这样的话，当Servlet容器视图序列化的时候会失败并报错。

HttpSession内部实现会话跟踪的原理：

所有保存在HttpSession的数据不会被发送到客户端，不同于其它会话管理技术， Servlet容器为每个 HttpSession 生成唯一的标识， 并将该标识发送给浏览 器， 或创建一个名为**JSESSIONID**的cookie， 或者在 URL后附加一个名为jsessionid 的参数。 在后续的请求中，浏览器会将标识提交给服务端，这样服务器就可以识别该请求是由哪个用户发起的。 Servlet容器会自动选择一种方式传递会话标识，无须开发人员介入 。

获取JESSIONID的代码如下：

```
String jsessionId= httpSession.getId();
```

HttpSession的有效性：

HttpSession是保存在内存中的，那么如果用户会话结束了，该怎么办呢？

比如：当用户退出登录的时候，就可以在代码中人为的设置HttpSession过期，`httpSession.invalidate();`

比如：有时候用户并非正常途径退出，像关闭浏览器，服务端很难知道这时候会话应该结束，这种情况下就需要设置一个过期时间，时间一到，容器帮助处理，清除HttpSession。

设置过期时间的方式：

- 调用HttpSession的方法

```
//HttpSession在用户30分钟之内无活动，设置过期，单位秒
//设置为0，表示永不过期
httpSession.setMaxInactiveInterval(60 * 30);
```

不建议设置为永不过期，因为HttpSession是占用堆内存的，如果用户特别多的情况下，堆内存将消耗特别大，而且永不释放，直到应用重新加载或者Servlet关闭。

- 在web.xml部署描述文件中配置

```
<session-config>
    <session-timeout>20</session-timeout>
</session-config>
```

- 不配置，使用容器默认的配置（默认30分钟），参见：`${TOMCAT_HOME}/conf/web.xml`

3. 过滤器Filter

Filter是拦截Request请求的对象，在用户的请求访问资源前处理ServletRequest以及ServletResponse,它可以用于日志记录，加解密，Session检查，图像文件保护。通过Filter可以拦截处理某个资源或者某些资源。

Filter的配置可以通过注解或者部署描述来完成。当一个资源或者某些资源需要被多个Filter所使用到，且它的触发顺序很重要时，只能通过部署描述来配置。

3.1 Filter的核心类

Filter的实现类必须继承`javax.servlet.Filter`接口。该接口包含了Filter的3个生命周期：`init,doFilter,destroy`。

- Filter的初始化方法

Servlet容器初始化Filter时，会触发Filter的init方法，一般来说是在应用开始时。init方法不是在该Filter相关资源使用到时才初始化，而且该方法只调用一次，用于初始化Filter。 方法的定义如下：

```
public void init(FilterConfig filterConfig) throws ServletException;
```

- Filter的过滤方法

Servlet容器每次处理Filter相关的资源时，都会调用该Filter实例的doFilter方法。方法定义如下：

```
public void doFilter(ServletRequest request, ServletResponse response,
                     FilterChain chain) throws IOException, ServletException;
```

在doFilter中访问ServletRequest，ServletResponse， 这意味着允许修改这两个对象的内容。在doFilter实现中，最后需要调用FilterChain中的doChain方法。

一个资源可能需要被多个Filter关联，也成为Filter链。这时候FilterChain#doFilter()将会触发Filter链中下一个Filter。只有在Filter链条中最后一个Filter里调用的FilterChain#doFilter()，才会触发处理资源的方法。

如果在Filter#doFilter()的实现中，没有在结尾处调用FilterChain#doFilter()的方法，那么该Request请求终止，后面的处理就会中断。

- Filter的销毁方法

Servlet容器销毁Filter时触发，一般在应用停止的时候进行调用。

```
public void destroy();
```

3.2 Filter的配置

当完成Filter的实现后，就可以开始配置Filter 。Filter的配置需要如下步骤：

- 确认那些资源需要使用该Filter拦截处理
- 配置Filter的初始化参数值，这些参数可以再Filter的init方法中读取
- 配置Filter的名称。通常该名字没有特别含义，在一些需要识别Filter的时候名字就非常有用

3.3 Filter的案例

3.3.1 日志过滤器

- 需求描述

实现一个简单的Filter将应用中的Request请求的URL记录到日志文件。日志文件的文件名通过Filter的初始化参数来配置，此外日志的每条记录都会有一个前缀，该前缀由初始化参数定义。

- 功能价值

通过日志文件，可以分析出有用的价值信息，比如：应用中那些资源访问最频繁，Web站点在一天中的那个时间段访问量最多。

- 实现LoggingFilter，通常Filter的实现类名称以*Filter结尾

```
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
public class LoggingFilter implements Filter {

    private PrintWriter logger;

    private String prefix;

    @Override
    public void init(FilterConfig filterConfig)
            throws ServletException {
```

```
        //获取prefix参数
        prefix = filterConfig.getInitParameter("prefix");
        //获取logFileName参数
        String logFileName = filterConfig.getInitParameter("logFileName");
        //获取应用的路径，该值取决于应用运行时的位置
        String appPath = filterConfig.getServletContext().getRealPath("/");
        try {
            File file = new File(appPath, logFileName);
            logger = new PrintWriter(file);
        } catch (FileNotFoundException e) {
            throw new ServletException(e.getMessage());
        }
    }

    @Override
    public void destroy() {
        if (logger != null) {
            logger.close();
        }
    }

    @Override
    public void doFilter(ServletRequest request,
                         ServletResponse response, FilterChain filterChain
    ) throws IOException, ServletException {
        //处理Http请求，这里进行类型转换
        HttpServletRequest httpServletRequest = (HttpServletRequest) request;
        //获取请求的URI
        String uri = httpServletRequest.getRequestURI();
        logger.println(new Date() + " " + prefix + uri);
        logger.flush();
        //继续下一个Filter
        filterChain.doFilter(request, response);
    }
}
```

- 配置Filter

```
<filter>
    <!--Filter名称-->
    <filter-name>LoggingFilter</filter-name>
    <!--Filter实现类-->
    <filter-class>com.bittech.javaweb.filter.LoggingFilter</filter-class>
    <!--logFileName参数配置-->
    <init-param>
        <param-name>logFileName</param-name>
        <param-value>filter.logging</param-value>
    </init-param>
    <!-- prefix参数配置 -->
    <init-param>
        <param-name>prefix</param-name>
        <param-value>Request:</param-value>
    </init-param>
```

```
</filter>
<filter-mapping>
    <filter-name>LoggingFilter</filter-name>
    <!--根据URL进行拦截-->
    <url-pattern>/*</url-pattern>
    <!--根据Servlet名称拦截 -->
    <!--<servlet-name>IndexServlet</servlet-name>-->
</filter-mapping>
```

3.3.2 请求计数器

上面案例中虽然可以通过分析日志文件，也是可以获取每个请求在一定时间范围内的访问量。当然我们也可以实现一个过滤器，单独进行请求访问次数的统计。

- 需求描述

统一拦截请求，将请求的URL路径作为属性名，请求次数作为属性值，并且以key-value的形式保存到属性文件。

- 问题难点

由于统计结果放到属性文件中，而Filter可以被多线程访问，因此涉及到线程安全问题。

- 实现Filter

```
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Properties;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class CounterFilter implements Filter {

    //单线程，用来处理计数，解决多线程访问修改文件带来的错误
    private ExecutorService executorService = Executors.newSingleThreadExecutor();
    private Properties requestCounterLog;
    private File logFile;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException {
        String appPath = filterConfig.getServletContext()
                .getRealPath("/");
        logFile = new File(appPath, "requestCounterLog.txt");
        if (!logFile.exists()) {
            try {
                logFile.createNewFile();
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        requestCounterLog = new Properties();
        try {
            requestCounterLog.load(new FileReader(logFile));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void destroy() {
        executorService.shutdown();
    }

    @Override
    public void doFilter(ServletRequest request,
                         ServletResponse response, FilterChain filterChain)
            throws IOException, ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest) request;
        final String uri = httpServletRequest.getRequestURI();
        executorService.execute(new Runnable() {
            @Override
            public void run() {
                String value = requestCounterLog.getProperty(uri);
                if (value == null) {
                    requestCounterLog.setProperty(uri, "1");
                } else {
                    int count = 0;
                    try {
                        count = Integer.parseInt(value);
                    } catch (NumberFormatException e) {
                    }
                    count++;
                    requestCounterLog.setProperty(uri,
                            Integer.toString(count));
                }
                try {
                    requestCounterLog.store(new FileWriter(logFile), "Request
Counter");
                } catch (IOException e) {
                }
            }
        });
        filterChain.doFilter(request, response);
    }
}
```

- 配置Filter

```
<filter>
    <filter-name>CounterFilter</filter-name>
    <filter-class>com.bittech.javaweb.filter.CounterFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>CounterFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.4 Filter顺序

如果多个Filter应用与同一个资源，Filter的触发顺序将变的非常重要，这时就需要使用部署描述来管理Filter指定Filter的触发先后顺序。

比如：Filter1需要在Filter2前被触发，那么在部署描述中，Filter1需要配置在Filter2之前。

```
<filter>
    <filter-name>Filter1</filter-name>
    <filter-class>
    the fully-qualified name of the filter class
    </filter-class>
</filter>
<filter>
    <filter-name>Filter2</filter-name>
    <filter-class>
    the fully-qualified name of the filter class
    </filter-class>
</filter>
```

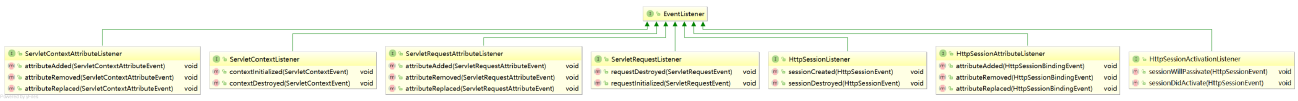
> 注意：通过部署描述之外的配置来指定Filter触发的顺序是不可能的。

4. 监听器Listeners

Servlet API提供了一些列的事件和事件监听接口。上层的Servlet/JSP应用能够通过调用这些API进行事件驱动开发。这些监听的所有事件都继承自`java.util.Event`对象。监听器接口可以分为三类：`ServletContext,HttpSession,ServletRequest`。

4.1 监听器接口

监听器接口主要在`javax.servlet`和`javax.servlet.http`的包中。



监听器接口分为三类：

- ServletContext的各类监听器
- HttpSession的各类监听器
- ServletRequest的各类监听器

编写监听器的方法：实现监听接口，并且在部署文件描述文件中指定监听接口的实现类。

```
public class ListenerClass implements ListenerInterface {
}
```

```
</listener>
     <listener-class>fully-qualified listener class</listener-class>
</listener>
```

4.2 应用级监听

- ServletContextListener能对ServletContext的创建和销毁做出响应。
- ServletContextAttributeListener会响应ServletContext范围的属性被添加，删除或者替换。

案例：通过监听ServletContext的创建和销毁以及属性监听

```
public class AppContextListener implements ServletContextListener ,
ServletContextAttributeListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("ServletContext Initialized "+sce.toString());
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("ServletContext Destroyed "+sce.toString());
    }

    public void attributeAdded(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Add ServletContext Attribute :" +
servletContextAttributeEvent.getName() + " = " + servletContextAttributeEvent
                .getValue());
    }

    public void attributeRemoved(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Removed ServletContext Attribute :" +
servletContextAttributeEvent.getName() + " = " +
                servletContextAttributeEvent
                        .getValue());
    }

    public void attributeReplaced(ServletContextAttributeEvent
servletContextAttributeEvent) {
        System.out.println("Replaced ServletContext Attribute :" +
servletContextAttributeEvent.getName() + " = " +
                servletContextAttributeEvent
                        .getValue());
    }
}
```

```
<listener>
      <listener-class>com.bittech.javaweb.listener.AppContextListener</listener-class>
</listener>
```

使用场景：

- 应用全局共享数据加载
- 监听容器初始化和销毁时机，进行准备工作和清理工作
- 监听ServletContext的属性操作

4.3 会话级监听

- HttpSessionListenern能够监听HttpSession的创建和销毁。
- HttpSessionAttributeListener和ServletContextAttributeListener类似，它会响应HttpSession范围的属性的添加，删除，替换。

案例：通过监听HttpSession的创建和销毁来统计HttpSession的数量

```
public class HttpSessionCountListener implements HttpSessionListener,
ServletContextListener, HttpSessionAttributeListener {

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        ServletContext context = se.getSession().getServletContext();
        int sessionCount = Integer.parseInt(
                (String) context.getAttribute("sessionCount")
        );
        sessionCount++;
        context.setAttribute("sessionCount", String.valueOf(sessionCount));
        System.out.println("Current Http Session :" + sessionCount);
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        ServletContext context = se.getSession().getServletContext();
        int sessionCount = Integer.parseInt(
                (String) context.getAttribute("sessionCount")
        );
        sessionCount--;
        context.setAttribute("sessionCount", String.valueOf(sessionCount));
        System.out.println("Current Http Session :" + sessionCount);
    }

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        sce.getServletContext().setAttribute("sessionCount", "0");
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        sce.getServletContext().removeAttribute("sessionCount");
    }
```

```
    @Override
    public void attributeAdded(HttpSessionBindingEvent event) {
        System.out.println("HttpSession attributeAdded " + event.getName() + "=" +
event.getValue());
    }

    @Override
    public void attributeRemoved(HttpSessionBindingEvent event) {
        System.out.println("HttpSession attributeRemoved " + event.getName() + "=" +
event.getValue());
    }

    @Override
    public void attributeReplaced(HttpSessionBindingEvent event) {
        System.out.println("HttpSession attributeReplaced " + event.getName() + "=" +
event.getValue());
    }
}
```

```
<listener>
    <listener-class>com.bittech.javaweb.listener.HttpSessionCountListener</listener-
class>
</listener>
```

当我们在Servlet实现中创建了HttpSession那么这里的统计就会生效，HttpSession销毁则需要人工的销毁或者等待到过期。

4.4 请求级监听

- ServletRequestListener监听器会对ServletRequest的创建和销毁事件进行响应。容器会通过一个池子来存放并复用多个ServletRequest，ServletRequest的创建时从容器池里被分配出来的时刻开始，而它的销毁时刻是放回容器池里的时间。
- ServletRequestAttributeListener会响应ServletRequest范围的属性被添加，删除，或者替换。

案例：计算每个Http请求完成的时间

```
public class PerfStatListener implements ServletRequestListener {
    @Override
    public void requestInitialized(ServletRequestEvent sre) {
        long startTime = System.nanoTime();
        ServletRequest request = sre.getServletRequest();
        request.setAttribute("start_time", startTime);
    }
    @Override
    public void requestDestroyed(ServletRequestEvent sre) {
        ServletRequest request = sre.getServletRequest();
        long startTime = (long) request.getAttribute("start_time");
        long endTime = System.nanoTime();
        String url = ((HttpServletRequest) request).getRequestURI();
```

```
        System.out.println("Request " + url + " Cast time " + (endTime - startTime) /
1000 + " microseconds ");
    }
}
```

```
<listener>
    <listener-class>com.bittech.javaweb.listener.PerfStatListener</listener-class>
</listener>
```

```
//结果示例:
Request /query Cast time 4387 microseconds
Request /form Cast time 485 microseconds
Request /index Cast time 1206 microseconds
```

> 备注：Filter中为Request添加属性，此时就可以监听到

5. 注解描述

案例：通过注解的方式来描述Servlet，Filter，Listener对象

主要注解：

- WebServlet：标识Servlet类
- WebFilter：标识Filter类
- WebListener：标识Listener类
- WebInitParam：标识初始化参数
- MulitpartConfig：标识上传附件的配置

总结

知识块	知识点	分类	掌握程度
Servlet	1. Servlet核心接口类 2. 功能特性 3.会话跟踪	实战型	掌握
Filter	1. Filter核心接口类 2. 过滤器使用案例	实战型	掌握
Listener	1. Listener核心接口类 2. 监听器使用案例	实战型	掌握
Web注解	1. 3.0API提供的注解配置Servlet，Filter，Listener	实战型	掌握