@Description：Report of Servlet Basic, Part 3

@Date：2018.6.8

@Auther：Sujin Guo

@Reference: [1] 《Servlet & Jsp &Spring MVC 初学指南》

[2] http://www.runoob.com/servlet/servlet-cookies-handling.html

Cookie & Session

Http 是无状态的协议，要保存客户端的状态（如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，应用需要记录用户登录的状态），就需要用到 Session 和 Cookie 技术。

1、Cookie

原理：

Cookie 是存储在客户端计算机上的文本文件，并保留了各种跟踪信息。Cookie 通常设置在 HTTP 头信息中，作为响应发送给客户端，Cookie 中信息包括键值对、路径和作用域、过期时间。当客户端再次发送 HTTP 请求时，如果用户的浏览器指向任何匹配该 Cookie 的路径和域的页面，它会重新发送 Cookie 到服务器。

基本操作：

1. 通过 Servlet 在响应中设置 cookie。

```
//设置cookie
Cookie cookie = new Cookie(name, value);
httpServletResponse.addCookie(cookie);
```

2. 通过 Servlet 读取请求中的 cookie。
3. 删除 cookie：没有直接的方法来删除cookie，只能创建一个同名的cookie，并将maxAge属性设置为0。
4. cookie 的属性：其中 domain 是 cookie 的有效域，maxAge 是 cookie 有效期，isHttpOnly=true 禁止 javascript 操作cookie。

```
//Cookie的属性
private String name;     // NAME= ... "$Name" style is reserved
private String value;    // value of NAME

private String comment; // ;Comment=VALUE ... describes cookie's use
// ;Discard ... implied by maxAge < 0
private String domain;  // ;Domain=VALUE ... domain that sees cookie
private int maxAge = -1;     // ;Max-Age=VALUE ... cookies auto-expire
private String path;     // ;Path=VALUE ... URLs that see the cookie
private boolean secure; // ;Secure ... e.g. use SSL
private int version = 0;     // ;Version=1 ... means RFC 2109++ style
private boolean isHttpOnly = false;
```

作用：

Cookie 可以实现多个页面传递信息。案例：通过 cookie 设置用户偏好，如字体、每页显示多少条数据等。

2、HttpSession

session 跟踪

Web 服务器可为每个客户端分配 session，在服务端保存访问的状态信息。跟踪 session 的方式有下列几种：

1. cookies 跟踪：Web 服务器为每个会话分配一个唯一的 session id，并将 session id 存在 cookie 中发送给客户端，对于客户端的后续请求可以使用接收到的 cookie 来识别。
2. 隐藏表单字段：一个 Web 服务器可以发送一个隐藏的 HTML 表单字段，以及一个唯一的 session 会话 ID。
3. URL重写：在每个 URL 末尾追加数据来标识 session 会话，服务器会把该 session 会话标识符与已存储的有关 session 会话的数据相关联。

HttpSession 基本操作

session 保存参数

```
public void setAttribute(String name,Object value)
public Enumeration<java.lang.String> getAttributeNames()
public Object getAttribute(String name);
public void removeAttribute(String name);
//setValue、getValueNames、getValue、removeValue 对于Version2.2被标记为过时
```

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。

放入 HttpSession 中的 Object 最好实现 java.io.Serializable，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，如果将没有实现序列化的对象放入 HttpSession，会在这些时候报序列化错误。

session 唯一标志：JSESSIONID

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。

```
public String getId();
```

session 过期

```
public void setMaxInactiveInterval(int interval); //设置超时时间
public int getMaxInactiveInterval(); //查看超时时间,单位为秒
public void invalidate(); //强制会话过期，并清空其保存的对象。
```

默认 session 过期时间可通过 web.xml 配置，如果没有配置，则取决于 Servlet 容器设置的默认值。

大部分情况下，你应该主动销毁无用的 HttpSession，以便释放相应的内存。

若设置为0，则该HttpSession 永不过期。通常这不是一个好的选择。

案例

购物车的浏览、添加和查看。

3、Cookie 和 Session 区别与联系

- Session是在服务端保存的一个数据结构，用来跟踪用户的状态。在服务端保存Session的方法很多，内存、数据库、文件都有。
- Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息。Cookie 可以保存 JSESSIONID 用来识别特定的用户。也可以为了方便用户保存个性化设置信息，购物车信息，保存上次输入的用户名密码等。

Listener

1、创建和注册 Listener

Servlet API提供了一系列的事件和事件监听接口。当触发一定的事件时，会执行已注册的监听器中对应的方法。监听器接口可以分为三类：ServletContext、HttpSession 和ServletRequest 。

监听器的注册方法：编写一个监听器，只需要写一个Java类来实现对应的监听器接口就可以了。注册的方法有两种，第一种是使用 WebListener 注解，另一种是在 web.xml 文件中添加 listener 元素。代码如下：

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

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

下面介绍各种监听器的作用，哪些事件会触发监听器的行为。

2、ServletContext Listeners

（1）javax.servlet.ServletContextListener

它能够响应 ServletContext 生命周期事件，它提供了 ServletContext 创建之后和 ServletContext 关闭之前的会被调用的方法。ServletContext 创建和销毁时触发下列函数，这两个方法都会从容器获取到一个 ServletContextEvent，通过此事件可以获取 ServletContext。

```
void contextInitialized(ServletContextEvent event)
void contextDestroyed(ServletContextEvent event)
```

（2）javax.servlet.ServletContextAttributeListener

它能够响应 ServletContext 范围的属性添加、删除、替换事件。改变属性的行为触发下列函数，这三个方法都能获取到一个 ServletContextAttributeEvent 的对象，通过这个对象可以获取属性的名称和值。

```
void attributeAdded(ServletContextAttributeEvent event)
void attributeRemoved(ServletContextAttributeEvent event)
void attributeReplaced(ServletContextAttributeEvent event)
```

3、Session Listeners

（1）javax.servlet.http.HttpSessionListener

它能够响应 HttpSession 的创建、超时和失效事件。当一个 HttpSession 创建或者销毁时，触发下列两个方法，这两个方法都可以接收到一个继承于 java.util.Event 的 HttpSessionEvent 对象。可以通过调用HttpSessionEvent 对象的 getSession 方法来获取当前的 HttpSession。

```
void sessionCreated(HttpSessionEvent event)
void sessionDestroyed(HttpSessionEvent event)
```

（2）javax.servlet.http.HttpSessionAttributeListener

它能响应HttpSession范围的属性添加、删除、替换事件。

```
void attributeAdded(HttpSessionBindingEvent event)
void attributeRemoved( HttpSessionBindingEvent event)
void attributeReplaced( HttpSessionBindingEvent event)
```

这三个方法都能获取到一个HttpSessionBindingEvent 的对象，通过这个对象可以获取属性的名称和值：

```
java.lang.String getName()
java.lang.Object getValue()
```

（3）javax.servlet.http.HttpSessionActivationListener

它在一个 HttpSession 激活或者失效时被调用。

在分布式环境下，会用多个容器来进行负载均衡，有可能需要将session保存起来，在容器之间传递。例如 当一个容器内存不足时，会把很少用到的对象转存到其他容器上。这时候，容器就会通知此监听器。

当HttpSession被转移到其他容器之后，sessionDidActivate方法会被调用。当一个HttpSession将要失效时，容器会调用 sessionWillPassivate 方法。两个方法中，容器将一个 HttpSessionEvent 方法传递到方法里，可以从这个对象获得HttpSession。

```
void sessionDidActivate(HttpSessionEvent event)
void sessionWillPassivate(HttpSessionEvent event)
```

（4）javax.servlet.http.HttpSessionBindingListener

可以实现这个接口来保存 HttpSession 范围的属性。当有属性从 HttpSession 添加或删除时，监听器能够做出响应。

```
void valueBound(HttpSessionBindingEvent event)
void valueUnbound(HttpSessionBindingEvent event)
```

4、ServletRequest Listeners

（1）javax.servlet.ServletRequestListener

它能够响应一个ServletRequest的创建或删除。

ServletRequestListener监听器会对ServletRequest的创建和销毁事件进行响应。容器会通过一个池来存放并重复利用多个ServletRequest，ServletRequest的创建是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。

当一个ServletRequest创建（从容器池里取出）时，requestInitialized方法会被调用，当ServletRequest销毁（被容器回收）时，requestDestroyed方法会被调用。这两个方法都会接收到一个ServletRequestEvent对象，可以通过使用这个对象的getServletRequest方法来获取ServletRequest对象。

```
void requestInitialized(ServletRequestEvent event)
void requestDestroyed(ServletRequestEvent event)
```

（2）javax.servlet.ServletRequestAttributeListener

它能响应ServletRequest范围的属性值添加、删除、修改事件。

当一个ServletRequest范围的属性被添加、删除或替换时，ServletRequestAttributeListener接口会被调用。ServletRequestAttributeListener接口提供了三个方法：attributeAdded、attribute Replaced和attributeRemoved。这些方法都可以获得一个继承自ServletRequestEvent的ServletRequestAttributeEvent对象。通过ServletRequestAttributeEvent类提供的getName和getValue方法可以访问到属性的名称和值：

```
void attributeAdded(ServletRequestAttributeEvent event)
void attributeRemoved(ServletRequestAttributeEvent event)
void attributeReplaced(ServletRequestAttributeEvent event)
```

5、Listener 代码示例

- 计算每个HTTP请求的完成时间（监听 Request）
- 统计当前存活的 session 个数（监听 Session）

Filters

- 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，它可用于日志记录、加解密、Session 检查、图像文件保护等。

- 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 Annotation 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。

1、Filter 的生命周期

- Filter 也是单实例多线程的，Filter 实例在容器启动时初始化。在编写代码时要注意线程安全问题。
- Filter的实现必须继承javax.servlet.Filter接口。这个接口包含了Filter的3个生命周期：init、doFilter、destroy。
- 在Filter的doFilter的实现中，最后一行需要调用FilterChain中的doChain方法。否则FilterChain后面的处理会终止。

```
void init(FilterConfig filterConfig)
void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
    filterChain.doFilter(request, response)
void destroy()
```

2、Filter 的配置

Filter 的配置可以通过注解和 xml 两种方式配置。

1. 哪些资源需要使用拦截器处理：urlpattern
2. 配置Filter的初始化参数值，这些参数可以在Filter的init方法中读取到；
3. Filter 的其他属性。

WebFilter 的属性：

属性	描述
asyncSupported	Filter是否支持异步操作
description	Filter的描述
dispatcerTypes	Filter所生效范围
displayName	Filter的显示名
filterName	Filter的名称
initParams	Filter的初始化参数
largeIcon	Filter的大图名称
servletName	Filter所生效的Servlet名称
smallIcon	Filter的小图名称
urlPatterns	Filter所生效的URL路径
value	Filter所生效的URL路径

3、Filter 执行顺序

如果多个Filter应用于同一个资源，Filter的触发顺序将变得非常重要，这时就需要使用部署描述来管理 Filter：指定哪个Filter先被触发。

例如：Filter 1需要在Filter 2前被触发，那么在部署描述中，Filter 1需要配置在Filter 2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>
```

4、Filter 代码示例

- 日志 Filter
- 图像文件保护 Filter
- 下载计数 Filter
- 结合 session、cookie、filter 实现用户登录与登出