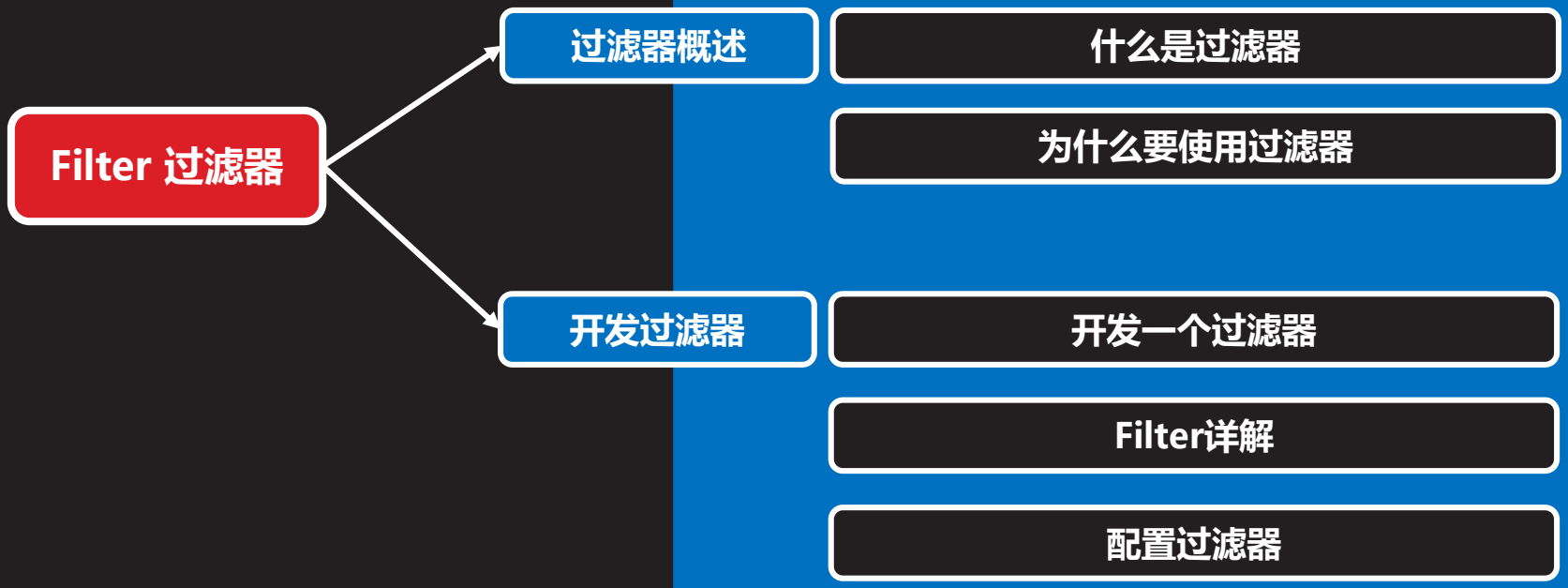
过滤器、监听器

WebBasic 过滤器、监听器 Unit01

内容

上午	09:00 ~ 09:30	作业讲解与回顾
	09:30 ~ 10:20	Filter过滤器
	10:30 ~ 11:20	
	11:30 ~ 12:00	
下午	14:00 ~ 14:50	Listener 监听器
	15:00 ~ 15:50	
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑

Filter 过滤器

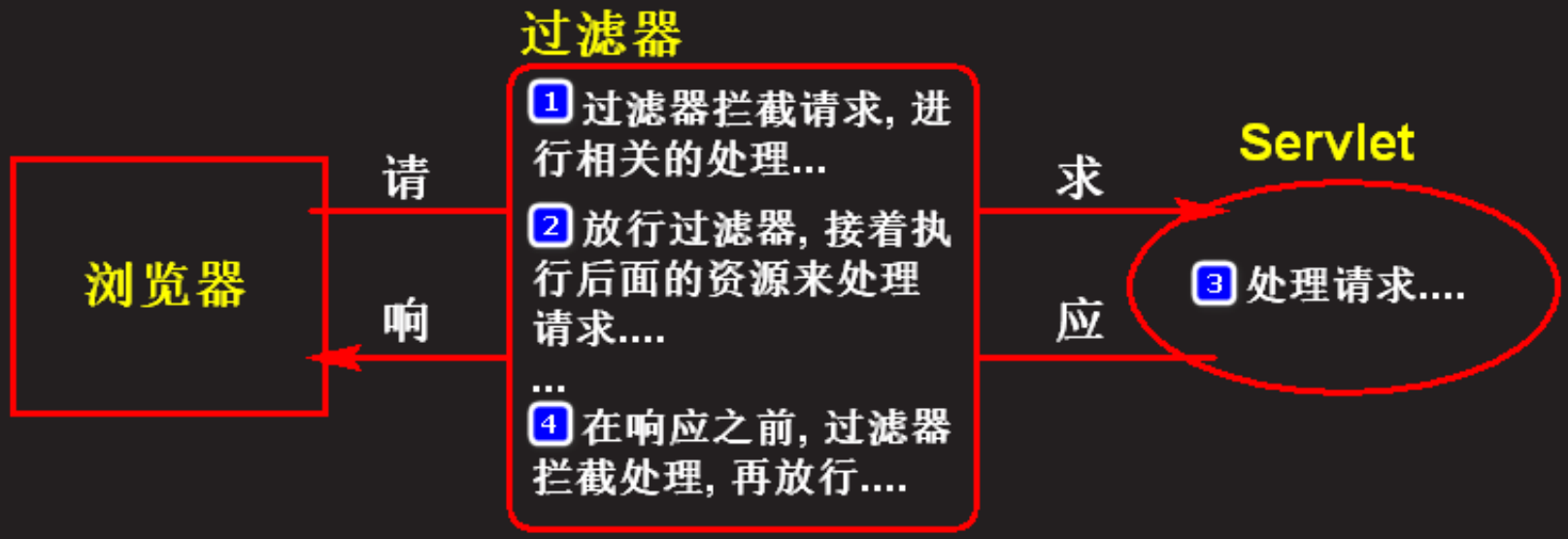

Filter 过滤器
过滤器概述
什么是过滤器
为什么要使用过滤器
开发过滤器
开发一个过滤器
Filter详解
配置过滤器

过滤器概述

什么是过滤器

- Servlet 技术规范中，定义了 Servlet、**Filter**、Listener 三门技术，统称为 JavaWeb 的三大组件。
  - 其中 Filter 也叫做过滤器，它与 Servlet 很相似，不同的是 过滤器不是用来处理请求的，而是用来拦截请求的。
  - 通过过滤器可以实现用户在访问某个资源之前或之后，对访问的请求和响应进行拦截，从而做一些相关的处理。

知识讲解



什么是过滤器（续1）

- 当用户请求某个Servlet之前，会先执行能够拦截这个请求的 Filter，如果 Filter "放行"，那么会继续执行用户请求的 Servlet，否则，将不会执行用户请求的 Servlet！

知识讲解

- 总结：
  - 所谓的过滤器就是拦截用户对资源的访问。
  - 一个过滤器可以拦截多个资源，一个资源也可以配置多个过滤器进行拦截。
  - 其实所谓的拦截，就是将 request 和 response 对象拦截下来，拦截下来后，可以做如下类似的操作：
  - （1）判断用户是否登陆，对某个资源的访问是否具有权限。
  - （2）在请求之前或响应之前做一些处理，如全站乱码解决。

为什么要使用过滤器

- 为什么要使用过滤器?
  - Servlet 可以用来处理请求，有时我们可能需要在处理请求之前或者响应之前实现一些特殊的功能，例如，在用户访问某些资源之前检查用户是否登陆或是否具有一定的权限等。使用传统的方式实现起来比较繁琐。
  - 而 Filter （过滤器）可以对用户的请求进行拦截，即可以在请求访问 web 资源之前，对请求进行拦截，或者在响应发送给客户端之前，进行拦截，从而做相应的处理。
  - 通过过滤器处理，只需要将代码在过滤器编写一次，所有能够拦截的请求，在处理之前或响应之前都可以执行过滤器中的代码，实现了代码的复用，也提高了程序的维护性和扩展性。

知识讲解

Filter 过滤器（1）

- 【参见COOKBOOK】
  - 演示 Servlet 在接受请求参数时和做出响应时的乱码问题。
  - 按照传统方式处理乱码问题，并思考其中的缺点。

课堂练习

开发过滤器

开发一个过滤器

- 开发一个 Filter和开发一个 Servlet很相似。步骤如下：
  - （1）写一个类，实现 javax.servlet.Filter 接口，实现 doFilter 方法来处理所拦截到的请求。
  - （2）在 web.xml 文件中配置 Filter 及所拦截的路径。
- Hello World实例
  - （1）创建 HelloFilter 类，实现 Filter 接口，并添加未实现的方法

知识讲解

```
public class HelloFilter implements Filter {
    public void destroy() { }
    public void doFilter(ServletRequest request, ServletResponse resp
            FilterChain chain) throws IOException, ServletException {
        System.out.println("hello Filter....");
        chain.doFilter(request, response);
    }
    public void init(FilterConfig arg0) throws ServletException { }
}
```

开发一个过滤器（续1）

- （2）打开 web.xml 文件，配置 HelloFilter 相关信息及所拦截的路径。

```
<filter>
    <!-- Filter 的名字 -->
    <filter-name>HelloFilter</filter-name>
    <!-- Filter 类的路径 -->
    <filter-class>cn.tedu.filter.HelloFilter</filter-class>
</filter>
<filter-mapping>
    <!-- Filter 的名字，必须和上面保持一致 -->
    <filter-name>HelloFilter</filter-name>
    <!-- Filter 所拦截的路径 -->
    <url-pattern>/index.jsp</url-pattern>
</filter-mapping>
```

知识讲解

Filter详解

- （1）过滤器中的方法
  - 过滤器是一个实现了 javax.servlet.Filter 接口的 Java 类。javax.servlet.Filter 接口定义了三个方法：

序号	方法 & 描述
1	public void init(FilterConfig filterConfig) 该方法在 Filter 实例创建完成后立即执行，进行初始化的操作。
2	public void doFilter (ServletRequest, ServletResponse, FilterChain) 该方法负责对拦截到的请求进行处理。其中参数 FilterChian 表示多个过滤器组成的过滤器链对象
3	public void destroy() 该方法在 Filter 实例销毁之前调用，进行善后的处理

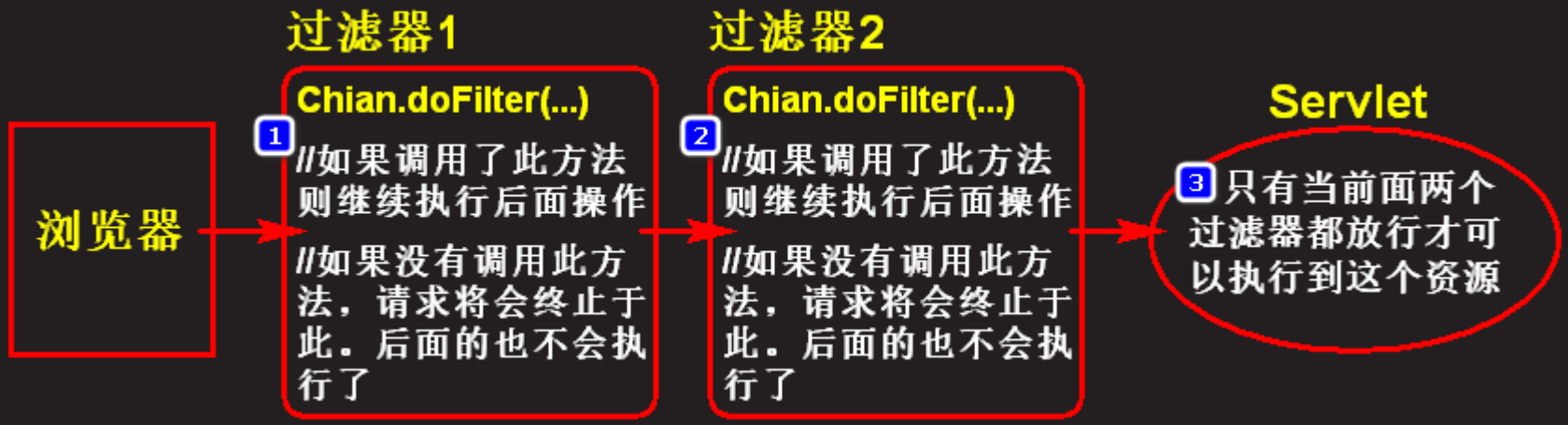
  - Filter 中的 init 及destroy 方法和 Servlet 中的非常相似，这里不再讲解。

知识讲解

Filter详解（续1）

- 过滤器中的 doFilter 方法
  - doFilter 方法用于对过滤器所拦截的请求进行处理 及 是否放行对后续过滤器或后续资源的访问。
  - 只有当 doFilter 方法中的参数 FilterChian 调用了 自身的 doFilter 方法，才会放行请求，接着执行后面的过滤器或资源。

知识讲解



Filter详解（续2）

- （2）过滤器 doFilter 方法中的参数 – FilterChain
  - doFilter 方法的参数有一个类型为 FilterChain 的参数，它只有一个方法：doFilter(request，response)
  - 前面我们说调用 FilterChain 的 doFilter 的方法可以放行，接着执行后面的操作，其实调用该方法的意思是，"我（当前 Filter）" 放行了，但是不代表其他的过滤器也放行。
  - 如果一个 web 资源被多个过滤器所拦截，可以比作是过年回家的路上有多个劫匪（过滤器），如果第一伙劫匪将你放行了（即调用 FilterChain 的 doFilter 的方法），但不代表第二伙劫匪也会放行，依次类推。
  - 所以调用 FilterChain 的 doFilter 的方法表示执行下一个过滤器，如果没有下一个过滤器，就执行对应的web资源。

知识讲解

配置过滤器

- 在开发完一个 Filter 之后，需要在 web.xml 文件中配置 Filter 的信息及所拦截的路径。
  - 需要配置的信息包括两个内容，一个是 filter 标签，另一个是 filter-mapping 标签
  - （1）<filter> 标签的配置

```
<filter>
    <!-- Filter 的名字 -->
    <filter-name>HelloFilter</filter-name>
    <!-- Filter 类的路径 -->
    <filter-class>cn.tedu.filter.HelloFilter</filte
</filter>
```

  - 其中 filter-name 仅仅是 Filter 的名字，可以任意指定。
  - filter-class 用来指定 Filter 的类路径。千万不要写错！

知识讲解

配置过滤器（续1）

知识讲解

- （2）<filter-mapping> 标签的配置

```
<filter-mapping>
    <!-- Filter 的名字，必须和上面保持一致 -->
    <filter-name>HelloFilter</filter-name>
    <!-- Filter 所拦截的路径 -->
    <url-pattern>/index.jsp</url-pattern>
</filter-mapping>
```

- 其中 <filter-name> 仅仅是 Filter 的名字，需要和 filter 标签中的name保持一致。
- **<url-pattern>** 用来指定过滤器所拦截的路径。路径的配置可以直接写一个固定的路径，也可以采用 * 号配置路径。
- 其实，过滤器所拦截的路径还可以通过在 <filter-mapping> 中通过 **<servlet-name>**标签进行配置，用于配置所拦截的 Servlet 的名字。

配置过滤器（续2）

- **<url-pattern> 和 <servlet-name>** 标签都可以配置多个，多个路径所拦截的资源都可以通过该过滤器处理。
- 例如：

```
<filter-mapping>
    <filter-name>HelloFilter</filter-name>
    <!-- Filter 所拦截的路径 -->
    <url-pattern>/index.jsp</url-pattern>
    <url-pattern>/login.jsp</url-pattern>
    <servlet-name>HelloFilter</servlet-name>
    <servlet-name>HelloFilter2</servlet-name>
</filter-mapping>
```

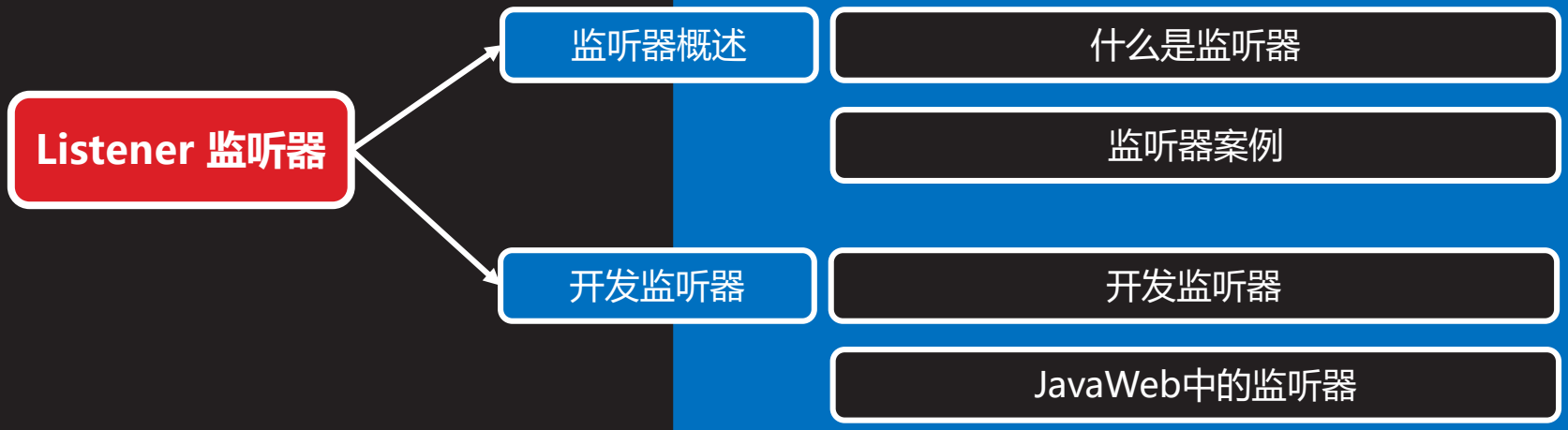
- 上面的配置中可以拦截对 index.jsp、login.jsp页面以及名称为 HelloFilter 和 FilterDemo 两个 Servlet 的访问。

Filter 过滤器（2）

- 【参见COOKBOOK】
  - 实现一个过滤器案例：允许用户在访问Servlet之前，在请求中携带两个“暗号“，接着配置两个过滤器对访问Servlet的请求进行拦截，在过滤器中判断暗号是否正确。例如第一个过滤器判断第一个暗号，第二个过滤器判断第二个暗号。只有当两个暗号都正确时，才允许访问Servlet。只要有一个过滤器判断暗号错误，将会跳转到提示页面提示暗号错误。

课堂练习

Listener 监听器

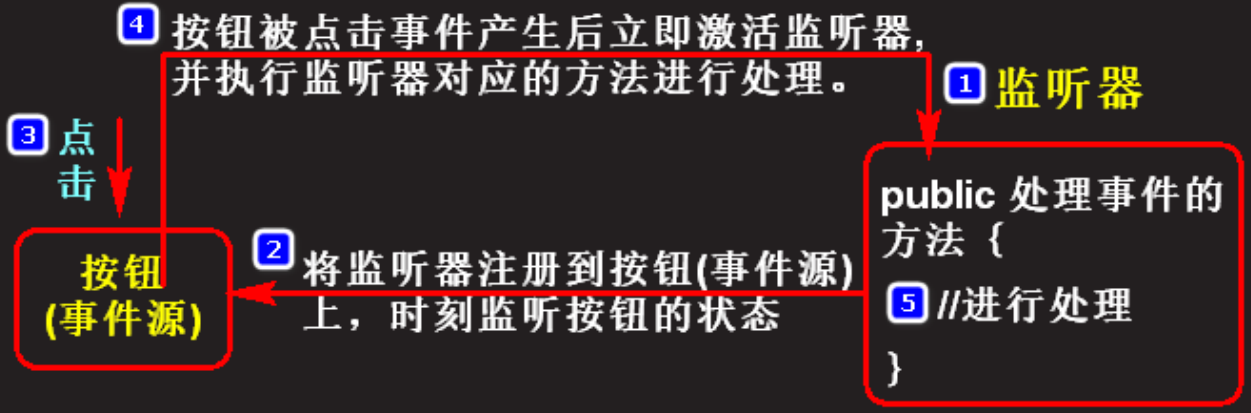


监听器概述

什么是监听器

- Servlet 技术规范中，定义了 Servlet、Filter、**Listener** 三门技术，统称为 JavaWeb 的三大组件。
  - 其中 Listener 就是监听器，监听器就是一个实现特定接口的 java 程序。
  - 这个程序专门用于监听另一个 java 对象状态变化（比如：创建、销毁或属性改变等），当被监听对象发生上述事件后，就会通知监听器，监听器中的某个方法就会立即执行，来处理该事件。

知识讲解



监听器案例

- 在 JavaSE 的 GUI 编程中，其中事件处理机制就有监听器的身影。通过实现 Listener 接口的类，可以在特定事件发生时，呼叫特定的方法来对事件进行响应。
- 下面通过一个 Demo，来深入的了解一下监听器的使用。
  - 创建一个 ListenerDemo 类，在 main 函数中创建一个窗口，并在窗口中添加一个按钮，代码如下:

```
// 1.创建一个窗口，并设置窗口的大小及位置
JFrame frame = new JFrame();
frame.setSize(250, 200);
frame.setLocation(350, 250);
// 2.创建一个按钮, 并添加到窗口中
JButton btn = new JButton("按钮");
```

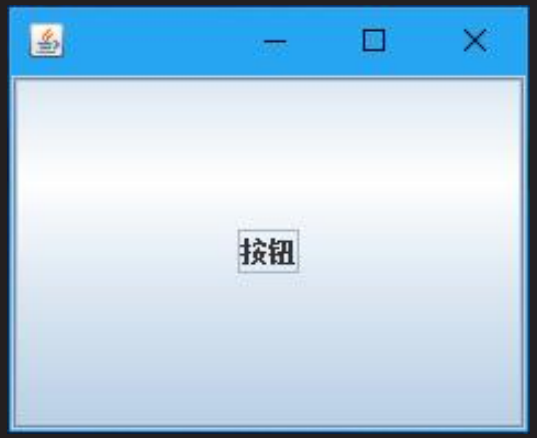
知识讲解

监听器案例（续1）

```
// 3.将按钮添加到窗口中
frame.add(btn);
// 4.设置窗口为显示状态
frame.setVisible(true);
```

知识讲解

- 运行上述代码，结果如下：



  - 点击按钮没有任何效果。
- 现有需求：
  - 实现点击按钮，在控制台打印 "Hello btn..."

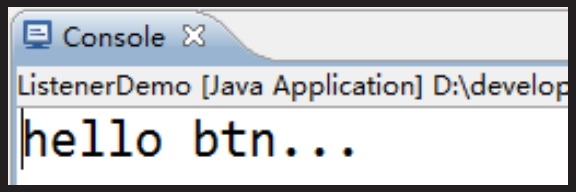
监听器案例（续2）

知识讲解

- 添加如下代码：

```
// 5.创建一个监听器(行为监听器)
ActionListener listener = new ActionListener() {
    // 处理事件的方法(事件处理器)
    public void actionPerformed(ActionEvent e) {
        System.out.println("hello btn...");
    }
};
// 6.将监听器注册到按钮上（此后监听器会一直监听按钮的状态）
btn.addActionListener(listener);
```

- 再次点击按钮，控制台效果：



监听器案例（续3）

知识讲解

- 在上述案例中，采用了事件监听机制，实现了按钮在被点击之后，立即在控制台打印 "Hello btn..."，其中在按钮被点击事件中可以分为：
  - 事件源：按钮也称为事件源。
  - 事件："按钮被点击" 为发生的事件
  - 监听器：listener 称为监听器。
- 小结：
  - 在按钮和监听器刚创建之后，其实二者之间并没有关系，即使按钮被点击也不会产生任何输出，所以后面需要将监听器注册在按钮上，之后监听器将会一直监听按钮的状态，只要按钮被点击，就会执行监听器中的 actionPerformed 方法，从而实现特点的功能。

监听器案例（续4）

- 总结：通过上述按钮，可以得出监听器具有如下特点：
  - 它是一个接口，内容由我们自己来实现
  - 它需要注册，例如注册在按钮上
  - 监听器中的方法，会在特殊事件发生时被调用
- 思考：
  - ListenerDemo 程序中的监听器 ActionListener 为什么是一个接口？

知识讲解

开发监听器

开发监听器

- JavaWeb 中编写监听器：
  - （1）写一个监听器类：要求必须实现对应的监听器接口，如：

```
public class MyXxxListener implements XxxListener{...}
```

  - （2）注册监听：在 web.xml 中配置监听器完成注册。

```
<listener>
    <listener-class>
        cn.tedu.listener.MyXxxListener
    </listener-class>
</listener>
```

知识讲解

JavaWeb中的监听器

- 在 JavaWeb 的监听器监听的事件源为：ServletContext、HttpSession、ServletRequest，即三个域对象。
  - 按照类型的划分，监听器分为三类：
  - （1）监听域对象 "创建" 与 "销毁" 的监听器、
  - （2）监听域对象 "操作域属性" 的监听器、
  - （3）监听 HttpSession 的监听器。
- （1） 监听域对象 "创建" 与 "销毁" 的监听器，也称之为生命周期监听器，共三个：

知识讲解

JavaWeb中的监听器（续1）

- 接上↑

监听器接口	方法 & 描述
ServletContextListener	void contextInitialized(ServletContextEvent sce) 创建 ServletContext 对象时调用
	void contextDestroyed(ServletContextEvent sce) 销毁 ServletContext 对象时调用
HttpSessionListener	void sessionCreated(HttpSessionEvent se) 创建 Session 对象时调用
	void sessionDestroyed(HttpSessionEvent se) 销毁 Session对象时调用
ServletRequestListener	void requestInitialized(ServletRequestEvent sre) 创建request时调用
	void requestDestroyed(ServletRequestEvent sre) 销毁request时调用

知识讲解

JavaWeb中的监听器（续2）

- （2）监听域对象 "操作域属性" 的监听器，也称之为属性监听器，共三个：

监听器接口	方法 & 描述
ServletContextAttributeListener	void attributeAdded(ServletContextAttributeEvent scae)； 添加属性时调用
	void attributeReplaced(ServletContextAttributeEvent scae)； 替换属性时调用
	void attributeRemoved(ServletContextAttributeEvent scae)； 删除属性时调用
HttpSessionAttributeListener	void attributeAdded(HttpSessionBindingEvent se)；添加属性时调用
	void attributeReplaced(HttpSessionBindingEvent se)； 替换属性时调用
	void attributeRemoved(HttpSessionBindingEvent se)； 删除属性时调用

知识讲解

JavaWeb中的监听器（续3）

- 接上↑

监听器接口	方法 & 描述
ServletRequestAttributeListener	void attributeAdded(ServletRequestAttributeEvent srae) 添加属性时调用
	void attributeReplaced(ServletRequestAttributeEvent srae) 替换属性时调用
	void attributeRemoved(ServletRequestAttributeEvent srae) 删除属性时调用

知识讲解

JavaWeb中的监听器（续4）

- （3）监听 HttpSession 的监听器

监听器接口	方法 & 描述
HttpSessionBindingListener	valueBound(HttpSessionBindEvent e); 当前JavaBean感知到自己被添加到Session时调用。
	valueUnbound(HttpSessionBindEvent e); 当前JavaBean感知到自己被移出Session时调用
HttpSessionActivationListener	sessionWillPassivate(HttpSessionEvent e); 当前JavaBean感知自己随着Session一起钝化时调用
	sessionDidActive(HttpSessionEvent e); 当前JavaBean感知自己随着Session一起活化时调用

知识讲解

Listener 监听器（1）

- 【参见COOKBOOK】
  - 编写一个 ServletContextListener 的实例（监听器），用于监听 ServletContext 对象的创建和销毁。

课堂练习

总结和答疑