监听器就是一个**Java**类用来监听其他的**JavaBean**的变化.

* 监听器就是监听三个域对象的状态的。**request　　session　　ServletContext**

监听器和过滤器属于Servlet中的高级技术.

Servlet的监听器：

监听ServletContext,HttpSession,ServletRequest

* 事件源和监听器绑定的过程:通过配置web.xml完成.

Servlet的监听器分成三类8个:

* 一类：监听三个域对象的创建和销毁的监听器.
* 二类：监听三个域对象的属性的变更.
* 三类：监听HttpSession中JavaBean的状态的改变.

一类：监听三个域对象的创建和销毁的监听器

ServletContextListener：监听ServletContext对象的创建和销毁.

ServletContext对象何时创建和销毁:应用在，它就在

* 创建：服务器启动加载应用时创建。一个应用只有一个ServletContext对象
* 销毁：服务器停止或应用卸载时销毁。

案例

1.编写一个类实现监听器的接口.

public class MyServletContextListener implements ServletContextListener

2.通过配置完成监听器和事件源的绑定. 在web.xml中配置

```
<listener>
    <listener-class>com.itheima.weblistener.MyServletContextListener</listener-class>
</listener>
```

HttpSessionListener:监听**HttpSession**对象的创建和销毁的监听器.

HttpSession对象何时创建和销毁的?

* 创建:第一次执行getSession()时，或第一次访问JSP时创建。
* 销毁：

默认30分钟销毁

setMaxInactiveInterval(int interval)

服务器非正常关闭。

Invalidate() 直接销毁

案例

1.编写监听器:　在web.xml中配置

public class MyHttpSessionListener implements HttpSessionListener

2.配置监听器:

```
<listener>
    <listener-class>com.itheima.weblistener.MyHttpSessionListener</listener-class>
```

</listener>

访问一个Servlet是否创建session对象？ :不一定 看是否有getSession()

访问一个jsp是否创建session对象? :默认会，如果JSP关闭了这个功能就不会

ServletRequestListener:监听ServletRequest对象的创建和销毁的监听器

ServletRequest对象何时创建和销毁?

* 创建:每次发送请求都会创建。

* 销毁:每次响应结束都销毁了。

案例

1.编写一个监听器

public class MyServletRequestListener implements ServletRequestListener

1.配置监听器 在**web.xml**中配置

```
<listener>
<listener-class>com.itheima.weblistener.MyServletRequestListener</listener-class>
</listener>
```

二类：监听三个域对象属性变更的监听器

ServletContextAttributeListener：监听ServletContext对象中的属性变更的监听器

HttpSessionAttributeListener：监听HttpSession对象中的属性变更的监听器

ServletRequestAttributeListener:监听ServletRequest对象中的属性变更的监听器

三类：监听**HttpSession**中的**JavaBean**的状态改变的监听器

HttpSessionBindingListener:监听HttpSession中的JavaBean的绑定和解除绑定的状态

HttpSessionActivationListener:监听HttpSession中的JavaBean的钝化和活化的状态

过滤器**Filter**

Filter：一个实现了特殊接口的**Java**类.实现对请求资源的过滤的功能.

* 过滤器是**Servlet**技术中最为实用的技术.

➢· 过滤器的作用：

对目标资源进行过滤.

* 自动登录,解决网站乱码,进行脏字、敏感词汇的过虑，进行页面静态化,进行响应压缩...

➢· 使用过滤器：

1.编写一个类实现过滤器的接口.

包 javax.servlet.Filter;

public class FilterDemo1 implements Filte

真正过滤的方法

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

2.对过滤器进行配置. 在web.xml

```
<filter>
    <filter-name>FilterDemo1</filter-name>
    <filter-class>com.itheima.filter.FilterDemo1</filter-class>
</filter>

<filter-mapping>
    <filter-name>FilterDemo1</filter-name>
    <url-pattern>/*</url-pattern>        随Servlet类配置
</filter-mapping>
```

过滤器的生命周期（了解）

*** Servlet的生命周期**

实例化（**new**） 初始化（**init**） 服务（**service**） 销毁（**destroy**）

过滤器从创建到销毁的过程.

实例化（**new**）初始化（**init**）过滤（**doFilter**） 销毁（**destroy**）

Filter与Servlet生命周期的区别：

Filter实例化（**new**）初始化（**init**）服务器启动时执行。**Servlet**实例化（**new**）初始化（**init**）第一次访问时执行

* 服务器启动的时候,服务器就会创建过滤器的对象,每次访问被拦截目标资源,过滤器中的**doFilter**的方法就会执行.当服务器关闭（或应用卸载）的时候,服务器就会销毁**Filter**对象.

过滤器链：多个Filter能对同一资源进行拦截，就构成了过滤器链。如果过滤器是链中的最后一个，则调用目标资源。

过滤器链中的过滤器的执行的顺序跟<filter-mapping>的配置顺序有关.

方法

void	**doFilter**(ServletRequest request, ServletResponse response) Causes the next filter in the chain to be invoked, or if the calling filter is the last filter in the chain, causes the resource at the end of the chain to be invoked.

Filter的配置

【**url-pattern**的配置】与**servlet**中的配置一样：

* 三种配置**:**

  * 完全路径匹配：以 / 开始**/aaa /aaa/bbb**

  * 目录匹配**:**以 / 开始**/* /aaa/***

  * 扩展名匹配：不能以 / 开始***.do *.jsp *.action**

【**servlet-name**的配置】通过**url-pattern**拦截一个**Servlet**的资源**.**也可以通过**servlet-name**标签进行拦截**.**

dispatcher的配置 ： 告知服务器如何拦截 注：要配合url-pattern标签一起使用

*** REQUEST** ： 默认值**.**

*** FORWARD** ： 拦截转发

*** ERROR** ： 拦截跳转到错误页面**.**全局错误页面**.**

*** INCLUDE** ： 拦截在一个页面中包含另一个页面**.**

使用Filter解决请求数据乱码见案例代码

实现步骤：

1、创建Filter

2、使用request.setCharacterEncoding(“UTF-8”); 解决post乱码

问题：如果get方式提交的数据如何解决？

使用代码增强：

1、继承

2、装饰者模式

3、动态代理

自动登录核心代码：

```
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException {
    //1、强转
    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) resp;
    //2、处理业务
        //2.1从session域对象中获得user
        HttpSession session = request.getSession();
        User user = (User) session.getAttribute("user");

        if(user==null){ //说明没有登录
            String username = "";
            String password = "";
            Cookie[] cookies = request.getCookies();
            for (int i = 0;cookies!=null && i < cookies.length; i++) {
                if("autologin".equals(cookies[i].getName())){
                    String[] values = cookies[i].getValue().split("_");
                    username = values[0];
                    password = values[1];
                    break;
                }
            }
            UserService us = new UserService();
            user = us.findUserByUsernameAndPassword(username, password);
            //把user对象保存到session域对象中
            session.setAttribute("user", user);
        }
```

FilterChain：过滤器链