一、Listener（监听器）：

常用监听器“6+2”：

6个常规监听器：

|----ServletContext:

|---ServletContextListener（生命周期监听器）

|---ServletContextAttributeListener(属性监听)

|---HttpSession

|---HttpSessionListener(生命周期监听)

|---HttpSessionAttributeListener(属性监听)

|---ServletRequest

|---ServletRequestListener(生命周期监听)

|---ServletRequestAttributeListener(属性监听)

2个感知监听器：

|---HttpSessionBindingListener

|---HttpSessionActivationListener

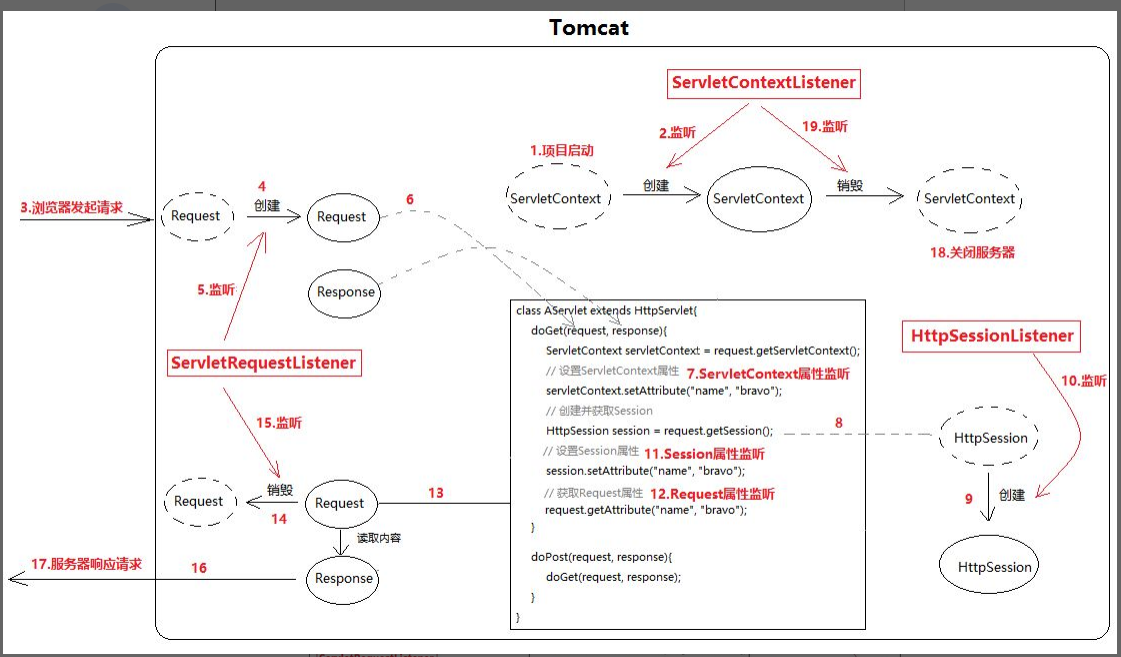
感知监听都是和session有关的

6个常规监听器，分属三类，分别对应JavaWeb三大域对象（除去JSP的Page域）：ServletContext、HttpSession、ServletRequest。共三对，每一对都包括1个生命周期监听和1个属性监听。

所谓生命周期监听器，就是监听三大域对象的创建和销毁。每当Tomcat创建或销毁三大域对象，都会被这些监听器察觉，然后它们会做相应的操作（调用自身的特定方法）。

属性监听器则专门监听三大域对象get/setAttribute()。每当我们给域对象设置值或者从里面取值，都会被它们监听到，然后还是触发它们特定的方法。

如下图：查看监听器到底在做什么事：



为了帮助大家理解接下来这张图的细节，我问几个问题，大家带着问题去看：

1、三大生命周期监听器，各自在何时创建、销毁，顺序是怎么样的？

2、访问一个Servlet，HttpSessionListener一定会触发吗？（换个角度就是，访问Servlet，Session一定会创建吗？）

3、三大属性监听何时触发？

答案：

问题一：

**1、在项目启动时**ServletContextListener监听到ServletContext对象创建。

**2、每一次请求**Tomcat都会创建一个Request，它的创建会被ServletRequestListener监听到。

**3、如果**Servlet中调用了request.getSession()，则Tomcat会创建Session（如果根据JSESSIONID找不到对应的），这会被HttpSessionListener监听到。

4、请求结束，Request销毁，被监听到。

5、用户30分钟未访问，Session过期销毁，被监听到。

6、项目关停，ServletContext销毁，被监听到。

问题二：

只有当在Servlet中调用request.getSession()，且根据JSESSIONID找不到对于的Session时，才会创建新的Session对象，才会被监听到。第二次请求，浏览器会带上JSESSIONID，此时虽然还是request.getSession()，但是会返回上次那个。根据JSESSIONID去Session这个过程是隐式的，我们看到的就是getSession()。

问题三：

对于图中的步骤7/11/12，也就是get/setAttribute()时，会触发属性监听。

二：观察者模式(Observer)：

监听器的底层涉及到的观察者模式：

观察者（Observer）模式的定义：指多个对象间存在一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并被自动更新。这种模式有时又称作发布-订阅模式、模型-视图模式，它是对象行为型模式。

观察者模式是一种对象行为型模式，其主要优点如下：

1、降低了目标与观察者之间的耦合关系，两者之间是抽象耦合关系。

2、目标与观察者之间建立了一套触发机制。

它的主要缺点如下：

1、目标与观察者之间的依赖关系并没有完全解除，而且有可能出现循环引用。

2、当观察者对象很多时，通知的发布会花费很多时间，影响程序的效率。

模式的结构与实现：

实现观察者模式时要注意具体目标对象和具体观察者对象之间不能直接调用，否则将使两者之间紧密耦合起来，这违反了面向对象的设计原则。

**模式的结构**

观察者模式的主要角色如下：

1、抽象主题（Subject）角色：也叫抽象目标类，它提供了一个用于保存观察者对象的聚集类和增加、删除观察者对象的方法，以及通知所有观察者的抽象方法。

2、具体主题（Concrete Subject）角色：也叫具体目标类，它实现抽象目标中的通知方法，当具体主题的内部状态发生改变时，通知所有注册过的观察者对象。

3、抽象观察者（Observer）角色：它是一个抽象类或接口，它包含了一个更新自己的抽象方法，当接到具体主题的更改通知时被调用。

4、观察者（Concrete Observer）角色：实现抽象观察者中定义的抽象方法，以便在得到目标的更改通知时更新自身的状态。

观察者模式理解：

一个对象的状态发生改变时，所有依赖于它的对象都得到通知，并自动更新。

被观察者发生改变时，会通知观察者，然后观察者再调用自身的逻辑，进行处理。

## 模式的扩展

在[Java](https://link.zhihu.com/?target=http%3A//c.biancheng.net/java/" \t "_blank)中，通过 java.util.Observable 类和 java.util.Observer 接口定义了观察者模式，只要实现它们的子类就可以编写观察者模式实例。

**1、Observable类**

Observable 类是抽象目标类，它有一个 Vector 向量，用于保存所有要通知的观察者对象，下面来介绍它最重要的 3 个方法。

1. void addObserver(Observer o) 方法：用于将新的观察者对象添加到向量中。
2. void notifyObservers(Object arg) 方法：调用向量中的所有观察者对象的 update。方法，通知它们数据发生改变。通常越晚加入向量的观察者越先得到通知。
3. void setChange() 方法：用来设置一个 boolean 类型的内部标志位，注明目标对象发生了变化。当它为真时，notifyObservers() 才会通知观察者。

**2、Observer 接口**

Observer 接口是抽象观察者，它监视目标对象的变化，当目标对象发生变化时，观察者得到通知，并调用 void update(Observable o,Object arg) 方法，进行相应的工作。

简单代码模拟：

//被观察者， 继承Observable

public class OilFutures extends Observable {  
 *//原油价格* float price;  
  
 public float getPrice() {  
 return price;  
 }  
 /\*\*  
 \* 由于价格变动会引发其他对象的相应行为，所以此处应该被监听  
 \*/  
 public void setPrice(float price) {  
 this.price = price;  
 *//设置内部标志位，注明数据发生变化* super.setChanged() ;  
 *//通知观察者价格改变了* super.notifyObservers(price);  
 }  
}

//观察者 实现了Observer， 其内只有一个update方法，需实现类实现

public class Bull implements Observer {  
 @Override  
 public void update(Observable o, Object arg) {  
 float price = ((Float) arg).floatValue();  
 if(price > 0){  
 System.***out***.println("油价上涨" + price + "元，多方高兴了。");  
 }else if(price < 0){  
 System.***out***.println("油价下跌" + price + "元，多方伤心了。");  
 }  
 }  
}

//测试方法

public class ObserverTest\_01 {  
 public static void main(String[] args) {  
 OilFutures oilFutures = new OilFutures();  
 Bear bear = new Bear();  
 Bull bull = new Bull();

//需将观察者对象添加进栈中，以便事件触发能通知对应的观察者  
 oilFutures.addObserver(bear);  
 oilFutures.addObserver(bull);  
 oilFutures.setPrice(100F);  
 }  
}