一． GUI 和 AWT Event Model

GUI： Graphic User Interface 图形用户界面

下面我们简介一下两种主要的产品模型：

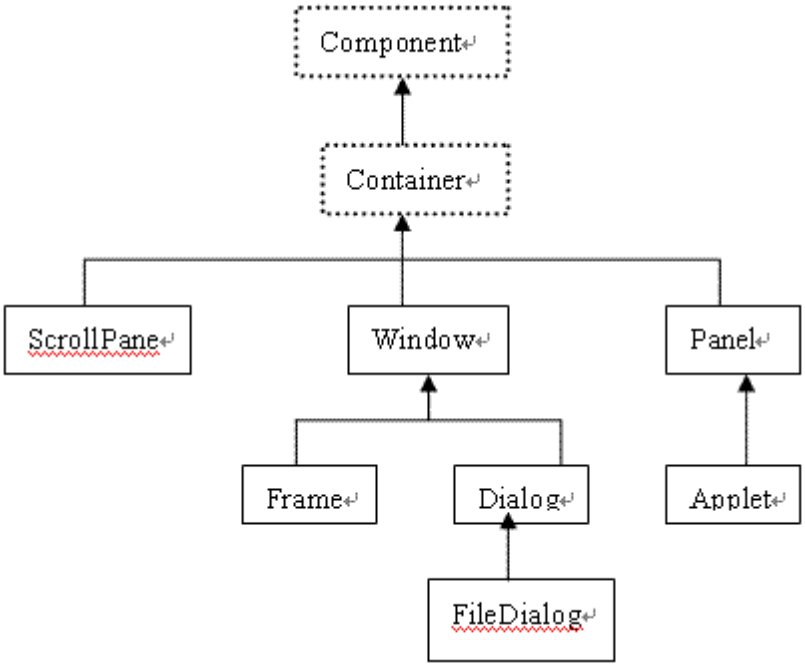
- B/S 模型： 特点：一切都在 Server 端运行，客户端只需要有浏览器，支持 HTTP 协议，如果软件更新或重新部署，只需要更新 Server 端既可，对用户不会产生影响。
- C/S 模型： 特点：用户必需下载客户端软件，如果服务改更或更新，客户端都有必需要重新下载或升级，并且对硬件也有一定的要求。

二． AWT： (Abstract Window Tookit) 抽象窗口工具包

做 SWING 的基本步骤：

1. 选择一个容器。 如 JFrame, JPanel
2. 使用哪种布局管理器
3. 添加组件
4. 给组件加上事件临听器

我们来看一下“ 容器层次图”。 如下：



注：Container 是一个组件管理容器，容器本身也是一个特殊的组件（Component），它是 Component 的子类，（由图中可以看出），这其实是一种‘组合模式’

另：Frame 是一个顶层容器，这有一个完整的窗体，包括有最小化图标，最大化图标，关闭，可以存放一些其它容器和组件，默认的布局管理器是：BorderLayout

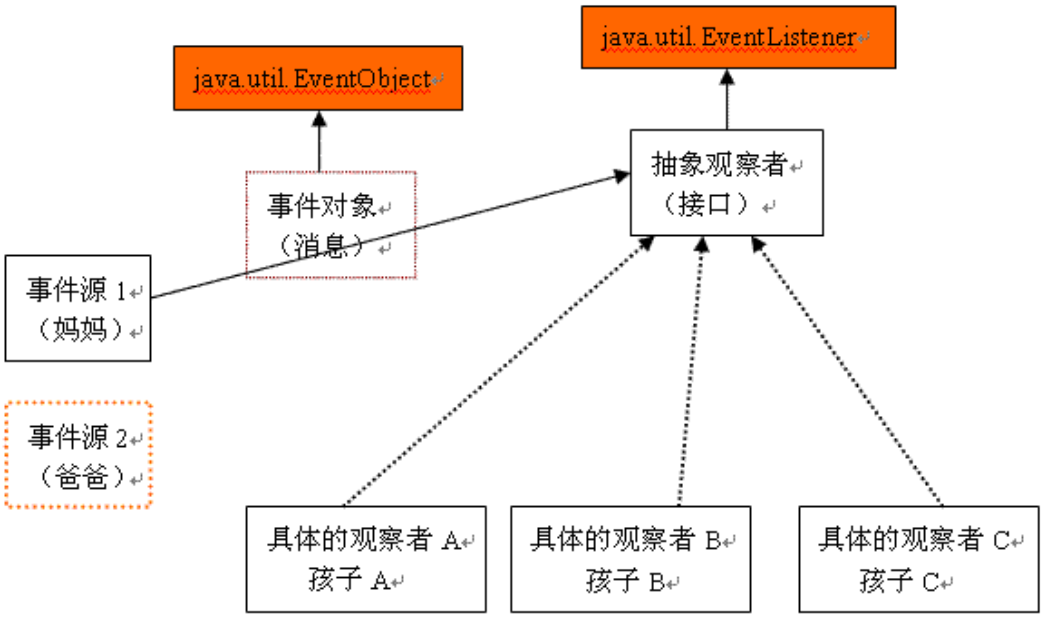
Panel 不是个顶层容器，它没有图标，本身并不可见，就是一个面板，可以存放组件，然后整个 Panel 放在一个 Frame 中，默认的布局管理器为：FlowLayout

三． 现在我们要重点学习 AWT 事件模型中的‘观察者模式‘：

让我们一起来分析一下一个日常生活中的例子：妈妈（Mother）和三个孩子

妈妈（Mother）到了晚上 6：00 会通知三个孩子回来，我们要求，妈妈只会喊一声，而孩子们就必须要各自回来，并做它们自己的事情。

那么这其实就是一个‘典型’的观察者模式。我们把‘妈妈’当做事件源，而孩子们则是‘观察者’，他们会接受妈妈的消息。
我们来看一下图：如下：



我们这里多出一个‘事件对象’和‘抽象观察者’，这就好比‘妈妈’要确保与‘小孩子们’进行消息传递，必定要有某种‘底层支持‘一样，那么这两个对象事实上 JDK 已经提供了，我们所要做的无非是继承而以。
所以，我们可以从下面三个对象来着手：

1. **EventSource** 事件源， 触发事件 （妈妈）
2. **EventObject** 事件对象。 它封装了事件源的信息 （消息）
3. **EventListener** 事件监听器： 为事件实现者提供标准的接口。 （抽象观察者）

请记住：事件源会以事件对象为参数去调用事件监听器的相应方法。
从上图中还可以看出，一个主题（事件源）可以有多个观察者（事件监听者）
一个观察者（事件监听者）也可以有多个主题（事件源）

注：事件对象一定要继承于： java.util.EeventObject 类
事件监听接口一定要继承于： java.util.EventListener 接口
代码如下：

```
class Mother {   //事件源
    private List list = new ArrayList();
    //注册一个事件
    public void addHomeWorkListener(HomeWorkListener hwl) {
        list.add(hwl);
    }
    //删除一个事件
    public void removeHomeWorkListener(HomeWorkListener hwl) {
        list.remove(hwl);
    }
    public void notify() { //发出通知： 触发事件方法
```

```
            //6 O CLOLK
            HomeWorkEvent event = new HomeWorkEvent(this);
            Iterator it = list.iterator();
            while(it.hasNext()) {
                HomeWorkListener h = (HomeWorkListener)it.next();
                h.homework();
        }
    }
    class HomeWorkEvent extends EventObject{ //事件对象：  消息
        public HomeWorkEvent(Object o) {
            super(o);
        }
    }
    interface HomeWorkListener extends EventListener{ //抽象事件接口
        void homework(HomeWorkEvent o);
    }
    class XiaoQiang implements HomeWorkListener { //具体的监听者 1
        public void homework(HomeWorkEvent o) {
            //做小学五年级的作业
        }
    }
    class XiaoMing implements HomeWorkListener {//具体的监听者 2
        public void homework(HomeWorkEvent o) {
            //做小学三年级的作业
        }
    }
    class XiaoFend implements HomeWorkListener { //具体的监听者 3
        public void homework(HomeWorkEvent o) {
            //做大班的作业
        }
    }
```

从上面这个例子可以看出， 我们的命名规则：

对于事件对象： XXXEvent 如： HomeWorkEvent

对于事件监听接口： XXXListener 如： HomeWorkListener

注：事件对象中的 **getSource()** 方法，可以得到事件源对象，这对事件监听者来说至关重要，因为有时候一个事件监听者会有多个事件源，从而它可以做出正确的回应。

下面我们再来写几个例子：好好体会一下：

例 **1**： 用一个女孩类（**Gril**）做为事件源，她 **1**，**3**，**5**，**7**，**9** 会发出一个开心的消息给男朋友类**(Boy)**， **Boy** 类会做出相应回应，**2**，**4**，**6**，**8**，**10** 会发出一个悲伤的消息。**Boy** 同样做出回应。

分析：

1. 事件源： **Gril**
2. 事件对像： 这是一个情感事件： **EmotionEvent extends EventObject**
3. 事件监听接口： EmotionListener
4. 实现类： **Boy**

Coding:

```
public class Gril { //事件源
    private String name;
    private List list = new ArrayList(); //用来维护一组临听者
    public Gril(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    /*****************************************************
     * 此方法用来注册一个情感临听者
     * @param e: 情感临听者
     */
    public void addEmotionListener(EmotionListener e) {
        list.add(e);
    }
    /*****************************************************
    * 此方法用来去除一个情感临听者
    * @param e 情感临听者
    */
    public void removeEmotionListener(EmotionListener e) {
        list.remove(e);
    }
    //事件触发方法
    public void fire() {
        EmotionEvent event = new EmotionEvent(this);
        for (int i = 0; i < 10; i++) {
            if(i % 2 == 0 ) {
                Iterator it = list.iterator();
                while(it.hasNext()) {
                    EmotionListener el =
(EmotionListener)it.next();
                    el.whatCanIdoWhenHappy(event);
                }
            } else {
                Iterator it = list.iterator();
                while(it.hasNext()) {
                    EmotionListener el =
(EmotionListener)it.next();
```

```
                    el.whatCanIdoWhenSad(event);
                }
            }
        }
    }
    //事件对象：消息
    public class EmotionEvent extends EventObject {
        public EmotionEvent(Object o) {
            super(o);
        }
    }
    // 事件临听接口
    public interface EmotionListener extends EventListener {
        void whatCanIdoWhenHappy(EmotionEvent e);
        void whatCanIdoWhenSad(EmotionEvent e);
    }
    //实现类: Boy
    public class Boy implements EmotionListener {
        private String name;
        public Boy(String name) {
            this.name = name;
        }
        public String getName() {
            return name;
        }
        public void whatCanIdoWhenHappy(EmotionEvent e) {
            Object o = e.getSource();
            Gril g = (Gril)o;
            System.out.println(name+" said to "+g.getName()+", you
happy,I am happy");
        }
        public void whatCanIdoWhenSad(EmotionEvent e) {
            Object o = e.getSource();
            Gril g = (Gril)o;
            System.out.println(name+" said to "+g.getName()+", you
sad,I am so sad.");
        }
    }
```

例二： 有一个时间类（Time）,它从 2006 年开始每隔一年就像 Olympics（奥林匹克）和 WorldCup（世界杯）这个两个类发送年份，这个两个类会进行相应的计算，如果是奥运年，Olympics 就会打印： ‘年份’是奥运年，如果是世界杯年，WorldCup 就会打印：‘年份’是世界杯年。 请使用事件模型来设计。

Coding:

```
class Time {  //事件源
    private List lt = new ArrayList();//管理事件临听者的集合
    private int year; //用来保存当前的年份
    public int getYear() { return year; }
    public void addTimeListener(TimeListener t) {
        lt.add(t);
    }
    public void removeTimeListener(TimeListener t) {
        lt.remove(t);
    }
    public void action() {
        TimeEvent event = new TimeEvent(this);
        for(int i=2006;i<=3000;i++) {
            this.year = i; //设置当前年份
            Iterator it = lt.iterator();
            while(it.hasNext()) {
                TimeListener tl = (TimeListener)it.next();
                //事件源以事件对象做为参数调用事件临听接口的 disp()方法
                tl.disp(event);
            }
        }
    }
}
//事件对象
class TimeEvent extends EventObject {
    public TimeEvent(Object o) {
        super(o);
    }
}
//事件临听接口
interface  TimeListener extends EventListener {
    void disp(TimeEvent event);
}
//实现类 1：奥林匹克
class Olympics implements TimeListener {
    public void disp(TimeEvent event) {
        Time t = (Time)event.getSource();
        if(t.getYear() % 4 == 0 ){
            System.out.println(t.getYear()+" 奥运年");
        }
    }
}
//实现类 2： 世界杯年
class WorldCup implements TimeListener {
```

```
    public void disp(TimeEvent event) {
        Time t = (Time)event.getSource();
        if(t.getYear() % 4  == 2 ) {
            System.out.println(t.getYear()+" 是世界杯年");
        }
    }
}
```

作者：叶加飞
mailto: yejf@tarena.com.cn
加拿大.达内科技 (上海分中心)

好了，经过以上几个例子的演示，相信大家对事件临听模型已经有了一定的了解。那么交给大家一个作业：

1. 用事件模型来实现 N 个小孩围成圈，然后报数，报到 7 的那个小孩出列，接着从一下小孩又重新开始数数。依次类推，直到所有的小孩全部出列。

分析：以老师做为事件源，它负责数数，小孩子则做为临听者，如果老师数到 7，就会向他发出一个出列的消息，小孩负责临听。