1.适配器模式

基本介绍

1） 适配器模式(Adapter Pattern)将某个类的接口转换成客户端期望的另一个接口表示，主的目的是兼容性，让原本因接口不匹配不能一起工作的两个类可以协同工作。其别名为包装器(Wrapper)

2） 适配器模式属于结构型模式

3） 主要分为三类：类适配器模式、对象适配器模式、接口适配器模式

工作原理

1） 适配器模式：将一个类的接口转换成另一种接口.让原本接口不兼容的类可以兼容

2） 从用户的角度看不到被适配者，是解耦的

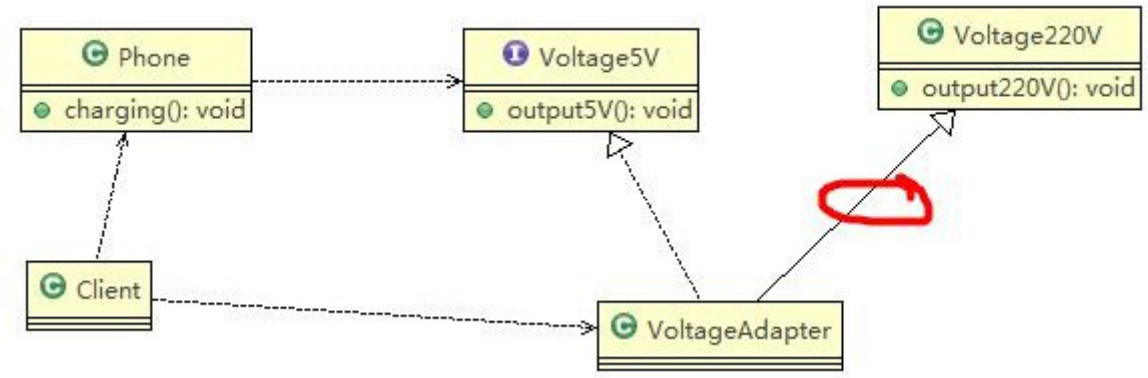
3） 用户调用适配器转化出来的目标接口方法，适配器再调用被适配者的相关接口方法

4） 用户收到反馈结果，感觉只是和目标接口交互，如图

1.1类适配器模式

基本介绍：Adapter 类，通过继承 src 类，实现 dst 类接口，完成 src->dst 的适配。

以生活中充电器的例子来讲解适配器，充电器本身相当于 Adapter，220V 交流电相当于 src （即被适配者），我们的目 dst(即 目标)是 5V 直流电



```
//适配器类
public class VoltageAdapter extends Voltage220V implements IVoltage5V {
  @Override
  public int output5V() {
    // TODO Auto-generated method stub
    //获取到220V电压
    int srcV = output220V(); //调用父类的方法获取电压
    int dstV = srcV / 44 ; //转成 5v
    return dstV;
  }
}
```

类适配器模式注意事项和细节

1) Java 是单继承机制，所以类适配器需要继承 src 类这一点算是一个缺点，因为这要求 dst 必须是接口，有一定局限性；

2) src 类的方法在 Adapter 中都会暴露出来，也增加了使用的成本。

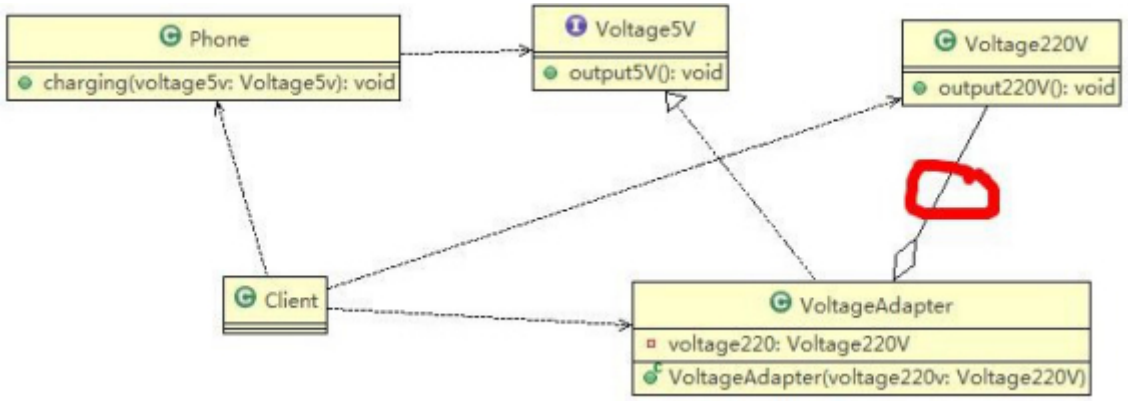
3) 由于其继承了 src 类，所以它可以根据需求重写 src 类的方法，使得 Adapter 的灵活性增强了。

1.2对象适配器模式(聚合)

基本介绍：基本思路和类的适配器模式相同，只是将 Adapter 类作修改，不是继承 src 类，而是持有 src 类的实例，以解决兼容性的问题。 即：持有 src 类，实现 dst 类接口，完成 src->dst 的适配

2) 根据“合成复用原则”，在系统中尽量使用关联关系（聚合）来替代继承关系。

3) 对象适配器模式是适配器模式常用的一种




```
//适配器类
public class VoltageAdapter  implements IVoltage5V {
  private Voltage220V voltage220V; // 关联关系-聚合
```

```
    //通过构造器，传入一个 Voltage220V 实例
    public VoltageAdapter(Voltage220V voltage220v) {
        this.voltage220V = voltage220v;
    }

    @Override
    public int output5V() {
        int dst = 0;
        if(null != voltage220V) {
            int src = voltage220V.output220V();//获取220V 电压
            System.out.println("使用对象适配器，进行适配~~");
            dst = src / 44;
            System.out.println("适配完成，输出的电压为=" + dst);
        }
        return dst;
    }
}
```

对象适配器模式注意事项和细节

1） 对象适配器和类适配器其实算是同一种思想，只不过实现方式不同。根据合成复用原则，使用组合替代继承， 所以它解决了类适配器必须继承 src 的局限性问题，也不再要求 dst必须是接口。

2） 使用成本更低，更灵活。

1.3 接口适配器模式

1） 一些书籍称为：适配器模式(Default Adapter Pattern)或缺省适配器模式。

2） 核心思路：当不需要全部实现接口提供的方法时，可先设计一个抽象类实现接口，并为该接口中每个方法提供一个默认实现（空方法），那么该抽象类的子类可有选择地 覆盖父类的某些方法来实现需求

3） 适用于一个接口不想使用其所有的方法的情况。

应用实例

1） Android 中的属性动画 ValueAnimator 类可以通 addListener(AnimatorListener listener)方法添加监听器， 那么常规写法如右：

2） 有时候我们不想实现 Animator.AnimatorListener 接口的全部方法，我们只想监听 onAnimationStart，我们会如下写

4) AnimatorListener 是一个接口.

```
public static interface AnimatorListener {
    void onAnimationStart(Animator animation);
    void onAnimationEnd(Animator animation);
    void onAnimationCancel(Animator animation);
    void onAnimationRepeat(Animator animation);
}
```

```
public abstract class AnimatorListenerAdapter implements Animator.AnimatorL
    Animator.AnimatorPauseListener {
  @Override //默认实现
  public void onAnimationCancel(Animator animation) {
  }

  @Override
  public void onAnimationEnd(Animator animation) {
  }

  @Override
  public void onAnimationRepeat(Animator animation) {
  }

  @Override
  public void onAnimationStart(Animator animation) {
  }

  @Override
  public void onAnimationPause(Animator animation) {
  }

  @Override
  public void onAnimationResume(Animator animation) {
  }
}
```

```
new AnimatorListenerAdapter() {
        @Override
        public void onAnimationStart(Animator  animation) {
            //xxxx具体实现
        }
    }
```

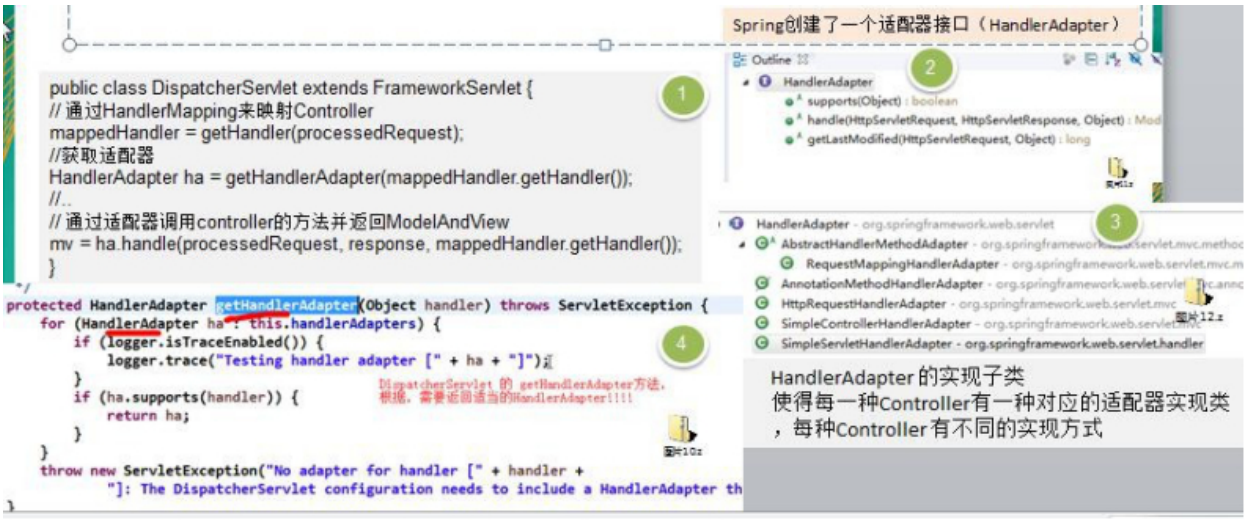
适配器模式在 SpringMVC 框架应用的源码剖析

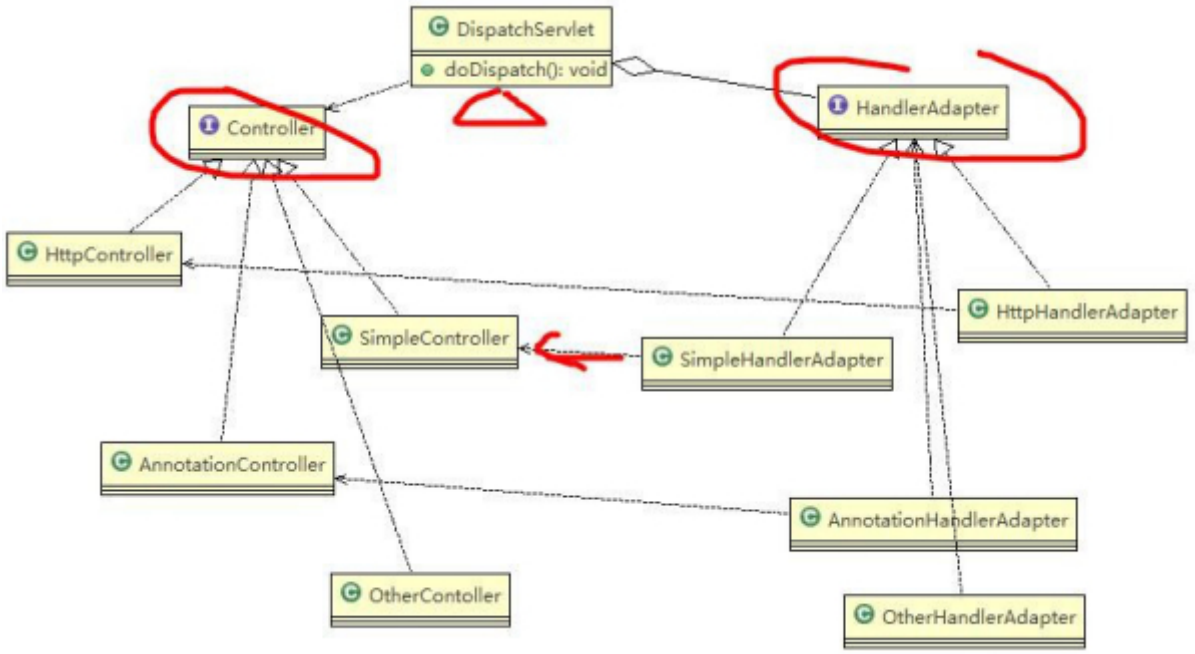
1） SpringMvc 中的 HandlerAdapter，就使用了适配器模式

2） SpringMVC 处理请求的流程回顾

3） 使用 HandlerAdapter 的原因分析：

可以看到处理器的类型不同，有多重实现方式，那么调用方式就不是确定的，如果需要直接调用 Controller 方法，需要调用的时候就得不断是使用 if else 来进行判断是哪一种子类然后执行。那么如果后面要扩展 Controller，就得修改原来的代码，这样违背了 OCP 原则。





适配器模式的注意事项和细节

1） 三种命名方式，是根据 src 是以怎样的形式给到 Adapter（在 Adapter 里的形式）来命名的。

2） 类适配器：以类给到，在 Adapter 里，就是将 src 当做类，继承对象适配器：以对象给到，在 Adapter 里，将 src 作为一个对象，持有接口适配器：以接口给到，在 Adapter 里，将 src 作为一个接口，实现

3） Adapter 模式最大的作用还是将原本不兼容的接口融合在一起工作。

4） 实际开发中，实现起来不拘泥于我们讲解的三种经典形式