# 1.概念

适配器就是一种适配中间件，它存在于不匹配的二者之间，用于连接二者，将不匹配变得匹配，简单点理解就是平常所见的转接头，转换器之类的存在。

　　适配器模式有两种：类适配器、对象适配器、接口适配器

前二者在实现上有些许区别，作用一样，第三个接口适配器差别较大。

# 2.类适配器

原理：通过继承来实现适配器功能。

当我们要访问的接口A中没有我们想要的方法 ，却在另一个接口B中发现了合适的方法，我们又不能改变访问接口A，在这种情况下，我们可以定义一个适配器p来进行中转，这个适配器p要实现我们访问的接口A，这样我们就能继续访问当前接口A中的方法（虽然它目前不是我们的菜），然后再继承接口B的实现类BB，这样我们可以在适配器P中访问接口B的方法了，这时我们在适配器P中的接口A方法中直接引用BB中的合适方法，这样就完成了一个简单的类适配器

ps2接口:

|  |
| --- |
| 1 public interface Ps2 {  2 void isPs2();  3 } |

usb接口:

|  |
| --- |
| 1 public interface Usb {  2 void isUsb();  3 } |

usb的实现类:

|  |
| --- |
| package adaptor;  public class UsbImpl implements Usb {  @Override  public void isUsb() {  System.out.println("USB口");  }  } |

适配器:

|  |
| --- |
| package adaptor;  public class Adapter extends UsbImpl implements Ps2{  @Override  public void isPs2() {  isUsb();  }  } |

测试:

|  |
| --- |
| Ps2 p = new Adapter();  p.isPs2(); |

ps2为接口是,当时这个接口中没有我们想要访问的方法,

但是usb接口中有我们想要访问的方法,这时,我们就构建一个适配器

这个适配器继承usb,实现目标接口ps2.

# 3.对象适配器

原理：通过组合来实现适配器功能。

　　当我们要访问的接口A中没有我们想要的方法 ，却在另一个接口B中发现了合适的方法，我们又不能改变访问接口A，在这种情况下，我们可以定义一个适配器p来进行中转，这个适配器p要实现我们访问的接口A，这样我们就能继续访问当前接口A中的方法（虽然它目前不是我们的菜），然后在适配器P中定义私有变量C（对象）（B接口指向变量名），再定义一个带参数的构造器用来为对象C赋值，再在A接口的方法实现中使用对象C调用其来源于B接口的方法。

ps2接口:

|  |
| --- |
| 1 public interface Ps2 {  2 void isPs2();  3 } |

usb接口:

|  |
| --- |
| 1 public interface Usb {  2 void isUsb();  3 } |

usb的实现类:

|  |
| --- |
| package adaptor;  public class UsbImpl implements Usb {  @Override  public void isUsb() {  System.out.println("USB口");  }  } |

适配器:

|  |
| --- |
| package adaptor;  public class Adapter implements Ps2{  private Usb usb;  public Adapter(Usb usb){  this.usb = usb;  }  @Override  public void isPs2() {  usb.isUsb();  }  } |

测试:

|  |
| --- |
| package adaptor;  public class Test {  public static void main(String args[]){  Ps2 p = new Adapter(new UsbImpl());  p.isPs2();  }  } |

# 4.接口适配器

原理：通过抽象类来实现适配，这种适配稍别于上面所述的适配。

当存在这样一个接口，其中定义了N多的方法，而我们现在却只想使用其中的一个到几个方法，如果我们直接实现接口，那么我们要对所有的方法进行实现，哪怕我们仅仅是对不需要的方法进行置空（只写一对大括号，不做具体方法实现）也会导致这个类变得臃肿，调用也不方便，这时我们可以使用一个抽象类作为中间件，即适配器，用这个抽象类实现接口，而在抽象类中所有的方法都进行置空，那么我们在创建抽象类的继承类，而且重写我们需要使用的那几个方法即可。

接口A:

|  |
| --- |
| package adaptor;  public interface A {  void a();  void b();  void c();  } |

适配器:

|  |
| --- |
| package adaptor;  public class Adapter implements A{  @Override  public void a() {}  @Override  public void b() {}  @Override  public void c() {}  } |

实现类:

|  |
| --- |
| package adaptor;  public class AImpl extends Adapter{  public void a(){  System.out.println("方法a");  }  public void b(){  System.out.println("方法b");  }  } |

测试:

|  |
| --- |
| A a = new AImpl();  a.a();  a.b(); |

# 5.使用场景

类适配器与对象适配器的使用场景一致，仅仅是实现手段稍有区别，二者主要用于如下场景：

　　（1）想要使用一个已经存在的类，但是它却不符合现有的接口规范，导致无法直接去访问，这时创建一个适配器就能间接去访问这个类中的方法。

　　（2）我们有一个类，想将其设计为可重用的类（可被多处访问），我们可以创建适配器来将这个类来适配其他没有提供合适接口的类。

　　以上两个场景其实就是从两个角度来描述一类问题，那就是要访问的方法不在合适的接口里，一个从接口出发（被访问），一个从访问出发（主动访问）。

接口适配器使用场景：

　　（1）想要使用接口中的某个或某些方法，但是接口中有太多方法，我们要使用时必须实现接口并实现其中的所有方法，可以使用抽象类来实现接口，并不对方法进行实现（仅置空），然后我们再继承这个抽象类来通过重写想用的方法的方式来实现。这个抽象类就是适配器。