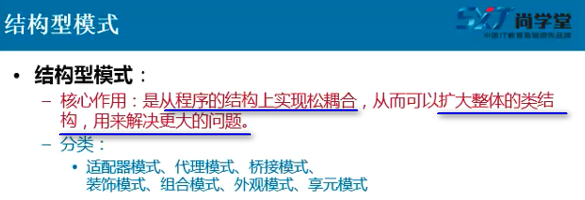
之前是创造者模式

现在是结构型模式 结构型模型对程序的整体性结构起到解耦的作用 对程序的升级和维护起到很大的作用

学习设计模式 是为了软件项目不断地升级 升级就用到了设计模式 如果软件遵循比较好的设计模式 对软件以后的扩展非常有好处



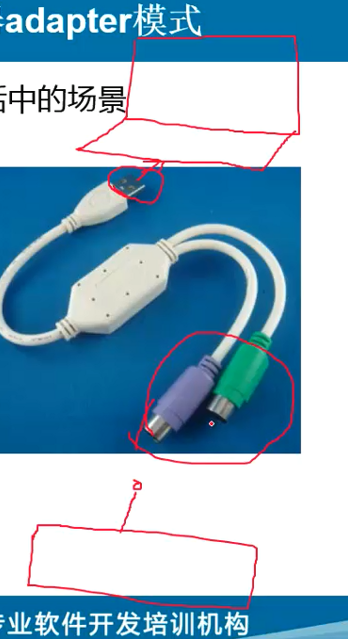
开源的产品中用的非常多

====== 适配器模式



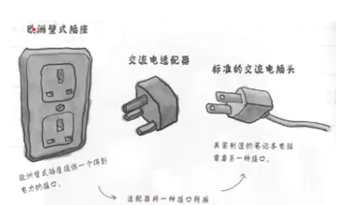
实际生活中，一边是链接USB接口 另一个是PC串口

一个键盘是Ps2接口 但是笔记本没有这种接口 只有usb接口

 也就是在键盘上和需求不一样 可以打字的基本需求 重买一个买不起 但是 花低成本买一个转接口是没有问题的

通过这个转接口 把usb和ps2适配器起来

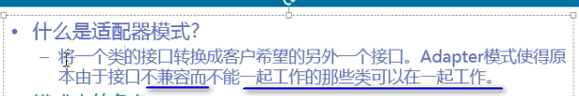
适配器模式也是起到这样的效果

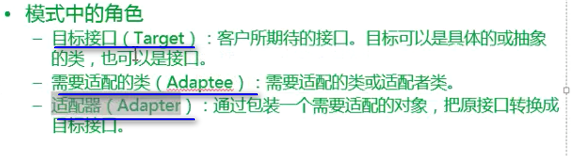


这样实现一个转接

非常多见

适配器模式就是用来解决这样的问题的



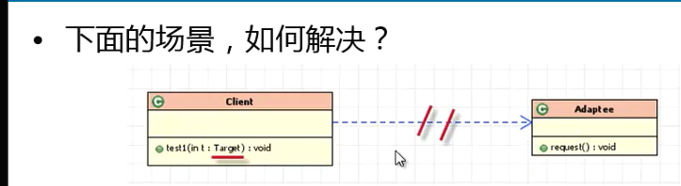
 --- 分工了

键盘就是需要适配的类 可以打字了 有这样的功能了

适配器就是这个产品本身 的ps2接口

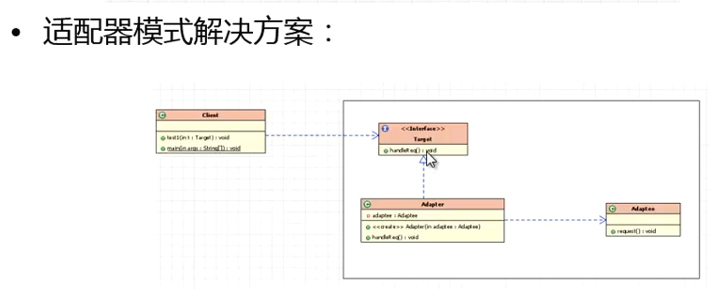
目标接口就是usb接口

大概就是这样的三个角色



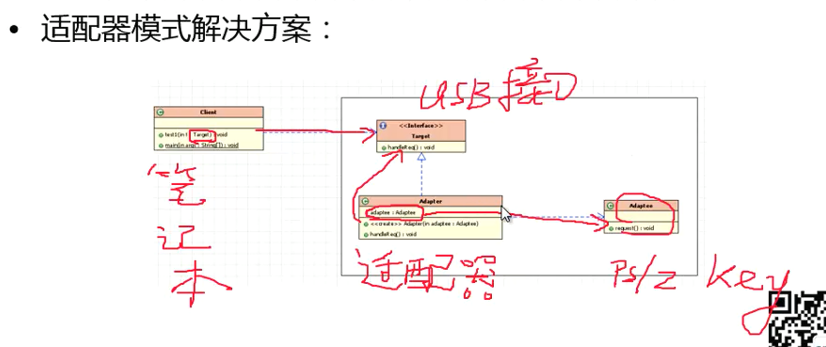
我的这个客户现在想使用这个Adaptee ---- client无法使用

现在增加这样几个类



Client直接和适配器打交道 不和Adaptee打交道

【谁是Client 谁是Adaptee 就看是谁使用谁 A使用B, B有自己的功能 ---- A就是Client B就是Adaptee --- 这里面是电脑使用键盘 键盘就是Adaptee】



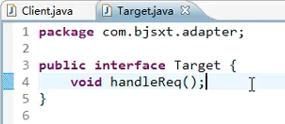
写代码

建立一个新的包



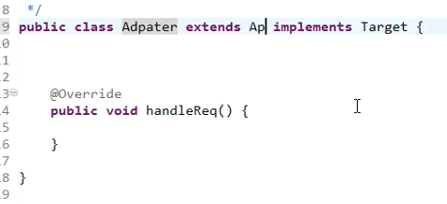
这就对应一个键盘可以打字

建立Target方法



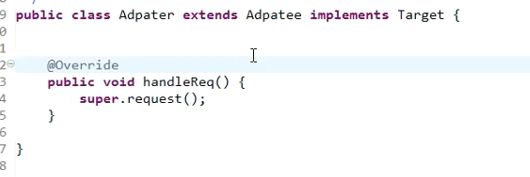
我们这个客户端就调用了这个Target接口

Target需要一个实现类 这个实现类就是Adapter

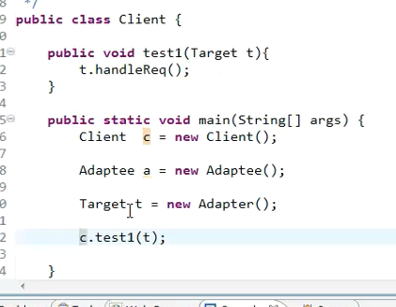


这个时候Adapter要关联这个Adaptee 有两种方式 一种是继承 另一种是聚合

现在继承一次



现在有一个Client类 相当于笔记本



需要的是Target 实际上调用的是Adaptee

Client就能调用了



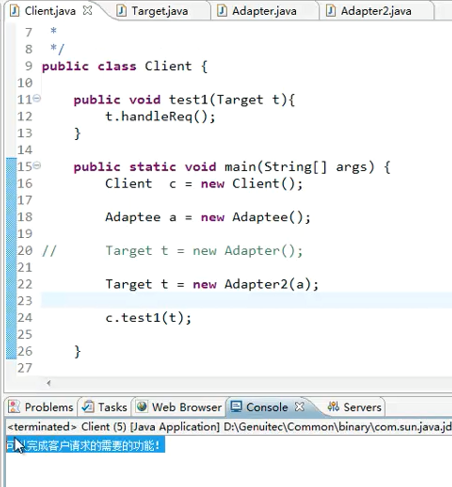
这里面问题 就是继承 Java只有单继承

如果这个类要继承其他类 就完蛋了 这种类适配器不好用

现在使用聚合 就是对象适配器

构建一个Adapter2



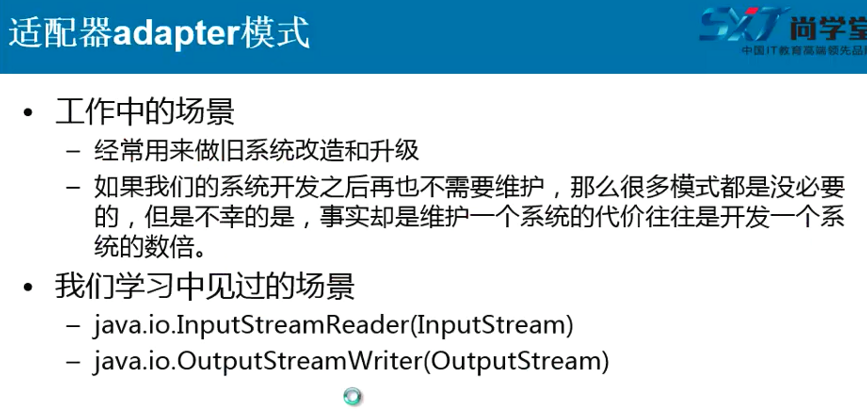


我们这PPT上面写的就构建完成了

======

类适配器

对象适配器



旧的系统升级会使用大量的升级模式 使用者就是2.0系统 要用1.0的功能

JDK非常经典的场景使用了适配器模式



我要使用字符流 但是 只有字节流 这样 InputStreamReader就是一个适配器

简单画一个

