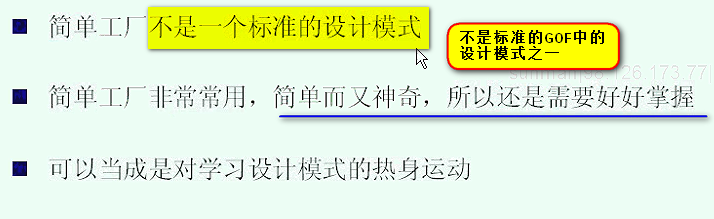
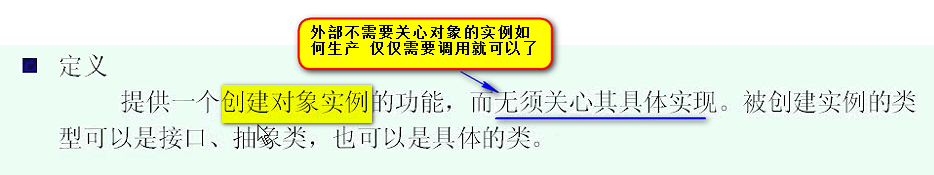
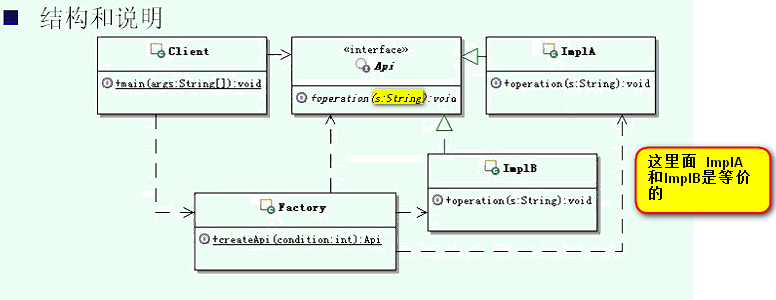
简单工厂设计模式

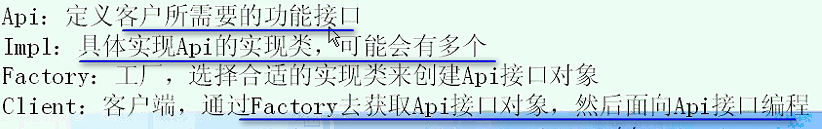






这样 简单工厂可以创建各种类型的





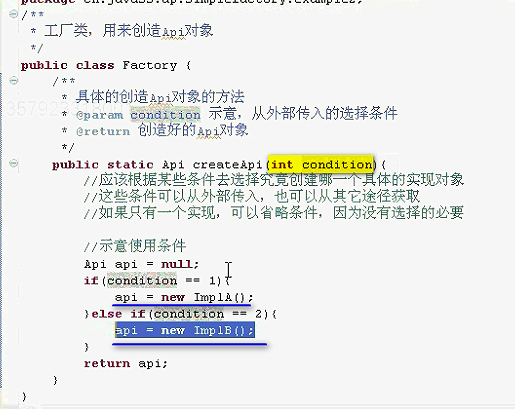
看完这个结构之后 看一下参考示例代码



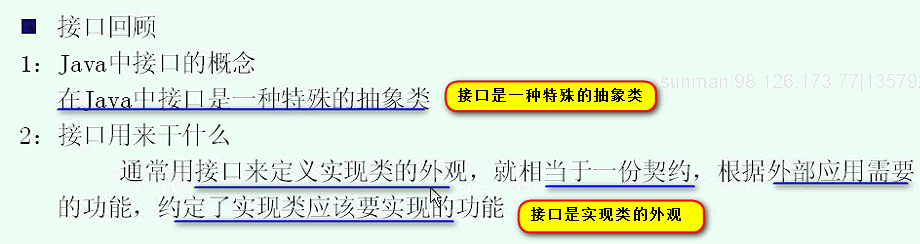
有两个实现







============ 回顾 =================

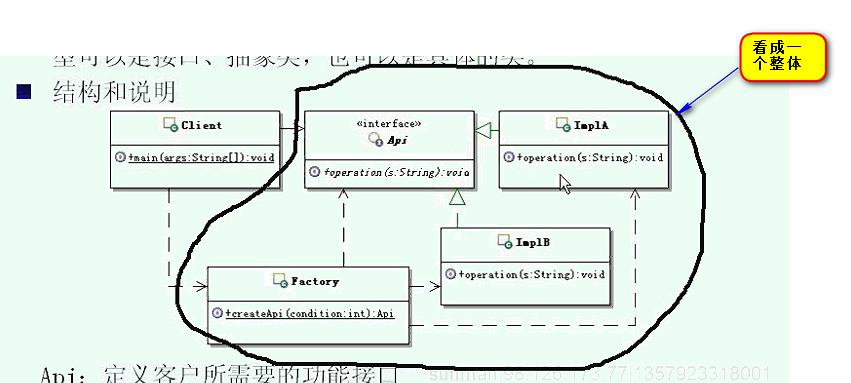




封装和隔离 看成两个词

封装 指的是对**隔离体行为**的封装 或者 **职责**的封装 ----- 也就是**通过接口 封装了这个隔离体的功能**

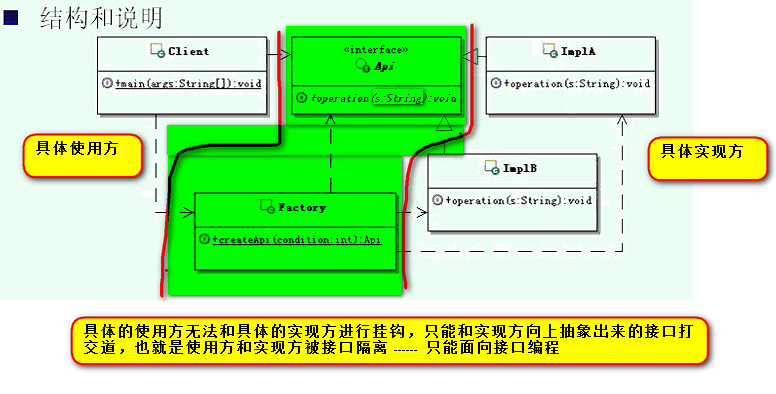
如果把 client以外 当成一个模块



接口就是**封装**了隔离体的职责或者功能（行文） ------ 从这个图上面看 这个黑色框住的就是一个隔离体，这个接口API就是从client角度来看，能看到的这个隔离体有哪些功能。

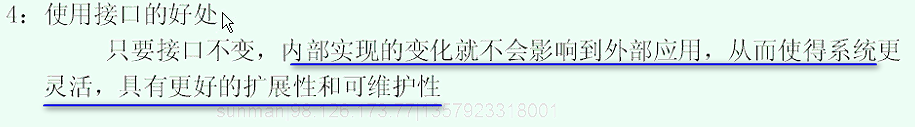
-------------这个是封装的意思

**隔离**的意思：还是这个图 client和具体的实现没有联系 直接使用的都是接口 不是和实现直接发生联系 ===== 这样 client和具体实现就被接口分离开了。



【总结一下 ------- 接口**对隔离体的职责或者行为进行**封装之后，接口就可以在功能上全全负责 ---- 这样接口通过某种方式就取代了具体实现，具体使用的人只和接口打交道 ------- 封装的功能进一步 就是隔离 隔离了**具体调用者和实现者的相互联系**】

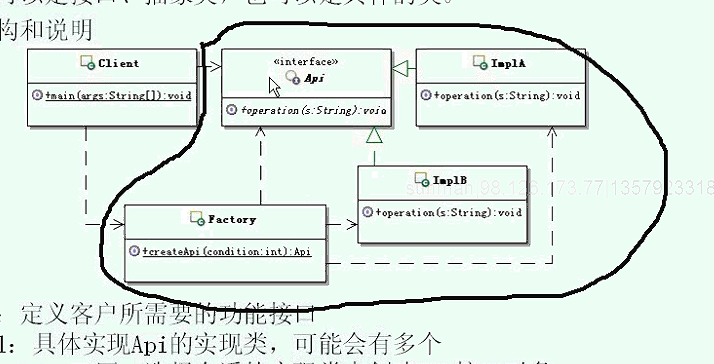
【再进一步说 就是 你不是把我和别人隔离了么？好 我同意 但是 你必须给我标记好 我有啥内容 ----- 接口标记好之后 就是封装 前面就是隔离】



外部就看到了接口 没有看到内部的实现

**接口是系统可插拔性的保证**

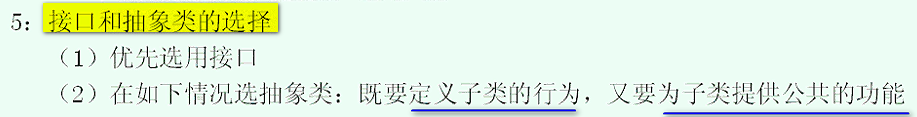
解释如下：



用着用着 说想新加一种实现 ------ ImplC ---- 插入了一段功能 ---- 可以插入

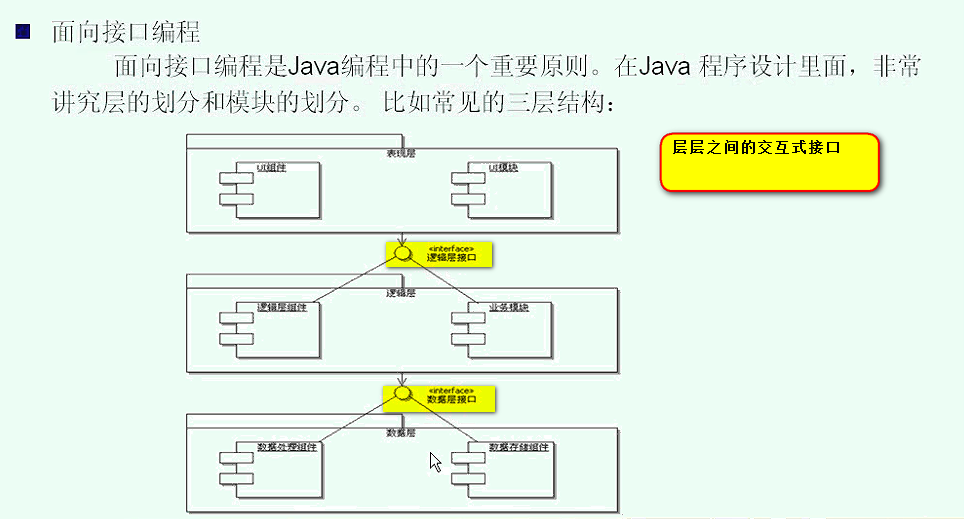
用着用着 觉得ImplA没用了 拿掉 ------ 可以拔掉

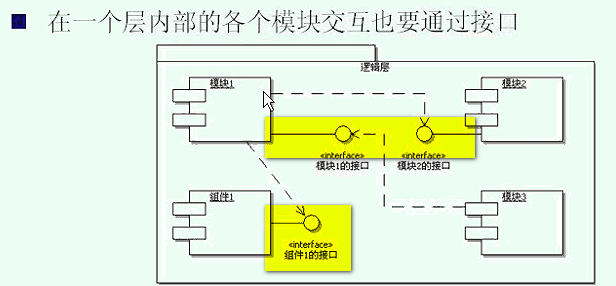
但是 这种插拔对client没有任何影响！！！！



面向接口编程 是Java中非常重要的编程原则

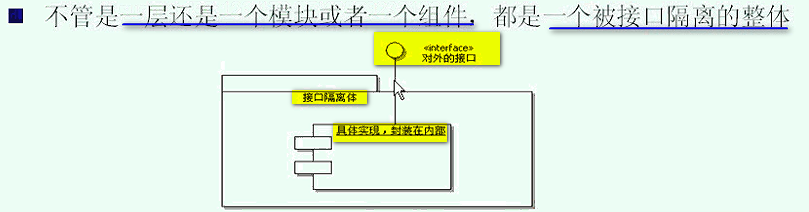
Java中非常讲究分层和模块的开发

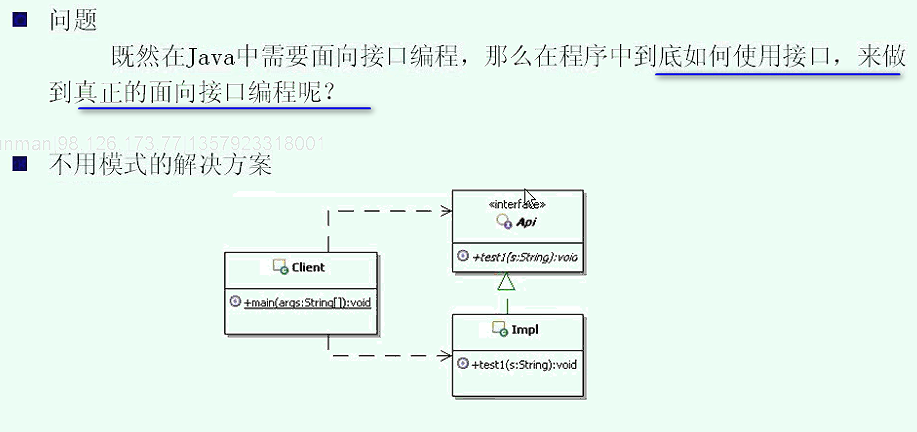




【虚线就是依赖 使用 ---- 这个图中就可以看到 模块1使用了模块2的接口和组件1的接口======= 模块3使用了模块1的接口 而不是直接和模块1打交道】

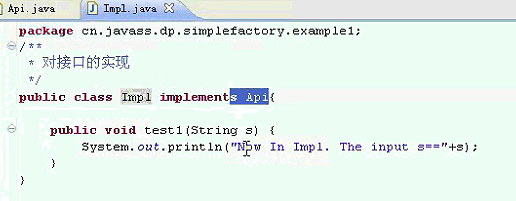
模块1看模块2 没有看到2中其他的功能 只能是因为模块2的接口对其进行了封装隔离





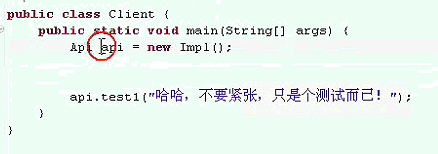
这个时候的代码示例





这时候 客户端用法

面向接口编程

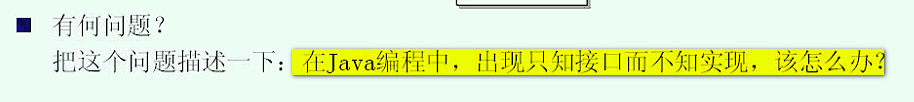


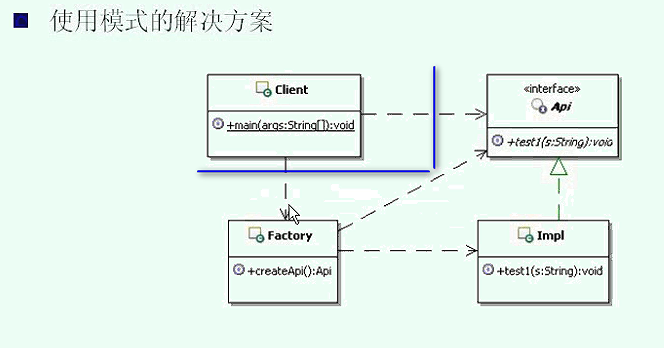
这个是真真的面向接口编程呢？ 这个不能！

如何得到接口的？ New Impl() ----问题就在这 Impl就是具体的实现类 ----- 这样client不但是知道了你的接口和你的内部实现 ------ 刚才说了接口是封装隔离的 ------ 这样 既没有封装住 也没有隔离开 ======= 这里面接口的优势全部浪费掉了

下面就是不用模式的解决方案的问题

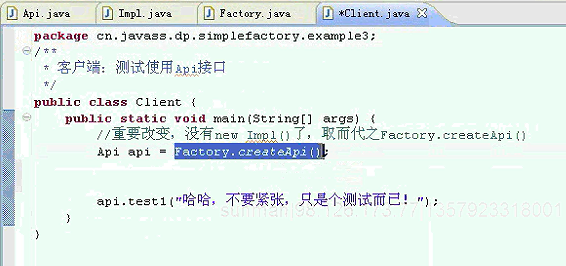
------ 只知道接口 不知道实现该怎么办？



 这次的接口的优势全部体现出来了



客户端的写法：



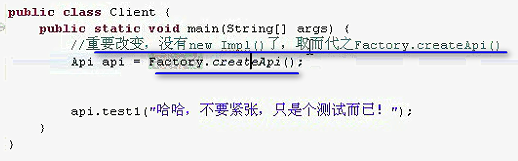
这个就面向对象编程了

从客户端的角度 仅仅找到了一个工厂 就得到了一个接口对象 但是根本不知道是谁实现的

这样 client和实现就被接口隔离开 体现了隔离

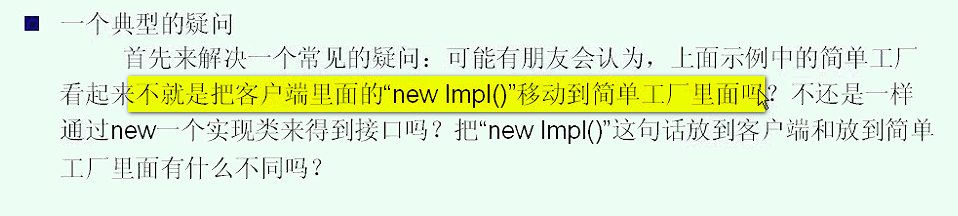
====== 使用了简单工厂 真正的面向的接口就没有问题了

 之后仅仅用了多态 其他接口的优势都没有体现出来

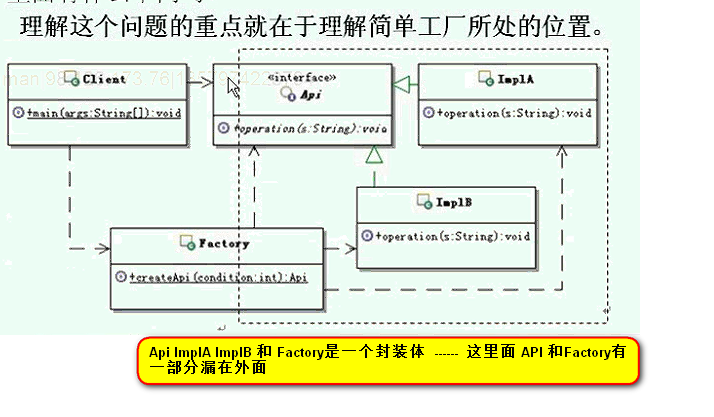


看起来就是把new Impl();搬到了工厂Factory中 ------ 这岂不是换汤不换药？

要理解这个问题 先来典型的疑问：







漏在外面的Factory和API就是因为要提供给客户端Client使用

简单工厂不是位于客户端 而是封装体的内部

如果在客户端写 接口 xxx = new xxx() ---- client不但是知道接口 还知道实现 又和具体的实现发生了耦合

但是 放到了工厂之后，工厂不会对外说这些，仅仅需要给客户端返回一个类型就可以了 ----- 客户端并不知道工厂到底是返回了ImplA还是ImplB

搬到工厂里面，客户端真正实现了和具体实现的隔离（也就是解耦合）

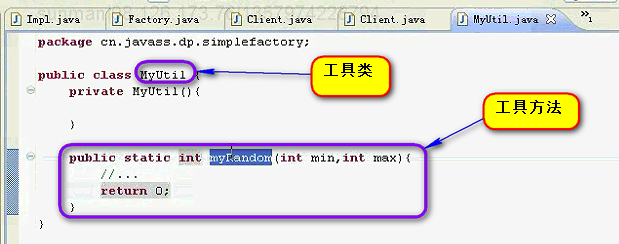
【所以 接口也可以叫封装解耦 封装隔离】



工具类：Util

构造方法私有化 不让new ----- 然后体外提供静态方法

下面是一个例子

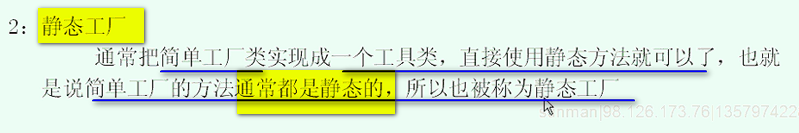


工具类和工具方法之间不需要特别关系的 ---- 就是为了帮助你实现单独的功能

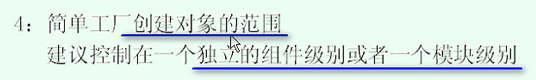
一般一个工具类中有多个工具方法

**放到简单工厂中 ----- 就是通常把一个工厂类实现为一个工具类【Util和工厂类的关系】**

 一般最好把Factory私有化 ---- 不需要多个工厂 这个根本不用类实例

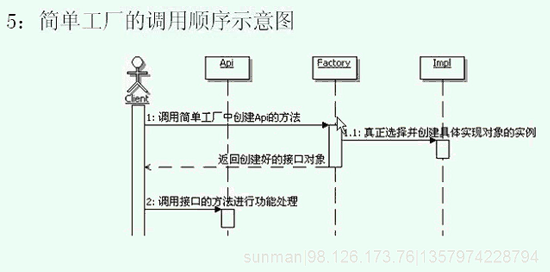


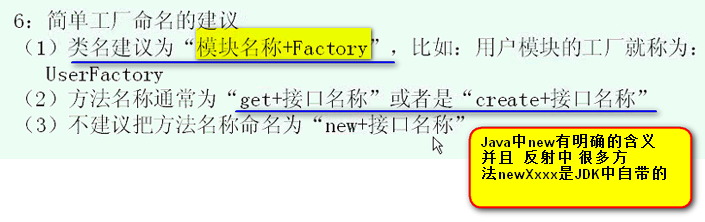




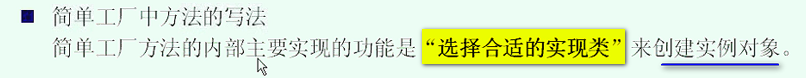
**如果有一个封装体 就应该有一个简单工厂 ====== 这是因为 既然是一个封装体 那就要有一个接口 ---- 有了接口 为了彻底面向接口编程 ----- 简单工厂自然出现**

【范围就是独立的封装体就可以】

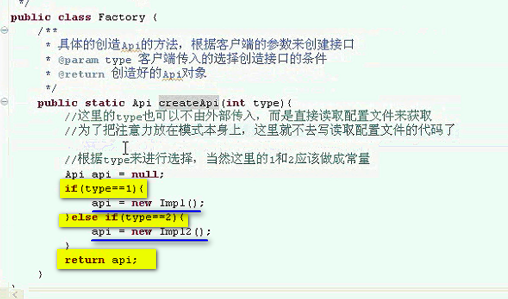




用get或者create增加方法名称



现在这个关键就落在了选择上面



选择就涉及到 如何进行选择 ----- 这个就是选择的条件或者参数

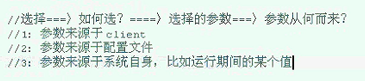


归结为

\*\*参数来源于客户端 client --- 从模块外部来

\*\*参数来源于配置文件

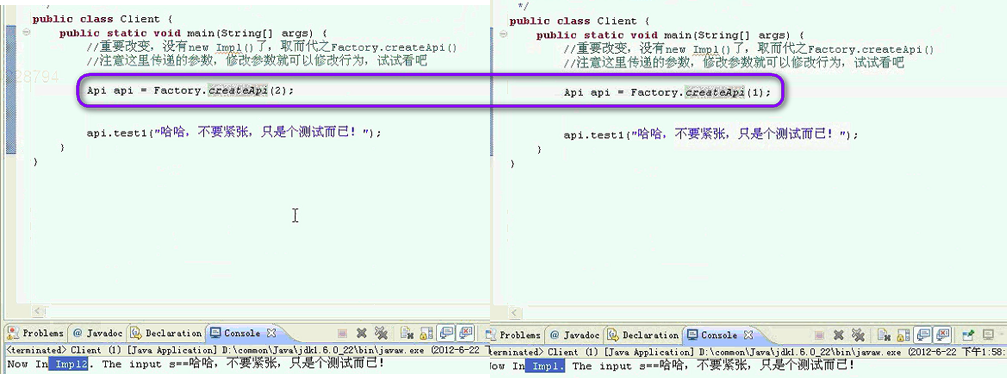
\*\*参数来源于系统自身 ----- 比如运行期中的某个值 ---- 这个值就在内存里面



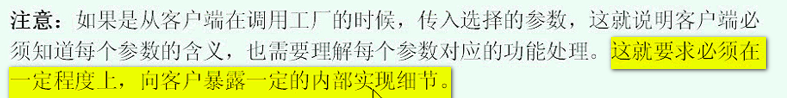
=======

\*\*参数来源于client

客户端调用的时候，要进行传参

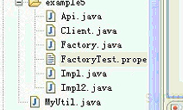


这种做法有问题 ----- 客户端要传入参数 这样客户端就要知道参数的含义 ------ 我还知道你的参数有多少个选项 ---- 这其实内部的实现暴露给了客户端 ----- 知道有了两种 这样一来 并不是很好



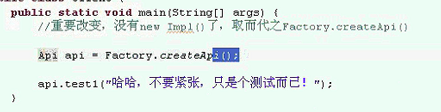
如果不使用IOC DI容器的化 ---- 采用可以配置的简单工厂 ==== 也就是参数的第二种来源 参数来源于配置文件 ---- 不是客户端来选择 而是配置文件传进来 ----- 因为配置文件是属于模块这边 就是属于封装体这边的

到底该怎么来实现？

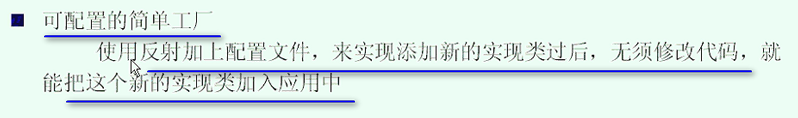


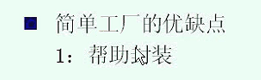






这种做法 有更强的优势 ----- 真正实现了开闭原则





这样针对客户端 ----- 真正面向接口编程

 客户端和具体实现类的解耦

解除了耦合 这样 客户端和实现类都可以独立的变化了

======

 因为是选择实现么 所以 客户端必须知道哪些参数对应哪些实现

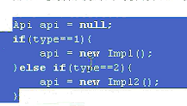
 ----- 因为工厂方法构造私有化 ----- 但是 工厂一般很少扩 所以 没所谓

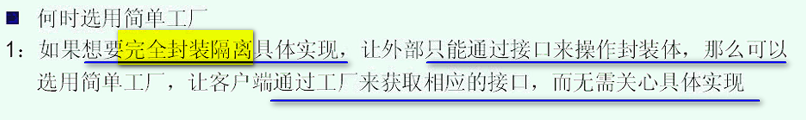


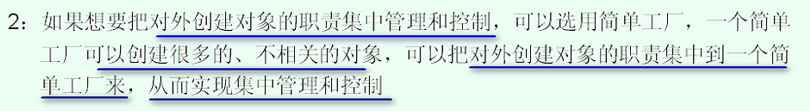
实现是先做好的，简单工厂就是选 一选 就需要参数 ----- 来源客户端 要么来源于配置文件 还有就是来源于程序的运行期间

总而言之 简单工厂的重心是选择 而不是实现

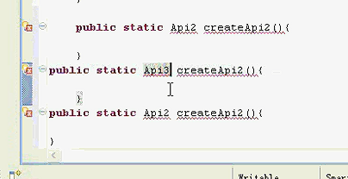
在工厂里面







如果创建多个接口 都可以集中到这里面来



一个简单工厂 是可以创建很多的不相关的对象 --- 把对外创建的职责 都集中到一个简单工厂里面来

这样 得到了简单工厂 就得到需要的内容 ----- 建议选用简单工厂