接下来我们来看到最后一种建造者模式，**原型模式**

这种模式的核心需求是**通过复制已有对象来创建对象而不是通过类构造函数**。核心思想就是**克隆对象**

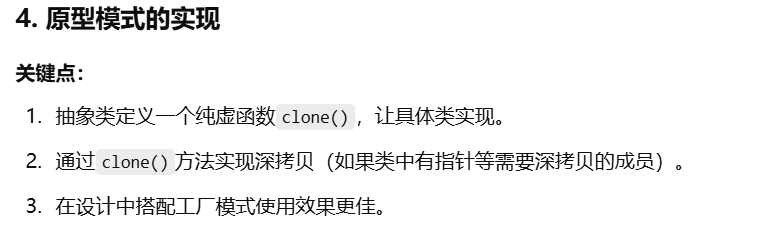
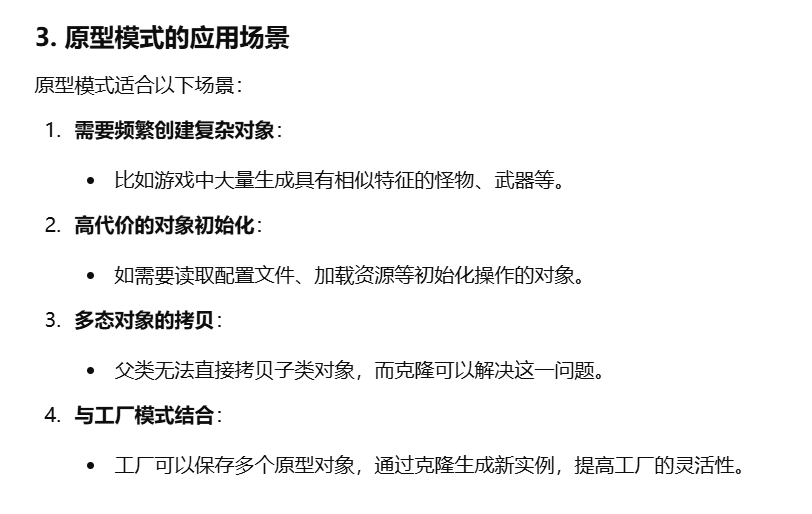
其实吧，你看到这上面的时候你其实可能会疑惑这种模式存在的意义，毕竟即使复制已有对象和通过构造函数来构造对象存在一定的差异，但是也不会存在多么离谱的区别，所以这种模式的存在意义到底在哪。

首先我们知道在一个类的默认设计中，编译器都会默认提供一个拷贝构造函数，即使这个拷贝构造函数只能提供浅拷贝的功能。也就是说，要在一个类的外面来拷贝一个类其实是相对简单的，最多就是添加一个拷贝三件套即可。

我们来深入探讨一下，我们如果使用多态下的类对象拷贝，在这种情况下，我们在类的外部对于任何一个同层次的类的拷贝都是简单的(这里指的是用相应的类名称指针来拷贝对应的类对象)。但是，我们来考虑下使用父类指针来拷贝下子类对象的情况呢，这种是我们的拷贝构造函数所无法提供的，而这个情况就是我们的原型模式的一种运用场景了。

这里我们讨论下这个原型模式的应用场景。相对来说，原型模式反而是相对来说应用广泛的一个模式了，毕竟这个就是对于扩展方法的一种扩展。只需要一个clone方法即该类对应的拷贝构造函数三件套。需要注意的是，我们在使用这种原型模式时，需要考虑设计上的效率问题，如果我们要拷贝的类对象是直接构造起来比较费时的话，那原型模式能够很好的提高效率。

在接下去之前我们需要解释下原型模式的实际实现，本质就是在类内使用拷贝构造函数来实现拷贝。那么我们就需要考虑下我们的拷贝构造和一般的构造的效率之间区别，在一般情况下，这俩种的情况是差不多的，甚至于在大多数情况下，使用无参构造来实现是更加简便的。但是吧，就比如我们前面学到的抽象工厂模式和生产者模式，这些要产生的类是一系列相对来说更加复杂的，都是无法通过简单的无参构造实现的，还需要一系列的额外函数来添加属性的。这时候原型模式的作用就体现出来了，其绕开了构建一个复杂对象时需要的流程，直接对现有的对象进行复制，获取一个可以直接使用的对象。即使我们后续需要对这个对象进行修改，也只用去修改对应的成员即可，跟普通的构造出来的对象没有什么大区别，但是在构造的消耗上优化了很多。



而且，这是一个可以简单的嵌入到其他模式中的模式，这也就体现了它应用的广泛性，毕竟它应用简单而且能优化性能。