# 第1条：考虑使用静态工厂方法代替构造器

静态工厂方法相较于构造器有几个优势：

1. **静态工厂有名字。**

这样，就可以在具有相同的参数列表的时候，通过名字区分他们的区别，而构造器如果多个构造器之间参数列表相同，只能通过调整参数类型的顺序来进行区分，通常这会给调用这些构造器的开发人员造成很大的困扰，容易调用错误的构造器，而使用可以用名字来区分的静态工厂方法，就可以有效的区分这些静态的“代替构造器”之间的区别。

1. **静态工厂方法不必每次都创建一个对象。**

这一点主要是在单例模式中的应用。

1. **静态工厂可以直接返回原类型的子类型对象。**

这样做可以增加返回对象的灵活性。这种灵活性的应用是，API可以返回对象，而且不会使对象的类变成共有。以这种方式隐藏实现类会使API变得非常简洁。即客户端程序可以直接通过接口来引用被返回的类，而不是通过它的实现类来引用被返回的对象。

**（虽然，在java7之前我们不能在接口中定义静态方法，但是在java8中，由于引入了default方法，因此静态方法也是被允许的了，因此，可以直接在接口中定义静态方法，并返回接口的实现类。）**

另外，返回的子类对象的类型可以根据参数进行变化，只要是子类就可以。为了提升软件的可维护性和性能，返回对象的类型也可能随着发行版本的不同而不同。

# 第2条：遇到多个构造器参数时要考虑用构建器

在大量可选参数的情况下，静态工厂方法和构造器都不能很好的扩展。考虑，如果一个JavaBean有很多属性，其中某些属性是必须的，而另外一部分属性是非必须的，那么使用构造器的话，就需要为客户端代码提供多种参数列表的构造器。这种方式叫做“重叠构造器”。很明显，这种重叠构造器的方式可读性非常差，客户端必须知道每个构造器的用途才可以调用正确的构造器，一长串类型相同的参数可能导致微妙的差别，如果客户端不小心颠倒了其中两个参数的顺序，编译器也不会报错，这样的方式是糟糕的。因此，重叠构造器的方式可行，但是当有许多参数的时候，客户端代码会很难编写，并且难以阅读。

遇到许多构造器参数的时候，另一种解决方式是JavaBeans模式，也就是我们最常用的setter赋值，即通过无参构造来创建一个对象，然后通过setter来为属性赋值。但是JavaBeans也有自身的缺点：因为构造过程并不是一气呵成，而是被分到几个调用中，在构造过程中JavaBean可能处于不一致的状态。类无法仅仅通过校验构造器参数的有效性来保证一致性。这样的JavaBean可能会产生线程安全问题，可能会需要程序员付出额外的努力来确保它的线程安全。

第三种方式，既可以保证像重叠构造器模式那样的安全性，也能保证像JavaBeans模式那样好的可读性，就是Builder模式（建造者模式）的一种形式。使用方法如下：

**不直接生成想要的对象，而是让客户端利用所有必要的参数调用构造器（或静态工厂），得到一个builder对象。然后客户端在builder对象上调用类似于setter的方法，来设置每个相关的可选参数。最后，客户端调用无参的builder方法来生成不可变的对象。这个builder是它构建的类的静态成员类。**

# 第3条：用私有构造器或枚举类型强化Singleton属性

# 第4条：通过私有构造器强化不可实例化的能力

对于只包含静态方法、静态域的类（比如工具类等），

# 第5条：避免创建不必要的对象

# 第6条：消除过期的对象引用

# 第7条：避免使用终结方法

# 第8条：覆盖equals时请遵守通用约定

# 第9条：覆盖equals时总要覆盖hashCode

# 第10条：始终要覆盖toString

# 第11条：谨慎地覆盖clone

# 第12条：考虑实现Comparable接口

compareTo()方法是Comparable接口中唯一的方法。compareTo()方法不但允许进行简单的等同性比较，而且允许执行顺序比较，它与Object的equals方法具有相似的特征，它还是个泛型。**类实现了Comparable接口，就表明它的实例具有内在的排序关系**。包含了实现Comparable接口对象的数组进行排序可以如下这样：

Arrays.sort(array);

compareTo方法的通用约定与equals方法的相似：将这个对象与指定的对象进行比较。当该对象小于指定对象，则返回负整数；等于返回0；大于，则返回正整数。如果由于指定对象的类型而无法与该对象进行比较，则抛出ClassCastException异常。

# 第13条：使类和成员的可访问性最小化

# 第14条：在公有类中使用访问方法而非公有域

# 第15条：使可变性最小化

# 第16条：复合优先于继承

# 第17条：要么为继承而设计，并提供文档说明，要么就禁止继承

# 第18条：接口优先于抽象类

# 第19条：接口只用于定义类型

# 第20条：类层次优于标签类

# 第21条：用函数对象表示策略

# 第22条：优先考虑静态成员类

# 第23条：请不要在新代码中使用原生态类型

# 第24条：消除非受检警告

# 第25条：列表优先于数组

# 第26条：优先考虑泛型

# 第27条：优先考虑泛型方法

# 第28条：利用有限制通配符来提升API的灵活性

# 第29条：优先考虑类型安全的异构容器

# 第30条：用enum代替int常量

# 第31条：用实例域代替序数

# 第32条：用EnumSet代替位域

# 第33条：用EnumMap代替序数索引

# 第34条：用接口模拟可伸缩的枚举

# 第35条：注解优先于命名模式

# 第36条：坚持使用Override注解

# 第37条：用标记接口定义类型

# 第38条：检查参数的有效性

# 第39条：必要时进行保护性拷贝

# 第40条：谨慎设计方法签名

# 第41条：慎用重载

# 第42条：慎用可变参数

# 第43条：返回零长度的数组或者集合，而不是null

# 第44条：为所有导出的API元素编写文档注释

# 第45条：将局部变量的作用域最小化

# 第46条：for-each循环优先于传统的for循环

# 第47条：了解和使用类库

# 第48条：如果需要精确的答案，请避免使用float和double

# 第49条：基本类型优先于装箱基本类型

# 第50条：如果其他类型更适合，则尽量避免使用字符串

# 第51条：当心字符串连接的性能

# 第52条：通过接口引用对象

# 第53条：接口优先于反射机制

# 第54条：谨慎地使用本地方法

# 第55条：谨慎地进行优化

# 第56条：遵守普遍接受的命名惯例

# 第57条：只针对异常的情况才使用异常

# 第58条：对可恢复的情况使用受检异常，对编程错误使用运行时异常

# 第59条：避免不必要地使用受检异常

# 第60条：优先使用标准的异常

# 第61条：抛出与抽象相对应的异常

# 第62条：每个方法抛出的异常都要有文档

# 第63条：在细节消息中包含能捕获失败的信息

# 第64条：努力使失败保持原子性

# 第65条：不要忽略异常

# 第66条：同步访问共享的可变数据

# 第67条：避免过度同步

# 第68条：executor和task优先于线程

# 第69条：并发工具优先于wait和notify

# 第70条：线程安全性的文档化

# 第71条：慎用延迟初始化

# 第72条：不要依赖于线程调度器

# 第73条：避免使用线程组

# 第74条：谨慎地使用Serializable接口

# 第75条：考虑使用自定义的序列化形式

# 第76条：保护性地编写readObject方法

# 第77条：对于实例控制，枚举类型优先于readResolve

# 第78条：考虑用序列化代理代替序列化实例