## 第二章 创建和销毁对象

### 1.考虑用静态工厂方法代替构造器

优势：

1. 静态工厂方法有名称！
2. 不必每次调用的时候，都创建一个新对象！
3. 可以返回原类型的任何子类型
4. 在创建参数化类型实例的时候，使代码更加简洁

缺点：

1. 类如果不含公有的或者受保护的构造器，就不能被子类化

（对于公有的静态工厂所返回的非公有类，也同样如此）

1. 与其他静态方法实际没有区别

valueOf：方法返回的实例和参数具有相同的值，实际上是类型转换

of：valueOf的一种更简洁的替代

getInstance：返回实例

newInstance：可以确保每一次获取的实例都不同！

getType：像getInstance一样，表示工厂方法返回的对象类型

newType：像newInstance一样，表示工厂方法返回的对象类型

### 2.遇到多个构造器参数的时候，要考虑用构建器

builder构建者模式

### 3.用私有构造器或者枚举强化Singleton属性

1. 在私有构造器中，加入创建第二个实例的时候，抛出异常，可以避免通过反射强行破坏单例
2. 通过重写readResolve方法，可以避免，反序列化的时候，破坏单例
3. 枚举天然就支持！

### 4.通过私有构造器强化不可实例化的能力

当一些工具类，不希望被实例化的时候，直接private构造方法，并且在里面抛异常

### 5.避免创建不必要的对象

1. 注意自动装箱的对象消耗
2. 注意一些重量级对象的创建

### 6.消除过期对象的引用

1. 注意数组对象的引用，及时置空
2. WeakHashMap
3. 监听和回调

### 7.避免使用终结方法

1. finalizer的调用时机不确定
2. 推荐显示的创建终止方法，与try-finally结合
3. “安全网”：在类的方法结束的时候，手动。比如FileInputStream，FileOutputStream，Connection，Timer
4. 本地Native C/C++对象，java无法回收，需要手动处理！
5. 创建匿名内部类，重写finalize方法，来达到释放资源的效果！（如果需要把终结方法与公有的非final类关联，则使用终结方法守卫者，以确保即使子类的终结方法没有调用，也能释放资源）

public class Foo {

private final Object finalizerGuardian = new Object(){

@Override

public void finalize() throws Throwable {

super.finalize();

// 释放Foo的资源

// 当System.gc的时候，Foo会被回收

// finalizerGuardian内部匿名类的finalize方法，也会被调用。

// 可以释放Foo的资源

}

};

// doSomeThing

}

这样的话，即使Foo的子类，没有手动释放资源。Foo的匿名内部类也可以释放资源！

## 第三章 对于所有对象都通用的方法

### 1.覆盖equals时遵守的通用约定