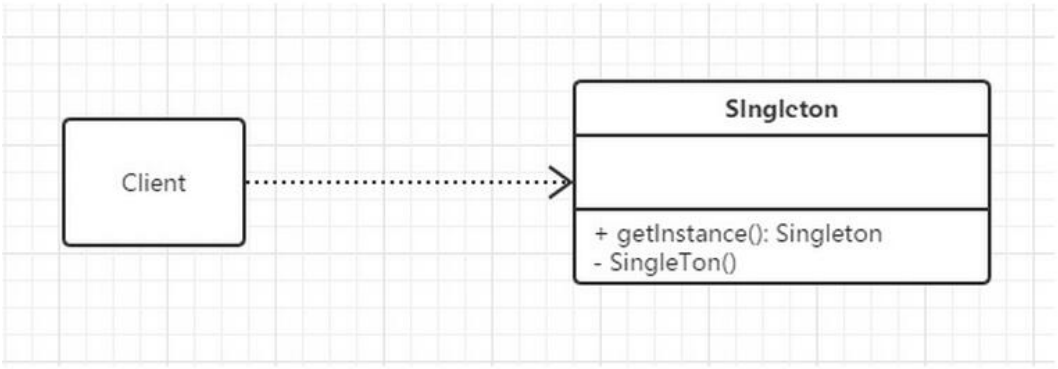
auth ： biwh　2018/08/02 14:04

生活中的单例模式

单例模式：就像是大力神杯，每4年就有一个冠军，但是却只有一个大力神杯，每个新冠军会把自己国家刻到神杯的底座上，就像是单例模式，只有一个实例，但是可以被所有获胜的国家队调用，这就是单例模式

在Spring中创建的Bean实例默认都是单例模式存在的。



MainTest.java
487B

单例模式的关键点

通过上面的UML图，我们可以看出单例模式的特点如下：

1、构造器是私有的，不允许外部的类调用构造器

2、提供一个供外部访问的方法，该方法返回单例类的实例

根据单例对象的创建时机不同，可以分为饿汉模式和懒汉模式。

饿汉是指在类加载的时候，就创建了对象。但是创建对象有时比较消耗资源，会造成类加载很慢，但是优点是获取对象的速度很快，因为早已经创建好了嘛。懒汉就是相对饿汉而言，在需要返回单例对象的时候，在创建对象，类加载的时候，并不初始化，好处与缺点也不言而喻

根据是否实现线程安全，可以分为普通的懒汉模式这种线程不安全的写法，和饿汉模式，双重检查锁的懒汉模式，以及通过静态内部类或者枚举类等实现的线程安全的写法。

```
public class SimpleSingle {
   public static SimpleSingle SimpleSingle;

   private SimpleSingle(){
   }
   public synchronized static SimpleSingle getInstence(){
      if(SimpleSingle==null){
         SimpleSingle = new SimpleSingle();
      }
      return SimpleSingle;
   }
}
```

HungarySingle.java
250B

首先，我们可以看出这是一个懒汉模式的实现。因为只有在getInstance的时候，才会真正创建单例的对象。(如果不加synchronized)但是为什么他是线程不安全的呢，是因为可能会有2个线程同时进入if (simpleSingleton == null)的判断，就是同时创建了simpleSingleton对象。

使用单例模式，可以使多个程序来公用一个对象，里面的方法和变量可以进行共享，通过synchronize来使线程安全，保证多个线程共用一个单例的对象

单例模式实际的操作步骤：

1.给DCLSingleton实例分配内存

2.调用DCLSingleton()的构造函数，初始化成员字段

3.将singleton对象指向分配的内存空间。

SimpleSingle.java
737B

饿汉的单例模式

饿汉写法是在类加载的时候，就完成了对象的初始化，类加载保证了他们天生是线程安全的通过类.方法或变量来调用这个对象的！

InnerSingle.java
280B

内部类实现单例模式

枚举类实现单例模式

使用的时候，直接通过EnumSingleton.SINGLETON.doSomethings()。枚举类天生特性是保证不会有两个实例，并且只有在第一次访问的时候才会被实例化，是懒加载的情况。

```
public enum EnumSingle {
    SINGLE;
   public void doSomeThing(){
   }
}
```

真的不会再次创建新的对象吗？

在常规调用单例类的getInstance（ ）方法的情况下，使用线程安全的写法确实不会创建新的对象，但是Java提供了很多奇特的技巧和使用，下面这些使用会破坏掉常规的单例。

- 反序列化
- 反射
- 克隆
- 分布式环境下，多个类加载器

在除了枚举实现单例模式的方法以外，其余所有方法碰到上述四种情况，都会重新创建对象。原因如下：

- 反序列化会调用一个特殊的readResolve()方法来创建新的对象。我们可以重写该方法，让他返回原来的instance，而不是重新创建一个。
- 反射会得到私有的构造函数，只能在构造函数中加一个判断，如果对象不为null,则扔出一个运行时异常，如果不这样，只有枚举能解决，因为枚举自带的特性。

- 克隆，因为直接拷贝的内存空间的内容，所以只有自己重写单例类的clone方法，如果不这样，也只有枚举能解决，因为枚举没有克隆方法。
- 多分布式环境，因为我们上述很多种单例的写法，都是依赖于类加载器的特性，但是static的作用只负责到类加载器，所以当工程中存在多个类加载器的时候，就会创建多个实例，这种通常就需要第三方库来解决。

什么时候用单例模式，用哪一种写法的单例模式

单例模式有两种比较适合的使用场景。

第一种是创建某个对象，需要的代价比较大，为了避免频繁的创建和销毁对象从而引起的对资源的浪费，会考虑使用单例模式。

第二种是这个对象必须只有一个，有多个会造成不可预估的错误，或者程序的混乱，比如只会有一个序号生成器，一个缓存等等。

针对使用的单例模式，如果需要理解的加载资源，就是用饿汉写法，在Android应用中，很多对象需要在启动的时候，立即就使用，比如启动时，需要拉取相机配置的类管理缩略图的cache类等等。如果不是立即需要，或者不是贯穿应用始终的，就不需要使用饿汉写法，可以考虑懒汉写法用（DCL或者静态内部类实现）这两种在一般情况下都不会出现问题。