java泛型设计

泛型的使用场景：
（1）编译时强类型检查
（2）避免类型强转： 在赋值的时候会编译报错（其实运行时泛型擦除，但是不会通过编译）
（3）实现通用算法：比如集合中的一些add、remove之类的算法

这里ArrayList是实现了 RondomAccess标记型接口，代表可以随机访问。

一、 泛型类设计

- 调用泛型类型
  其实就是利用泛型去声明

- 实例化泛型
  实例化就是给用泛型声明的变量赋值

- java7 Diamond语法
  就是 List<String> a = new ArrayList<>(); 后边的ArrayList不需要定义泛型

- 类型参数命名约定
  E：表示集合元素（Element)
  K：表示键
  V：表示值
  T：表示类型

泛型有界类型参数

1. 单界限
   extends
   但是运行时会擦除，都是Object，还是有一定瑕疵的
2. 多界限
   <T extends C & I & I2>

多界限泛型参数类型 extends 第一个类型允许是具体类（也可以是接口）
但是第二个或者更多参数必须是接口

3. 泛型方法和有界类型参数

增强通用性

知识点：Arrays.asList方法的强转 --> Arrays.<Integer>asList()

泛型通配符（? 的使用）

通配符也分为上界通配和完全通配

1.上、下界通配：

比如像 List的addAll方法 就使用了泛型的通配符。表示 可以合并更广泛（有一定上限）的数据。

通配符在代码编译上有些麻烦，常用语方法参数的抽象。

一个使用技巧

读取数据（生产者）使用extends

操作数据（消费者）使用super

2. 完全通配：

在运行时和非泛型通配可能有方法冲突 （都是Object）

但是完全通配更灵活的是可以逃过一定的类型检查的限制（比如List<?> 和 List<Object>）， 所以还是有使用场景的。

一个问题：当泛型定义为字段属性，那么可以通过字节码去反推这个泛型的类型。

但也只是泛型定义的字符串而已。这里可以看看Spring框架中的ResolvableType。如果泛型是在类型上，那么就可以直接通过反射去拿到类泛型。

泛型擦除

擦除是运行时的，编译生成的字节码还是有对应的泛型信息的。

java方法设计

具体的设计方法见demo

方法名称设计

方法参数设计

方法返回值设计