java接口设计

类和接口设计

课程中的一些问题：

（1）String a + String b 到底产生了几个对象

（2）String 是不可变的吗

（3）String内部是线程安全的吗

（4）java8新增的Executable接口

访问性设计

四种访问修饰符：

（1）public

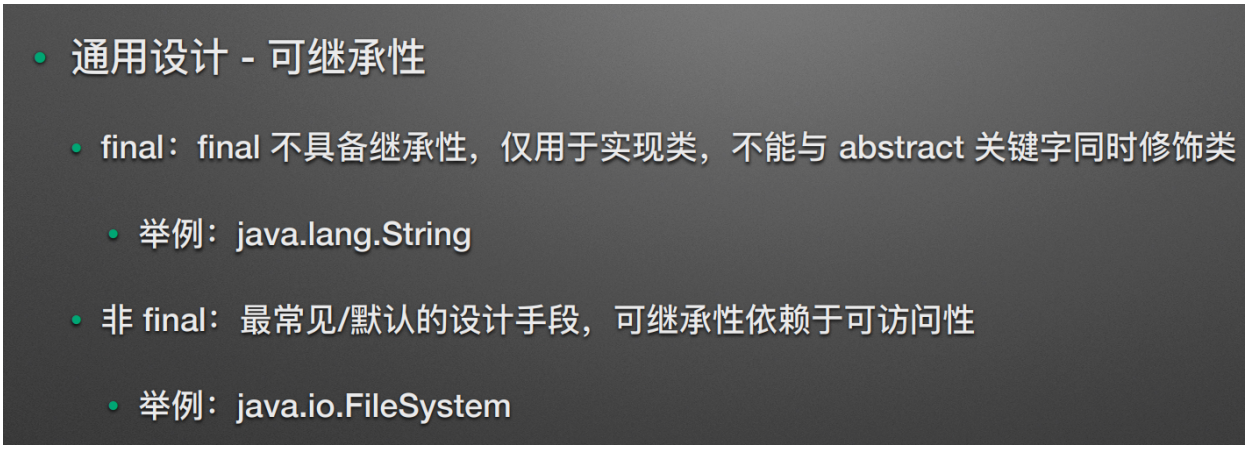
（2）protected 不能用来修饰最外层class， 但可以用在内置类

（3）default

（4）private 不能用来修饰最外层class

注意 FileSystem是default的， 主要用于内部

可继承性



String为啥设计成不可变的?

- 不变性主要体现在内部的 final char[] val变量
- 如果是可继承的，那么子类就可以覆写equals和hashCode方法，给方法带来困难。
- 为了一定的安全设计、效率的思考，final的话就可以常量优化（网上都有）
- jdk升级的时候，加入方法对子类的影响

String 对象到底是不是可变的?

- 设计上char[] 是final的，但是1.5之后的反射可以修改对象属性（不是类属性）~
- 通过反射（getDeclaredField）来拿到private的方法

怎么防止反射入侵

- 可以看反射的setAccesible方法中，有check RuntimePermission，如果设置了不可以反射获取，那么就不能通过反射去修改了

具体类设计

命名模式：

- 前缀
- 后缀

抽象类设计

- 接口通用实现（模板模式）
  Spring xxxTemplate
  ArrayList AbstractList
  HashMap AbstractMap
- 状态/行为继承
- 工具类
  Objects

常见的模式：

- 抽象程度介于类与接口之间（java8+通常可由接口替换）
- 以Abstract 或 Base 类名前缀
  - AbstractCollection
  - javax.sql.rowset.BaseRowSet

接口设计

上下游系统（组件）通讯、契约

API 这里其实为了一个强类型约束

RPC

常量定义

java中的标记型接口?

Saerializable

Cloneable

AutoCloseable

EventListener

这里问一下Cloneable的一个用处?

- 是一个标记接口，用于表示该类可以clone，否则会抛出异常
- Object中的clone方法 是protected的，通常会重写的时候扩充为public, 并且强制返回值为当前类型
- 深浅拷贝~
- clone的代价 比 new 对象() 操作要小，new的时候要运行的指令是很多的。

接口设计中的常见模式

（1）无状态（stateLess）

（2）完全抽象（< java8）

（3）局部抽象（java8+）

（4）单一抽象（java8函数式接口）

内置类的设计

常见场景举例：

（1）临时数据存储类：ThreadLocal.ThreadLocalMap

（2）特殊的API实现：Collections.UnmodifiableCollection

- 可以看到这里不可变的集合属于一种包装模式，内置类实现了Collection接口，把写操作都实现抛出不可操作异常。
- 这里不可变通常用于返回值的不可变性，是只读的。（比较 java doc中的snapshop和view）

（3）Builder模式（接口） java.util.stream.Stream.Builder

java枚举设计

枚举类

场景：enum引入之前的模拟枚举实现

模式：

- 成员用常量表示，且类型为当前类型（类的类型）
- 常被设置为final类
- 非public构造器（自己内部来创建）

缺陷：

- 相对于常量 （强类型约束）
- 那定义的成员是第几个定义的？ 如果打印全部的枚举项？

枚举

基本特性：

- 类结构（强类型）
- 继承 java.lang.Enum
- 不可显示地继承和被继承

注意：

（1）枚举中 values方法是jvm（字节码提升）给枚举提升的方式

（2）Enum基类中的valueOf方法 也同上

父类中的name、ordinal提供了一些遍历

（3）枚举中的构造器字节码观察之后，默认枚举的构造函数是private的

枚举实际上就是个final的class，它的成员修饰符 public static final

（4）枚举是可以 定义抽象方法的（虽然是final的，这里比较矛盾） 然后成员在定义的时候实现的 （比如TimeUtil，但是这里的方法没有显式定义为abstract method，但是是可以定义抽象方法）

问题：

1. 子类在初始化时，会复制初始化一份父类的实例。
2. Integer 的那个值缓存 是可以修改的（一个VM参数）
3. 枚举的序列化问题（留个尾巴）