线程池(主要参数)

1. corePoolSize 核心线程数 一直维护的线程的个数，这种线程不会消亡
2. maximumPoolSize 最大线程数 并发执行的最大线程数
3. 闲置时间 空闲大于等于60s且超过了核心线程数的线程会被回收
4. 任务队列

当一个任务通过excute(Runnable)方法添加到线程池当时:

1. 线程数量小于corePoolSize(设定的核心线程数)，会新建核心线程直到线程数量等于corePoolSize
2. 线程数量大于corePoolSIze时，新任务被添加到等待队列、

若添加成功 执行线程

若添加失败

1. 线程数量小于maximumPoolSize(设定的最大线程数量) -----> 新建线程执行任务
2. 线程数量小于maximumPoolSize ------> 使用RejectedExcutionHandler拒绝策略

Okhttp的分发器处理策略

Running队列 正在执行的队列

Ready队列 等待的队列

异步请求

1.一个任务T被添加

如果 当前正在进行的请求个数大于64(可以设置，64是默认值),则将T加入等待队列

如果 当前正在进行的请求个数小于64

判断对同一域名的主机发起的请求数是否大于5个

是 --> 加入Ready队列等待

否 --> 加入Running队列执行

2.从Ready队列加入到Running队列中的条件是什么

每个请求执行完成就会从running移除，移除后会按顺序从等待队列中的一个符合执行的任务加入Running队列中执行

1. 分发器线程池的工作行为 无等待，高并发

为什么呢，首先得了解三种不同数据结构的等待队列(选取什么样的数据结构会决定线程池的排队机制)

SynchronousQueue<Runnable> //没有容量的容器

LinkedBlockingQUeue<Runnable> // 链表 有容量的容器

ArrayBlockingQueue<Runnable> // 数组 有容量的容器

Okhttp的分发器选择了SynchronousQueue<Runnable>，这是一个没有容量的容器，也就是i该队列无法添加东西，所以在执行addWork操作的时候会失败，会直接新建一个线程来执行该任务(线程池的源码，ctrl然后点击excute查看即可)

同步请求

1. 一个任务T被添加

加入running队列中，结束后移除

2.同步线程会在主线程中执行，所以在等待数据的时候会造成线程阻塞

OKhttp的不足

1. 用户网络请求的接口配置繁琐，尤其是需要配置复杂请求的body，请求头，参数的时候
2. 数据解析过程需要用户手动 拿到responseBody进行解析，不能复用
3. 无法适配自动进行线程切换
4. 网易我们存在嵌套网络请求就会陷入”回调陷阱”

补充:回调陷阱就是当你发起一次网络请求的时候，在第一次网络请求的callback中再次发起一次网络请求(也就是嵌套网络)，造成代码可读性低和维护性差。

retrofit的简介

1. retrofit本身不提供http请求，它是通过okhttp来实现http请求的。Retrofit是对okhttp的高度封装
2. Retrofit是一个RESTful的HTTP网络请求框架的封装

原因: 网络请求的工作本质上是okhttp完成的，而Retrofit仅负责网络请求接口的封装

1. App应用程序通过Retrofit请求网络，实际上是使用Retrofit接口层封装请求参数、Header、Url等星系，之后由OkHttp完成后续的操作
2. 在服务器返回数据之后，okHttp将原始的结果交给Retrofit、retrofit根据用户的需求对结果进行解析

学习Retrofit 主要学习它的架构思路

Retrofit封装的点

1. Okhttp创建的是OkhttpClient，然而Retrofit创建的是Retrofit实例
2. 构建Request，Retrofit是通过注解来完成
3. 配置Call的过程中，retrofit利用Adapter适配的Okhttp的Call，为call提供了多样性
4. 相对okhttp，retrofit会对responseBody进行解析(Gson)，提供了可服用，容易拓展的数据解析方案
5. 相对okhttp，retrofit会自动的完成线程的切换

Retrofit这个类主要使用了外观设计模式

而对于**外观设计模式的定义**是为子系统的一组接口提供了一个统一的入口。外观模式定义了一个高层接口，这个接口使得这一子系统更加容易使用

显然这么专业的定义不好看懂，那就举个例子，就比如

电脑启动的过程 启动cpu-->启动Memory-->启动Disk



如果不用外观设计模式对它们进行封装的话 用户启动电脑必须做出以下操作

CPU cpu = new CPU();

Memory memory = new Menory();

Disk disk = new Disk();

cpu.startUp();

memory.startUp();

disk.startUp();

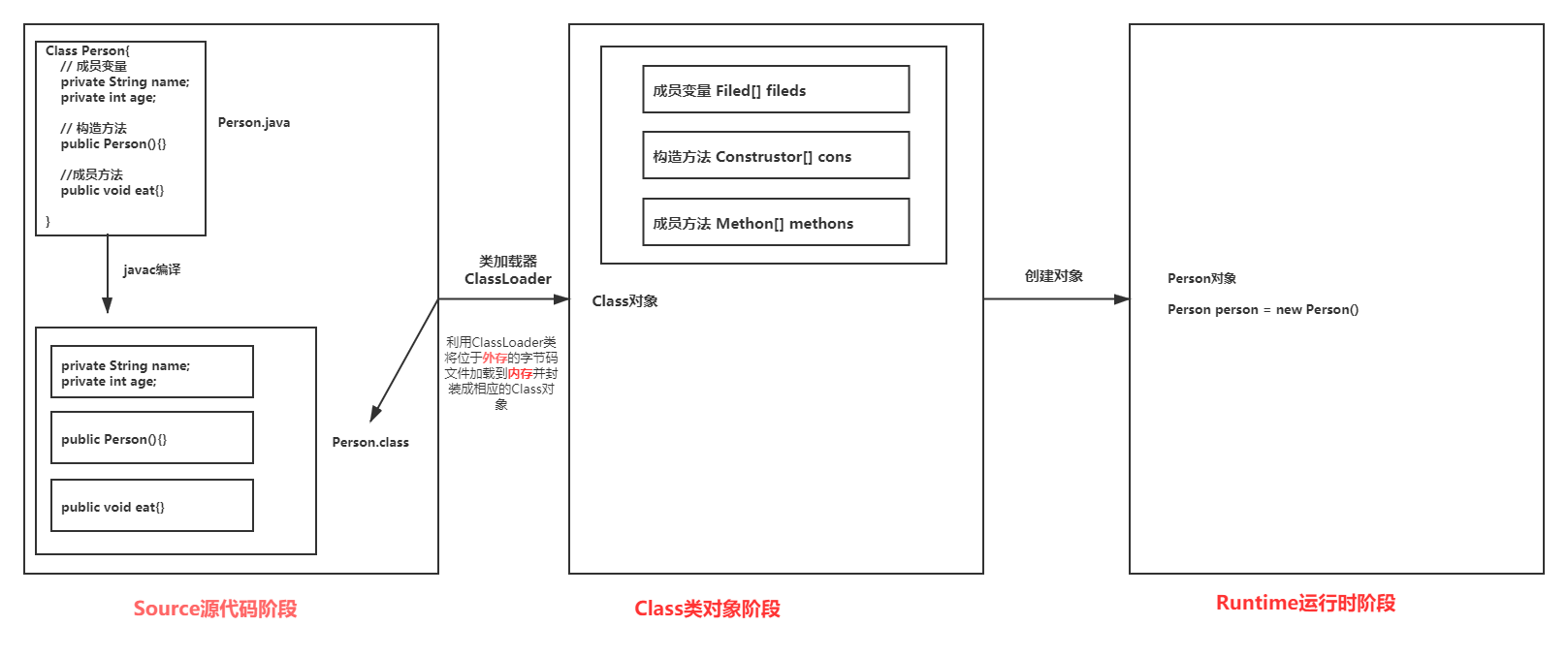
而利用外观设计模式，利用Computer类对这三个类进行封装，用户只需要

Computer computer = new Computer();

Computer.startUp();



**了解retrofit前，先来聊一聊反射(图可以自行放大)**

****

上面的图描述了一个类在编码阶段到运行阶段的流程

其中，Java利用ClassLoader类将位于**外存**的Person.class的字节码文件加载入**内存**并封装成相应的Class对象，这个过程就叫**反射**

1. 三种方式
2. 加载外存中的字节码文件，返回Class对象

Class object = Class.forName(“全类名”) 全类名也就是包名com.example.demo.Person

多用于配置文件，将类名定义在配置文件中。读取文件，加载类

1. 类名.class : 通过类名的属性class获取

多用于参数的传递

1. 通过实例化对象的getClass方法获取(这时字节码文件已经加载到内存并且已经封装成Class类了)

多用于判断该对象是什么类

Person person = new Person();

person.getClass();

1. Class对象功能

\*获取功能

1. 获取成员变量们

Field[] getFields() : 获取所有public修饰的成员变量

Field getField(String name) : 获取指定public修饰的成员变量

Field[] getDeclaredFields() : 获取所有声明的成员变量(包括private等)

Field getDeclaredField(String name) : 获取指定声明的成员变量

获取值 field.get(Object o);

设置值 field.set(Object o)

Field：成员变量

\*操作:

1. 设置值

\* void set(Object obj, Object value);

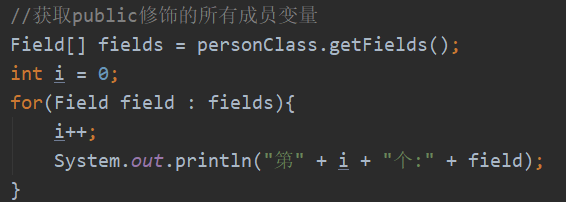
1. 获取值

\* void get(Object obj) 参数中的obj指的是持有该引用的类

1. 忽略访问修饰符的安全检查

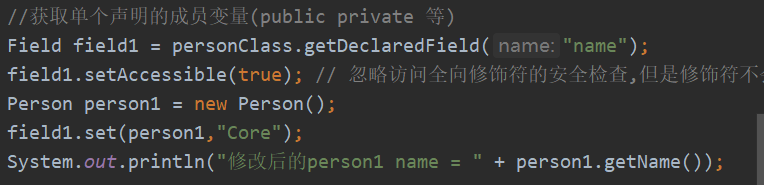
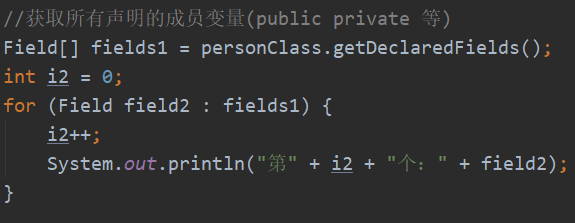
\*setAccessible(true) ：暴力反射

**实例代码**



Set方法需要传入一个该类的实例化对象，修改的是该实例化对象对应变量的值





1. 获取构造方法们（有没有Declared的区别和获取成员变量的一样，注意也需要setAccessible(true)）

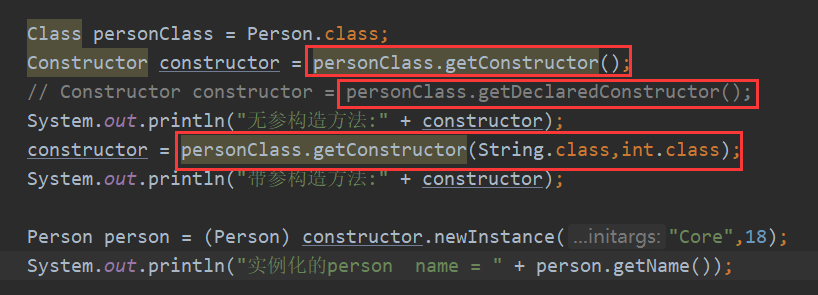
\* Constructor[] getConstructors()

\* Constructor[] getConstructor(String name(方法名字)，参数1，参数2 ...) 没有参数就不用填

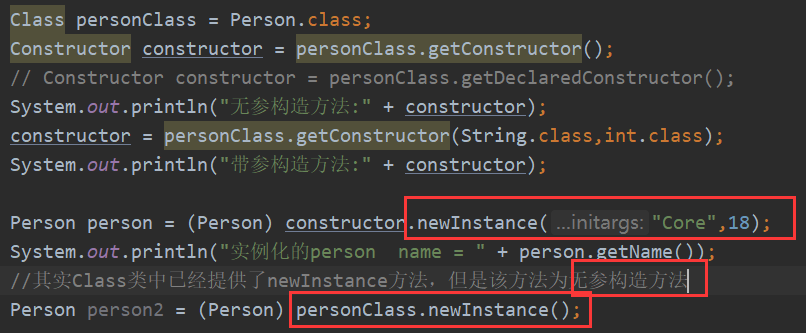
\* Constructor[] getDeclaredConstructors()

\* Constructor[] getDeclaredConstructor(String name(方法名字)，参数1，参数2 ...) 没有参数就不用填

**\* setAccessible(true)**



操作: 创建实例化对象



1. 获取成员类方们(注意setAccessible)

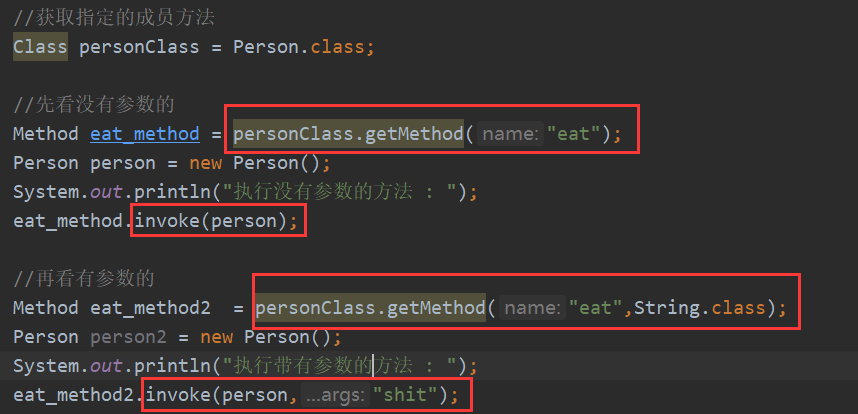
\* Methon[] getMethons() 这里需要注意，获取到的方法会包含子类从父类中继承的方法，而所有类默认继承Object类，所以都会有Object的方法

\* MethongetMethon(String name(方法名) , 参数1 ，参数2 ....) 有几个参数写几个，没有就不用写

\* Methon[] getDeclaredMethons()

\* Methon getDeclaredMethon(String name , 参数1 ，参数2 ....)有几个参数写几个，没有就不用写

**\* setAccessible(true)**



操作:执行方法

\* invoke( 实例化对 , 方法所需参数 ) 注意和获取到的Methon参数对应



1. 获取类名

\* getName()

结论: 同一个字节码文件(\*.class) 在一次程序运行的过程中，只会被加载一次，无论通过以上三方式方式中的哪种方式获取的class对象都是同一个对象

首先来了解一下retrofit的注解

成功建立一个Retrofit对象的标准: 配置好Retrofit类里面的成员变量

1. baseUrl: 网络请求的地址
2. CallFactory: 网络请求工厂
3. CallbackExecutor: 回调方法的执行器
4. AdapterFactories: 网络请求适配器工厂的集合
5. ConverterFactories: 数据转换器工厂的集合

retrofit的call对象是对okhttp的call的封装1

1. 请求类型的注解

GET、POST、PUT、DELETE、PATCH、HTTP、HEAD、OPTIONS一共8个

* 1. GET 这里path注解表示到时候参数id会取代@GET(“new/{id}”)中的id

@GET("new/{id}")

Call<ResponseBody> getNew(@path("id") int id);