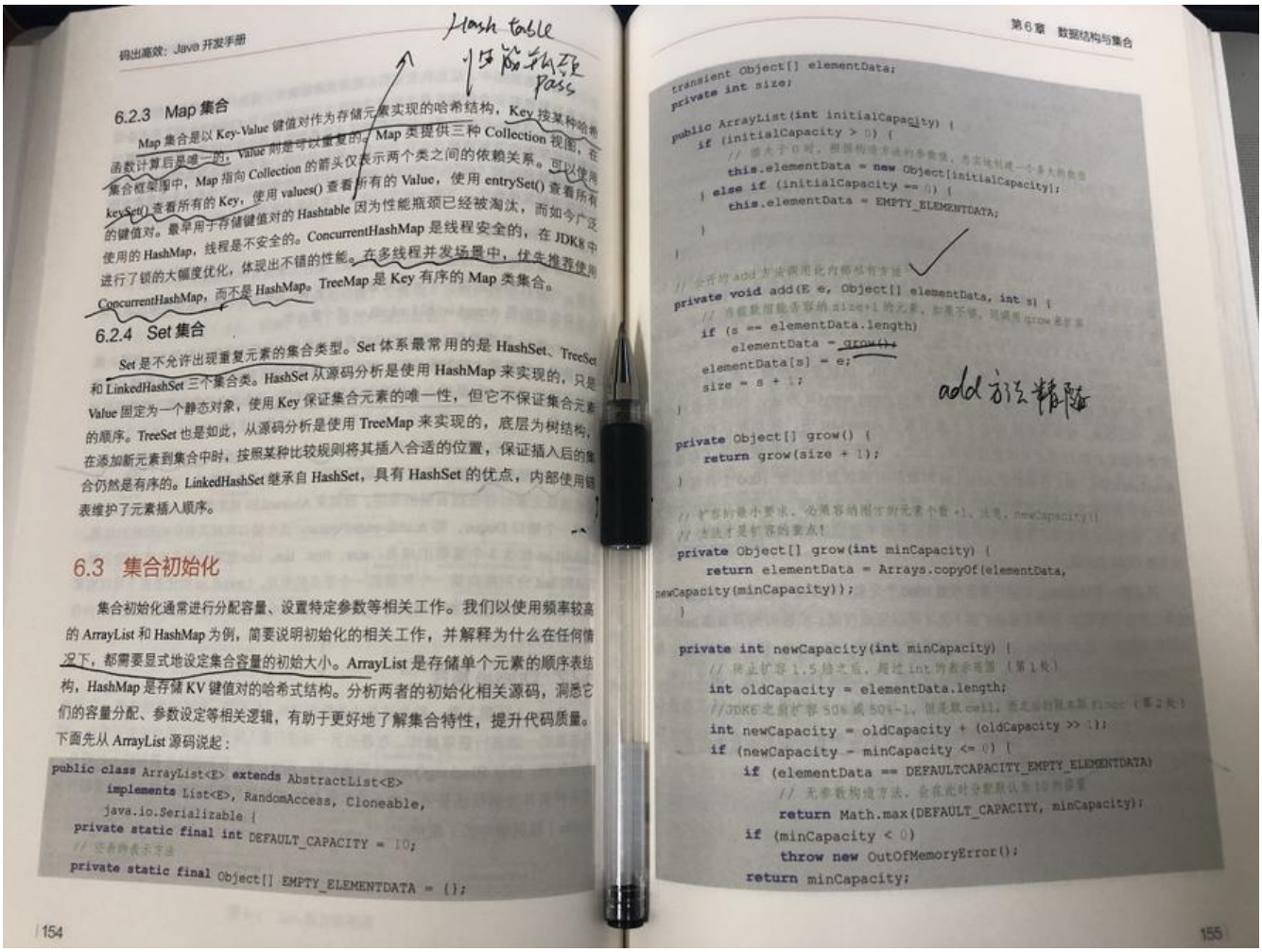
链滴

《码出高效》系列笔记（一）：面向对象中的方法

作者：matthewhan

原文链接：https://ld246.com/article/1579163958179

来源网站：链滴



良好的编码风格和完善统一的规约是最高效的方式。

前言

本篇汲取了本书中较为精华的知识要点和实践经验加上读者整理，作为本系列里的第一篇章第二节：向对象之方法篇。

本系列目录:

- 《码出高效》系列笔记（一）：面向对象中的类
- 《码出高效》系列笔记（一）：面向对象中的方法
- 《码出高效》系列笔记（一）：面向对象中的其他知识点
- 《码出高效》系列笔记（二）：代码风格
- 《码出高效》系列笔记（三）：异常与日志
- 《码出高效》系列笔记（四）：数据结构与集合的框架
- 《码出高效》系列笔记（四）：数据结构与集合的数组和泛型
- 《码出高效》系列笔记（四）：元素的比较

方法签名

方法签名包括方法名称和参数列表，将 JVM 标识方法的唯一索引，不包括返回值，更加不包括访问限控制符、异常类型等。

参数

参数一般又分为实参和形参，在代码注释中用 @param 表示参数类型，属于方法签名的一部分，包参数类型和参数个数，在代码风格中，约定每个逗号猴年必须要有一个空格，不管是形参和实参。

想到有一次面试时，就问到了成员变量和局部变量的传递过程，虽然很简单，说明还是有一些面试官注意到这些方面。

可变参数

一种特殊的参数————可变参数。在 JDK5 版本中引进，在实际应用中不算多见，适用于不确定参个数的场景。

有时候我们需要打印多个变量或参数的时候，用字符串拼接并不是很省力的方式，我们可以利用 form t 或者 printf 来进行格式化输出。其中 PrintStream 类中 printf() 方法就是使用了可变参数：

```
public PrintStream printf(String format, Object... args) {
    return format(format, args);
}
// 第一处
System.out.printf("%d", n);
// 第二处
System.out.printf("%d %s", n, "str");
```

在第一处调用传入了两个参数，在第二处调用传入了三个参数，他们调用的都是 printf(String format, Object... args) 方法。

这种方式即是 语法糖 ，也可能是 小恶魔 ，在实际开发中处理不当，容易影响代码可读性和可维护性书中也建议不要使用 Object 作为可变参数。

```
public static void late(Object... args) {
    System.out.println(args.length);
}
public static void main(String[] args) {
    // 第一处，此处打印结果为2
    late(4, new Integer[] {1, 2, 3});
    // 第二处，此处打印结果为3
    late(new Integer[] {1, 2, 3});
}
```

由于 Object 参数过于灵活，在第一处 new Integer[] {1, 2, 3} 和 4 都转型成了 Object[] ，作为 2 个象，在第二处则是作为 3 个 int[] 对象，所以不同类型的参数尽量避免使用该种方式传参。

入参保护

入参保护是对服务提供商的保护，常见于批量接口。因为批量接口虽然能够处理一批数据，但其处理力是有限的，因此要对入参的数据进行判断和控制，超出处理能力的，直接返回错误给客户端。

入参校验

需要进行参数校验的场景：

1. 调用频度低的方法。

2. 执行时间开销很大的方法。此情形中，参数校验的时间相对于就是小开销了，但是如果遇到因为参错误导致中间执行回退或者错误，则得不偿失。

3. 需要极高稳定性和可用性的方法。

4. 对外提供的开放接口。

5. 敏感权限入口。

不需要参数校验的场景：

1. 极有可能被循环调用的方法。但是在方法说明里必须注明外部参数检查。

2. 底层调用平度较高的方法。由于是频度较高，反而不容易是参数问题而引发，一般 DAO 和 SERVIC 都在同一个应用中，部署在同一台服务器中，所以可以省略 DAO 的参数校验。

3. 声明 private 只会被自己的代码调用的方法。如果能够确定调用方法的代码传入参数已经做过检查一般就不太会出现这种问题，此时便可不需要参数校验。

构造方法

构造方法是方法名与类名相同的特殊方法，在新建对象时调用，可以通过不同的构造方法实现不同方的对象初始化，他有如下特征：

1. 构造方法名称必须与类名相同。

2. 构造方法是没有返回类型的，即使是 void 也不能有。 他返回对象的址，并赋值给引用变量。

3. 构造方法不能被继承，不能被覆写，不能被直接调用。 调用途径有三：一是通过 new 关键字，二是在子类的构造方法中通过 super 关键字调用父类的构造方法（前面 super 关键字那里可以看下），三是通过反射方式获取。

4. 类定义时提供了默认的无参构造方法。 但是如果显式的定义了有参构方法，此无参构造方法就会被覆盖。

5. 构造方法可以私有。 外部无法使用私有构造方法创建对象。

6. 一个类可以有多个参数不同的构造方法，称为构造方法的重载。

类内方法

在面向过程的语言中，所有的方法都是全局静态方法。在引入面向对象理念后，某些方法才归属于具对象，即类内方法。除了构造方法外，还有三类方法：实例方法、静态方法、静态代码块。

实例方法

又称为非静态方法，依附于某个对象，可以通过引用变量调用其方法。类内部各个实例方法互相调用但是不包含 this 。实例方法可以调用静态变量和静态方法，当从外部创建对象后，应该尽量使用「类。静态方法」来调用，而不是对象名，依赖为编译器减负，二来提升代码可读性。

静态方法

又称为类方法。其中注意以下两点：

- 静态方法中不能使用实例成员变量和实例方法。
- 静态方法不能使用 super 和 this 关键字，这两个关键字指代的都是需要被创建出来的对象。

通常静态方法用于定义工具类的方法等，静态方法使用了可修改的对象，那么在并发时会存在线程安问题。所以工具类常见静态方法和单例相伴而生。

静态代码块

在书中是极不推荐的一种处理方式，这里就不提啦~

getter 与 setter

这是一类比较特殊的方法，一般自身不包含任何业务逻辑，仅仅为了类成员属性提供读取和修改的方，这样设计的好处就是：

- 满足面向对象语言封装的特性。将类成员属性设置成 private，访问与修改统统交由 getter 与 setter 方法处理。
- 有利于统一控制。

以下情况不推荐使用

1. getter 与 setter 中添加了业务逻辑。
2. 同时定义 isXxx() 和 getXxx() 。
3. 相同的属性名容易带来歧义。主要体现在子类与父类之间。