

第三章第一节 数据类型与类型检查

本章将介绍软件构造的理论基础——ADT，软件构造的技术基础——OOP

Outline

- 数据类型
 - 基本数据类型
 - 对象数据类型
- 类型检查
 - 静态类型检查
 - 动态类型检查
- 可变性与不可变性
 - 不可变性
 - 可变性
 - 防御性拷贝
- 快照图（Snapshot diagram）

Notes

##数据类型及其表达

Primitives	Object Reference Types
int, long, byte, short, char, float, double, boolean	Classes, interfaces, arrays, enums, annotations
No identity except their value 只有值，没有ID (与其他值无法区分)	Have identity distinct from value 既有ID，也有值
Immutable 不可变的	Some mutable, some not 可变/不可变
On stack, exist only when in use 在栈中分配内存	On heap, garbage collected 在堆中分配内存
Can't achieve unity of expression	Unity of expression with generics
Dirt cheap 代价低	More costly 代价昂贵

【基本数据类型】（引自 [菜鸟教程](#)）

Java语言提供了八种基本类型。六种数字类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型。

- **byte**：数据类型是**8**位、有符号的，以二进制补码表示的整数；
 - 最小值是 -128 (-2^7)；最大值是 127 (2^7-1)；默认值是 0；
 - byte 类型用在大型数组中节约空间，主要代替整数，因为 byte 变量占用的空间只有 int 类型的四分之一；
- **short**：数据类型是 **16** 位、有符号的以二进制补码表示的整数
 - 最小值是 -32768 (-2^{15})；最大值是 32767 ($2^{15} - 1$)；默认值是 0；
 - Short 数据类型也可以像 byte 那样节省空间。一个short变量是int型变量所占空间的二分之一；
- **int**：数据类型是**32**位、有符号的以二进制补码表示的整数；

最小值是 (-2^{31}) ；最大值是 $(2^{31} - 1)$ ；默认值是0；

- 一般地整型变量默认为 int 类型；
- **long**：数据类型是 64 位、有符号的以二进制补码表示的整数；
 - 最小值是 (-2^{63}) ；最大值是 $(2^{63} - 1)$ ；默认值是 0L；
 - 这种类型主要使用在需要比较大整数的系统上；
- **float**：数据类型是单精度、32位、符合IEEE 754标准的浮点数；
 - float 在储存大型浮点数组的时候可节省内存空间；
 - 默认值是 0.0f；
- **double**：数据类型是双精度、64 位、符合IEEE 754标准的浮点数；
 - 浮点数的默认类型为double类型；
 - double类型同样不能表示精确的值，如货币；
 - 默认值是 0.0d；
- **boolean**：数据类型表示一位的信息；
 - 只有两个取值：true 和 false；
 - 默认值是 false；
- **char**：是一个单一的 16 位 Unicode 字符
 - 最小值是 \u0000（即为0）；最大值是 \uffff（即为65,535）；
 - char 数据类型可以储存任何字符；例子：char letter = 'A'；

【对象数据结构】

- 对象：对象是类的一个实例，有状态和行为。
- 类：类是一个模板，它描述一类对象的行为和状态。
- Java作为一种面向对象语言，支持多态、继承、封装、抽象、重载等概念（[Java教程](#)）

【包装类】

- 对于包装类说，这些类的用途主要包含两种：
 - 作为和基本数据类型对应的类类型存在，方便涉及到对象的操作。
 - 包含每种基本数据类型的相关属性如最大值、最小值等，以及相关的操作方法。
- java中有八种基本数据类型对应的封装类型是：Integer([BigInteger](#)表示一个任意大小的整数),Double ,Long ,Float, Short,Byte,Character,Boolean；
- 包装类的效率更低，编译器会自动进行转换；
- 参考 [Java API java.lang](#)

类型检查

Java是一种静态类型的语言；所有变量的类型在编译的时候就已经知道了（程序还没有运行），所以编译器也可以测出每一个表达式的类型。在动态类型语言中（例如Python），这种类型检查是发生在程序运行的时候。

【动态检查】关于“值”的检查

- bug在运行中被发现
- 倾向于检查特定值才出发的错误
- 动态分析检查的类型：
 - 非法的变量值。例如整型变量x、y，表达式x/y 只有在运行后y为0才会报错，否则就是正确的。
 - 非法的返回值。例如最后得到的返回值无法用声明的类型来表明。
 - 越界访问。例如在一个字符串中使用一个负数索引。

- 空指针，使用一个null 对象解引用。

【静态检查】：关于“类型”的检查

- 静态检查>>动态检查>>无检查
- 在编译阶段发现错误，避免将错误带入到运行阶段，提高程序的正确性\健壮性
- 静态分析检查的类型
 - 语法错误，例如多余的标点符号或者错误的关键词。即使在动态类型的语言例如Python中也会做这种检查：如果你有一个多余的缩进，在运行之前就能发现它。
 - 类名\函数名错误，例如Math.sine(2)。(应该是 sin)
 - 参数数目错误，例如 Math.sin(30, 20)
 - 参数的型错误 Math.sin("30")
 - 返回值类型错误，例如一个声明返回 int 类型函数 return "30"
- 更多 关于静态\动态检查 请查看 [MIT Software Construction Reading 1](#)

可变性和不可变性

- 改变一个变量：是将该变量指向另一个值得存储空间
- 改变一个变量的值：是将该变量当前指向的值的存储空间中写入一个新的值

【不变性】（immutability）

- final 变量能被显式地初始化并且只能初始化一次。不变数据类型，一旦被创建，值不可修改；
- 基本类型及其封装对象类型都是不可变的
- 不可变的引用是指一旦指定引用位置后，不可再次指定。
- 如果编译器不能确定final变量不会改变，就提示错误，这也是静态类型检查的一部分
- 注意：
 - final类无法派生子类
 - final变量无法改变值\引用
 - final方法无法被子类重写

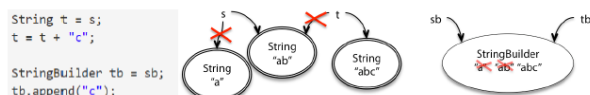
【可变性】（mutability）

- 不变对象：一旦被创建，始终指向同个值\引用
- 可变对象：拥有方法以修改自己的值\引用
- Sting与StingBuilder
 - String：不可变数据类型，在修改时必须创建一个新的String对象

```
String s = "a";  
a = s + "b";//s = s.concat("b");
```

- StringBuilder:可改变的数据类型，可以直接修改对象的值

```
StringBuilder sb = new StringBuilder("a");  
sb.append("b");
```



【可变性与不可变性的优缺点】


- 可变数据类型最小化的拷贝以提高效率；使用 不可变类型，对其频繁修改会产生大量的临时拷贝 (需要垃圾回收)
- 可变数据类型，可获得更好的效能；
- 可变数据类型也适合在多个模块之间共享数据；
- 不可变数据类型更安全，更易于理解，也更方便改变；

【防御性拷贝】

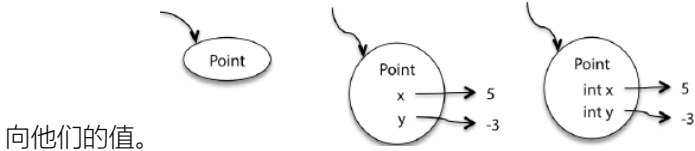
- 如果一个方法或构造函数允许可变对象进/出，那么就要考虑一下使用者是否有可能改变它。如果是的话，那你必须对该对象进行保护性拷贝，使进入方法内部的对象是外部时的拷贝而不它本身（因为外部的对象有可能还会被改变）。

```
• public Date getEnd() {  
    return new Date(end.getTime());  
} // 示例请参考 TKD03072010的专栏
```

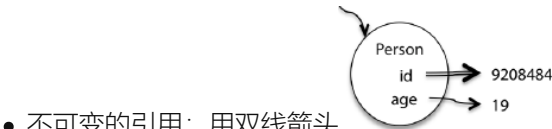
快照图(Snapshot diagram)

- 基本类型的值：原始值由裸露的常量表示。传入箭头引用变量或对象字段的值。

- 对象类型的值：一个对象值是一个由它的类型标记的圆。当我们想要显示更多的细节时，我们在它里面写字段名称，箭头指向他们的值。



- 不可变对象：用双线椭圆。



- 不可变的引用：用双线箭头
 - 引用时不可变的，但指向的值可以是可变的
 - 可变的引用，也可指向不可变的值