

Java HW3 文档

不变类分析与设计

目录

1	不变类源码分析	2
1.1	String 类	2
1.2	Math 类	3
1.3	System 类	3
1.4	总结	4
2	String 相关类分析	4
2.1	String 类	4
2.2	StringBuilder 类	5
2.3	StringBuffer 类	5
2.4	总结	5
3	不变类设计与测试	6
3.1	Vector 及 UnmodifiableVector 类	6
3.2	Matrix 及 UnmodifiableMatrix 类	7
3.3	MathUtils 类	8
3.4	运行效率测试	8

1 不变类源码分析

我从 `java.lang` 中选择了三个具有一定代表性的不变类进行源码分析。它们分别是：`String` 类，一个基本数据类；`Math` 类，一个纯工具类；`System` 类，一个与运行平台相关的标准系统类。

1.1 `String` 类

`String` 在 Java 中是作为一种基本数据类存在的，因此具有许多适用于字符串处理的方法。

1.1.1 数据表示

`String` 类的字符串数据用 `private final char value[]` 存储。除此之外，类中存储一个该字符串的哈希值 `private int hash`，用以快速进行语法中的字符串比较等操作。

1.1.2 构造方法

`String` 类的构造函数主要是完成对 `value[]` 的填充。除去直接从另一个字符串复制所用的构造函数，所有构造函数皆不置 `hash` 值。

其他具体细节不做分析。

1.1.3 功能方法

`String` 类的许多方法是用以从字符串中取数据，这些方法基本上是直接在 `value[]` 上进行操作，故不做分析。

一个值得注意的方法实现是 `equals()`；`equals()` 中先进行了引用的比较，如果不相等再进行依照字符串的比较。

一些功能性上需要返回 `String` 和 `String[]` 对象的方法都是返回新创建的 `String` 对象。方法中基本上会先创建一个 `char[]` 类型的字符串缓存并在其上进行操作，之后会调用一个特定的构造函数直接让 `value[]` 指向这个缓存，达到节省空间和时间的目的。

1.1.4 说明

字符串的不变类构造可以针对常量的可复用性，可以在构造时就让所有引用指向一个地址，从而实现快速字符串相等判断等操作。除此之外，基本数据类型的线程安全也是一个重要的考虑因素。

String 类中需要用到的 hash 值也需要 String 是个不变类，否则在每次取 hash 时都需重新遍历整个字符串计算，平白无故增加计算量。又 String 类有对应的可变类 StringBuilder，所以 String 设计成不变类很大程度上是出于对基本数据类型的性能的考虑。

1.2 Math 类

Math 类主要用来为程序员提供一系列数学方法。因此，Math 类基本不需要数据表示，也不需要创建新实例。

1.2.1 数据表示

Math 类中的数据只需要保存一些数学常量，如自然对数底 E 和 PI。这些值直接声明为 `public static final`，因此程序员也可以直接在外进行引用。

1.2.2 构造方法

Math 类只提供一个 `private` 的构造方法，方法中不做任何操作（因为所有功能性都已经由 `static` 方法和数据提供）。这避免了程序员创建该类的新实例，造成不必要的开销。

1.2.3 功能方法

Math 类提供许多静态数学方法，但这些方法与该类的特性无太大关联，故不对其做分析。

1.2.4 说明

Math 类作为一个无需创建新实例、只是为程序员提供工具的工具类，是完全不需要可变特性的。因此将其声明为不变类是较为合理的。

1.3 System 类

System 类直接与 JVM 相关，且是程序员常用的工具类，因此需要严格限制程序员对该类的操作。

1.3.1 数据表示

该类中的数据全部都声明为 `static`，以便于无法创建实例时程序员对其的使用。这些静态数据可能会被功能性的类方法改变。

1.3.2 构造方法

System 类与 Math 类类似，只提供一个 private 的构造方法，方法中不做任何操作（因为所有功能性都已经由 static 方法和数据提供）。这避免了程序员创建该类的新实例，造成不必要的开销。

1.3.3 功能方法

System 类的其他方法基本上都与系统状态有关。这些方法中有一些声明为 native 的方法，需要由虚拟机在语言外进行实现。其余的方法大多也是用来调用以及设置系统状态。

1.3.4 说明

System 类是一个 Java 语言中极其特殊的类，需要直接依赖外部编写的代码进行实现。如果将其声明为可变类或可以创建实例，则每次都会重新进行大量依赖于系统的设置，是不应该出现的情况。因此将其声明为不变类。

1.4 总结

我在以上部分选取了三个 Java 源码中具有一定代表性的不变类进行分析，可以看出把类设置为不变类主要是基于以下考虑：

1. 就性能来说，不变类应该可以缓存到某些常量池或其他内存池以提高利用该类的性能，或者是只提供方法而不需要创建实例。
2. 就功能来说，如果一个类在创建时需要进行大量依赖于其他代码的设置，则最好在代码中只实例化有限的实例，并把这个类设计成不变类，以避免多余的开销。
3. 就安全来说，对于一些数据类型，线程安全是比较重要的，这些数据可以设计为不变类以避免多线程运行可能造成的问题。

2 String 相关类分析

2.1 String 类

String 类是一个不变类。其数据用 final char value[] 存储，同时存储一个特定的 hash 值。String 类提供一些字符串功能操作，但因为不变类特性，这些操作基本上是新创建一个 char[] 对象，在其上进行操作，最后从这个对象通过构造方法新建一个 String 类对象。

String 类的不变类特性是适宜于作为基本数据类型的使用的。在程序编写中可能出现很多字符串常量，Java 把这些常量都创建对应的 String 类对象并存进常量池中，可以提升有关常量的语句的运行效率。但如果要进行字符串的修改，String 的效率是较低的。

2.2 StringBuilder 类

StringBuilder 是一个可变类，基于 AbstractStringBuilder 类继承而来。因此，其数据表示是基于 AbstractStringBuilder 类中的 char value[]。

StringBuilder 类提供许多字符串修改的操作。它直接在 char value[] 中进行修改，为此在申请内存时还预留了 16 个 char 的空间。许多方法是直接依赖 AbstractStringBuilder 进行实现。在修改操作完成时，它直接返回自己作为修改后的结果，从而（当不需要利用原字符串时）相对于 String 类极大地节省了空间和时间开销。

但同时，由于它是直接对自身进行修改，它是线程不安全的。

2.3 StringBuffer 类

StringBuffer 类被设计成一个可变类，和 StringBuilder 类同样由 AbstractStringBuilder 类继承而来。其数据表示同样是基于基类中的 char value[]。

它与 StringBuilder 类的不同主要是在于它的所有需要修改对象的方法都是带有 synchronized 关键字的，因此能够实现线程安全。其余情况下，它与 StringBuilder 类的表现相同。

2.4 总结

String 类被设计成不变类，主要是为了通过缓存快速处理代码中使用的常量字符串。但由于 String 类处理其他字符串的效率不高，所以 Java 提供了一个可变类 StringBuilder 来辅助进行字符串修改的操作。许多有关可变字符串的操作可以利用 StringBuilder 类进行，从而能够提高运行效率。

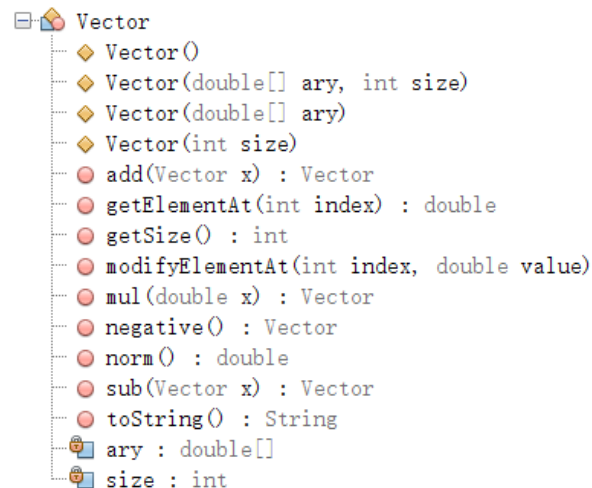
然而，StringBuilder 类的可变又会产生新问题，即其不是线程安全的。为解决这个问题，Java 又提供了 StringBuffer 类，用以在可变的基础上保证线程安全。程序员应该根据不同情况对这些类的使用加以区分。

3 不变类设计与测试

在这一部分，我将依照作业要求进行 Vector、Matrix 及其对应不变类的编写以及测试。

3.1 Vector 及 UnmodifiableVector 类

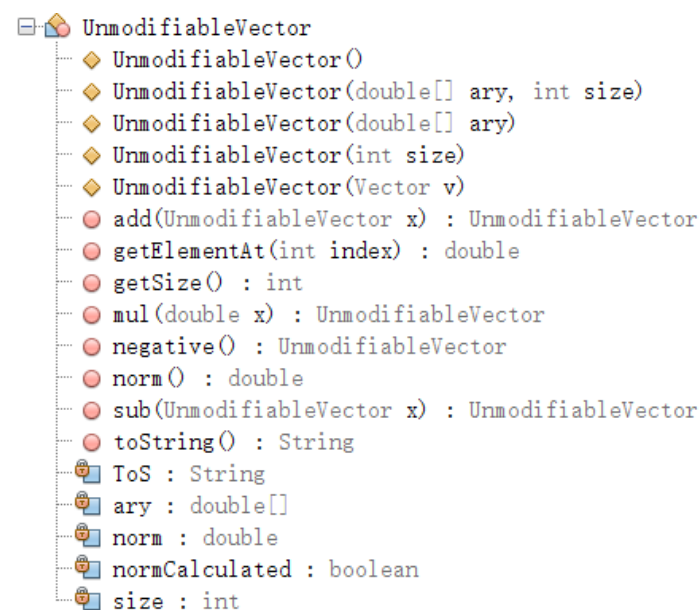
我设计的 Vector 类中有以下数据成员和方法：



```
Vector
  Vector()
  Vector(double[] ary, int size)
  Vector(double[] ary)
  Vector(int size)
  add(Vector x) : Vector
  getElementAt(int index) : double
  getSize() : int
  modifyElementAt(int index, double value)
  mul(double x) : Vector
  negative() : Vector
  norm() : double
  sub(Vector x) : Vector
  toString() : String
  ary : double[]
  size : int
```

ary 和 size 是数据成员，add()、sub()、mul()、negative()、norm() 是向量的基础运算（分别对应加、减、数乘、取反、取范数，返回一个新建的对象），getSize() 和 getElementAt() 提供对数据成员的访问，modifyElementAt() 进行对数据成员的修改。

相对应，UnmodifiableVector 类有以下成员和方法：



```
UnmodifiableVector
  UnmodifiableVector()
  UnmodifiableVector(double[] ary, int size)
  UnmodifiableVector(double[] ary)
  UnmodifiableVector(int size)
  UnmodifiableVector(Vector v)
  add(UnmodifiableVector x) : UnmodifiableVector
  getElementAt(int index) : double
  getSize() : int
  mul(double x) : UnmodifiableVector
  negative() : UnmodifiableVector
  norm() : double
  sub(UnmodifiableVector x) : UnmodifiableVector
  toString() : String
  ToS : String
  ary : double[]
  norm : double
  normCalculated : boolean
  size : int
```

其与 Vector 类的主要不同点如下：

1. 去除了 `modifyElementAt()` 方法，这是不变类中所不应当实现的功能；
2. 添加一些数据成员，以节省 `toString()` 和 `norm()` 方法的运行时间。

其中各方法的具体实现可参考代码。需要注意的是，在设计的不变类中，`ary` 和 `size` 都是被置为 `final` 的。

3.2 Matrix 及 UnmodifiableMatrix 类

我设计的 `Matrix` 类中有以下数据成员和方法：

```

Matrix
├── Matrix()
├── Matrix(double[][] ary, int xsize, int ysize)
├── Matrix(double[][] ary)
├── Matrix(int xsize, int ysize)
├── add(Matrix x) : Matrix
├── getElementAt(int x, int y) : double
├── getXSize() : int
├── getYSize() : int
├── modifyElementAt(int x, int y, double value)
├── mul(double x) : Matrix
├── mul(Matrix x) : Matrix
├── negative() : Matrix
├── sub(Matrix x) : Matrix
├── toString() : String
├── transposition() : Matrix
├── ary : double[][]
├── xsize : int
└── ysize : int

```

`Ary`、`xsize` 和 `ysize` 是数据成员，`add()`、`sub()`、`mul()`、`negative()`、`transposition()` 是矩阵的基础运算（分别对应加、减、数乘或矩乘、取反、取转置，返回一个新建的对象），`getXSize()`、`getYSize()` 和 `getElementAt()` 提供对数据成员的访问，`modifyElementAt()` 进行对数据成员的修改。

相对应，`UnmodifiableMatrix` 类有以下成员和方法：

```

UnmodifiableMatrix
├── UnmodifiableMatrix()
├── UnmodifiableMatrix(double[][] ary, int xsize, int ysize)
├── UnmodifiableMatrix(double[][] ary)
├── UnmodifiableMatrix(int xsize, int ysize)
├── UnmodifiableMatrix(Matrix v)
├── add(UnmodifiableMatrix x) : UnmodifiableMatrix
├── getElementAt(int x, int y) : double
├── getXSize() : int
├── getYSize() : int
├── modifyElementAt(int x, int y, double value)
├── mul(double x) : UnmodifiableMatrix
├── mul(UnmodifiableMatrix x) : UnmodifiableMatrix
├── negative() : UnmodifiableMatrix
├── sub(UnmodifiableMatrix x) : UnmodifiableMatrix
├── toString() : String
├── transposition() : UnmodifiableMatrix
├── ToS : String
├── ary : double[][]
├── trp : UnmodifiableMatrix
├── xsize : int
└── ysize : int

```

其与 Matrix 类的主要不同点如下：

1. 去除了 `modifyElementAt()` 方法，这是不变类中所不应当实现的功能；
2. 添加一些数据成员，以节省 `toString()` 和 `transposition()` 方法的运行时间。

其中各方法的具体实现可参考代码。需要注意的是，在设计的不变类中，`ary`、`xsize`、`ysize` 都是被置为 `final` 的。

3.3 MathUtils 类

参考 Java 库代码，我把 MathUtils 类实现如下：

```
public class MathUtils {
    private MathUtils() {
    }

    static UnmodifiableVector getUnmodifiableVector(Vector V) {
        return new UnmodifiableVector(V);
    }

    static UnmodifiableMatrix getUnmodifiableMatrix(Matrix M) {
        return new UnmodifiableMatrix(M);
    }
}
```

MathUtils 类只包含静态方法因此而不需要实例化，故把其唯一构造函数设定为 `private` 的。

由于设计的两个不变类都把数据置为 `final`，所以只能通过调用构造函数的方式进行从可变类到不变类的转换，构造函数写法不作说明。

3.4 运行效率测试

由于正确性测试比较显然，故在此处略去；以下大多是运行效率测试。各测试直接按以下表格体现，测试代码请参考原代码中的 `Main.java`。

编号	说明	重复规模	运行时间 1 (ms)		运行时间 2 (ms)	
1	矩阵递推自乘	1e7	1439	Matrix	1524	UnMatrix
2	矩阵乘原矩阵转置	1e7	2549	Matrix	1326	UnMatrix
3	向量加减后取范数	1e8	4929	Vector	4417	UnVector

诸如此类的测试可以表明：

1. 对象内容频繁发生改变且内容不易重复时，可变类的效率略比不变类高；

2. 需要进行读取的对象内容几乎不改变时，不变类的效率远远高于可变类；
3. 对象内容变化多但变化后可能与原对象重复时，不变类效率略高。

这样的结果与不变类的“空间换时间”方式的类实现以及 Java 对象的缓存特性有关。因此在编写 Java 程序时，应该依照情况灵活考虑可变类和不变类的不同使用。