(19)国家知识产权局

(12)发明专利

(10)授权公告号 CN 113835688 B
(45)授权公告日 2024.04.12

(21)申请号 202110991635.2

(22)申请日 2021.08.27

(65)同一申请的已公布的文献号
申请公布号 CN 113835688 A

(43)申请公布日 2021.12.24

(73)专利权人 河南理工大学
地址 454000 河南省焦作市高新区世纪大道2001号

(72)发明人 李晓斌 崔少北 海四洋

(51)Int.Cl.
G06F 8/30(2018.01)

(56)对比文件
US 6142684 A,2000.11.07
US 7669191 B1,2010.02.23
CN 112306472 A,2021.02.02
CN 111610966 A,2020.09.01
CN 109375899 A,2019.02.22
CN 103279358 A,2013.09.04
CN 110874213 A,2020.03.10
CN 103150199 A,2013.06.12
CN 112306473 A,2021.02.02
US 2012124564 A1,2012.05.17
杨有;李晓虹;尚晋.C~(++)中超长整型类型的构造与实现.计算机科学.2008,(01),全文.
杨有;李晓虹;尚晋.C~(++)中超长整型类型的构造与实现.计算机科学.2008,(01),全文.

审查员 黄云雪

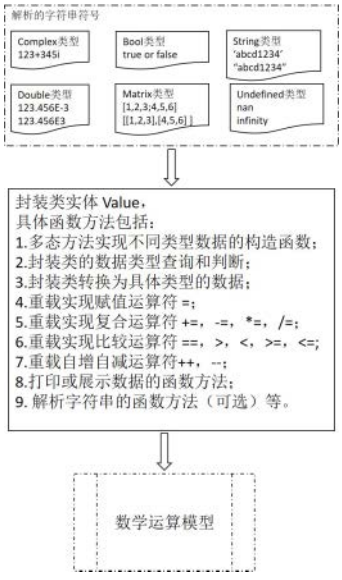
权利要求书2页 说明书5页 附图3页

(54)发明名称

一种科学计算语言解释器的对象封装方法

(57)摘要

一种科学计算语言解释器的对象封装方法，科学计算语言解释器的需要运算处理的字符串分为bool、complex、double、matrix、string和undefined等6种类型，其中：bool类型采用单词true和false作为标识；double类型采用数字、字母E和小数点作为标识；complex类型采用1个double标识和double紧随字母i为标识；string类型采用单引号或双引号作为标识；undefined类型采用单词nan或infinity作为标识。对上述字符串解析后调用相应的构造函数创建封装类Value实体；该实体然后输入值数学运算模型中，由于封装类实体重载实现了各种赋值运算符等，数学运算模型将对实体进行赋值和运算处理，最后调用处理后的Value实体的打印或展示数据函数方法，从而实现科学计算语言解释器的对象封装的目的。



CN 113835688 B

1.一种科学计算语言解释器的对象封装方法,其特征在于:该方法包括以下步骤:

第一步,创建封装对象class Value的基本数据类型索引及数据实例联合体;

首先,根据科学计算为目的不同类型数据建立枚举enum索引,包括Undefined, Bool, Double, Complex, String, Matrix, Vector 和Error类型,其中Int可看作Double类型,Error类型将指示赋值或运算不符规则的信息;然后,创建一个计算机位或字长的联合体union,用于存储实际需要处理的数据实例或指针,由于字符串、复数和矩阵数据类型大于一个计算机位或字长,采用指针连接字符串对象或矩阵数据类型,即封装的数据是上述枚举enum索引的一种类型;进一步,分别为字符串数据类型和矩阵数据类型创建对象,用于存储实际需要处理的字符串或矩阵数据对象;此外,创建一个枚举enum索引变量,用于指示封装对象的数据类型;

第二步,分别为上述不同数据类型创建构造函数,即:

```
Value(Type type = Undefined);
Value(bool b); Value(double n);
value(int64_t n);
Value(const std::string &s);
Value(const char *s);
Value(const matrix & mat);
Value(const std::vector<Value>&vec);
Value(const comlex& cc);
Value(double real, double image);
Value(const Value &other);
```

为了不同类型的原始数据实例化为封装对象,实例化同时对enum索引变量赋值,指示该封装对象的数据类型;

第三步,创建函数方法读取或判断封装对象的数据类型,包括:

```
Type type() const { return t; }
bool isError() const { return t == Error; }
bool isBool() const { return t == Bool; }
bool isDouble() const { return t == Double; }
bool isString() const { return t == String; }
bool isComplex() const { return t == Complex; }
bool isMatrix() const { return t == Matrix; }
bool isVector() const { return t == Vector; }
bool isUndefined() const { return t == Undefined; }
```

其中type()函数方法用于返回封装对象的数据类型,其他函数方法用于判断封装对象是否属于某一种类型;

第四步,创建函数方法将封装对象转换为实际的数据,使其能够被编译型语言对其进行赋值和运算;

第五步,创建设置不符运算规则的函数方法;

第六步,创建函数方法:printValue();

该函数方法用于打印或输出数据封装对象的数据,首先判断封装对象的数据类型,然后根据不同类型对象采用不同的输出方式;

第七步,实现科学计算的运算符重载方法。

2.如权利要求1所述的对象封装方法,所述第四步创建的函数方法包括:

bool toBool(bool defaultValue = false) const;

int toInt(int defaultValue = 0) const ;

double toDouble(double defaultValue = 0) const ;

std::string toString(const std::string &defaultValue = std::string()) const;

Matrix::Scalar toComplex(const Matrix::Scalar &defaultValue = std::complex<double>()) const;

Matrix toMatrix(Matrix defaultValue = Matrix());

const Matrix toMatrix(Matrix defaultValue = Matrix()) const ;

std::vector<Value>toVector(std::vector<Value>defaultValue = std::vector<Value>());

const std::vector<Value>toVector(std::vector<Value>defaultValue =std::vector<Value>()) const;

void setErrorInfo(const std::string &err = std::string());

std::string errorInfo() const。

3.如权利要求1所述的对象封装方法,所述第五步创建的函数方法包括:

setErrorInfo(const std::string &err = std::string());

该函数方法用于实现运算符重载时,存储不符规则的错误信息。

4.如权利要求1所述的对象封装方法,所述第七步包括:实现科学计算的运算符重载方法算术运算符的正+a、负-a、加 a + b、减a - b、乘 a * b、除 a / b、求余 a % b;自增a++、++a;自减a--、--a;比较运算大于a > b、小于 a < b、等于 a == b、不等于如 a != b、大于等于a >= b、小于等于 a <= b;位运算取反~a、左移 a << 2、右移 a >> 3、按位与a & b、按位或 a | b;赋值运算a = 5;复合赋值+=、-=、*=、/=、%=,实现重载方法时,首先判断封装对象的类型是否符合规则,然后进行运算,若不符运算规则,则调用其setErrorInfo()方法记录错误信息。

一种科学计算语言解释器的对象封装方法

技术领域

[0001] 本发明属于编程语言的技术领域,具体涉及一种适用于科学计算为目的对象封装方法,从而实现科学计算语言解释器能够灵活地处理不同的数据类型。

背景技术

[0002] 计算机科学领域的高级编程语言所编写的代码转换为机器码的方式可分为编译型和解释型2类,其中编译型语言需要先由编译器将程序代码编译并生成可执行文件,然后运行生成的可执行文件,程序执行效率高;而解释型语言在运行时由解释器将程序代码翻译成易于执行的中间代码,并由解释器(例如浏览器、虚拟机)引擎逐一将该中间代码解释成机器码并执行。

[0003] 编译型语言软件通用使用方法是编写代码、编译代码为可执行文件、然后运行,编译型语言对处理固定格式数据是很好的范例,但对于融合代码属性的元数据(代码作为数据的属性特征)处理不甚方便,以及编译型语言不能交互使用(例如,每编写一句或一行代码,执行运行结果的人机交互),对于需要探索和解决问题的工程师或数学家来说不是那么有用。而解释型语言可逐条解释运行,方便工程师或科学研究者交互使用;也可以添加到应用程序中,允许用户动态添加代码作为“脚本”运行。

[0004] 最典型的解释型语言为JavaScript、Python、Ruby和Perl等,以科学计算为目标的解释性语言包括Matlab、Octave、Maple和Mathematics等。这些语言需要对不同类型的数据对象进行赋值或加、减、乘、除、求余数或指数运算等,最为常见的数据对象为整数、实数、字符串和布尔等类型,以及特殊的复数和矩阵等类型。

[0005] 目前,解释型语言引擎分别需要对不同类型的数据对象分别建立模型,进行赋值和运算处理(实数和整数可视为同一类型),这使得语言解释器开发规则变得非常复杂,增加了开发难度,亦容易造成编译器更多的稳定性缺陷等。

发明内容

[0006] 本发明提供一种科学计算语言解释器的对象封装方法,以克服现有科学计算解释器开发的上述问题,或者至少部分解决上述问题。

[0007] 为了实现上述目的,本发明所述的一种科学计算语言解释器的对象封装方法,包括如下内容或步骤:

[0008] 第一步,创建封装对象class Value的基本数据类型索引及数据实例联合体。首先,根据科学计算为目的不同类型数据建立枚举enum索引,包括Undefined, Bool, Double, Complex, String, Matrix, Vector 和Error类型,其中Int可看作Double类型,Error类型将指示赋值或运算不符规则的信息;然后,创建一个计算机位或字长的联合体union,用于存储实际需要处理的数据实例或指针,由于字符串、复数和矩阵数据类型大于一个计算机位或字长,故采用指针连接字符串对象或矩阵数据类型,即封装的数据仅可能是上述枚举enum索引的一种类型;进一步,分别为字符串数据类型和矩阵数据类型创建对

象,用于存储实际需要处理的字符串或矩阵数据对象,该对象将可能被上述创建联合体union的指针连接指向实质的数据存储地址,同时对字符串或矩阵数据对象进行原子性封装,这样在多线程访问原子对象的不会造成竞争-冒险,从而实现字符串或矩阵数据对象的无锁设计;此外,需要创建一个枚举enum索引变量,用于指示封装对象的数据类型。

[0009] 第二步,分别为上述不同数据类型创建构造函数,即:

[0010] Value(Type type = Undefined);

[0011] Value(bool b); Value(double n);

[0012] value(int64_t n);

[0013] Value(const std::string &s);

[0014] Value(const char *s);

[0015] Value(const matrix & mat);

[0016] Value(const std::vector<Value> & vec);

[0017] Value(const comlex & cc);

[0018] Value(double real, double image);

[0019] Value(const Value &other);

[0020] 方便不同类型的原始数据实例化为封装对象,实例化同时对enum索引变量赋值,指示该封装对象的数据类型。

[0021] 第三步,创建函数方法读取或判断封装对象的数据类型等,包括:

[0022] Type type() const { return t; }

[0023] bool isError() const { return t == Error; }

[0024] bool isBool() const { return t == Bool; }

[0025] bool isDouble() const { return t == Double; }

[0026] bool isString() const { return t == String; }

[0027] bool isComplex() const { return t == Complex; }

[0028] bool isMatrix() const { return t == Matrix; }

[0029] bool isVector() const { return t == Vector; }

[0030] bool isUndefined() const { return t == Undefined; }

[0031] 其中type()函数方法用于返回封装对象的数据类型,其他函数方法用于判断封装对象是否属于某一种类型。

[0032] 第四步,创建函数方法将封装对象转换为实际的数据,使其能够被编译型语言对其进行赋值和运算等,包括:

[0033] bool toBool(bool defaultValue = false) const;

[0034] int toInt(int defaultValue = 0) const ;

[0035] double toDouble(double defaultValue = 0) const ;

[0036] std::string toString(const std::string &defaultValue = std::string()) const;

[0037] Matrix::Scalar toComplex(const Matrix::Scalar &defaultValue = std::complex<double>()) const;

[0038] Matrix toMatrix(Matrix defaultValue = Matrix());

[0039] const Matrix toMatrix(Matrix defaultValue = Matrix()) const ;

[0040] std::vector<Value> toVector(std::vector<Value> defaultValue = std::vector<Value>());

[0041] const std::vector<Value> toVector(std::vector<Value> defaultValue = std::vector<Value>()) const;

[0042] void setErrorInfo(const std::string &err = std::string());

[0043] std::string errorInfo() const;

[0044] 这些函数方法用于将封装类转换为具体类型的数据。

[0045] 第五步,创建设置不符运算规则的函数方法:

[0046] setErrorInfo(const std::string &err = std::string());

[0047] 该函数方法用于实现运算符重载时,存储不符规则的错误信息,例如字符串封装对象和实数封装对象运算时不符规则,将调用该方法存储出错信息;上述的errorInfo()函数方法可以读取该错误信息等。

[0048] 第六步,创建函数方法:

[0049] printValue();

[0050] 该函数方法用于打印或输出数据封装对象的数据,首先判断封装对象的数据类型,然后根据不同类型对象采用不同的输出方式;例如,字符串类型的数据直接输出至人机交互界面,实数或矩阵类型转换为字符串输出至人机交互界面等。

[0051] 第七步,实现科学计算的运算符重载方法,C++语言的运算符重载函数如下:

[0052] Value &operator = (const bool & b);

[0053] Value &operator = (const double & n);

[0054] Value &operator = (const int & n);

[0055] Value &operator = (const int64_t & n);

[0056] Value &operator = (const std::string &s);

[0057] Value &operator = (const char *s);

[0058] Value & operator = (const std::complex<double> & c);

[0059] Value &operator = (const Derived & mat);

[0060] Value &operator = (const Value & other);

[0061] Value &operator = (const std::vector<Value> &vec);

[0062] Value &operator ++(); // 前置++重载

[0063] Value &operator --(); // 前置++重载

[0064] Value &operator ++(const int); // 后置--重载

[0065] Value &operator --(const int); // 后置--重载

[0066] Value operator + (const Value & val) const ;

[0067] Value operator - (const Value & val) const ;

[0068] Value operator * (const Value & val) const ;

[0069] Value operator / (const Value & val) const ;

[0070] Value & operator - ();

[0071] Value & operator +=(const Value & val) ;

[0072] Value & operator -=(const Value & val) ;

[0073] Value & operator *=(const Value & val) ;

[0074] Value & operator /=(const Value & val) ;

[0075] bool operator == (const Value & val) const;

[0076] bool operator != (const Value & val) const;

[0077] bool operator < (const Value & val) const;

[0078] bool operator <= (const Value & val) const;

[0079] bool operator > (const Value & val) const;

[0080] bool operator >= (const Value & val) const;

[0081] 即:算术运算符的正+a、负-a、加 a + b、减a - b、乘 a * b、除 a / b、求余 a % b等;自增a++、++a;自减a--、--a;比较运算大于a > b、小于 a < b、等于 a == b、不等于如 a != b、大于等于 a >= b、小于等于 a <= b;位运算取反~a、左移 a << 2、右移 a >> 3、按位与a & b、按位或 a | b;赋值运算a = 5;复合赋值+=、-=、*=、/=、%=等,实现重载方法时,首先判断封装对象的类型是否符合规则,然后进行运算,若不符运算规则,则调用其setErrorInfo()方法记录错误信息,方便人机交互查询出错原因等。

[0082] 与现有技术相比,本发明的有益效果是:使得科学计算语言解释器模型简化同一,运行规则简单,降低开发难度,即建立同一的数据运算模型,然后针对数据的封装对象赋值和运算处理,从而更容易实现以科学计算为目的语言解释器等。

附图说明

[0083] 为了能够更清楚地理解本发明的上述目的、特征和优点,下面结合附图和具体实施方式对本发明进一步的详细描述。

[0084] 图1 为科学计算语言解释器的对象封装方法原理图。

[0085] 图2 为不同类型数据的枚举和共同体定义方法。

[0086] 图3 为字符串对象的定义方法。

[0087] 图4 为重载组合运算符的函数实现方法。

具体实施方式

[0088] 需要说明的是,在不冲突的情况下,本申请的实施例及实施例中的特征可以相互组合。在下面的描述中阐述了很多具体细节以便于充分理解本发明,但是,本发明还可以采用其他不同于在此描述的其他方式来实施,因此,本发明的保护范围并不受下面公开的具体实施例的限制。

[0089] 图1 所示为科学计算语言解释器的对象封装方法的原理图,首先,人机交互计算机语言的程序代码编辑器输入为空格或特殊符号分割的字符串,科学计算语言解释器需要运算处理的字符串分为bool、complex、double、matrix、string和undefined等6种类型,其中:bool类型采用单词true和false作为标识;double类型采用数字、字母E和小数点作为标识;complex类型采用1个double标识、“+”和double紧随字母i为标识,例如“2+3i”;string类型采用单引号或双引号作为标识;undefined类型采用单词nan或infinity作为标识。对上述字符串解析后调用相应的构造函数创建封装类Value实体;然后将该实体输入值数学

运算模型中或解释器引擎中,由于封装类实体重载实现了各种赋值运算符等,数学运算模型将对实体进行赋值和运算处理,最后调用处理后的Value实体的打印或展示数据函数方法,从而实现科学计算的目的。

[0090] 图2 所示为封装对象Value实体中不同类型数据的枚举和共同体定义方法,其中:不同数据类型采用枚举enum定义,具体的数据采用联合体union定义,union占1个计算机位数字长(例如32bit字长或64bit字长),由于字符串、矩阵、复数和矢量超过了1个计算机位数字长,则采用指针的指向相应的数据对象。

[0091] 图3所示为图2联合体字符串指针所指向的字符串对象的定义,该对象中除了字符串std::string所定义的s对象外,还采用原子锁方法,确保多线程不能在同一时间内访问相同的资源,实现改变数据结构数据时的无锁设计;当然,Value实体访问字符串对象时可采用locked或unlocked方法的有锁设计,确保安全访问或修改数据结构。

[0092] 如图3方法,实现超过占用1个计算机位数字长的矩阵类对象、复数类对象的定义。

[0093] 图4为求和组合运算符+=的重载函数实现方法,利用该方法可完成a += b 数学表达式的功能,其中a和b分别为封装对象value实体。该实现方法将调用查询或判断实体封装类型函数方法,若符合运算规则,则返回运算后的封装对象实体,若不符合运算规则,则调用赋值错误信息的函数方法,从而返回错误信息。

[0094] 如图4方法实现其他赋值运算符重载函数。

[0095] 以上所述实施例仅表达了应用了具体个例对本申请的原理及实施方式进行了阐述,其描述较为具体和详细,用于帮助理解本申请的方法及其核心思想;同时,对于本领域的一般技术人员,依据本申请的思想,在具体实施方式及应用范围上均会有改变之处,基于本申请中的实施例,本领域普通技术人员在没有做出创造性劳动前提下所获得的所有其他实施例,都属于本申请保护的范围。

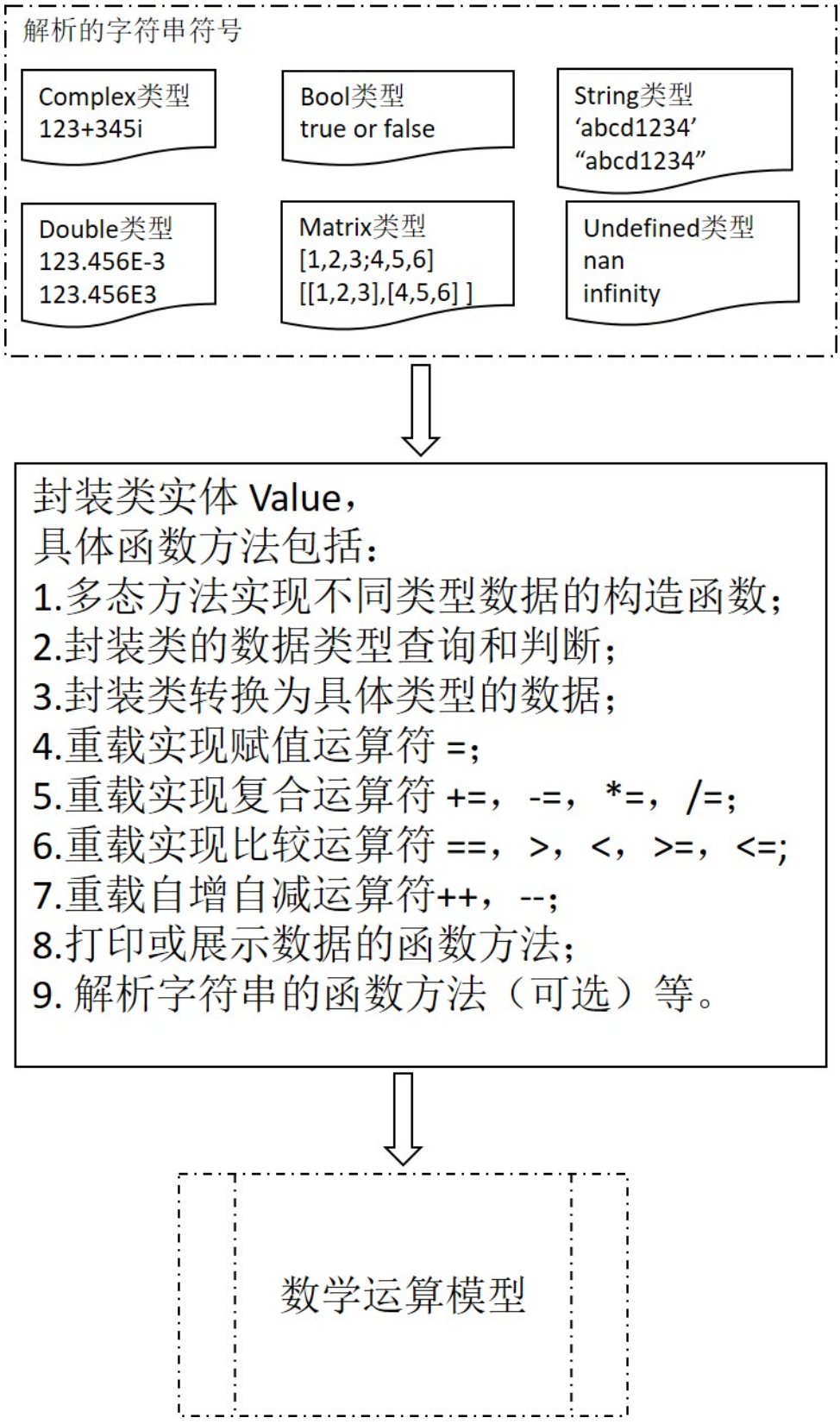

解析的字符串符号
Complex类型
123+345i
Bool类型
true or false
String类型
‘abcd1234’
“abcd1234”
Double类型
123.456E-3
123.456E3
Matrix类型
[1,2,3;4,5,6]
[[1,2,3],[4,5,6]]
Undefined类型
nan
infinity
封装类实体 Value，
具体函数方法包括：
1.多态方法实现不同类型数据的构造函数；
2.封装类的数据类型查询和判断；
3.封装类转换为具体类型的数据；
4.重载实现赋值运算符 =；
5.重载实现复合运算符 +=，-=，*=，/=；
6.重载实现比较运算符 ==，>，<，>=，<=；
7.重载自增自减运算符++，--；
8.打印或展示数据的函数方法；
9. 解析字符串的函数方法（可选）等。
数学运算模型

图1

```
class Value
{
  public:
      enum Type {
        Undefined  = 0x0,
        Bool       = 0x1,
        Double     = 0x2,
        Complex    = 0x3,
        String     = 0x4,
        Matrix     = 0x5,
        Vector     = 0x6,  // 用于多个返回值
        Error      = 0x80   // 用于返回错误信息
      };
inline Value(Type type = Undefined) {ui=0, t=type};
……
private:
      union {
        uint64_t ui;  // 整数
        bool b;       // bool类型
        double dbl;   // 实数
        SharedString *stringData;// 指向字符串对象;
        SharedData  *d;   // 指向矩阵或复数对象
        SharedList *vlist; // 指向封装对象数组
      };

      Type t;  // 封装对象的数据类型
};
```

图2

```
class SharedString
{
public:
   std::string s;

   std::string operator()(){return s;}
   bool ref() { return ++x != 0; }
   bool deref() { return --x != 0; }
   int load() { return x.load(std::memory_order_seq_cst); }
private:
   std::atomic<int> x { 0 };

};
```

图3

```
Value &Value::operator +=(const Value &val)
{
   if(t == Double && val.t == Double )
     this->dbl += val.dbl;
   else if(t == Complex && val.t == Complex )
     this->d->m += val.d->m;
   else if(t == Matrix && val.t == Matrix )
     this->d->m += val.d->m;
   else if(t == String && val.t == String)
     this->stringData->s += val.stringData->s;
   else
     this.setErrorInfo("Error!! The assignment operator does not fit the rules");

   return *this;
}
```

图4