

(19) 中华人民共和国国家知识产权局



(12) 发明专利

(10) 授权公告号 CN 104809008 B

(45) 授权公告日 2016. 06. 08

(21) 申请号 201510119258. 8

(22) 申请日 2015. 03. 18

(73) 专利权人 广东电网有限责任公司电力科学
研究院

地址 510080 广东省广州市越秀区东风东路
水均岗 8 号

专利权人 威海欣智信息科技有限公司

(72) 发明人 高雅 杜双育 王红斌 范颖
谢善益 杨强 王彬 马金宝

(74) 专利代理机构 广州知友专利商标代理有限
公司 44104

代理人 周克佑

(51) Int. Cl.

G06F 9/445(2006. 01)

审查员 张坦

权利要求书1页 说明书6页

(54) 发明名称

一种动态加载性质的 CIM 内存库生成方法

(57) 摘要

本发明的目的在于提供一种动态加载性质的 CIM 内存库生成方法，包括如下步骤：第一步：生成元信息代码；第二步：生成性质信息代码；第三步：定位性质；第四步：设置性质取值；以及，第五步：获取性质取值。本方法不对每个 CIM 性质生成存储变量和 bool 型的取值设置标识，也不通过占内存较多的映射表来存储性质取值，而是通过一种基于数组的动态寻址算法来实现所有性质的存储和访问，对于重量级稀疏型电力模型，节省内存特别明显。

1.一种动态加载性质的CIM内存库生成方法,其特征在于包括如下步骤:

第一步:生成元信息代码

a)将CIM模型中的每一个类生成相同名称的代码类;

b)为所述代码类中的属性生成代码中的字段,所述字段的名称和数据类型与CIM模型相同;

c)为所有字段生成缺省值静态常量,CIM模型里有缺省值的常量取值为缺省值,没有缺省值的常量取值为空;

d)为每个字段生成一个整型性质静态索引号,从0开始;

第二步:生成性质信息代码

在每个所述代码类中定义一个性质取值数组,用于动态存储性质取值,数组的大小初始为0,即空数组,在加载数据的时候根据实际有取值的性质个数动态调整数组的大小;另外根据CIM模型中的类中性质的个数,定义一个或多个long型数据标识,用来管理性质是否被设置取值以及配合在性质取值数组中定位性质;

第三步:定位性质,计算出用于在性质取值数组中定位性质的动态索引号;动态索引号的计算过程为:根据性质静态索引号找到对应的long型数据标识中的bit位,如果此bit位为1,计算出在此bit位之前的bit位为1的个数,就是此性质的动态索引号;对于bit位为0的性质,不存在动态索引号;

第四步:设置性质取值

设置性质取值首先判断long型标识中,性质的静态索引号所对应的bit位的取值;如果取值为1,则根据第三步找到相应的存储单元,将性质取值存放到存储单元中;

如果对应的bit位取值为0,说明当前数组中没有对应的存储单元,需要扩充性质取值数组;首先开辟新的性质取值数组,大小比原来大1,然后将当前性质静态索引号之前所有有值的性质按顺序拷贝到新的性质取值数组,再将当前性质要设置的值放到新的性质取值数组的下一个元素,最后将当前性质静态索引号之后的全部有值的性质按顺序拷贝到新性质取值数组中;将对象中记录数组的引用指向新的性质取值数组;

第五步:获取性质取值

如果long型标识中表明进行读取访问的性质没有设置值,即bit位为0,如果性质定义了缺省值就返回缺省值,否则返回空;

如果long型标识中表明进行读取访问的性质有设置值,即bit位为1,根据第三步找到相应的存储单元,直接返回存储单元中的性质取值。

一种动态加载性质的CIM内存库生成方法

技术领域

[0001] 本发明涉及电力系统数据处理方法,具体来说涉及一种动态加载性质的CIM内存库生成方法。

背景技术

[0002] 国际电工技术委员会IEC定义了一种电力系统通用信息模型CIM(Common Information Model,通用信息模型)。CIM提供了一个关于电力系统信息的全面逻辑视图,是一个代表电力企业所有主要对象的抽象模型,包括了这些对象的公有类和属性,以及它们之间的关系。

[0003] 使用CIM描述电力系统数据使得每种业务对象都有了一个特定的类名,包含明确的属性和关联定义,便于对数据进行统一描述和管理,也使得将电力设备对象之间映射成面向对象编程语言中的内存对象成为可能。CIM类中属性和关联的统称为性质。当对象性质取值为空时读取该性质得到的值为缺省值。

[0004] 在电力自动化分析等领域,将按照CIM组织的对象加载到内存中进行业务逻辑分析处理作为已经很常见,但多数是轻量级模型,数据量在几十万,而百万甚至千万级的数据平台很少有通过内存库提供数据访问效率的做法。

[0005] 一种是通过代码生成工具直接将CIM类中的类、属性、关联映射成编程语言中的类、属性和引用是常用的内存库生成方法。CIM中的每个性质,都被生成了实体变量,加载数据后,无论是否取值都会占用8个字节的内存空间。这种方法的缺点在于:第一:对于取值稀疏的电力模型,即CIM模型中某类的属性多,而实际数据中属性取值多数为空的情况下,内存浪费很多。现实情况中,由于CIM模式的业务覆盖率广,对象性质定义很多,多数CIM属性有值的概率不到一半,对于千万对象级数据,至少有几个G的内存被浪费,也使得低配置的硬件环境无法支撑大数据量的内存库。第二:由于CIM对象的性质还有缺省值的概念,对于未设置值的属性不能简单的用null代替,因为属性有可能被设置了null值,这种情况不能应用缺省值。因此需要额外为每个对象定义一个bool型的是否设置了取值的标识,每个性质多占4个字节的内存。

[0006] 此外,采用通用的对象描述构建内存库是另一种构建方式。此种方式使用一个通用的数据类型来描述所有对象,所有的属性、关联以映射表的方式存储在内存中。这种做法只保存有值的对象,没有值的性质不占用内存。这种基于对象通用描述和性质映射表的CIM内存库缺点在于:第一:不能发挥面向对象特性,无法使用get、set方法直接访问对象属性,也不能为特定的类增加会修改方法实现。第二:性质映射表采用key-value方式存储,每个key都是一个字符串,还有每个对象映射表本身所占的内存,使得有对象的性质内存占用量翻倍。第三:当进行性质访问时,必须进程字符串比较才能从映射表中定位到性质取值,字符串方式比较效率低,即使采用hash算法,也有一定的性能损失。

发明内容

[0007] 本发明的目的在于提供一种动态加载性质的CIM内存库生成方法,本方法可以在基本不影响效率的情况下,有效的减少基于CIM的面向对象内存库的资源占用,对于取值稀疏的电力模型尤其明显的。内存占用的大幅减少使得重量级数据平台的内存库构建不再受过高的内存要求限制,能够进行快速业务分析和提供高速数据访问能力。

[0008] 本发明的目的可通过以下的技术措施来实现:

[0009] 一种动态加载性质的CIM内存库生成方法,包括如下步骤:

[0010] 第一步:生成元信息代码

[0011] 根据CIM模型生成静态代码,包括类名、性质名称到静态索引号的映射、性质名称到数据类型的映射、性质的缺省值。

[0012] a)将CIM模型中的每一个类生成相同名称的代码类;

[0013] b)为所述代码类中的属性生成代码中的字段,所述字段的名称和数据类型与CIM模型相同;

[0014] c)为所有字段生成缺省值静态常量,CIM模型里有缺省值的常量取值为缺省值,没有缺省值的常量取值为空;

[0015] d)为每个字段生成一个整型性质静态索引号,从0开始;

[0016] 第二步:生成性质信息代码

[0017] 在每个所述代码类中定义一个性质取值数组,用于动态存储性质取值,数组的大小初始为0,即空数组,在加载数据的时候根据实际有取值的性质个数动态调整数组的大小。另外根据CIM模型中的类中性质的个数,定义一个或多个long型数据标识,用来管理性质是否被设置取值以及配合在性质取值数组中定位性质。

[0018] 一个long型变量有64个bit,可以管理64个性质。当某个bit取值为1时,代表该bit所对应的性质被设置了取值,性质取值数组中为此性质分配一个存储单元;当某个bit取值为0时,代表此bit所对应的性质未设置取值,性质取值数组中不会为其分配存储单元。

[0019] 第三步:定位性质,计算出用于在性质取值数组中定位性质的动态索引号;动态索引号的计算过程为:根据性质静态索引号找到对应的bit位,如果此bit位为1,计算出在此bit位之前的bit位为1的个数,就是此性质的动态索引号;对于bit位为0的性质,不存在动态索引号。

[0020] 第四步:设置性质取值

[0021] 设置性质取值首先判断long型标识中,性质的静态索引号所对应的bit位的取值。如果取值为1,则根据第三步找到相应的存储单元,将性质取值存放到存储单元中。

[0022] 如果对应的bit位取值为0,说明当前数组中没有对应的存储单元,需要扩充性质取值数组。首先开辟新的性质取值数组,大小比原来大1,然后将当前性质静态索引号之前所有有值的性质按顺序拷贝到新的性质取值数组,再将当前性质要设置的值放到性质取值数组的下一个元素,最后将当前性质静态索引号之后的全部有值的性质按顺序拷贝到新性质取值数组中。将对象中记录数组的引用指向新的性质取值数组,如果必要,释放旧的性质取值数组的内存。

[0023] 第五步:获取性质取值

[0024] 如果long型标识中表明进行读取访问的性质没有设置值,即bit位为0,如果性质定义了缺省值就返回缺省值,否则返回空。

[0025] 本发明对比现有技术,有如下优点:

[0026] 本发明是一种根据CIM模型生成面向对象程序代码的方法。本方法不对每个CIM性质生成存储变量和bool型的取值设置标识,也不通过占内存较多的映射表来存储性质取值,而是通过一种基于数组的动态寻址算法来实现所有性质的存储和访问,对于重量级稀疏型电力模型,节省内存特别明显。

具体实施方式

[0027] 一种动态加载性质的CIM内存库生成方法,包括如下步骤:

[0028] 第一步:生成元信息代码

[0029] 根据CIM模型生成静态代码,包括类名、性质名称到静态索引号的映射、性质名称到数据类型的映射和性质的缺省值。

[0030] 第二步:生成性质信息代码

[0031] 根据CIM类和性质定义生成程序类,每个类中定义一个空数组,用于动态存储性质取值。另外根据CIM类中性质的个数,定义一个或多个long型数据标识,用来管理性质是否被设置取值以及配合在数组中寻址。

[0032] 一个long型变量有64个bit,可以管理64个性质。当bit取值为1时,代表该性质被设置了取值,性质取整数组中会有一个与之对应的存储单元;当bit取值为0时,代表此性质未设置取值,性质取值数组中不会为其分配存储单元。

[0033] 第三步:定位性质

[0034] 因为性质取值数组并不是为所有性质分配存储单元,不存在一一对应关系,因此第一步生成的性质索引号不能直接用于访问数组中的存储单元。根据long型数据标识的取值和索引号可以计算出用于在数组中定位性质的动态索引号。例如,性质“name”的静态索引号是5,在它之前,索引号为0和3的也有值,其他为空,那么性质“name”的动态索引号为2。动态索引号并不是一直不变的,如果静态索引号为4的性质也被设置的取值,这时性质“name”的动态索引号就变成了3。

[0035] 因为性质取值数组并不是为所有性质分配存储单元,不存在一一对应关系,因此第一步生成的性质静态索引号不能直接用于访问数组中的存储单元。根据long型数据标识的取值和静态索引号可以计算出用于在数组中定位性质的动态索引号。

[0036]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | 63 |
|---|---|---|---|---|---|---|---|---|---|----|----|----|----|-------|----|
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 0 |

[0037] 动态索引号的计算过程为:根据性质静态索引号找到对应的bit位,如果此bit位为1,计算出在此bit位之前的bit位为1的个数,就是此性质的动态索引号;对于bit位为0的性质,不存在动态索引号。

[0038] 例如,性质“name”的静态索引号是5,在它之前,静态索引号为0和3的性质也有值,其他为空,那么性质“name”的动态索引号为2。动态索引号并不是一直不变的,如果静态索引号为4的性质也被设置的取值,这时性质“name”的动态索引号就变成了3。

[0039] 第四步:设置性质取值

[0040] 设置性质取值首先判断long型标识中,性质的静态索引号所对应的bit位的取值。

如果取值为1，则根据第三步找到相应的存储单元，将数据设置进去。

[0041] 如果对应的bit为取值为0，说明当前数组中没有对应的存储单元，需要扩充数组。首先开辟新的数组，大小比原来大1，然后将当前性质静态索引号之前所有有值的性质按顺序拷贝到新的数组，再将当前性质要设置的值放到数组的下一个元素，最后将当前性质静态索引号之后的全部有值的性质按顺序拷贝到新数组中。将对象中记录数组的引用指向新数组，如果必要，释放旧数组的内存。

[0042] 第五步：获取性质取值

[0043] 对于性质的读访问，有了第三步中的动态索引号，就可以取到数组中的实际取值，根据元信息中的性质取值类型，对数据进行强制转换，如Java语言中的Object对象转换为String对象，C++语言中void*指针转换成std::string*指针。

[0044] 如果long型标识中表面进行读取访问的性质没有设置值，即bit位为0，如果性质定义了缺省值就返回缺省值，否则返回空。

[0045] 数据的读写访问是通过生成的get和set方法代码进行的，因此数据的转换过程和动态寻址过程对内存库的使用者是完全透明的，用户只需要了解类中有哪些属性和属性本身的数据类型。

[0046] 假设CIM类Substation具有如下性质：

[0047]

| 索引号 | 名称 | 取值类型 | 默认值 |
|-----|---------------|----------------|-------|
| 0 | aliasName | String | 无 |
| 1 | category | String | “敞开式” |
| 2 | description | String | 无 |
| 3 | localName | String | 无 |
| 4 | mRID | String | 无 |
| 5 | name | String | 无 |
| 6 | VoltageLevels | VoltageLevel[] | 无 |
| 7 | Bays | Bay[] | 无 |
| 8 | Measurements | Measurement[] | 无 |

[0048] 以Java语言为例，第一步生成元信息代码，包括类名、性质静态索引号、性质缺省值：

```
class Substation {
    // static index
    static int aliasName_index = 0;
    static int category_index = 1;
    static int description_index = 2;
    static int localName_index = 3;
    static int mRID_index = 4;
    static int name_index = 5;
    static int VoltageLevels_index = 6;
    static int Bays_index = 7;
    static int Measurements_index = 8;

    // default value
    static String aliasName defaultValue = null;
    static String category defaultValue = "敞开式";
    static String description defaultValue = null;
    static String localName defaultValue = null;
    static String mRID defaultValue = null;
    static String name defaultValue = null;
    static VoltageLevel[] VoltageLevels defaultValue = new
    VoltageLevel[0];
}

[0049]
```

[0050]

```
static Bay[] Bays defaultValue = new Bay[0];
static Measurement[] Measurements defaultValue = new Measurement[0];
```

}

[0051] 第二步生成性质信息代码,包括性质取值数组和long型数据标识:

[0052] Object[]values=new Object[0];

[0053] //bit

[0054] long bits0=0;

[0055] 以name性质为例,第三步生成性质定位辅助代码,根据name的静态索引号name_index和bits0中的各bit位,计算出动态索引号idx:

```

int name_DynamicIndex() {
    int idx = 0;
    int b = 1;
    for (int i = 0; i < name_index; i++) {
        if ((bits0 & b) != 0) {
            idx++;
        }
        b <<= 1;
    }
    return idx;
}

```

[0056] 以name性质为例,第四步设置性质取值,先判断当前是否有存储单元,如果没有,将bit位置1,扩展数组大小以分配存储单元,然后将值设置到数组中。如果当前存在值,直接将值存放于数组中。

```

void setName(String name) {
    int b = 1;
    b <<= name_index;

    if ((bits0 & b) == 0) {
        // 设置bit位
        bits0 |= b;
        // 重新调整数组大小
        .....
    }
    values[name_DynamicIndex()] = name;
}

```

[0057] 以name性质为例,第五步,获取性质取值,先根据bit位判断name当前是否有值,如果没有,直接返回缺省值,如果有值,返回数组中的值。

```

String getName() {
    int b = 1 << name_index;
    if ((bits0 & b) == 0) {
        return nameDefaultValue;
    }
    // 类型转换
    return (String) values[name_DynamicIndex()];
}

```

[0058] 本发明的实施方式不限于此,在本发明上述基本技术思想前提下,按照本领域的普通技术知识和惯用手段对本发明内容所做出其它多种形式的修改、替换或变更,均落在本发明权利保护范围之内。