

文章编号:1001-9081(2010)02-0398-04

## 嵌入式 Java 反射机制的设计与实现

丘志杰, 罗 蕾

(电子科技大学 计算机科学与工程学院, 成都 610054)

(qzhijie@uestc.edu.cn)

**摘 要:**Java 反射是提高软件系统灵活性的重要技术,它是 Java 被视为动态语言的一个关键特性。虽然 CLDC 规范并不支持反射,但是在基于 Java 技术的嵌入式领域,反射仍有其应用需求。Sun 公司根据 CLDC 规范发布了一套嵌入式 Java 技术的参考实现,其虚拟机被称作 KVM。通过分析 KVM 的相关数据结构和机制,详细描述了反射机制的设计原理,并提出了一种在 KVM 中扩展反射功能的实现方案,经过测试证明该方案是可行的。

**关键词:**CLDC;KVM;Java;J2ME;反射

**中图分类号:**TP311.5 **文献标志码:**A

## Research and implementation of embedded Java reflection mechanism

QIU Zhi-jie, LUO Lei

(School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan 610054, China)

**Abstract:** Java reflection is an important technology, which can improve software system flexibility and is a key feature for Java to be regarded as dynamic language. Though CLDC specification does not support reflection, it is still required in embedded area based on Java technology. Sun Microsystems released a reference implementation of embedded Java technology according to CLDC specification. The authors described the reflection design principle in detail by analyzing related data structure and mechanism of KVM, as well as it presented a method to extend reflection for KVM. The method is proved feasible by test.

**Key words:** Connected Limited Device Configuration (CLDC); K Virtual Machine (KVM); Java; Java 2 Micro Edition (J2ME); reflection

### 0 引言

反射(reflection)的概念是由 MIT 的 Brian C. Smith 于 1982 年在他的博士论文中首次提出的<sup>[1]</sup>,主要是指程序可以访问、检测和修改它本身状态或行为的一种能力。许多程序设计语言都提供了程序包以支持反射,我们所熟悉的 Java 从 1.1 版本开始也增加 java.lang.reflect 程序包来提供反射功能。Java 的反射机制是一种强大的工具,我们使用 Java 反射能够创建灵活的软件框架。应用程序利用反射能够在运行时加载、探知和使用编译期间完全未知的 Java 类,因此,反射是 Java 被视为动态语言的一个关键特性。在各种 Java 版本中,只有 J2EE、J2SE,以及 J2ME 中的 CDC 提供了反射机制,基于性能、内存和安全等因素的考虑,在 J2ME 的 CLDC 规范中并不支持反射<sup>[2]</sup>,但是在基于 Java 技术的嵌入式应用领域却有着使用反射的需求。例如 JUnit 是一个优秀的用于单元测试的框架,在 J2SE 和 J2EE 的开发中被广泛使用。由于 JUnit 依赖于反射技术,因此程序员无法在不支持反射功能的 J2ME 平台上使用 JUnit。虽然网上也有其他一些类似 JUnit 的 J2ME 平台上的测试工具(如 J2MEUnit),但是都没有 JUnit 方便和好用。因此在提供反射功能的 J2ME 平台上,程序员可以充分利用 JUnit 的优势来提高测试效率,从而更好地监控和维护代码质量。

Sun 公司于 2000 年根据 CLDC 规范发布了一套嵌入式 Java 技术的参考实现(Reference Implementation, RI),其核心

是一个被称作 KVM<sup>[3]</sup>(K Virtual Machine)的 Java 虚拟机。由于可以从 Sun 公司的官方网站上自由获得该 RI 的源码,这为研究嵌入式 Java 反射提供了便利。本文的主要工作是以 KVM 为参考,提出一种嵌入式 Java 反射机制的设计思路。

### 1 KVM 中类和对象的表示形式

Java 反射机制主要涉及到了 java.lang.Class 以及 java.lang.reflect 包中的类,Class 类提供的如下三组反射 API 可以获取指定类的信息。

1) 获取类的构造方法信息:

```
Constructor getConstructor(Class[] params);  
Constructor[] getConstructors();  
Constructor getDeclaredConstructor(Class[] params);  
Constructor[] getDeclaredConstructors();
```

2) 获取类的方法信息:

```
Method getMethod(String name, Class[] params);  
Method[] getMethods();  
Method getDeclaredMethod(String name, Class[] params);  
Method[] getDeclaredMethods();
```

3) 获取类的字段信息:

```
Field getField(String name);  
Field[] getFields();  
Field getDeclaredField(String name);  
Field[] getDeclaredFields();
```

获取得到的类信息以对象的形式存放,即分别被保存于 java.lang.reflect 包的 Constructor、Method 以及 Field 类的对象

收稿日期:2009-07-29;修回日期:2009-09-17。

**作者简介:**丘志杰(1979-),男,广西百色人,博士研究生,主要研究方向:嵌入式 Java 技术、嵌入式富媒体;罗蕾(1967-),女,四川丹陵人,教授,博士生导师,主要研究方向:嵌入式富媒体、嵌入式操作系统、实时软件形式化。

中,应用程序调用这些类中相应的接口方法不但可以获知类的详细信息,而且还可以动态创建对象、动态调用方法以及动态存取字段<sup>[4]</sup>。Java之所以能够支持反射,根本原因就在于Java虚拟机在运行时拥有完整的类型信息,因此弄清楚虚拟机中类和对象的结构形式是实现Java反射的关键之一。

Java的class文件以二进制格式精确地描述了Java类或接口,它包含了Java虚拟机所能识别的、关于类或接口的所有信息<sup>[5]</sup>。一个以class文件表示的类需要被Java虚拟机加载后才能被使用。不同的Java虚拟机对类和对象的描述不尽相同,在KVM中,类和对象分别由instanceClassStruct和instanceStruct结构体来表示<sup>[6]</sup>,如图1所示。

在instanceStruct结构体中,ofClass成员代表了该对象所

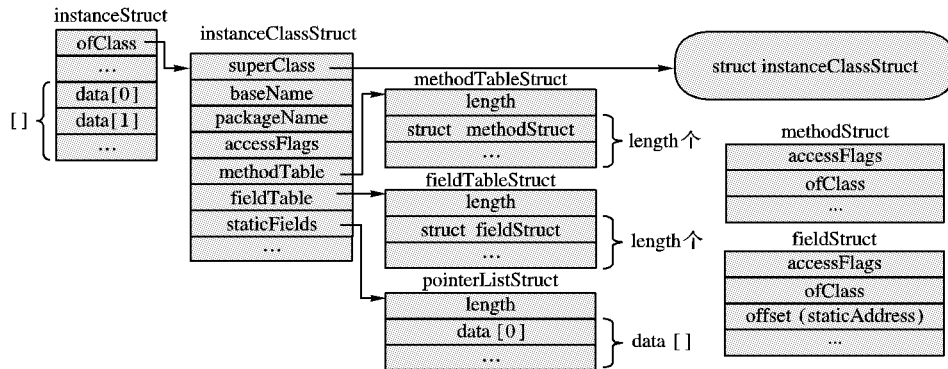


图1 KVM中类和对象的表示形式

由于静态字段属于类,一个类的所有对象共享该类的所有静态字段,因此KVM在加载类的时候专门为静态字段开辟了一段内存区,该内存区由pointerListStruct结构体来描述,由staticFields指针所指向。在pointerListStruct结构体中,length表示静态字段的个数,data成员是一个数组,数组大小则由length来决定,一个数组单元对应一个静态字段的存储空间。

## 2 Java反射机制的设计

### 2.1 动态调用Java方法

Java虚拟机的工作原理就是解释Java指令,根据指令的语义进行相应的操作。虚拟机以栈帧形式在Java栈中保存方法调用的状态<sup>[5]</sup>,当调用一个Java方法时,虚拟机会在Java栈中压入一个新的栈帧;方法执行完毕后,虚拟机也会将相应的栈帧从Java栈中弹出。程序中的每一条方法调用语句被编译成字节码后都对应于一组指令序列,图2给出了一个简单的方法调用示例及相应的指令序列,图3给出了虚拟机执行这组指令序列的基本过程:1)执行load\_0和iload\_1指令时,将被调方法callee所需的参数压入调用方法caller的栈帧中,如图3中①所示;2)为callee方法生成一个新的栈帧,如图3中②所示;3)将参数从caller方法的栈帧复制到callee方法的栈帧中,如图3中③所示;4)将程序计数器指向callee方法的第一条指令,如图3中④所示。此后虚拟机就可以开始执行callee方法。KVM在执行invokevirtual指令时主要完成以下两项工作:1)在方法表中(包括递归查找父类的方法表)找出与callee方法相对应的methodStruct结构体;2)将所找到的methodStruct结构体的指针作为参数来调用pushFrame函数,而该函数主要就是完成图3中②~④所示的工作。

由于class文件中的内容是在编译期间就静态确定的,因

属的类;由于同一个类的不同对象都各自拥有非静态字段的存储空间,因此data数组就是用来保存对象中每一个非静态字段的值。

在instanceClassStruct结构体中,methodTable是一个指向方法表(由methodTableStruct结构体表示)的指针,而每一个方法则由methodStruct结构体来描述,它记录了与一个Java方法相关的所有详细信息,例如方法的访问标志、方法所属的类、方法的名称、方法返回值、方法的参数列表等信息。fieldTable是一个指向字段表(由fieldTableStruct结构体表示)的指针,而每一个字段则由fieldStruct结构体来描述,它保存了与一个字段相关的所有详细信息,例如访问标志、字段所属的类等信息。

此要实现方法的动态调用,就必须解决一个问题:如何在编译时就确定了指令流中插入一组额外的、进行方法调用的指令序列。本文采取的解决方案是:1)在调用方法的栈帧中准备好被调方法所需的参数;2)找到与被调方法相对应的struct methodStruct结构体;3)以该结构体的指针作为参数来调用pushFrame函数。由于这三个步骤正是虚拟机执行方法调用指令序列所要做的,这等效于插入另外一组方法调用指令,因此也可以达到动态调用方法的效果。

```
public int callee(int v){ /*"v=callee(v)"对应的指令序列
    return v++;
}
public void caller(){
    int v=6;
    v=callee(v);
}
```

3: aload\_0 //将"this"指针压入caller方法的栈帧  
4: iload\_1 //将变量v压入caller方法的栈帧  
5: invokevirtual #2; //调用callee方法  
8: istore\_1 //将callee方法的返回值存入变量v中

图2 “v=callee(v)”语句对应的指令序列

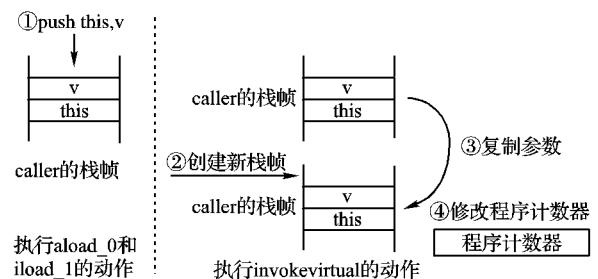


图3 虚拟机执行方法的过程

### 2.2 动态创建对象

Java程序可以调用new、Class类的newInstance方法或反射API(即Constructor类的newInstance方法)来创建对象,但无论采用那种方式,虚拟机除了为对象分配空间外,更关键的就是还要调用合适的构造方法去初始化对象。对于前两种方式,在编译期就静态绑定了所要调用的构造方法,这取决于在使用这两种方式来创建对象时所给的参数列表;而对于最后一种方式,却是在运行时完成构造方法的动态绑定,从而达到

动态创建对象的目的。而要实现动态绑定,其依据在于 Java 程序为了获取构造方法信息而调用反射 API (例如 `getConstructor`) 时所指定的参数列表。要实现反射机制中的动态创建对象的功能,本质上还是要解决如何动态调用构造方法的问题。因此,在设计动态创建对象的机制时采用了与动态调用 Java 方法相类似的方式,主要过程为:1) 分配对象空间;2) 在调用方法的栈帧中准备好构造方法所需的参数;3) 找到最佳匹配的构造方法的 `struct methodStruct` 结构体;4) 以该结构体的指针作为参数来调用 `pushFrame` 函数。

### 2.3 动态存取字段

Java 应用在调用诸如 `getField` 的反射 API 时,通过传递字段名字就可以动态地获得相应字段的详细信息。要实现字段的动态存取,关键问题就在于在运行时能够动态地定位字段的存储空间。由于 `fieldStruct` 结构体记录了一个字段的详细信息,因此只要找到字段的 `fieldStruct` 结构体,就可以完全定位该字段的存储空间。在 `fieldStruct` 结构体中,如果字段是静态的,那么 `staticAddress` 指针就指向该字段的存储空间,否则 `offset` 则是用于访问该字段存储空间的索引,如图 4 所示。

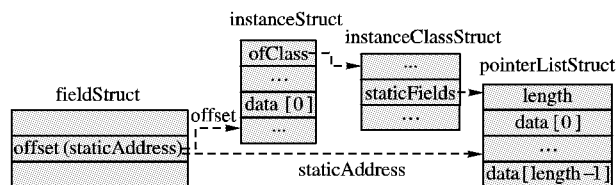


图 4 访问字段的方式

## 3 Java 反射机制的实现

在实现反射时,首先在 `Class` 类中扩展第 1 章所列举的反射 API,其次就是按反射规范<sup>[4]</sup> 定义并实现 `java.lang.reflect` 包。由于反射机制向应用提供的 API 较多,因此本文只以一些典型的 API 为例来描述反射机制的实现过程。这些 API 均被定义成本地方法(如下所示),并通过 KVM 中的 KNI<sup>[7]</sup> (K Native Interface) 机制实现本地化。

```
public native Constructor getConstructor(Class[] paramTypes);
//在 Class 类中定义

public native Method getMethod(String methodName,
Class[] paramTypes);
//在 Class 类中定义

public native Field getField(String fieldName);
//在 Class 类中定义

public native Object newInstance(Object[] args);
//在 Constructor 类中定义

public native Object invoke(Object obj, Object[] args);
//在 Method 类中定义

public native void setInt(Object obj, int value);
//在 Field 类中定义

public native int getInt(Object obj);
//在 Field 类中定义
```

### 3.1 收集类信息

除了规范要求提供的 `public` 方法以外, `Constructor`、`Method` 以及 `Field` 类的定义如下所示。通过 2.1 ~ 2.3 节的分析可知,找到描述 Java 方法(包括构造方法)的 `methodStruct` 结构体和描述字段的 `fieldStruct` 结构体是实现动态调用 Java 方法、动态创建对象和动态存取字段的关键。因此在这些类的定义中, `slot` 字段的作用就在于保存所找到的 `methodStruct` 或 `fieldStruct` 结构体的地址。

```
public final class Constructor implements Member
{
    private Class clazz;
    private int slot;
    private Class[] parameterTypes;
```

```
private Class[] exceptionTypes;
private int modifiers;
...
}

public final class Method implements Member
{
    private Class clazz;
    private int slot;
    private String name;
    private Class[] paramTypes;
    private Class returnType;
    private Class[] exceptionTypes;
    private int modifiers;
    ...
}

public final class Field implements Member
{
    private Class clazz;
    private int slot;
    private String name;
    private Class type;
    private int modifiers;
```

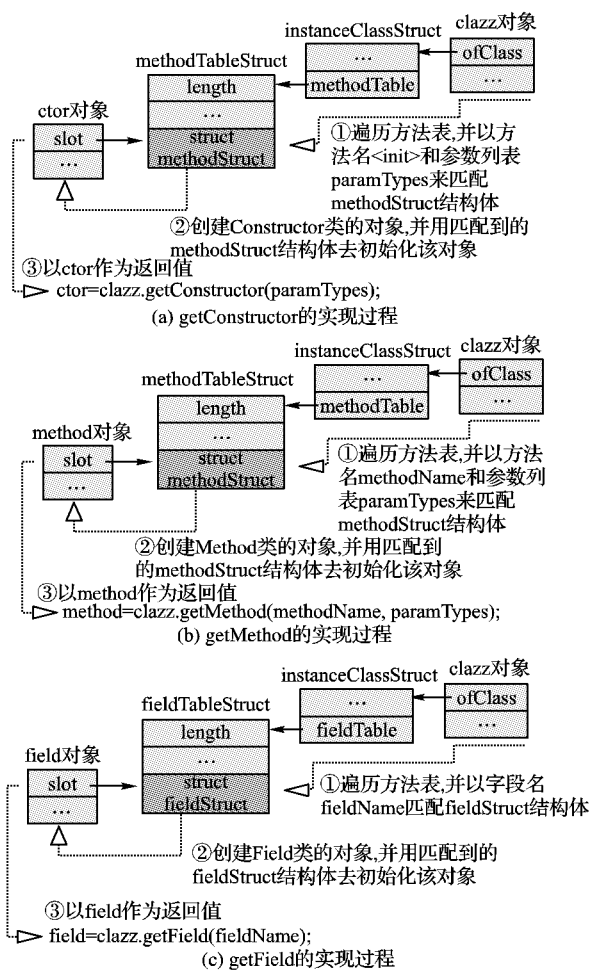


图 5 收集类信息的过程

在实现 `getConstructor` 方法时(如图 5(a)), 首先遍历 `instanceClassStruct` 中的方法表 `methodTable`, 找到最佳匹配的 `methodStruct` 结构体。匹配依据是: 因为 Java 允许方法重载, 因此除了匹配方法名以外(构造方法的方法名默认为 `<init>`), 还需要匹配方法的参数列表; 然后创建 `Constructor` 类

的对象 `ctor`, 用 `methodStruct` 结构体中的信息去初始化该对象, 并且将找到的 `methodStruct` 结构体的地址保存到该对象的 `slot` 字段中; 最后将该对象作为 `getConstructor` 的返回值返回给调用者。 `getMethod` 方法的实现过程相类似 (如图 5(b) 所示), 所不同的是创建和返回的是 `Method` 类的对象 `method`。在实现 `getField` 方法时 (如图 5(c) 所示), 首先遍历 `instanceClassStruct` 中的字段表 `fieldTable`, 找到最佳匹配的 `fieldStruct` 结构体。匹配原则: 由于在一个类中不允许定义同名字段, 因此匹配字段名称是唯一依据; 然后创建 `Field` 对象 `field`, 并用 `fieldStruct` 结构体中的信息去初始化该对象, 并且将找到的 `fieldStruct` 结构体的地址保存到该对象的 `slot` 字段中; 最后将该对象作为 `getField` 的返回值返回给调用者。

### 3.2 动态创建对象和调用 Java 方法

应用通过 `ctor` 对象来调用 `newInstance` 可以实现动态创建对象, `newInstance` 的实现过程如图 6(a) 所示: 首先根据 `ctor` 中的 `clazz` 字段, 调用 KVM 中的 `instantiate` 函数创建出对象 `newObj`; 然后将 `newObj` 以及参数列表 `args` 依次压入 Java 栈; 最后根据 `ctor` 中的 `slot` 字段调用 `pushFrame` 函数。KVM 从而将按照 2.2 节所述方式开始初始化 `newObj` 对象。

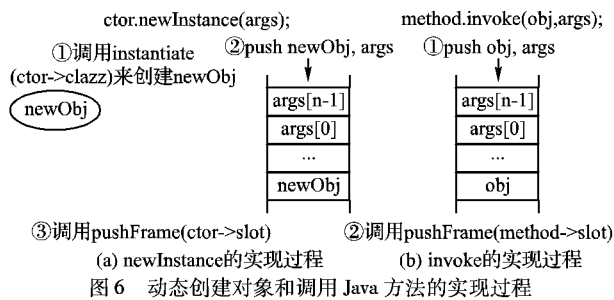


图6 动态创建对象和调用 Java 方法的实现过程

应用通过 `method` 对象来调用 `invoke` 可以 Java 方法的动态调用, `invoke` 的实现过程如图 6(b) 所示: 首先将参数 `obj` 及 `args` 依次压入 Java 栈, 然后根据 `method` 中的 `slot` 字段调用 `pushFrame` 函数。KVM 从而将按照 2.1 节所述方式开始动态调用 Java 方法。

### 3.3 动态存取字段

应用通过 `field` 对象来调用 `setInt` 和 `getInt` 就可以动态地存取整数类型的字段。 `setInt` 的实现过程如图 7(a) 所示: 首先根据 `field -> modifiers` 判断是否是静态字段, 如果是静态字段, 则将 `value` 赋值给 `field -> slot -> staticAddress` 所指向的内存空间; 否则将 `value` 赋值给 `obj -> data[field -> slot -> offset]`。 `getInt` 的实现过程如图 7(b) 所示: 首先根据 `field -> modifiers` 判断是否是静态字段, 如果是静态字段, 则将 `field -> slot -> staticAddress` 所指向的内存空间中保存的数据作为返回值; 否则将 `obj -> data[field -> slot -> offset]` 中保存的数据作为返回值。

## 4 测试

针对本文提出的反射方案, 测试分为三个部分。第一部分进行的是功能测试, 测试方式为: 按照反射规范编写测试用例, 并且在 PC 环境下使用扩充反射功能的 KVM 运行这些测试案例。测试结果表明本反射功能是正确的。第二部分进行的是性能对比, 测试方式为: 1) 使用具有 416 MHz 主频的 Xscale CPU、操作系统是 Windows Mobile 的 PDA 作为测试设备; 2) 为了进行性能比较, 将 KVM 和 `phoneME Advanced`<sup>[8]</sup> 都移植到了该 PDA 上; 3) 在 KVM 和 `phoneME Advanced` 上分别

运行文献[9]所提供的测试用例, 测试结果如图 8 所示。

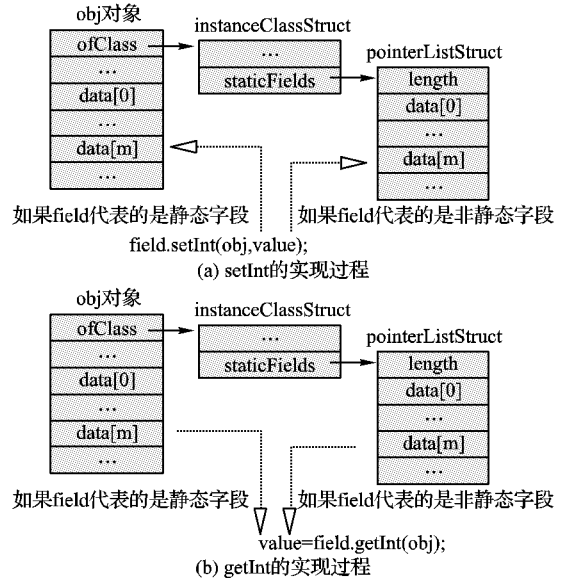


图7 动态存取整数类型字段的过程

从图 8 中可以看出, KVM 反射机制的性能与 `phoneME Advanced` 相当, 并且还略高一些。 `phoneME Advanced` 是一款符合 J2ME CDC 规范的虚拟机, 虽然它已经集成了反射功能, 但是它是针对高端嵌入式设备而设计的, 对平台能力 (如 CPU 主频、内存大小、内存管理、多任务管理和动态链接等) 的要求很高, 因此 `phoneME Advanced` 并不适合那些资源受限的中低端嵌入式设备。而 KVM 本身就是按照 CLDC 规范而设计, 因此扩展了反射功能的 KVM 在中低端嵌入式设备中比 `phoneME Advanced` 更具有优势。

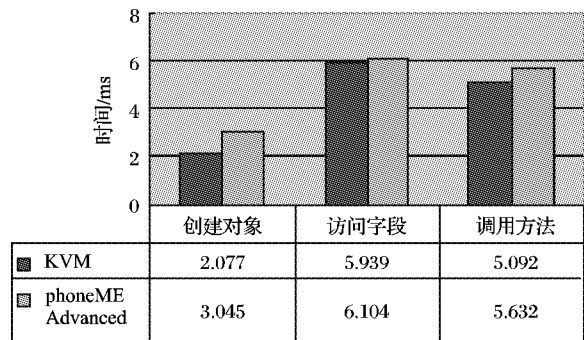


图8 KVM 与 `phoneME Advanced` 反射机制的性能比较

第三部分测试的目的是验证在中低端嵌入式设备的可行性, 为此将 KVM 移植到了采用 `nucleus` 实时操作系统、107 MHz ARM7 处理器的 `Feature Phone` 上, 图 9 给出了相应的测试数据。从结果来看, 虽然因 CPU 主频较低使得性能要比在 PDA 上差两倍左右, 但性能差距并不大, 对于诸如 `Feature Phone` 这类能力有限的中低端嵌入式设备来说, 这样的性能是可以满足应用需求的。

## 5 结语

本文通过分析 KVM, 提出了一种嵌入式反射机制的实现方案。反射在提供强大功能的同时也有其不足之处, 主要体现在性能和安全方面<sup>[9]</sup>。性能问题对应用程序的影响程度取决于在程序中如何使用反射, 或者什么样的程序使用反射。在性能关键的应用的核心逻辑中使用反射, 性能问题才变得

(下转第 422 页)

树在常用查询操作时快了 17.6%。

## 5 结语

本文分析 R\*-树在时间效率上的改进并非完善,其中的强制插入算法只是理论上的考虑,实际上很难达到 R\*-树所要求的两个目的,以一种新型的存储结构 R<sup>0</sup>-树,打破以前只局限于对算法上进行改进的方式,在 R\*-树的基础上做出新的调整,并给出了新的存储结构。通过 106 组数据实验,实验结果新的存储结构在常用查询操作快了 17.6%,说明改进结构在查询中具有相当好的效果。

### 参考文献:

- [1] 陈敏,王晶海. R\*-树空间索引的优化研究[J]. 计算机应用, 2007, 27(10): 2581-2583.
- [2] 过志峰,王宇翔,杨崇峻. 空间数据索引与查询技术研究及其应用[J]. 计算机工程与应用, 2002, 38(23): 176-178.
- [3] 谈国新. 一体化空间数据结构及其索引机制研究[J]. 测绘学报, 1998, 27(4): 293-299.
- [4] 顾军. R-树空间索引的优化研究[D]. 南京: 南京师范大学, 2002.
- [5] 翻腾. 基于 R-树空间索引系统的研究与应用[D]. 北京邮电大学, 2003.
- [6] 罗琪,李军,陈萃. 基于 GIS 平台的 R-树索引模型研究与实现[J]. 计算机工程与科学, 2003, 26(6): 24-27.
- [7] BECKMANN N, KRIEDEL H-P, SCHNEIDER R, *et al.* The R\*-tree: An efficient and robust access method for points and rectangles [J]. ACM SIGMOD Record, 1990, 19(2): 322-331.
- [8] BOHM C, BERCHTOLD S, KEIM D A. Searching in high-

dimensional spaces: Index structures for improving the performance of Multimedia databases [J]. ACM Computing Surveys, 2001, 33(3): 322-373.

- [9] ROUSSOPOULOS N, KELLEY S, VINCENT F. Nearest neighbor queries [C]// Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data. New York: ACM, 1995: 71-79.
- [10] HJALTASON G R, SAMET H. Distance browsing in spatial databases [J]. ACM Transactions on Database Systems, 1999, 24(2): 265-318.
- [11] 刘宇,朱仲英,施颂椒. 空间 k 近邻查询的新策略[J]. 上海交通大学学报: 自然科学版, 2001, 35(9): 1298-1302.
- [12] ZHANG DONG-HUI, XIA TIAN. A novel improvement to the R\*-tree spatial index using gain/loss metrics [C]// Proceedings of the 12th Annual ACM International Workshop on Geographic Information Systems. New York: ACM, 2004: 204-213.
- [13] PAPADIAS D, TAO Y, FU G, *et al.* An optimal and progressive algorithm for skyline queries [C]// Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2003: 467-478.
- [14] BRINKHOFF T, KRIEDEL H, SEEGER B. Efficient processing of spatial joins using R-trees [J]. ACM SIGMOD Record, 1993, 22(2): 237-246.
- [15] HUANG Y, JING N, RUNDENSTEINER E. Spatial joins using R-trees: Breadth-first traversal with global optimizations [C]// Proceedings of VLDB. [S. l.]: Morgan Kaufmann, 1997: 396-405.
- [16] 郭伦,刘瑜,张晶,等. 地理信息系统——原理、方法和应用 [M]. 北京: 科学出版社, 2001: 125-156.

(上接第 401 页)

至关重要;如果在程序运行过程中不会那么频繁地执行反射调用,或者相对很少的部分涉及到反射,性能问题才变得不那么重要。在安全方面,由于 J2SE 的安全机制庞大而耗内存,因此该安全机制并不适合于那些支持 CLDC/MIDP<sup>[10]</sup> 规范、内存和处理能力受限的嵌入式设备。CLDC/MIDP 使用了一种相对简单的安全模型,它通过除去 JNI<sup>[11]</sup>、反射等机制来为应用营造安全的运行环境<sup>[12]</sup>。在今后的工作中,将针对硬件资源有限的嵌入式设备的特点,进一步分析和解决反射所带来的性能和安全问题。

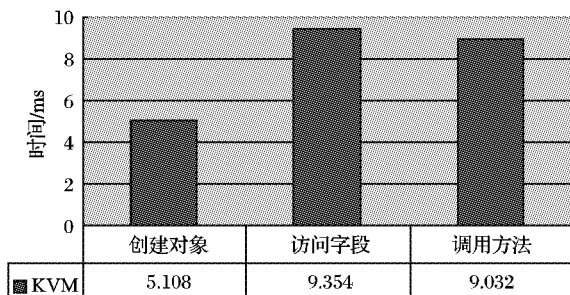


图9 在 Feature Phone 上的性能测试

### 参考文献:

- [1] SMITH B C. Reflection and semantics in a procedural language [D]. Boston, MA: Massachusetts Institute of Technology, 1982.
- [2] JCP. JSR-000030 J2 ME connected, limited device configuration [EB/OL]. (2000-05-30) [2009-07-10]. <http://jcp.org/aboutJava/communityprocess/final/jsr030>.
- [3] Sun Microsystems. J2ME building blocks for mobile devices [EB/

OL]. (2000-05-19) [2009-07-07]. <http://java.sun.com/products/cldc/wp/KVMwp.pdf>.

- [4] Sun Microsystems. Java core reflection [EB/OL]. [2009-07-01]. <http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflection.doc.html>.
- [5] VENNERS B. 深入 Java 虚拟机 [M]. 曹晓刚, 蒋靖, 译. 北京: 机械工业出版社, 2003.
- [6] 探砂工作室. 深入嵌入式 Java 虚拟机 [M]. 北京: 中国铁道出版社, 2003.
- [7] Sun Microsystems. K native interface [EB/OL]. [2009-07-01]. <http://java.sun.com/javame/reference/docs/kni/KNISpec.pdf>.
- [8] Sun Microsystems. phoneME advanced software project [EB/OL]. (2007-03-06) [2009-05-01]. [https://phoneme.dev.java.net/content/phoneme\\_platforms.html#phonemeadvanced](https://phoneme.dev.java.net/content/phoneme_platforms.html#phonemeadvanced).
- [9] SOSNOSKI D. Java programming dynamics, Part 2: Introducing reflection. [EB/OL]. (2003-06-03) [2009-06-01]. <http://www.ibm.com/developerworks/library/j-dyn0603/#resources>.
- [10] JCP. JSR-000118 mobile information device profile 2.0 [EB/OL]. (2002-11-20) [2009-06-01]. <http://jcp.org/aboutJava/communityprocess/final/jsr118>.
- [11] Sun Microsystems. Java native interface specification [EB/OL]. [2009-06-10]. <http://java.sun.com/javase/6/docs/technotes/guides/jni/spec/intro.html#wp9502>.
- [12] Qusay Mahmoud. Wireless Java security [EB/OL]. [2009-06-14]. <http://developers.sun.com/mobility/midp/articles/security/index.html>.