DOI：10.7652/xjtuxb201306002

以虚拟机为核心实现动态行为调整的方法

赵银亮，朱常鹏，韩博，曾庆花
（西安交通大学计算机科学与技术系，710049，西安）

摘要：针对面向上下文的编程语言未提供支持动态层添加机制的缺陷，提出了一种以虚拟机为核心的支持层动态添加的编程框架，并给出了实现方法。编程框架的核心是一个扩展后的Java虚拟机将谓词测试融入到Java虚拟机来实现程序对上下文的感知，将对象组合与代理融入到Java虚拟机来实现层的动态激活，利用虚拟机自身提供的功能实现动态层添加。实验结果表明，基于扩展后的Java虚拟机的编程框架可以实现层激活和动态层添加，编程框架的层激活效率较基于编译器的最多提高10%左右。

关键词：Java虚拟机；对象组合；代理；动态行为调整
中图分类号：TP314 **文献标志码**：A **文章编号**：0253-987X(2013)06-0006-06

Virtual Machine-Centric Approach for Dynamic Behavior Adjustment

ZHAO Yinliang, ZHU Changpeng, HAN Bo, ZENG Qinghua
(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: A programming framework is proposed to address the problem that the context-oriented programming languages do not provide any mechanisms to support dynamic layer addition, and its implementation is presented. The key component of the framework is an extended Java virtual machine. It incorporates predicate for checking object composition and delegation to implement layer activation, and uses existing virtual machine services to support dynamic layer addition. Therefore, the framework provides the capabilities of layer activation and dynamic layer addition. Experimental results show that the performance of layer activation in the proposed framework is approximately 10% higher than that in ContextJ.

Keywords: Java virtual machine; object composition; delegation; dynamic behavior adjustment

上下文感知是移动和普适计算领域中的一个关键特征，它要求程序能根据上下文动态地调整行为，这需要程序包含行为变体。目前，主流的编程语言缺少相应的机制来封装这类行为变体，这样变体会横切上下文感知程序的多个模块，从而导致出现代码交织和分散，影响程序的维护与扩展[1]。为了解决该问题，面向上下文编程[2](COP)建立了新的封装机制——层，来封装行为变体，即通过层的激活来支持程序根据上下文来动态调整编程行为，同时保持程序代码的简洁性，提高程序的可维护性。

文献[3-4]的COP语言是通过扩展Java编译器实现层和层的激活的，但不是所有的上下文信息在开发阶段可知[5]，程序在运行时需要增加新的层。因此，基于编译器扩展实现的COP语言无法有效支持动态的层添加。

Subramanian等提出了利用扩展Jike RVM虚拟机实现Java类的动态更新的方法[6]。MalaBarb等提出了通过扩展虚拟机实现Java类的动态更新

收稿日期：2012-09-25。 作者简介：赵银亮(1960—)，男，教授，博士生导师。 基金项目：国家自然科学基金资助项目(61173040)。
网络出版时间：2013-03-11 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20130311.1749.003.html

的方法，同时形式化地描述了该方法，并证明了它的正确性[7]。为了保证更新前后程序状态的一致性，上述方法在更新类的同时必须转化类的所有对象，这会导致额外开销和安全点问题[6-7]，延迟类的动态更新。Bettini 等提出了一种对象组合和代理的方法，通过对象组合操作实现对象的动态改变，将代理应用于对象方法的调用[8]，但是该方法在改变对象时并未考虑上下文信息，因此程序无法根据上下文动态地调整编程行为，引起程序运行时出错误。

本文提出了一种支持动态层添加的编程框架，核心是扩展了 JamVM 虚拟机。该虚拟机通过融入谓词测试[9-10]实现了程序对上下文的感知，利用对象组合和代理、根据上下文来动态调整程序行为，利用自身功能实现了层的动态添加。

1　编程框架

编程框架由一个编译器和一个扩展的 JamVM 虚拟机构成，是整个框架的核心组件。基于该框架开发的应用程序由一个标准 Java 程序、一组层和一个配置文件构成，其中：标准 Java 程序称为基程序，它的类称为基类；层通过关键字 layer 定义，并用 class-in-layer[11] 方式编写；配置文件是一个 XML 文件，描述定义程序包含的层。每一个层都包含一组类和一个特殊的谓词 predicate，层中的类称为层化类，它的定义方式与基类相同。层化类可以重写基类中的方法，层化类的实例称为部分对象。谓词函数 predicate 用于判断层是否可用于当前上下文[9-10]：返回 true 表示可用，否则不可用。层可用，层中的方法可在当前上下文中执行，层则视为上下文感知单元，它使应用程序具备了感知上下文的能力。

框架的工作流程如图 1 所示。流程分 3 步：①在程序运行中，当执行方法调用时，框架的运行系统将检查被调用方法是否被层化类重写；②如果被调用方法被层化类重写，则运行系统根据当前运行时的上下文来评估配置文件；③如果在配置文件中能发现适合当前上下文的层，则加载其中相应的类，通过对象组合和代理来执行该类的相应方法，否则直接执行被调用方法。

1.1　配置文件

配置文件是一个 XML 文件，是应用程序的重要组成部分，它将层从传统 COP 应用程序的逻辑中分离，形成相互独立的层和基程序两部分，这使得扩展的虚拟机能根据运行时的上下文将方法调用重定向到相应的层中方法，提供层的激活能力，确保层的动态添加不影响基程序的代码。

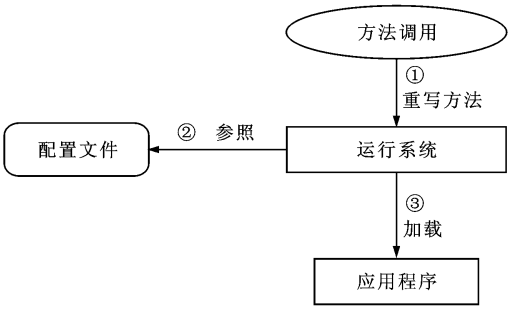

图 1　编程框架工作流程

配置文件格式如下

```
〈app_contract name=“App1”〉
  〈zone class=“C”〉
    〈layer constituent=“L1,C” location=“bin”/〉
    ⋮
  〈/zone〉
  〈zone class=“D”〉
    ⋮
  〈/zone〉
〈/app_contract〉
```

根据类名将文件内容划分为不同的区，与类名同名的层化类划分到该类名所对应的区。如，与类 C 同名的层化类划分到上述格式的第一个区，其中〈layer constituent=“L_1,C” location=“bin”/〉表示定义在层 L_1 中的层化类 C，并且 L_1 和类 C 编译后生成的文件位置位于当前目录的 bin 目录下。

运行时的方法调用要被运行系统重定向，这会导致出现动态的类型违背[7]，因此需对层化类制定相应的约束，以确保应用程序运行时类型的安全。

约束 1　层化类的公有方法必须是同名基类的重写方法。

约束 2　在配置文件中，同一区中的层化类必须属于不同的层，而且有相同的接口，即这些层化类的公有方法的签名和返回类型必须相同。

约束 1 可确保方法调用重定向到层中的相应方法时不会导致出现动态的类型违背；约束 2 可确保同一区中不同层化类的相同签名方法提供相互兼容的语义[12]。

1.2　层激活与动态层加载

扩展后的虚拟机不仅负责执行程序，还要负责层的激活和动态加载。层激活是一种方法重定向机制，它将基程序中的方法调用重定向到指定层中的相应方法。动态层添加框架通过对象组合和代理来

实现方法调用的重定向,这与 COP 语言中使用语法块实现重定向的方式有所不同,与基于建模的自适应方法[13]也不同。

假设方法调用为 $O.\,\mathrm{m}()$,其中 O 是类 C 的对象。为了将方法调用重定向到层中的相应方法,运行系统首先从配置文件类 C 所在的区查找到适合当前上下文的层,然后加载其中的层化类,实例化该类,产生对象 O_p,最后组合对象 O 和 O_p,生成一个组合对象,记为 $O::O_p$。这样,方法调用的接受者由对象 O 变为组合对象 $O::O_p$。

代理(delegation)与顾问(consultant)存在一些类似之处,为了更好地为基于上下文的方法调用[8]建模,本文使用了代理,而不使用顾问。代理是指调用对象 A 的方法会触发对象 B 的同签名同返回类型的方法,但在执行中,该方法的 this 变量仍指向 A 而不是 B。运行系统在执行 $O::O_p$ 上的方法调用时使用代理:先扫描组合对象中的部分对象,如果发现一部分对象有在当前上下文可执行的方法 m,则执行该方法,实现方法重定向,同时将该方法的 this 变量指向组合对象 $O::O_p$。运行系统将代理应用于组合对象的方法调用是为了使得程序更好地感知上下文[8]。

方法调用重定向到哪个层中的方法在运行时才能确定,因此当配置文件被扩展,新层和它的层化类被动态添加时,运行系统仍可以通过对象组合和代理将方法调用重定向到新添加的层中方法,成为正在运行程序的一部分,从而实现层的动态添加。

2 实现方法

扩展 Java 编译器和虚拟机实现编程框架的核心功能,是将层融入到 Java 编译器,使框架支持新的语法结构,将对象组合和代理融入到 Java 虚拟机,以实现层激活和动态层添加。扩展后的 Java 编译器和虚拟机不仅支持标准 Java 程序的编译和执行,而且支持层激活和动态层添加,根据当前上下文,实现程序可动态调整其行为。

2.1 Java 编译器的扩展

框架的编译器负责程序的编译和配置文件的检查,将类、层和层化类编译为标准的.class文件,是 JastAddJ[14]的一个扩展。采用扩展 JastAddJ 的扫描器和词法分析器可使关键字 layer 被正确识别;新模块添加到 JastAddJ 的抽象语法树和语义分析模块可使层和其中的类被编译成标准 Java 字节码文件,将层中的 predicate 方法编译成静态方法。层中的类与基类同名,编译时需重命名,以避免名字发生冲突。

编译时每个方法都赋予一个访问属性值,用于标记该方法的访问权限。为了使运行系统方便地识别基类中被层化类重写的方法,扩展后的编译器在编译时先给这类被重写的方法赋予一个特殊的访问属性值,该值应保留该类方法原有的访问属性。

2.2 Java 虚拟机的扩展

为了支持对象组合和代理确保标准 Java 指令仍被正确执行,应选择扩展 JamVM 虚拟机,该虚拟机符合 Java 虚拟机规范(第 2 版),而且小巧紧凑、易于扩展。

2.2.1 *对象结构* 用对象链表统一表示对象、部分对象和组合对象,因此对象组合操作表示链表的插入操作。为了能正确地执行代理,当前正在执行的方法中来自于链表的那个对象需要记录下来,其用链表对来实现。扩展对象链表是一个对象链表对,其中对的第二元表示对象链表本身,第一元表示对象链表的一部分,链头记录当前执行方法所属的对象。

扩展 JamVM 虚拟机中的 Object 结构体经扩展后的数据结构如图 2 所示,其中指针 l 和指针 p 为新添加的域,l 仅指向由部分对象组成的链表,p 指向另一链表。扩展后的 Object 将 3 种类型的对象统一表示为对象链表对:当表示对象和部分对象时,Object 中的 l 和 p 分别指向空和对象本身;当表示组合对象时,Object 中的 l 指向下一个部分对象,p 指向组合对象本身,即链表的第一个元素,该元素的 p 在运行时可指向链表中的部分对象。扩展 JamVM 的空间分配模块功能为:在创建对象时,相应的 l 和 p 分别指向空和对象本身。

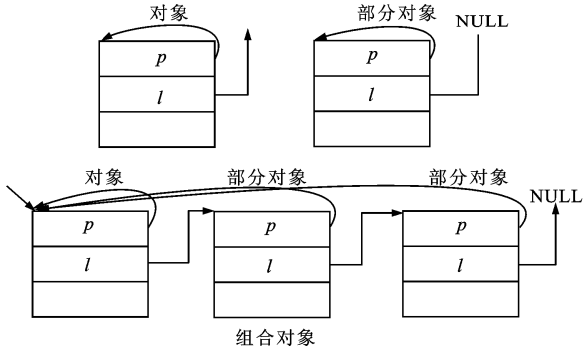

图 2 扩展对象的数据结构

2.2.2 *层激活和动态层添加* 层激活是一种方法重定向机制。JamVM 虚拟机的解释器负责执行方

法调用，因此扩展解释器将对象组合和代理融入到方法调用的执行中，实现方法的重定向，以支持层激活。

方法调用的执行流程如图 3 所示，图中左边为标准方法执行流程，右边为扩展后方法调用的执行流程。在执行方法的调用指令时，如 invokevirtual 指令，首先解析模块被调用，然后执行模块执行被解析的方法。被调用方法被层化类重写，方法的重定向需要应用对象组合和代理来实现。当被调用方法被解析时，虚拟机首先检测方法调用的接受者，即当前 this 指针指向的值。如果 this 的值是一个标准对象，则被调用方法正常执行；如果 this 的值是一个组合对象，并且被调用方法被层化类重写，则虚拟机首先调用部分对象查找模块（findpartobj）用于从 this 指向的值的 l 中查找所希望的部分对象，即该对象拥有与被调用方法同签名同返回类型，且是在当前上下文中可执行的方法。如果发现了这样的部分对象，虚拟机则在该对象的类中解析被调用方法，然后执行被解析的方法，执行时虚拟机将对象的 p 指向的值赋予被执行方法体中的 this。根据扩展后的对象结构，p 指向的是它所在的组合对象，即当前方法调用的接受者。因此，this 指向的也是方法调用的接受者，这与代理的语义一致。

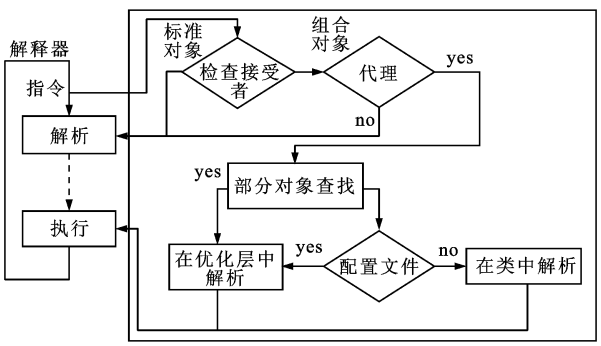

图 3　方法调用的执行流程

如果 findpartobj 没有发现符合要求的部分对象，虚拟机则从配置文件中查找适合当前上下文的层，然后调用虚拟机的类加载器加载其中相应的层化类，实例化该类，产生组合对象，该对象与当前方法调用接受者进行组合，产生新的组合对象。虚拟机在刚加载的层化类中解析被调用方法，最后执行被解析了的方法。同样地，被执行的方法体中的 this 指向新组合对象，因为配置文件可以动态地修改，新添加的层及层化类能够按需加载并通过对象组合和代理成为正在运行的程序的一部分，这样程序能够更有效地根据上下文的改变来调整行为。

如果 findpartobj 无法从配置文件中找到适合当前上下文的层，为了避免运行程序中断，虚拟机则在基类中解析被调用方法，然后执行被解析的方法。每个组合对象都由一个对象和一组部分对象组成，可确保虚拟机总在基类中成功地解析被调用方法。

3　实　验

在测试评估中，测试工具基于 Java Grande Forum Benchmark Suite[15]（JGFBS），并以文献[3]为例开发了一组微基准测试程序，以用于测试 Java 原方法（plain method）和层中方法调用的执行效率。基准测试程序包含实例方法调用、类方法调用、同步实例方法调用和同步类方法调用，每个方法都计算一给定数的幂，这些方法的定义都用标准的 Java 语言编写。测试计算机的 CPU 为 Intel Pentium Dual 2.2 GHz，内存为 3 GB，操作系统为 Ubuntu10.04 LTS，虚拟机的堆的最小值设置为 128 MB，最大值为 1 280 MB。

（1）Java 原方法和层中方法的执行效率比较。方法重定向就是将方法调用重定向到层中的相应方法。用基准测试程序测试了不同类型的 Java 原方法和层中方法的执行效率，结果如图 4 所示。

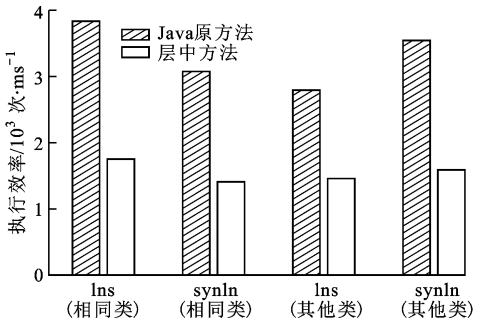

图 4　不同类型的 Java 原方法和层中方法的执行效率比较

图 4 结果表明：不同类型的层中方法的执行效率比 Java 原方法低大约 47%～54%；执行非同步实例的层中方法的执行效率比 Java 原方法低大约 54%。这是因为代理和对象组合需要额外的开销，其中执行代理需要 2 次额外的方法调用和 1 次方法表的查询操作。

（2）层中方法的执行效率比较。文献[3]是通过层感知消息传递（layer-aware message dispatch）来实现方法重定向的，在每次执行层中方法之前需要执行 3 次额外的操作和 1 次层列表遍历；本文是将对象组合和代理融入到虚拟机来实现方法重定向

的。以文献[11]的方式对上述 2 种方法的层中方法的执行效率进行了比较,结果如图 5 所示。图 5 结果表明:层感知消息传递在同一类中的层中方法的执行效率大约是 Java 原方法执行效率的 31%,在其他类中的层中方法的执行效率大约是 Java 原方法执行效率的 50%,该结果与文献[3]测试结果类似;2 种方法在其他类中的层中方法的执行效率基本相当,层感知消息传递在同一类中的层中方法的执行效率低于本文方法。

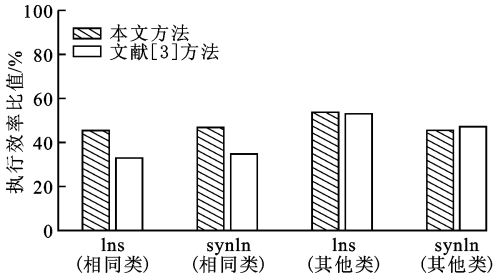

图 5 2 种方法的层中方法与 Java 原方法的执行效率比值

层感知消息传递效率低的原因,是其采用扩展编译器时需 3 次额外的方法调用和 1 次层列表遍历,这些需转化为相应的 Java 语句,虚拟机在执行了这些语句对应的指令后才能查找到相应层中的方法。本文方法采用的是扩展虚拟机,虚拟机可以更直接的方式找到相应层中的方法。此外,层感知消息传递无法优化执行层中方法[3],这也是在同一类中层中方法的执行效率比 Java 原方法低近 70%的原因。

扩展编译器实现方法重定向的效率较低,但这种方法使得被编译的应用程序能够运行在标准的 Java 虚拟机上,具有较好的跨平台应用。本文方法通过扩展虚拟机实现了重定向,并与 Java 应用程序兼容,但无法提供垮平台应用,这也是今后的研究工作,即如何将所提方法融入到中间件之中,以提供跨平台特性[16]。

4 结 论

针对当前基于编译器实现的 COP 语言无法支持动态层添加的缺陷,提出了一种以虚拟机为核心的方法,利用虚拟机自身已有的功能支持动态层添加,通过对象组合和代理来实现方法的重定向和层激活。测试结果表明,本文方法能够有效提高方法重定向的执行效率,由此而开发的编程框架支持程序行为的动态调整和层的动态添加,指导如何开发编程框架来支持类 Java(Java-like)应用程序的动态行为调整和动态层添加。

参考文献:

[1] KICZALES G, LAMPING J, MENDHEKAR A, et al. Aspect-oriented programming [C]// Proceedings of the ECOOP. Berlin, Germany: Springer-Verlag, 1997: 220-242.

[2] HIRSCHFELD R, COSTANZA P, NIERSTRASZ O. Context-oriented programming [J]. Journal of Object Technology, 2008, 7(3): 125-151.

[3] APPELTAUER M, HIRSCHFELD R, HAUPT M, et al. ContextJ: context-oriented programming with Java [J]. Journal of the Japan Society for Software Science and Technology on Computer Software, 2011, 28(1): 272-292.

[4] KAMINA T, AOTANI T, MASUHARA H. EventCJ: a context-oriented programming language with declarative event-based context transition [C]// Proceedings of the 10th International Conference on Aspect-Oriented Software Development. New York, USA: ACM, 2011: 21-25.

[5] ENGINEER B, JORGE V. Interruptible context dependent executions: a fresh look at programming context-aware applications [C]// Proceedings of the ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. New York, USA: ACM, 2012: 67-84.

[6] SUBRAMANIAN S, HICKS M W, MCKINLEY K S. Dynamic software updates: a VM-centric approach [C]// Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM, 2009: 1-12.

[7] MALABARBA S, PANDEY R, GRAGG J, et al. Runtime support for type-safe dynamic Java classes [C]// Proceedings of the ECOOP. Berlin, Germany: Springer-Verlag, 2000: 337-361.

[8] BETTINI L, BONO V, VENNERI B. Delegation by object composition [J]. Science of Computer Programming, 2011, 76(11): 992-1014.

[9] ZHAO Y L. Granule-oriented programming [J]. ACM SIGPLAN Notices, 2004, 39(12): 107-118.

[10] 赵银亮, 朱常鹏, 韩博, 等. 一种利用适合性测试支持方法重定向的演算 [J]. 软件学报, 2013, 24(7): 1495-1511.
ZHAO Yinliang, ZHU Changpeng, HAN bo, et al. A calculus using fitness testing for method redirection [J]. Journal of Software, 2013, 24(7):1495-1511.

[11] APPELTAUER M, HIRSCHFELD R, HAUPT M, et al. A comparison of context-oriented programming languages [C] // Proceedings of the International Workshop on Context-Oriented Programming. New York, USA: ACM, 2009: 1-6.

[12] KNIESEL G. Type-safe delegation for run-time component adaptation [C] // Proceedings of the ECOOP. Berlin, Germany: Springer-Verlag, 1999: 351-366.

[13] 唐蕾，周兴社，於志文，等. 普适环境下自适应行为实体的研究与实现 [J]. 西安交通大学学报，2011，45(2)，102-106.
TANG Lei, ZHOU Xingshe, YU Zhiwen, et al, Research and implementation of adaptive entity under ubiquitous computing environment [J]. Journal of Xi'an Jiaotong University, 2011, 45(2): 102-106.

[14] EKMAN T, HEDIN G. The Jastadd extensible Java compiler [C] // Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. New York, USA: ACM, 2007: 1-18.

[15] BULL J M, SMITH L A, WESTHEAD M D, et al. A methodology for benchmarking Java Grande applications [C] // Proceedings of the ACM 1999 Conference on Java Grande. New York, USA: ACM, 1999: 81-88.

[16] DELICATO F C, SANTOS I L A, PIRES P F, et al. Using aspects and dynamic composition to provide context-aware adaptation for mobile applications [C] // Proceedings of ACM SAC. New York, USA: ACM, 2009: 456-460.

（编辑　苗凌）

（上接第 5 页）

probability for physical layer security [J]. Journal of Applied Sciences, 2011, 29(4): 368-373.

[2] 洪涛，宋茂忠. 一种基于天线虚拟运动的跳空扩频调制方法 [J]. 南京航空航天大学学报，2011，43(4)：481-485.
HONG Tao, SONG Maozhong. Space hopping spread-spectrum modulation based on a virtual motion antenna [J]. Journal of Nanjing University of Aeronautics & Astronautics, 2011, 43(4): 481-485.

[3] 殷勤业，贾曙乔，左莎琳，等. 分布式多天线跳空收发技术：I [J]. 西安交通大学学报，2013，47(1)：1-6.
YIN Qinye, JIA Shuqiao, ZUO Shalin, et al. A distributed multi-antenna space hopping transceiver technique: I [J]. Journal of Xi'an Jiaotong University, 2013, 47(1): 1-6.

[4] 穆鹏程，殷勤业，王文杰. 无线通信中使用随机天线阵列的物理层安全传输方法 [J]. 西安交通大学学报，2010，44(6)：62-66.
MU Pengcheng, YIN Qinye, WANG Wenjie. A security method of physical layer transmission using random antenna arrays in wireless communication [J]. Journal of Xi'an Jiaotong University, 2010, 44(6): 62-66.

[5] AONO K, HIGUCHI T, OHIRA B, et al. Wireless secret key generation exploiting reactance-domain scalar response of multipath fading channels [J]. IEEE Transactions on Antennas and Propagation, 2005, 53(11): 3776-3784.

[6] WILSON R, TSE D, SCHOLTZ R A. Channel identification: secret sharing using reciprocity in UWB channels [J]. IEEE Transactions on Information Forensics and Security, 2007, 2(3): 364-375.

[7] LI Xiaohua, HWU J, RATAZZI E P. Array redundancy and diversity for wireless transmissions with low probability of interception [C] // Proceedings of 2006 IEEE International Conference on Acoustics, Speech, and Signal Processing. Piscataway, NJ, USA: IEEE, 2006: 525-528.

[8] LI Xiaohua, HWU J. Using antenna array redundancy and channel diversity for secure wireless transmissions [J]. Journal of Communications, 2007, 2(3): 24-32.

[9] WANG Jintao, JIA Shuyun, SONG Jian. Generalised spatial modulation system with multiple active transmit antennas and low complexity detection scheme [J]. IEEE Transactions on Wireless Communications, 2012, 11(4): 1605-1615.

[10] 穆鹏程，殷勤业. 空谱的描述与定义 [M] // 10000 个科学难题：信息科学卷. 北京：科学出版社，2011：449-451.

（编辑　刘杨）