计算机科学 1999Vol.26№.12

Java语言实现安全性的原理与技术*)

Principle and Implementation of Java Security

吴旭东 陈 翀 麦中凡
(北京航空航天大学计算机科学与工程系 北京 100083)

Abstract Java program language, with feature of platform independent and portable, is suitable for building distributed application in Internet environment. Java provides a set of security mechanism to protect host against malicious code. In this paper we analysis the principle of Java security model and constitution of it's architecture, and expatiate key implementation technology and deficiency. By comparing with requirements in distributed application, We put forward some extension for Java security.

Keywords Java language, Access control, Security service, Protected domain

1. 引言

Java是Sun Microsystem公司开发的新一代面向对象的程序设计语言,其最主要设计目标之一是平台无关性以及可移植性,能很好地解决国际互连网的异构问题。随着以WWW为代表的Internet的迅速崛起,Java语言已被广泛接受,成为主流程序设计语言之一。

Java语言的程序被编译后,生成的是Java独有的字节代码(bytecode),以class文件的形式存在,其格式独立于平台,通过语言本身的动态装入机制,可以在运行时从网络的任意节点下载、执行,因此语言本身支持分布式环境。字节代码运行在Java虚拟机上,它屏蔽了各种平台的差异性,使Java实现了平台无关性。

Java的兴起最初是因为可以在WWW网页内插入独立的可执行内容,而无须考虑浏览器所处的操作系统和处理器平台。然而Web技术发展至今,已不仅仅用于信息发布,Java也不再仅限于作为applet扩充html使web网页增加交互性。利用平台无关性构建基于WWW的分布式应用,是Java今后的主要发展方向。如通过servlet扩充web服务器端功能,构建基于Internet,易于操作、升级,功能强大的企业级信息系统;结合分布式对象技术,利用Javabeans技术实现面向构件的软件生产、发布方式,加强系统的灵活性和开放性等等,这些新的发展方向大大丰富了Java的应用背景。

由于Java支持移动代码,主机面临受外来代码攻击的危险,由此带来了与传统程序设计语言有所不同的安全性问题。本文首先讨论了系统安全机制的一般要求及Java语言的安全特性,然后介绍了Java语言的安全模型和实现的体系结构,并指出其不足,最后提出了若干扩展方向。

2. 系统安全机制的构成

为了保护系统资源和用户的利益,在多用户环境下的计算机系统需提供一定的安全保障机制。它由两个要素组成:计算环境的可靠性和系统服务的安全性[2]。

计算环境的可靠性要求系统内代表不同主体运行的计算单元(如进程或线程)之间保证数据的私密性和执行的互不干扰。在集中式系统中主要表现为内存保护机制。内存保护是指确保程序的执行不可访问未经授权的内存空间,访问其中的数据或代码。对软件实现而言,内存保护可以在操作系统一级实现,通常是对进程的控制。主要方法包括运行时孤立(isolation)软件故障,或在程序装入执行前静态校验,证明程序描述的行为是安全的。内存保护也可以由程序设计语言实现,即所谓的"安全语言"。它在语言层抽象定义出内存访问控制区域,如变量或对象,并通过可见性规则和类型系统实现对该区域访问的控制。类型安全的实现同样

*)本文得到国家自然科学基金项目(69673003)"可移动的软件Agent研究"资助。吴旭东 硕士研究生,主要研究领域为软件Agent,计算机系统安全。陈 翀 硕士研究生,主要研究领域为软件工程,软件Agent。麦中凡 教授,主要研究领域为软件工程,程序设计语言,面向对象方法学,数据库,分布式智能软件系统等。

可分为动态或静态的方式。动态的类型检查实现简单，但性能会有所降低；静态的方案只需在编译时做一次类型检查，但需要证明语言的类型安全性，以确保静态类型检查足以防止非授权访问。

系统服务的安全性是在计算可靠性的基础上实现的，通过使用者身份的识别和访问控制防止对系统服务和资源的非授权访问。传统的访问控制大都由操作系统实现，访问控制粒度是进程，安全策略作用于各自独立的地址空间。

在分布式环境下，进程间的通信可能跨越主机系统，计算环境的可靠性一般通过扩充安全的数据传输通道保证，系统服务的安全性则通过扩充分布式身份认证机制实现。

Java 语言的安全模型依据系统安全机制的要素构成，它独立于操作系统，通过 Java 虚拟机的安全特性和 JDK(Java Development Kit)提供的访问控制模型实现主机的内存保护和服务的安全性，并支持数字签名和数据加密功能保障分布式环境下的系统安全。

3. Java 语言安全性模型

3.1 安全语言特性

所谓"安全语言"，是指程序设计语言的类型安全性。类型安全性由语言的类型规则保证，具有两个作用。首先，它确保程序运行时变量的值始终与声明的类型一致，在函数或方法调用时形参与实参的类型匹配。其次，它通过某些关键字(如 private，protected)定义了代码的可见性范围。Java 的类型系统包括编译时的静态类型检查和动态装入时的类文件校验以及 Java 虚拟机的强制类型转换系统。其类型规则构建有较为成熟的类型安全理论，并取消了指针，加入了定局(final)类和方法。所有这些都保证了 Java 作为安全语言的特性。从实现中稍做简化的 Java 语言模型已被证明是类型安全的[3]。

在面向对象的语言环境中，类型安全有更重要的意义。面向对象的安全模型建立于类型安全的基础之上，权限的实现通过对象来表示，获取了对象就等于获取了它所代表的权限，也就获取了对应资源的操作能力。在 Java 语言中，可见性最高层次以包为单位来划分，除了声明为 public 的类以外，其他类在包外是不可见的；同时，Java 限制 cast 操作并取消了指针，不能通过直接对内存访问和类型转换来非法获取对象引用。因此，创建并使用对象的唯一途径是通过 new 操作符，使得资源保护可以通过对象的构造函数来实现。

3.2 访问控制模型

Java 提供内在的访问控制模型来保证系统资源和运行环境(包括运行代码本身)不受到外来恶意代码的侵犯，并覆盖了以下几个可能受到攻击的目标：内存、操作系统及程序状态、客户文件系统、网络资源。由于语言具有可扩展性，代码在运行时动态装入，在一个线程中可能出现不同主体，其控制域的粒度是类，需要动态确定运行的主体实现访问控制。在 JDK1.0 中，本地代码有访问全部系统资源的权限，而远程代码在 sandbox 内运行，sandbox 是指为来自于网络的远程代码提供的受限运行环境，使其不能访问本地文件系统和网络连接等资源，如建立主动或被动的网络连接(除其源主机以外)是不允许的。在 JDK1.1 中(见图 1)，扩充了"签名 applet"的概念，如果携带数字签名的 applet 经过解释器校验，确认为来自信任主机，则与本地应用程序具有同等的权限。该模型易于实现及使用，但很不灵活，没有实现动态的访问控制。

JDK1.2 使用了更小也更为灵活的访问控制粒度——保护域，其安全模型如图 2。新的安全模型同 1.1 版相比，有了一些变化。首先，安全性限制不只存在于 sandbox 之中，对本地 Application 和远程代码均可配置安全策略，本地代码不再默认为安全的；其次，策略与代码脱离，存在于配置文件中，易于动态修改，扩展。在这个模型中包括如下概念：

1)主体(Principal)：被授予系统内某种权能的实体。在 JDK1.2 中，通常用 CODEBASE(path＋classname)和用户签名来标识并需经数字签名校验才能被系统承认。

2)被保护域(Protected domain)：域是受安全策略控制的类代码集合。需要授权才可访问其他资源的域为被保护域，每个类有且仅有一个被保护域，由它代表的主体决定。

3)许可(Permision)：被保域内主体对资源访问的句柄。

4)资源及其目标(resource/target)：资源是许可对应的对象，目标是由主体与操作组成的二元组，表示对资源的使用方式。

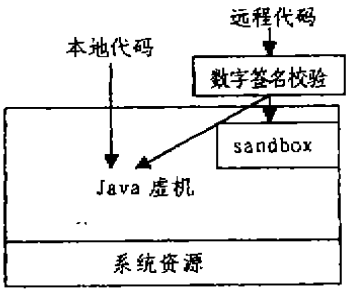

图 1　JDK1.1 安全模型

访问控制模型如图 3。主体是本地或远程类代码，

可以动态配置的安全策略是访问控制的依据，权限本身代表对资源的访问。

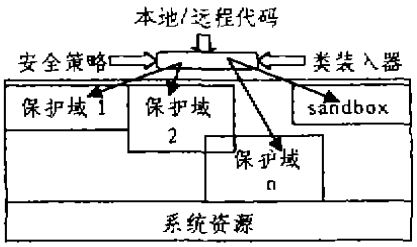

图 2　JDK1.2 的安全模型

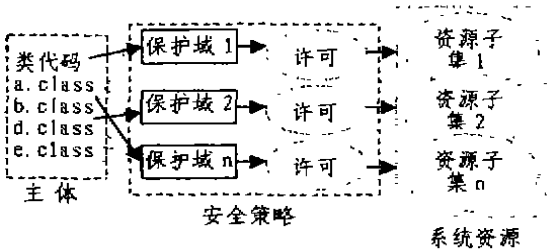

图 3　访问控制模型

3.3　安全策略实施原语

由于系统的移动代码特性，访问控制要实现两方面的功能：动态（运行时）确定代码来源；根据它实施安全策略。

Java 虚拟机有两个特点。首先，任何类均由类装入器读入运行系统，且每个类均包含对其类装入器的引用。所有来自于本地文件系统的类与来自网络的类由不同的类装入器装入，可以据此判断类的来源。其次，虚拟机调用栈的结构包含了对方法对应类的引用。这两个特点使得运行时可以判断当前代码来源于本地或网络。

在 JDK1.1 中，并不支持动态安全策略。在 JDK1.2 中，安全策略通过调用栈的操作来实现。在这种方式中，涉及三个对象：主体、策略引擎（policy engine）和目标（见 3.2 节）。当系统需要对某个资源进行保护时，在中心注册数据库中建立该资源对应的入口，并根据安全策略登记允许的目标。主体与资源通过调用栈操作的 3 个原语访问策略引擎：enablePrivilege (target)；disablePrivilege(target)；checkPrivilege(target)。

对资源的访问通过两个步骤进行：1）对目标资源进行 enablePrivilege 操作，策略引擎访问中心注册数据库，如果符合安全策略，将该权利写入调用者堆栈。调用返回后，进行 disablePrivilege 操作；2）受保护资源被使用前，进行 checkPrivilege 操作搜索调用堆栈，如未发现对该资源操作的权利，则返回异常。

4. Java 安全环境的实现

4.1　安全体系结构

安全体系结构的实现如图 4 所示，它包括以下几个部分：

1）Java 语言层的安全性。为了保证构建于 Java 之上的系统安全，首先应保证语言本身的安全性，其他的程序设计语言，如 C++，并不保证类型安全性，这使它在表达能力和执行速度上有所增强，同时以牺牲可靠性为代价。Java 语言可以保证类型安全性（见第 2 节）；

2）编译器。它的功能是进行强类型支持（类型检查）和动态类型转换安全检查；

3）Java 字节码校验器。确保字节码的合法性，如检查堆栈的上溢和下溢，检查寄存器存取的合法性，检查字节参数的正确性；

4）Java 运行系统。包括类装入机制，内存管理器和线程管理器；

5）类装入器（Class Loader）。定义了一个独立的本地名空间，使得不同的装入器之间的类不可见；提供策略网关（*strategic gateway*），禁止重载任何固有的 Java 类；

6）SecurityManager 是 Java 虚拟机与外部环境，如支持 Java 的网络浏览器的接口机制。通过对 SecurityManager 的编程，可以定制 applet 或 Application 的安全策略，以限制系统资源的访问；

7）可执行的代码以及它的允许使用权限，它们构成执行内容。

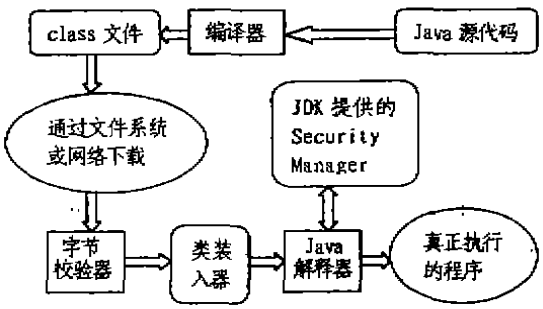

图 4　Java 的安全体系结构

在以上几个环节中，编译器、校验器是 Java 环境提供的固有安全机制，对所有的 Java 程序是相同的，而 ClassLoader 和 SecurityManager 是可定制的，它们支持资源的访问控制，sandbox 及控制域就是在此基础上实现的。

4.2　SecurityManager 类及实例

SecurityManager 是 JDK 中的一个抽象类，构造

安全策略时只要重载其权限控制方法。System 类提供了访问当前安全管理器的方法，并在未定义时设置缺省的安全管理器。当某些操作不符合安全管理器的限制时，系统将抛出异常。在 JDK1.2 中，类似的功能由 AccessController 实现，为了保证兼容性，仍然支持 SecurityManager。

4.3 类装入器

类装入器分为两种：原始类装入器(Primordial Classloader)，它是 Java 虚拟机的一部分，从 CLASSPATH 指定的路径装入类文件，并且装入的类被认为是可信的，具有访问本地资源的最大权限；自定义类装入器(Custom Classloader)，它是通过派生系统提供的 ClassLoader 类来实现的。类装入器使系统可以在运行时从本地或远程动态装载代码，并配置外来代码的访问权限。一个 Java 虚拟机上可以存在多个类装入器，相互之间具有独立的名空间，它们之间不能进行直接的交互；同时，也可以针对它们配置安全管理对象，使其行为受到控制。

使用类装入器的典型例子是支持 Java 的浏览器。当主页中包含 .class 文件时，浏览器向 HTTP 服务器请求下载该类文件，用自定义的类装入器，通常叫做 appletclassloader，装入该类，并配置 SecurityManager，以实现对 applet 行为的控制，包括读写本地磁盘；建立网络连接(除源主机外)；产生新进程；重新装载动态链接库和调用本地方法；创建类装入器和重载 SecurityManager。

由于类装入器是 Java 平台安全性的一个环节，其实现要遵循一些安全原则，如拒绝装入包名与本地类仓库(Java API 或 CLASSPATH 路径下存在的类)的包名相同的类，对远程类文件装入之前需要进行字节码校验。

4.4 安全策略算法

Java 安全策略按"最小权限"算法设计，涉及到多个调用者的操作，其操作权限是由各个调用者被保护域的交集决定。对直接访问的情况，处理很简单，如果资源位于 Permission 的申请者的域外，则抛出异常，否则成功返回，如果访问请求不是由主体直接发出的，中间涉及到多个不同被保护域的主体，形成一种调用的"链"，则按以下算法决定是否许可：若链中的某个调用者不具有直接访问许可，则访问拒绝；除非一个调用者具有"特权"(Privilege)，且它以后的调用者均具有直接访问许可，详见[4]。

控制的实现有两种策略：1)主动求值：每当线程执行到不同的被保护域，立即动态更新当前拥有的许可。2)被动求值：每当需要进行许可检验时，通过当前线程的调用堆栈(call stack)回溯访问涉及的每个主体，进而根据上述算法确定许可。

JDK1.2 是用被动求值的方法实现许可检验的。如某个线程派生出子线程，在 JDK1.2 中子线程继承了所有祖先线程的运行环境，避免了调用"链"的不完整。在多个无继承关系的子线程间，如果有调用关系，也需将调用者运行环境的快照(snapshot)传递给被调用者，用来最终决定是否许可访问。

4.5 分布式环境安全服务

为了适应分布式环境下的安全需求，JDK 包含了对 Java 加密体系结构的支持，它完成如下功能[5]：

1)数据指纹：根据不定长输入数据产生定长输出数据，且该输出不能通过其它的输入得到，即满足双向的对应关系。该功能用于验证数据未被修改。

2)数字签名：基于公开密钥的身份认证算法。该签名不可伪造、不可抵赖。因此，签名＋公开密钥与数据具有一一对应关系，私有密钥＋数据与签名具有一一对应关系。

3)数据加密/解密：使数据对第三方不可辨认。通常基于对称加密算法。

JCA 的设计原则是算法独立性和实现独立性，使 JCA 的实现不基于特定的算法和实现机构，具体的实现选择可动态配置。为了体现以上原则，JCA 的 API 层次如图 5，核心服务类有：1)MeassgeDigest(数据指纹)；2)Signature，KeypairGenerator(数字签名及公私密钥对的生成)；3)encryption，decryption(由 JCE 扩展包提供，实现加密/解密)。

它们实例化时，可以选择具体的算法和提供者，后者对应的类由不同的提供者实现。

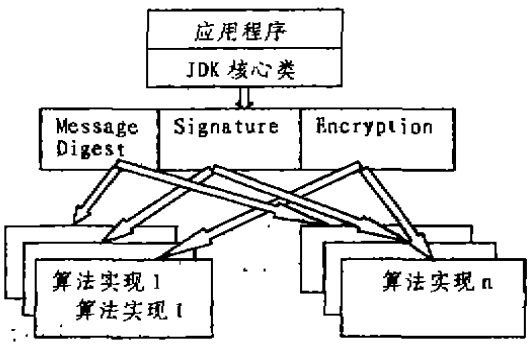

图 5 JCA 的 API 层次

4.6 安全代码发布

安全代码发布由 JAR 规范支持，它规定了 applet 加密、压缩数据存储的格式，是构建于 JCA 之上并扩展了 class 文件格式实现的，为移动代码的数字签名和身份认证提供支持。

5. 关于扩展的考虑

5.1 Java 安全性实现的局限性

对 Java 语言实现的分析可以看出，到目前为止(JDK1.2)其安全性是针对 applet 的，重点是为外来代码构造保护域，通过限制保护域对系统域访问来保护主机资源。在这种实现方式下，applet 对主机进行侵犯是困难的，但对于构造基于 Java 的分布式应用环境，有如下局限性：

1)主体的身份根据代码来源决定，且保护域基于代码而非实例构造。由此产生两个问题。首先，在实际应用如电子商务系统中，代码的生产者往往并不是代码的使用者，对不同的运行实例应根据使用者实行不同的安全策略，而非根据生产者指定，在目前的实现下不同的使用者无法区分。其次，即使可以区分使用者的不同，基于代码(类)构造的保护域无法对不同的实例(或运行时状态)分配不同的安全策略。

2)不支持权能(Capability)传递及委托授权(delegation)。这种缺点主要体现在扩充服务功能的情况。当外来代码作为服务功能的扩充而驻留在主机上时，这部分代码应被动态授予对系统的部分控制权以实施其功能，这包括授予第三方访问系统的权利。简单地改变该代码对应的主体的安全策略会导致两种情况：造成安全漏洞(该主机的其它代码有同样的权限)或使用的不便(扩充完代码后改变数字签名)。由于目前 Java 只支持被保护域和系统域之间的交互，外来代码所处不同的被保护域，相互之间不可见，因此目前 Java 的实现不能直接支持移动对象间的授权委托。

3)不支持资源管理。对服务没有数量的控制。一旦外来代码获得某种系统资源服务，如创建文件，则使用的磁盘空间在 Java 语言层无法限制。此外，由于设计上的局限性，虚拟机内所有的线程共用一个堆和线程调度器，对否定服务攻击(denial of service)的防范是困难的，如一个恶意的 applet 不断派生线程直至系统资源耗尽。

5.2 主要的扩展方案

针对 Java 安全性的不足，我们提出以下两个扩展方案：

1)支持权能及权能传递，实现基于对象实例的安全策略。权能是指向系统资源的不可伪造且受系统控制的指针。基于权能的访问控制在许多系统中都有实现，如 IBM System/38 操作系统，Java 电子商务体系结构中的网关安全模型(JECF)[6]。支持权能有两个好处。首先，突破了 Java 固有的安全模式，即只能由系统限制外来代码，而外来代码之间不可见。权能的实现为外来代码之间的安全交互提供了基础，其次，由于权能可以动态地生成和授与，权能可以实现基于对象实例而非类代码的访问控制。

2)基于角色构造保护域。Java 的保护域是基于主体的，实际上代表代码的制造者，不能完全反映出代码运行所代表的身份。在面向对象的设计领域中，角色(Role)是封装后的属性和行为的实例化表示，是功能和权利的抽象并与实现无关[7]。基于角色构造保护域是安全性扩展的第二个方法。在这种方法中，角色代表执行者，可以被动态地分配给主体，并具有相应的特权(privilege)，在这里，特权是预先分配的操作权限。由于角色更完整地反映了运行代码的身份信息，角色的授予成为安全策略的一个环节，可以实现更完整的访问控制；同时角色是功能的抽象，基于角色结合现有的认证协议，如 Kerberos[8]，可以构造更可靠和灵活的身份认证服务。

小结 面向分布式环境构造基于 Web 的应用系统，已成为 Java 走向应用的重要方向。Java 现有的机制对安全性提供了支持，但存在一定的局限性。因此，充分利用 Java 的特点并进行扩充以实现安全的应用环境是必要的。本文分析了 Java 安全体系结构原理与实现，并提出了基于权能和角色的扩充方法。

参考文献

1 Wallach D S. Extensible Security Architectures for Java; [Technical Report 546-97]. Department of Computer Science, Princetion University, April 1997

2 Dean D. The Security of Static Typing with Dynamic Linking. In: The 4th ACM Conf. on Computer and Communications Security. Zurich, 1997. 2～4

3 Java Security Architecture. Available at: http: // www. sun. com/products/jdk/1. 2/docs

4 Java Cryptography Architecture. Available at: http: // www. sun. com/products/jdk/docs/guide/security/CryptoSpec. html

5 The Gagteway Security Model In The Java Electronic Commerce Framework. Available at: http: // www. javasoft. com/

6 Lea D. Roles Before Objects. Available at: http: // goswegoe. edu

7 Neuman B C. Kerberos: An authentication Service for Computer Networks, IEEE Communication

8 Volpano D, Smith G. Language Issues in Mobile Program Security, Mobile Agents and Security. In: G. Vigna, ed. Volume 1419 of Lecture Notes in Computer Science. Springer Verlage, 1998. 25～43