摘 要

对于分布式应用资源的大量安全攻击来说，访问控制是重要的防范手段。现存的中间件访问控制技术存在这样几个问题：1）不能提供一个完整的解决方案，应用开发者不得不在应用系统中嵌入访问控制功能，这带来了大量的问题，包括实施困难、高开销、高错误率、资源所有权的归属等；2）就有效性来说，在一些应用领域，访问控制还要更细致，在鉴权判定方面对使用专门因素的支持，还有广大组织之间鉴权的一致性和可靠性的增强；3）中间件越来越异构化，相互作用方式复杂。在使用这些中间件时，要了解其安全特性，比如，机密性数据不会在网络上泄漏，但是，现在很难证实这些中间件有很好的安全特性。

针对中间件对访问控制不能提供完整的解决方案以及其有效性问题，本文提出了一个体系结构方法：基于 J2EE 的 RBAC 支持。首先，通过详细分析了 J2EE 访问控制机制的性能，设计了能够正式定义系统状态的 J2EE 保护系统配置。通过该定义，本文设计了一个鉴权判定算法。其次，本文研究了在进行 J2EE 安全服务时，对基于角色的访问控制模型的支持。利用 J2EE 保护系统定义的配置，用 J2EE 语言定义了 $RBAC_0$ 和 $RBAC_1$ 两种模型。此外，为了支持 $RBAC_0$-$RBAC_3$ 模型，本文分析并给出了实施 J2EE 安全服务所需的必要条件。本方法在遵循 J2EE 安全规范的前提下实现了对 $RBAC_0$ 的支持，并实现了 J2EE 安全规范范围以外的必要功能，来支持 $RBAC_1$ 和 $RBAC_2$。这加强了 J2EE 访问控制机制的性能。

J2EE 中间件访问控制机制与 RBAC 模型相结合的方法比较好的解决了完整性和有效性这两个问题，但是对于复杂策略，它们还存在灵活性差、粒度粗和扩展性较弱等问题。本文设计了鉴权服务体系结构（JCRADS），用它来解决分布式资源的访问控制问题。JCRADS 抽象了鉴权服务中的鉴权逻辑，该鉴权服务位于应用程序的外部，独立于下层的安全模型和安全规则。这种资源访问判定办法解决了保护大多数分布式资源所面临的问题。对于鉴权判定，允许使用多种类型的主体安全，服务结构允许使用从工作流和其它资源中获得的信息，这样它就能支持应用领域所特有的规则。它也能够使用在访问控制判定中的特殊信息。鉴权逻辑包装为一个独立的服务，可以很容易地在穿越应用边界时保持访问控制策略的一致性。此外，这个结构支持多重鉴权模型，并且它有助于安全管理员和应用开发者进行清晰的责任分离。

尽管当前的设计利用了 J2EE 所遵循的安全机制，JCRADS 几乎独立于下层的安全技术。JCRADS 方法能够应用到大多数分布式环境中去。这种结构不但简化了应用程序和安全系统开发，而且也允许管理部门一致地管理和实施其安全策略，很好的解决了分布式资源访问系统普遍存在的不够灵活、粒度粗、可扩展性差等主要矛盾。

为了在国家科技基础条件平台部署JCRADS，本文设计并配置了一个测试床－基于JCRADS的JAS，以便给出一个灵敏的、可扩展的，并且轻便的鉴权服务体系结构，以及测试使用JCRADS会产生什么样的性能。JAS附着于JCRADS服务，并且作为对JCRADS方法研究的一个框架。

针对不安全中间件的问题，本文重点研究了封装器的方法，它可以让这些中间件在安全的环境下（封装器）执行，它可以为中间件之间、中间件和其它系统资源之间提供细粒度的控制。本文使用calculus设计了四种封装器：过滤封装器、日志封装器、管道封装器、n元封装器，并探讨了各种封装器能够保证的安全性。

关键词：分布式，访问控制，安全，中间件，封装器

Abstract

Research on the Key Issues of Resource Access Control under Distributed Environments

Zhang Fangzhou

Resources access control (AC) is an essential defensive measure against a large number of security attacks. Under distributed environment, use of middleware technology is one common method to support access control, but it also incurs some problems to be overcome for the method to be fully effective. 1) It does not offer a complete solution, as application developers often have to resort to embedding AC into application code components. As a result, AC functionality is intertwined with application logic, making implementation difficult and inefficient with high error rates and unclear ownerships of resources. 2) For security protection to be effective in many application areas, AC requires fine granularity of control, use of application specific information in authentication and in granting resource access, and enhancement of authentication uniformity and reliability among multiple different organizations. 3) Middleware components are increasingly heterogeneous in structure and complicated for inter-component communication. When middleware components are used, one must be aware of their security characteristics. For example, one must know if there is danger of data leak in the network. However, it is difficult to prove if middleware components have good security functions.

In order to improve on the completeness and effectiveness of middleware access control, this thesis proposes a new architectural method: J2EE based RBAC support. First, we analyze in detail the workings and performance of J2EE access control mechanism. We then provide a design on the J2EE protection system configuration that allows formal definition of system states. The thesis proposes a design of an authentication algorithm in terms of such definition. Second, this thesis studies how to support Role Based Access Control model when providing J2EE security service. Using J2EE protection system configuration, we give a definition of $RBAC_0$ 和 $RBAC_1$ in J2EE language. We also give a description and analysis of necessary conditions for implementing J2EE security service to support $RBAC_0$-$RBAC_3$s models. Our proposed methods provide $RBAC_0$ support while conforming to the J2EE security requirements, and additional functionality beyond J2EE security requirements, in order to support $RBAC_1$ 和 $RBAC_2$. All of our work greatly enhances performance of J2EE Access Control mechanisms.

The combination of RBAC model and J2EE middleware access control mechanism works well to solve the problem of incompleteness and ineffectiveness. However, in the case of complicated security strategies, the combination still has problems of low flexibility, coarse granularity, poor scalability, etc. This thesis work includes a design of an authentication service system structure (JCRADS), to solve various access control issues for distributed resources. JCRADS extracted the logic of authentication service, and places the

authentication service outside of application code, thus making it independent of security model and rules in the lower layers of an application system. This resource access control method solves many issues incurred in protecting distributed resources. It allows multiple types of security strategies for authentication, and its service structure allows use of information obtained from work flows and other resources. As such, the authentication service system can support special rules in different specific application areas, as well as utilization of specific information for authentication. Wrapped up as an independent service, the authentication logic can easily maintain access control strategy consistency when it works across different application boundaries. In addition, this structure allows multiple authentication models and can assist security administrators and application developers in clearly separating their responsibilities.

Although our design makes use of security mechanisms used by J2EE, JCRADS is almost independent of lower level security techniques. JCRADS can be used in many different distributed environments. The structure not only simplifies application and security system development, but also enables an administration staff to consistently manage and enforce its security strategies, therefore solving problems of low flexibility, coarse granularity and poor scalability that often exist in distributed resource access control systems.

For deploying JCRADS on the Nationality Science and Technology Basic Platform(国家科技基础条件平台), the thesis work includes an implementation of a test-bed based on JAS of JCRADS. This implementation has the properties of agile, flexible, extensible and portable authentication service structure, and plays the role of a test bed for JCRADS performance in actual systems. With JAS affiliated to JCRADS service, the implementation provides a framework to study the JCRADS method.

One of the major works of this thesis is to study wrapper method for unsecured middleware components. This wrapper method allows middleware components to be executed in a secured environment. Wrappers are used to provide fine granularity control between middleware components and between such components and system resources. Several wrappers are implemented and described by using box-calculus: filter wrapper, logging wrapper, pipeline wrapper and n-element wrapper. The thesis provides a discussion on how security can be ensured by the use of such wrappers of various types.

Keywords: Distributing, Access Control, Security, Middleware, Wrapper

图目录

图 2.1 计算机安全的主要概念 5
图 2.2 参考监视器 6
图 2.3 中间件和应用系统访问控制的关系 7
图 2.4 JAAS Policy 词条的例子 13
图 2.5 鉴权处理和在基于 DCE 应用系统管理中的 ACL 管理 15
图 2.6 DCOM 中间件（摘自[Microsoft 1998]） 16
图 2.7 DCOM 鉴权策略的等级和它们的范围 17
图 2.8 SESAME 组件 18
图 2.9 CORBA 安全性中的策略实施 19
图 2.10 GAA API 模型中事件的顺序 20
图 2.11 策略代理 21
图 2.12 代理和拦截器 23
图 2.13 鉴权服务器 23
图 2.14 鉴权相关的交互 26
图 3.1 执行上下文的创建 33
图 3.2 J2EE 安全中的域和策略 34
图 3.3 J2EE 访问控制机制中关键元素间的关系 34
图 3.4 角色层次的例子（[Sandhu 1998b]） 43
图 3.5 EngineeringProject 接口 44
图 3.6 Employee 接口 44
图 3.7 EngineeringProject 接口层次 45
图 3.8 多域解决方案中域的层次 48
图 3.9 接口实例域成员 50
图 4.1 客户、应用系统和 JCRADS 服务之间的交互流程 54
图 4.2 JCRADS 构件之间的交互 56
图 4.3 一个交互流程的例子 56
图 4.4 主要元素及其附件 57
图 4.5 管理元素及其附件 58
图 4.6 JCRADS 体系结构的计算部分 58
图 4.7 角色层次（角色、层次间的关系，RH） 64
图 4.8 JCRADS 的基于角色策略的配置 65
图 5.1 JAS 主要组成部分 70
图 5.2 JAS 结构体系 71
图 5.3 J2EE 对象的实现 71
图 5.4 用策略方式实现服务器 72
图 5.5 模板的应用 73
图 5.6 JAS 组件的通用结构 73
图 5.7 参数配置 74
图 5.8 JAS 配置 74

图 5.9 DC 的设计 75
图 5.10 PE 的设计 76
图 5.11 JAS 体系结构 77
图 6.1 测量性能时间 80
图 6.2 信息穿过的界线 81
图 6.3 相关模型和 JAS 配置 83
图 6.4 各种 JAS 配置的响应时间增长% 86
图 7.1 三元中间件封装器 98

表目录

表 3.1 认证后的主题所具有的安全属性..... 35
表 3.2 需求权限矩阵..... 35
表 3.3 授予权限矩阵..... 35
表 3.4 每个主题授予的权限..... 36
表 3.5 每个主题允许的操作..... 36
表 3.6 访问矩阵..... 38
表 3.7 RBAC 与 J2EE 概念对照表..... 40
表 3.8 单域解决方案的需求权限矩阵..... 46
表 3.9 单域解决方案授予权限的矩阵..... 46
表 3.10 多访问策略域解决方案需求权限矩阵..... 48
表 3.11 多访问策略域解决方案接口实例域成员矩阵..... 49
表 3.12 多访问策略域解决方案授权矩阵..... 50
表 4.1 访问控制规则..... 63
表 4.2 资源记录条目..... 63
表 4.3 用户—角色的分配关系（UA）..... 64
表 4.4 许可—角色的分配关系..... 65
表 5.1 JAS 和 JCRADS 扩展的 IDL 间的关系..... 70
表 6.1 应用请求与 JAS 推荐配置的关系..... 86

声　　明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。就我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：　　日期：2006.04.30

关于论文使用授权的说明

中国科学院计算技术研究所有权处理、保留送交论文的复印件，允许论文被查阅和借阅；并可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存该论文。

作者签名：　　导师签名：　　日期：2006.04.30

第一章 引言

由于 Internet 的高速发展和普及，软件系统目前日益综合和互相连接。这样大范围内的综合，由自治的、异构的并且分布式的系统组成，称为分布式软件系统。每个组织内的应用系统可能是独立开发的，基于不同的设计和技术。国家安全、工业、商业、科技等国家重大项目越来越依赖这些系统的功能。

由于 Internet/Intranet 上分布式系统和信息资源互联的越来越庞大，越来越复杂，设计保护这些系统和资源的安全机制也变得越来越复杂和难以实现，这也成为每一个系统最核心的东西。

安全信息组织的问题一直集中在从行业来的巨大努力，因此，几个著名的用于网络、操作系统、数据库管理系统和中间件系统的安全系统体系结构和模型被开发出来用于配置分布式环境的可升级和灵活的安全，这表现了重大的进步，然而它只是达到目标的第一步。余下的问题还有：复杂的、更细粒度的安全策略的处理；不仅支持应用系统以及它们底层平台内的变化，而且支持在业务流程和安全策略的变化，还要支持用户群体和他们的角色的变化；支持企业应用的动态配置，而不影响安全的完整性；达到需要的性能。

本文考虑一个特殊的安全功能——访问控制（Sandhu 1994），对于针对信息组织资源的铺天盖地的安全攻击来说，它是一个必要的防范手段。然而，对应用资源的访问控制越来越需要更细的粒度和在鉴权判定中支持使用针对应用的一些因素。而且它还必须一致地、可靠地在企业应用间执行范组织的鉴权策略。

当前已有的网络、操作系统、数据库管理系统和中间件技术对于完成这样的控制是不适合的，并且以后也永远不会，这是因为它们是为一般目的的用途而设计的，它们的控制太粗糙，且只涉及到某些资源[CIST-NRC1999]。因此，应用开发者借助在应用系统嵌入访问控制功能的手段来支持复杂的、细粒度的、上下文依赖的鉴权策略。

访问控制功能和应用逻辑的耦合带来了大量的问题。系统安全管理员需要逐个应用的配置访问控制逻辑，疲于奔命[Beznosov 1998a]。这个基于应用的多点访问控制使得系统安全管理变得非常可怕，代价昂贵且错误百出[Beznosov 1997, Wilson 1997]，这样很难改变安全策略和控制机制，很难开发、改变和动态重配置应用软件[Beznosov 1999b, Grimm 1999, Hale 1999]。

为了解决保护应用资源的问题，构建应用系统的方法需要改变，而且，在分布式计算环境中，能以有效的方法对系统进行开发、整合和管理。

1.1 研究目标

本文提出了一个体系结构上的方法来提高和正规化访问控制机制的能力，用来定位对资源的访问控制问题。这个方法是双重的。一方面，使用 J2EE 安全机制提供的访问控制机制开发了一个实现基于角色的访问控制（Role Based Access Control，RBAC）模型的框架，对于那些 J2EE 控制粒度和 RBAC 模型的能足够表达的应用领域，这个框架解决了上述的问题。另一方面，本文开发了一个鉴权服务的体系机构，它解决了对分布式资源访问的控制问题：当中间件访问控制机制中的粒度和对复杂策略的支持不适合，应用开发者在他们的系统中嵌入附加的访问控制功能。

中间件越来越异构化，它们以复杂的方式相互作用。这些中间件中，有一些是从网络上下载的，不是十分的可信。用户想知道一些中间件拥有的安全特性，比如，机密数据不会在网络上泄漏，但是，现在很难证实这些中间件有很好的特性。取而代之的是，可以让这些中间件在安全的环境下执行，这个环境，称之为封装器，它可以为中间件之间、中间件和其它系统资源之间提供细粒度的控制。本文重点研究了封装器的表示方法，通过这种方法来阐述及严格证明其安全性。本文使用 calculus 描述了几种封装器：过滤封装器、日志封装器、管道封装器、n 元封装器，探讨了各种封装器能够保证的安全性。

1.2 主要研究成果

为了保护分布式应用及其资源，中间件技术提供的安全性非常重要和必不可少，因此，为了充分的利用它们，使用中间件访问控制机制来模型化鉴权策略的方法非常重要。本文定义了一个 J2EE 保护系统状态的配置，使用这个语言，本文设计了一个 J2EE 的鉴权判定算法。这个配置和鉴权判定算法一起在数学上定义了 J2EE 安全鉴权系统的状态和行为。

利用先前定义的 J2EE 保护系统的配置，给出了 J2EE 安全服务如何支持 RBAC 模型。在 J2EE 安全语言中提供了 $RBAC_0$ 和 $RBAC_1$ 的实现，而且，研究和分析了实现 J2EE 安全服务支持 $RBAC_0$－$RBAC_3$ 模型的需求。为了支持 RBAC 模型实现，本文设计了兼容 J2EE 安全规范的方法。这个工作通过最大化 J2EE 访问控制机制的效用来更好的展现它的能力，这一点对于使用中间件来保护系统资源是至关重要的。

第二个主要贡献是提出了基于 J2EE 的资源访问判定服务（J2EE-CORBA Based Resource Access Decision Service，JCRADS），一个新颖的配置鉴权机制的体系结构方法，这个鉴权机制在功能上适合在鉴权判定中使用针对应用的信息来保护更细粒度的应用资源，它允许应用与鉴权逻辑的分离，这样就使得应用开发、部署和管理更加有效。而且它能保证多个应用间策略的一致性。本文通过模型化鉴权策略的手段展示它的功能和性能，这些鉴权策略需要使用这样的针对的应用的信息作为用户和资源拥有者之间的联系。

通过实现基于J2EE资源访问判定服务的原型系统（J2EE-Based Antetype System，JAS），本文获得了一些有关基于JCRADS的鉴权服务设计的心得。另外，本文展示了JCRADS体系结构的主要特征，比如灵活性、可配置能力和可扩展能力，是如何使用J2EE中间件来实例化的。开发JAS的经验为基于JCRADS服务的设计提供了指导性的方针。

使用JAS作为测试床，本文获得了JAS构件的不同组件性能的定量评估。并发现，使用执行时间和鉴权请求的数量以及性能约束，JAS配置可以提供需要的性能。

本文研究了封装器的表示方法，通过这种方法来阐述及严格证明其安全性。本文使用calculus设计和描述了几种封装器：过滤封装器、日志封装器、管道封装器，并且以这三个封装器为基础，描述了n元封装器，探讨了各种封装器能够保证的安全性。

1.3 论文内容

下一章给出了访问控制主题的背景信息，解释了主要的概念和术语，并且介绍了分布式访问控制的范围。接下来，详细的论述了本文的问题定义。然后，定义了一个评价现存技术、相关工作和本文方法的框架。最后，本章提供了一个观点和技术的分析，可以看出，对于保护分布式系统的资源来说，现有的中间件技术非常重要和必须，但是，它们也有自己不充分的地方。本章还回顾了研究领域的一些相关工作。

第三章提出了J2EE保护系统的配置，它正规的定义了系统的状态，使用这个定义，设计了J2EE鉴权决定的算法，并且给出了使用J2EE安全怎么样来支持RBAC模型。

第四章介绍了JCRADS服务体系结构，并且通过访问控制策略的例子来论证JCRADS的功能的有效性。

第五章给出了JAS的设计，并且给出了JCRADS体系结构是如何使用J2EE来实例化的。

第六章讨论了使用JAS的性能试验，并且从结果中得出了相关结论。

第七章描述了4个中间件封装器，并证明了它们的安全性。

最后第八章讨论完成的结果和进一步的工作中应该解决的问题。

第二章 背景和相关问题研究

在详细的描述本文所定位的问题之前，非常有必要给出在计算机系统中访问控制有关主题的背景信息，解释主要的概念和术语，并且介绍分布式访问控制的领域。

另外，本文定义了一个标准，用来评价现存技术、相关工作和对本文提出的解决方案的分析。简要的说，这个标准是资源的粒度、针对策略的应用域支持、用于生成鉴权判定的可用信息的多样性、在鉴权判定中针对应用的信息的使用、多个应用间策略的一致性、对应用和环境机制变化的支持、性能和管理的可量测性等。

最后，本章对已有的和正在进行的研究工作进行了分析和研究，给出了它们的适用场景、优缺点等方面的情况。

2.1 背景信息和术语

现代软件系统的安全传统上都是通过保护和保证来实现的，如下图（图 2.1）所示。

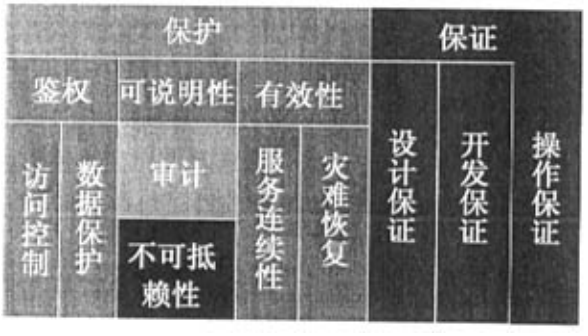


图 2.1 计算机安全的主要概念

保护机制通常由一些安全子系统/机制来提供，用来保护系统不受特定的威胁，一个威胁是发生在与计算机系统相关的资源上的、任何潜在的、会带来非预期的效果的事件[Amoroso 1994]。保护方式有一个前提：要能列出计算机系统内部可能发生的尽可能多的威胁，并且能建造一个能阻止这些威胁的机制[Blakley 1999]。保护机制可以分类为三个组：可说明性、有效性和鉴权，可说明性机制确保用户（或者他们执行的程序）——习惯上称为主题——能对他们访问系统资源和服务的行为负责。有效性机制确保服务连续性或者服务和资源中断后的恢复能力。鉴权机制确保执行使用系统资源和服务的规则。鉴权机制更有资格进行访问控制或者数据保护。当可以实现这些规则检查和执行的时候，访问控制机制允许系统拥有者执行这些规则。术语“鉴权”还暗含做出访问控制判定的过程。当规则检查和/或执行不可能时，使用数据保护机制。

传统访问控制机制的结构可用参考监视器的概念模型来说明，参考监视器是安全子

系统的一部分，负责仲裁主题到系统资源（客体）的访问，如下图（图 2.2）所示。

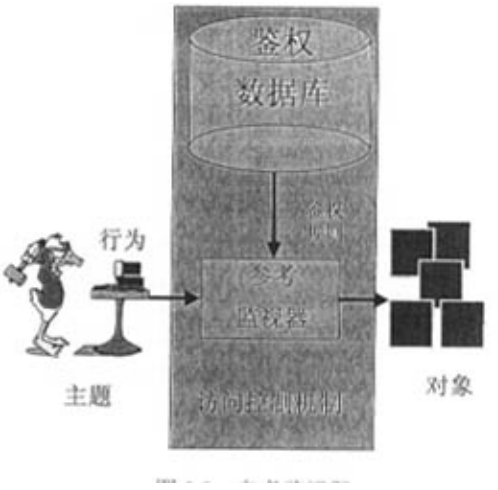


图 2.2　参考监视器

仲裁包括生成鉴权判定、根据鉴权数据库的鉴权规则检查访问请求、然后执行这些请求。一套规则有时被称为策略。通常根据主题－动作－客体体系结构，鉴权规则说明什么主题在什么客体上能执行什么动作。这些动作也称为访问权限。这样，根据定义在某个客体上的权限，如果一个主题能执行某个动作，它就拥有对这个客体的特定的访问权限。而且，所有鉴权规则就构成了访问矩阵[Lampson 1971]，其中每一行表示一个主题，每一列表示一个客体，矩阵中的每一个元素说明相应的主题对相应客体的访问权限。

为了生成鉴权判定，参考监视器把鉴权规则和三组信息作为输入，这三组信息是：

1）访问请求；

2）提出这个请求的主题；

3）要访问的客体。

因为要用它们来提出本文的问题、评价现有的方法和分析本文所做的工作，有必要讨论一下这三组信息的具体内容。

访问请求的信息常常和请求类型相关，例如读取文件的“读”请求。尽管如此，一些应用需要基于额外的请求属性的访问控制结果。比如如果访问超过自己权限范围以外的资源，系统可能拒绝此请求。

主题信息可以划分为两种类型——安全相关和安全不相关。最初，只有相关安全用于访问控制结果中。这种信息描述主题的特性、组织成员组成和其它的安全属性。有时候，本文会用特权属性来表示只用于访问控制的安全属性。

在一些应用领域，需要特别考虑主题的不相关安全信息。比如，访问公共图书馆的资料可以根据访问用户需要的页数来准许访问。另一个例子是工作流的信息处理。这个信息不受安全性和用户管理员控制，并且不会以安全属性的形式提供给参考监视器。这样，监视器就需要从其它渠道获得。客体信息的访问也可以划分为安全相关和安全不相

关。所有这些信息都用于评估鉴权规则。

取决于特殊访问控制机制的性能、主题、请求和目标信息的可用性，其有限的或者详细的信息可以用于得到鉴权判定。信息的可用性将作为评估访问控制机构表现力的一种评判标准。

2.2 应用资源的控制访问

应用资源可以以数据的形式来处理，应用本身、它们的服务（例如 Telnet [Postel 1983]， SMTP [Postel 1982] 或者 WWW）、针对它们的特殊操作（例如满足 HTTP 协议的 WWW 服务器上的 GET 访问请求）、甚至是应用接口都可以进行处理。

一些应用资源，例如文件、数据库记录或网络界面，会受到操作系统、DBM 系统或中间件系统的保护。然而，有一些只针对于应用的资源，并且它们不能被应用本身之外的任何程序识别，例如应用逻辑的特殊部分的运行。换句话说，特殊应用资源的粒度比一般用途的计算系统要好。图 2.3 阐明了中间件系统和应用层访问控制作用范围的不同，这也是应用层和一般用途访问控制的显著区别。

图 2.3 中间件和应用系统访问控制的关系

另一个明显的不同是应用层访问控制的鉴权规则需要用到访问操作、主题或者客体的信息，它们针对特殊的应用领域，或者说比一般系统的访问控制机制更有表现力。

2.3 问题陈述

本文阐述的中心问题是用体系结构的方法控制访问分布式应用及其资源的不充分性。为提高访问控制机制的充分性，它必须支持功能性需求。另外，这个机制上的体系与系统所在的信息体系必须互相支持。目前的方案之所以不充分，是因为它们或者策略保护细粒度应用资源功能性不足，或者不支持信息机构具体体系的目标，或者二者兼有。

首先，扩展信息组织体系结构问题及其起因的主题，并且为了支持组织体系结构，

本文分析了系统所必需的体系结构属性。然后，深入讨论在系统范围内控制访问资源的需求。最后，将国家科技基础条件平台的例子实例化，描述具体的访问控制问题。

2.3.1 信息环境机制前景

在庞大的背景下讨论任务的需求和有效性，将问题置于对资源的分布式访问控制中是很必要的。

在讨论这一层存在的问题前必须阐明信息组织的概念。直观上信息组织似乎是信息系统中的一种。虽然这样的描述不错，但是却远远不够严格。本文用下面这个更加精确的对信息组织的描述——“一种组织范围， 其上可以指定一套普通信息技术策略”。信息组织的技术范围定义为：对象、模块、模块收集、框架、程序、应用软件、系统、部门、信息机构和全球基础组织等。

信息组织体系结构（IEA）配置中遇到的主要问题是低语义兼容性、高重新联结和维护开销，并且随着部署的应用数量的增加而呈指数增加。另外，组织机构建模时间太长，又过时的太快。这个问题的主要原因是缺乏成功处理环境机制积累变化的有效方法；缺乏一个描述、分析和联结体系有效又精确的方法；体系失协；提取不当；对遗传、基于组件和多范例系统的支持性不高。主要约束是在应用级上数量和类别的改变，以及重用现存的信息基础组织的必要性。

很显然，一个机构的主要目标是满足功能性和非功能性请求。模板是它所支持的任务工作流。现在，任务工作流变化的越来越快，变化率从 20 世纪 70 年代和 80 年代的一个完整循环周期的 7 年缩减到 90 年代的 12 到 18 个月。另一个重要的目标是随着旧技术的消退和构成企业的系统的演进，系统能平滑的移植到新的技术。

网络访问控制

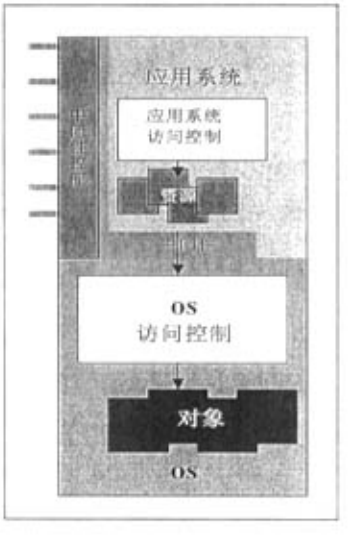


图 2.3 访问控制点

因此，本文建议在分布式环境下，一个系统的体系结构或者一个服务机能必须针对：

1). 减少与此系统或其它系统相关的变化数量；

2). 减少与此系统或其它系统相关的维修和重新联结所需开销；

3). 对部署的应用系统的数量的增加，能提供很好的测量方法。

例如，普遍用于分布式访问控制的方法在相应层次上进行资源保护，如图2.3所示。它们是网络（如防火墙）、中间件、数据库和操作系统等控制器。当有数以百计的应用系统和支持系统（如操作系统）时，使得所有这些控制器一起工作并保持整个范围内的访问控制策略的一致性是很困难的。这样的方法增加了鉴权策略和应用变化时的工作量，增加维护和重新联结的开销，且不能有效地测量应用数量的增加。

2.3.2 系统前景

系统级的主要问题是中间件访问控制机制不能对细粒度资源进行有效的保护，而且它们只提供有限的处理复杂策略的能力，而这些复杂策在一些应用领域中是需要的，例如国家科技基础条件平台。另外，也需要针对领域的一些因素（例如用户和资源提供者）用于访问控制策略。复杂性和粒度级别常常迫使应用设计者将针对领域的鉴权逻辑置于应用中。结果，这样增加了软件设计的复杂性，使之很难保证系统的完整性和质量，这也显著增加了系统管理和操作的难度，也增加了开销。

2.3.3 国家科技基础条件平台中的访问控制问题

前面的部分概述了分布式体系结构所面临的一般问题，建议了系统和服务应有的体系结构上的属性，也分析了对资源的访问控制应该具有的功能上的需求机制。现在借助科技部国家科技基础条件平台项目来讨论国家级的科技基础条件平台访问控制中的重要问题。国家科技基础条件平台项目是科技部 2003 年重大科技、应用项目，包括访问门户和全国 700 多个科技资源库。

2.3.3.1 介绍

国家科技基础条件平台[南 2004a]至少会在很长一段时间内是一种与众不同的环境。旧有计算技术和体系将与新的基于组件的系统以及新的中间件这样的其它技术和服务器共存。系统将一直调和新兴技术同即将过时的技术之间的关系。平台安全体系的主要目标是提供一个安全环境，平台用户可以一同浏览组件和资源，访问控制判定从针对平台机制的策略得到。[南 2004a]列出了令这个目标难以实现和维护的主要问题，提出了平台环境机制体系的设计思路，所以上述问题可以置于已有的约束领域中。

2.3.3.2 平台环境机制

国家科技基础条件平台是由大型科学仪器与实验基地、自然科技资源、科学数据、科技文献、网络科技环境、科技成果转化基地等物质与信息保障系统以及相关的共享制度和专业化队伍构成，是服务于全社会科技创新的数字化、网络化、智能化的基础性支

撑体系。

国家科技基础条件平台门户网站是门户应用系统的对外窗口，是科技资源信息发布、基于科技基础条件平台进行科普活动以及宣传科技基础条件平台建设成果的综合性、权威性科技站点，代表着国家科技基础条件平台在网络上的整体形象。

门户网站的主要功能需求包括[南 2004a]：

（1）发布科技信息，进行科普宣传，展示国家科技基础条件平台的建设成果；

（2）提供个性化服务功能；

（3）多终端统一接入功能；

（4）提供站点模板生成服务；

（5）提供用户管理功能。

国家科技基础条件平台门户应用系统的建设为我国的各种科技资源提供了一个集中统一的访问入口，将各种科技资源的应用、信息与流程有机地结合起来，为国家的科技基础条件平台提供统一的基础架构，实现统一的系统管理、端对端的安全架构。国家科技基础条件平台是由异地、异构的多个科技资源结点通过整合和互联而构成的。在门户系统的建设中，资源访问控制和用户身份认证系统是必不可少的重要组成部分。

统一身份认证和资源访问控制系统的实现，能够较好解决这一问题。国家科技基础条件平台应用系统的认证管理工作，将全部由该系统来承担和实现，从而达到一次注册即可通用的目的，实现了信息管理与系统安全的有效集成。门户应用系统通过对用户的统一身份进行认证管理的研究与开发，整合现有的资源系统，并为以后新建资源系统的集成创造规范的程序，使所有的资源系统在统一框架下得到高效、稳定而安全的管理，为国家科技基础条件平台建设提供必要的保障机制，增强科技资源的开放程度，促进科技资源共享。

建设统一身份认证和资源访问控制系统的主要功能需求[南 2004a]是：

（1）建立一套完整的国家科技基础条件平台分级认证体系，这样有利于实现用户的分级管理，使得不同的用户就对应于不同的权限，既能提高用户使用系统的安全性，又能简化用户的管理流程；

（2）建立统一身份认证中心和各子平台和资源认证系统，使用户不用频繁登录同一个系统，又保证用户的 ID 和密码以及其它身份相关信息得到更好的保护；

（3）设计门户认证中心和各子平台和资源认证系统的认证接口规范，根据该规范，使得平台用户认证系统实现统一化、正规化，减少系统开发和维护的成本，提高系统的易用性，并且使得各资源实体能够比较容易的接入到平台中来，提高系统的可用性；

（4）设计合理的资源访问控制机制，首先要有高效的访问控制机制；其次我们要求该访问控制机制要有足够的安全性，让用户、平台以及资源站点都能有足够的安全感，放心的去使用/提供资源或服务；第三，该资源访问控制机制要有易用性，尽可能的减少用户操作流程，减少门户应用系统的负担，尽可能的少改动或者不改动资源站点的现有

环境和机制；

（5）采取基于角色的资源访问策略，因为用户群数量巨大而类型比较鲜明，准确定义用户、角色、权限三者之间的关联映射，根据这个特点，可以把用户分成不同的角色来实行资源访问策略，准确定义用户、角色、权限三者之间的关联映射；

（6）针对分布在各子平台上大量多样化的资源建立有效的权限管理等支撑服务；

（7）考察各个资源实体的具体类型，按具体的环境把资源接口和策略进行分类，并分别对每一类接口和策略提供相应的参考实现。

2.3.4 问题描述

分布式环境中资源访问控制的中心问题有两种类型：功能性的和体系的。功能性是1)资源的粒度；2)专用于环境机制的任务域策略的执行；3)基于访问主题、访问操作和目标安全相关/非相关信息的判定。最后必须确定多种应用中策略执行的稳定性。

体系方法必须有效地支持平台系统的发展，也就是现有应用的变化，包括插入、删除、任务处理和安全策略里的变化、硬件/软件平台中的变化等等。在操作系统、应用系统以及它们包含的环境机制的发展过程中，这些性质需要以合理的开销来实现。

2.4 评估标准

在提出对问题的解决方法之前，本文定义了一个框架，用来评估现有技术和相关任务，以及分析本文的方法。

任何现有的或者已经被提议的解决方法都应该在适应性基础上加以评估。因此，问题陈述是主要的评估源。特别是针对以下问题：

1. 受保护资源的粒度。如果一项技术或者解决方法不允许在细粒度资源上的鉴权判定，那么它就不可以用于保护资源。本文采用下面的粒度层次：应用程序、接口、方法、专有资源。
2. 支持专用于应用域的策略。所支持的访问控制模型和策略有一个很宽的范围。一端的访问控制机制只能支持一个模型，另一端是允许执行任何鉴权逻辑的方法和只由逻辑接口限制的策略。一般的，一个机制可以轻松的支持更多的访问控制策略类型。
3. 用于取得判定的信息的种类。可行信息是有限的，例如有些技术仅允许获取主题的经过鉴定的特性，而不是它的组织成员信息或者个化性角色信息，因为它们基本上限制了基于这种技术的访问控制机制的功能。本文研究了什么信息是可行的，什么信息可用于自动化判定。
4. 专用信息的作用。专用信息是只有应用过程中客户要求时才可行的信息，它们的作用对一些应用域(比如科技基础条件平台)是很重要的。如果一个方法不允许这样的信息作用，那么保护资源的完全自动化是不可能的。

5. 多重应用中策略一致性的支持。前面讨论过，在平台环境下，一致性策略执行的问题是很重要、很关键的。通过检查可行的和已经提议的方法，本文将进一步考虑宽度一致性访问控制策略执行的支持。
6. 应用程序的插入和删除，策略的变化和计算环境的支持。无论资源的访问控制在功能性上如何完美，如果它调节这些变化的效率十分低下，那么它对环境机制的设定是没有用的。大多数可行方法都在某种程度上支持这些变化。本文将评估这种支持。遗憾的是，没有任何客观的定量标准来确定支持度。这也就是为什么本文以这种标准来进行本方法和其它方法的比较。
7. 方法的可测量性。执行和管理的可测量性很大程度的影响了方法的效用。无论有多少优点，如果一个方法不能很好的测量，它就只能是一个理论的演习。既然没有什么基准可用于评估访问控制方法的可测量性，就使用常识来推论出可测量性。比如，检查需要修改的数据的数量来调节策略的变化。另一个常用的测量手段是采用通讯复杂性，它仍旧被认为是分布式系统履行的主要因素。

2.5 已有的研究工作

把处理鉴权逻辑作为软件系统独立组件的想法已经存在很久了。参考监视器的抽象模型[Anderson 1972]就是在应用系统外形成和执行鉴权判定的典型事例。这一概念从有计算机安全意识的早期就一直应用于操作系统的访问控制（Access Control，AC）设计中。大多数操作系统在它们核心的安全部分实现了鉴权逻辑。[Benantar 1996, Curry 1992, DEC 1989， Gligor 1986，Grampp 1984，Heydon 1994，Hommes 1990，Karger 1991，Luckenbaugh 1986，McCauley 1979，McInerney1999，Mullender 1990，Pfleeger 1989，Quarterman 1985，Saltzer 1974，Walker 1980]。

这部分将重新考虑性能并且讨论在分布式软件应用中 AC 的主要技术。通过前面几节描述的标准评价它们的状况。理想情况下，所有安全功能都应该设计在应用系统之外，即所谓的“安全性不可知”。这就是为什么如果分布式安全技术能够在应用系统外实施 AC，还要经常检查的原因。

通常，有两类技术用于给分布式系统提供安全。一类技术是仅提供部分认证和通信保护，并且 AC 与基层通信独立。它们是 Kerberos [IETF 1993, Neuman 1994a]，GAA API [Ryutov 2000a]，SESAME [Kaijser 1998, Parker 1995]。系统开发者通过其它的方法（例如，ONC RPC [Bloomer 1992]）来传递内部应用信息。但是，这加重了开发者整合底层通信技术的负担。

另一个类型是中间件技术，例如 CORBA [OMG 1996b], DCE [Gittler 1995], Java [Lai 1999]，and DCOM [Microsoft 1998]，它们与安全子系统一起提供底层通信基础设施，与上一种技术接合的更好，这样开发者就可以合理的享用它们二者的集成。而且这种技术使访问控制完全独立于应用系统之外。

2.5.1 JAAS

Java 2 平台的发布引出了一个新的安全体系结构[Gong 1997]，它使用对连续代码判定访问许可的安全策略，使用与代码相关的因素来作鉴权判定，例如代码来自哪里的、通过谁等。这样以码源为中心的访问控制模式不同于常规计算环境支持的以用户为中心的鉴权策略。Java 最近已经广泛应用于不同用户运行相同代码的分布式应用系统中。Java 验证和鉴权服务（JAAS）提供了一种结构和标准接口程序，用于验证用户和鉴权用户。使用 JAAS，应用可以提供以码源为中心、用户为中心、或者是两种类型结合的认证。

作为使用 JAAS 认证机制的一个应用，它需要：

1）建造一个安全的 java 类，允许在问题中描述资源，

2）查找策略对象，

3）通过策略获得代码和主题许可：getPermissions()，

4）如果返回的许可包含请求，作出判定。

因为 JAAS 具有灵活的机制来定义特殊应用的资源，通过 Java 安全模型允许的访问权实现，JAAS 支持任何级别的资源粒度，与普通的 Java 类相同，Java 许可有安全性许可。例如 java.io.SocketPermission 是和图 2.4 [Lai 1999]中所列的都有联系。

```
//JAAS principal-based policy
grant
Codebase "http://cnic.cn",
Signedby "cnic",
Principal cnic.Principal "ark"
{
permission java.io.FilePermission "/jctjpt/ark", "read";
permission java.io.SocketPermission "*", "connect";
}
```

图 2.4 JAAS Policy 词条的例子

有一些预先定义的许可：file，socket，property, runtime, AWT, net, reflect, serializable 和 security。定义新许可子类来执行新类型的许可，包括特殊的申请。

JAAS 模式通过抽象类 java.security.Policy 定义了鉴权机制的一般概念。这个类的主要方式，即 getPermissions(subject,codesource)，返回一个许可集合，集合中 subject 表示带有特权的主题，并且执行码来自 codesource。Policy 的子类可以实行不同的鉴权策略。因此，语句 getPermissions()实施主要限制条件。JAAS 提供一个缺省的子类 Policy-File，它支持根据码源进行鉴权判定，代码签字人的身份认证，和主题拥有的特权属性值。这些都被用做判定特殊资源的访问许可。它不用局限于 JAAS AC 模式，能以各种各样的方式实现，这个鉴权属性使得 JAAS 具有很大的灵活性。因为 Policy 是一个抽象类，并且它的主要方法 getPermissions()能够用不同方法实现，所以 JAAS 不强制遵循任何特别的鉴权模型，这使得 JAAS 可以支持针对具体应用领域的特定策略成为可能，而且可以

用 Policy 实例来实现性能和测量管理。

JAAS 对于不同的认证因素的支持有很强的扩展能力。特权属性不只限制到预定义的范围。新的属性能通过新的 Java 类很容易的定义。而且，JAAS 支持特权属性的组合分级，这对于属性之间实施有关系的访问控制模式非常重要，例如 RBAC。另一方面，即使语义上属性相同，如果它们作为不同类来实现，JAAS 也会认为它们是不同的，这就容易造成混乱。

因为用于认证判定的 Policy 实例必须圈定在本地应用，JAAS 没有明确支持多个应用间鉴权判定的一致性。但是，不排除实施远程服务的代理鉴权判定的策略。

JAAS 体系结构相对适合于应用、策略鉴权和计算环境中的变化。通过 Policy 对象的替换，能调整策略的变化。正如与适应计算机环境中的各种变化一样，Java 的动态下载机制允许应用的增加和删除。

因为 JAAS 体系结构被定义成 Java 抽象类和接口的集合，允许各种不同的测量能力的实现，只能分析接口到其部件所限制的可测量性。然而，处理一个请求只需要一个鉴权判定，当签字人改变时，返回主题的全部许可好像效率不高。一个更可调节的解决办法将是只返回鉴权判定的结果。

2.5.2 分布式计算环境

开放软件基金会（OSF）分布式计算环境（DCE） [Kong1995]，是 RPC 基础设施，是在多台机器上支持分布式应用的综合服务的集合。DCE 安全服务提供的功能包括：用户认证；为保护在 DCE 基础设施上应用系统之间的通信而提供的安全数据传输；为应用鉴权；进行基于密钥的用户鉴别和应用，以便传达能相信彼此身份的信息。

DCE 安全服务基于 Kerberos[IETF 1993, Neuman 1994a]，它基于密钥技术来完成用户和应用系统的鉴权，这样通信各方就能信任这个标识。DCE 通过给服务器传送附加的权力属性来扩展 Kerberos。

这个服务不能控制对应用或者应用资源的访问，并且期望 DCE 应用来完成和提供对于它们自身的鉴权策略的可管理的访问。这样，应用必须实现访问控制功能，包括一个访问控制列表（ACL）管理器和一个 ACL 库，如图 2.5[CasWell 1995]中所示。

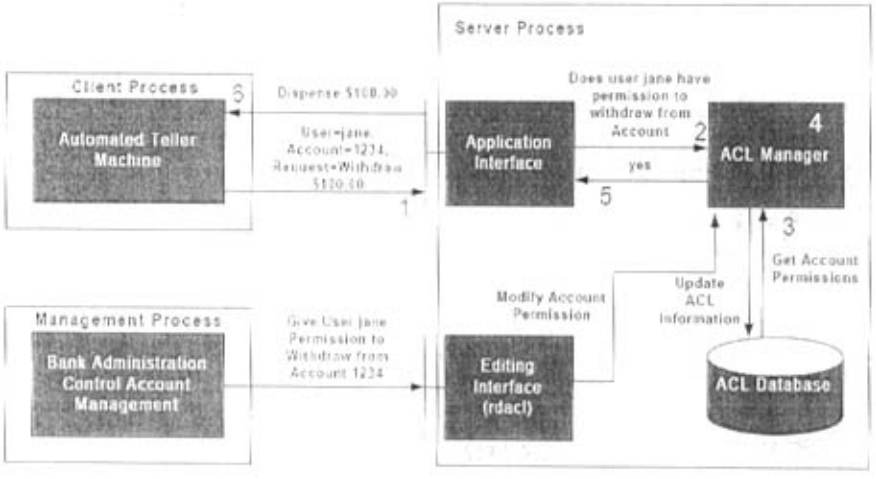


图 2.5 鉴权处理和在基于 DCE 应用系统管理中的 ACL 管理

如果一个应用系统使用 DCE ACL 模型进行鉴权，它通过 OIDs 把 ACL 与资源联系起来，由 ACL 管理器来确定正确的 ACLs。因此，OIDs 使资源的任何粒度成为可能。

DCE 安全服务给应用提供初步的帮助来做出 AC 决定，并且它对应用外部没有实施 AC。比较它的前任，Kerberos，它通过使用属性类型而不是使用主题身份而优化了特权属性的管理。但是，DCE ACL 语言的可表达性受到了极大的限制，并且不能确定如果利用 DCE ACL 机制，在鉴权判定过程中，如何使用针对应用的因素。似乎客户或者服务器人数的增加将不会严重影响 DCE 的系统性能，因为 AC 决定使用本地数据。不过，除非集中数据库，管理可量测性是不够的，因为策略变化必须反映到所有应用系统的 ACL 数据库上。如果这样性能可测量性又将受到影响。

2.5.3 Microsoft 分布式部件目标模型

分布式部件目标模型（DCOM）[Rubin1999，Grimes1997，Microsoft 1998]是微软公司的一项中间件技术，为了支持不同的、运行 MS Windows 操作系统的计算机上的 COM 对象之间的通信，这个技术拓展了部件目标模型（COM）。图 2.6 中显示了 DCOM 中间件技术。DCOM 协议，被称为“对象 RPC”或者 ORPC，扩展了标准 DCE RPC 协议。

因为 DCOM RPC 来源于 DCE RPC，所以它的安全模型与 DCE 的安全模型非常相似。ACL，类似于在 DCE 里的语言，用来编码鉴权策略。在 DCOM 里，命名为自主 ACL （DACL），表示对象拥有者有修改 DACL 入口的默认权限。DACL 可以使用 DCOMCNFG 配置工具或者按部就班地使用 Windows NT 注册和 Win32 安全功能来配置。在 DCOM 对象之外实施策略的能力，以及策略层次的存在，就比 DCE AC 模型具有相当多的优势，安全环境没有实施任何控制，应用系统必须自己实现。

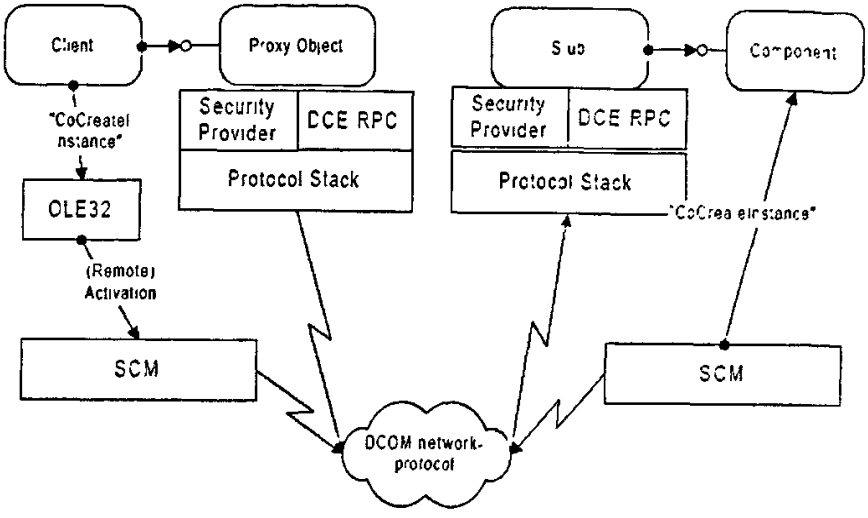


图 2.6 DCOM 中间件

DCOM 为控制应用系统及其资源的入口提供两个选择 [Eddon 1999]。由于“公布的安全性”，DCOM 可以不通过与对象或者对象的调用者协作而实施访问控制；用于应用系统的策略可以在外部配置并实施。

另一个，“纲领性的安全”，DCOM 通过安全 API 向开发者展现它的安全基础设施，以便客户和对象都能为任何粒度的资源实施它们自己的针对应用的鉴权策略，并且可以使用任何信息作为鉴权判定的输入。纲领性的安全能在注册表中覆盖默认和部件安全设置。

图 2.7 显示了 DCOM 鉴权策略的层次：

1）部件实施行为中嵌入的策略；

2）针对过程的公布；

3）针对主机的策略。

在调用被发送到对象方法之前，实施策略2 和3。在这个层次，内部策略按以下方式覆盖外部策略：在引用方法执行之前，如果有过程范围内策略的语句，覆盖主机范围内相应策略的语句；如果允许引用，它将被发送到方法进行实施，这将能执行它自己拥有的访问控制策略（策略3）。

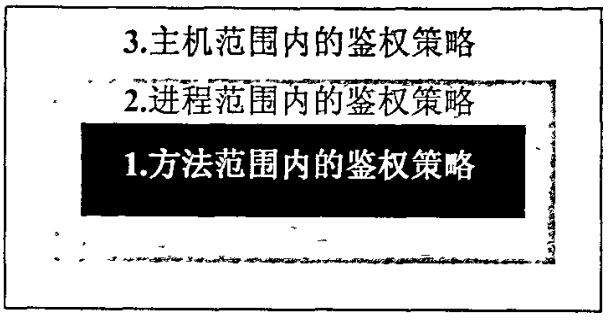


图 2.7 DCOM 鉴权策略的等级和它们的范围

对鉴权模型重大的阻碍是“针对部件”策略的粒度（图 2.7 中的策略 2）。这个粒度级别是操作系统过程，并且在不同的对象方法中相同。即在系统进程实现的对象上的策略使用相同的 DACL 控制对全部对象和方法的访问。

过程范围和主机范围内的策略（类型 2 和类型 3）隐含地引入了进入 DCOM 对象访问策略域的观念。令人遗憾的是，对象的划分可能只根据它们的位置而不根据它们的敏感性或者其它参数。把鉴权策略局限在主机边界领域内限制了基于 DCOM 的分布式应用管理的可测量性，因为它必须在每台主机甚至是每个进程上单独执行。

当公布访问控制时，不能使用针对应用系统的信息或者实施针对应用的策略。公布的鉴权策略及其变化必须在逐个机器上进行管理，这严重阻碍了管理的可测量性，并且除非应用系统位于相同的主机，否则不能自动保证穿越应用边界时策略的一致性。

2.5.4 SESAME

SESAME（Secure European System for Applications in a Multi-vendor Environment）是一个欧洲研究与开发工程。这项技术 [Kaijser 1998，帕克 1995]定义了提供根基的安全体系结构的部件，在此之上使用这个体系结构定义的下列服务可以建造完全可管理的安全产品：认证、鉴权、机密性、完整性和审计。

SESAME 组件（图 2.8）的工作方式如下：用户通过与用户代理（US）登录 SESAME 环境，然后通过认证特权属性（APA）客户与认证服务器（AS）联系。US 到 AS 进行自我认证，然后与特权属性服务器（PAS）联系并且从它那里得到用于 AC 判定的、包含主题特权的特权属性证书（PAC）。用户通过认证并且拥有一个 PAC。PAC 允许用户访问计算机上的应用，计算机对用户一无所知，但是能从 PAC 证实用户特权。如果用户要启动一个应用系统，US 为应用客户端与安全协作上下文管理器（SACM）联系。客户端 SACM 与服务器端 SACM 联系，交换主题的证书。然后，服务器 SACM 与 PAC 验证工具（PVF）联系来验证这个主题的 PAC。最后，用户能启动应用客户端并且与服务器交换数据。

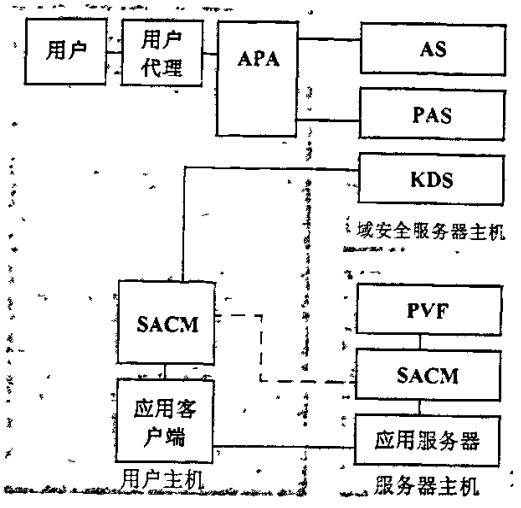


图 2.8 SESAME 组件

SESAME 技术不是中间件。相反它是用于安全服务的一个体系结构。它不提供通信方法，例如 ORB 总线。因此，它不能控制预调用或滞后调用事件。应用系统必须明确地调用访问控制和其它安全功能。这阻止了 SESAME 在应用系统以外提供访问控制。与 DCE 相反，它通过 SESAME 兼容的基础设施为应用系统提供鉴权逻辑。

SACM 为 SESAME 做鉴权判定，这个判定负责接收通用安全服务（GSS）标志，使用 GSS API 传给 SACM [Linn 1997]。目标 SACM 把收到的安全信息传给 PVF 用于分析和验证，如果都有效，SACM 从 PVF 收到一个完整性和机密性的会话密钥，用于保护客户端和对象应用系统之间的信息交换。但是如果 PVF 检测 PAC 失败，安全内容对应用不可用。

即使检测成功，除常规检测匹配发起者和 PAC 标识以外，SACM 根据鉴权规则进行附加的 AC 检查 [Parker 1995]，这个鉴权规则描述为一套符合 POSIX.6[IEEE]的访问控制入口（ACE）。一个入口能指定某种应用或者全部应用可不可以被一个角色、一个组或“全部发起者”访问。

在 SESAME 里的访问控制检查的最小单位是应用系统。因此，要不整个系统都准许访问，要不都不允许访问，这样的 AC 粒度的水平总是不充足。此外，这个体系结构不能把一个鉴权策略用于几个应用，因此要求每个应用分别配置来支持策略。

SESAME 中域的概念适于管理密钥、标识和特权属性的各种鉴权。SESAME 域这样影响访问控制：在不同域内相同用户可以授予不同的身份和特权属性，并且属性可以根据[Ashley 1997]映射到约束，这样就影响目标 SACM 做出的鉴权判定。对于大的或多个组织的环境来说，身份标识和特权属性域使用户安全管理更加可调。

访问控制扩展的缺乏、鉴权判定的粗粒度、以及在应用对应用基础上的对管理策略的需求，使 SESAME 与 JAAS、DCE、DCOM 或者 CORBA 技术相比，在分布式应用 AC 方面，缺乏吸引力。但是，SESAME 可以部署在大多数通信技术上，它是以特权属

性管理和传播模式而闻名，它最适于大的、多域的不同环境[Ashley 1997]。这使得它用于建造异类的、多运营的、高性能的分布式应用系统不可或缺。

2.5.5 CORBA 安全

CORBA（The Common Object Request Broker Architecture，通用对象请求代理结构）技术，包括 CORBA 安全服务，为了发展和部署大范围的专业应用领域内的分布式对象系统，提供通用基础设施。CORBA 计算模式的全部实体由 OMG 接口定义语言（IDL）定义的接口来标识[OMG 1999a]。一个 CORBA 接口是三件事情的集合：行动、属性以及异常。一个 CORBA 接口的实施称作一个 CORBA 实体。因此，我们使用“CORBA 目标”或者只是“目标”来表示“一个 CORBA 接口的实施”，这样不会引起混乱。对象的功能只通过相应接口展现给其它基于 CORBA 的应用。对象通过对象参考来引用。对象参考是一个句柄，通过它进行相应对象的请求操作。

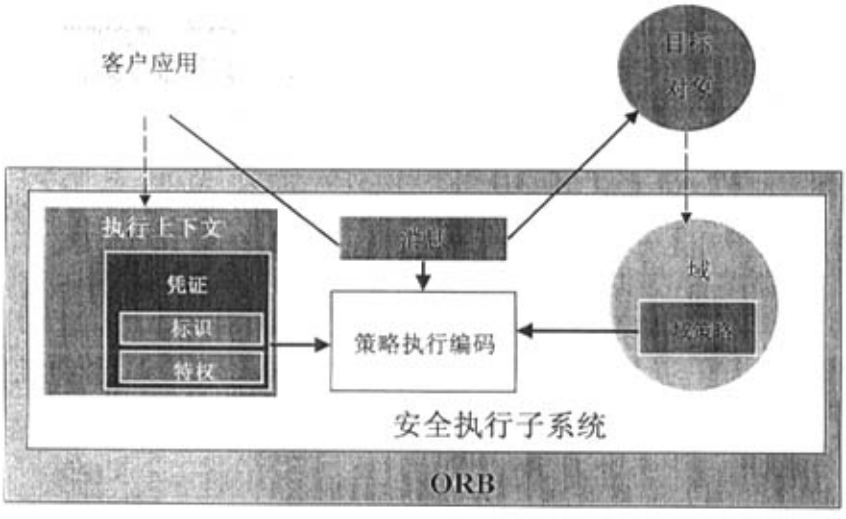


图 2.9 CORBA 安全性中的策略实施

CORBA 安全（CORBA SECURITY，CS）标准[OMG 1996 b]定义了应用程序开发者和安全管理者可见的下列功能性：鉴定和认证、鉴权和访问控制、审计、消息完整性和机密性保护、客户和目标对象认证、不可抵赖、安全策略和相关信息的管理。CS 目标之一是对应用程序开发者完全友好。不需要通过应用对象站点上的任何参与介入，没有安全意识的对象就能在 ORB 上安全地运行。同时，与 CS 实施的安全策略相比较，有安全意识的对象能运用更严格的安全策略。在 CS 模型方面，为了实施象访问控制这样各种各样的安全策略，由合适的安全功能来调停全部目标调用。

CS 控制客户对对象方法的访问。对象被依次放在 AC 策略域，这个域允许使用相同策略来管理对全部域成员方法的访问（图 2.9）。CS 允许用主题和对象安全属性启动 AC 策略。按照启动操作需求的权限对这些操作进行分组。根据主题的特权属性授予主

题的权利应该与操作所要求的权利相匹配。在策略所定义的域内，AC 策略控制什么主题能在什么对象上执行什么操作。

通过特权属性对用户进行分组、通过策略域对对象进行分组、以及通过要求的权利的概念对方法进行分组，这样使得 CS 管理的高量测性成为可能，这是在面向对象分布式环境的一个重要因素。还有应用，其中附加的 AC 必须实现。可知安全的应用能借助 CORBA 安全接口来实现管理的可量测性。为了实现常规 AC 策略，一个应用系统需要知道是谁想要访问、哪个受保护资源、以什么方法访问等。

2.5.6 通用鉴权和访问控制 API

通用鉴权和访问控制 API（GAA API）是 Ryutov 和 Neuman 在 [Ryutov 2000a] 公布的一份 IETF Internet 草稿。它为应用鉴权定义了一个框架，目标是为了定位基于 GSS API 的应用系统缺乏标准鉴权 API。Kerberos [IETF 1993, Neuman 1994a]是提供 GSS API 功能的第一个安全技术，并且它确实影响了 GSS API 之后的模型。Kerberos 在网络应用过程中只有简单地对 AC 的支持：如果客户没有一张准确的网络服务器认证的票据，就不能与这个服务器建立连接，这样访问也就被拒绝了。

框架由两个主要的部分组成：通过应用系统获得鉴权判定的一个编程接口；表示 AC 策略的一种“通用”语言，即扩展 ACL（EACL）[Ryutov 2000c]，是传统 ACL 模型的扩展。这里讨论的主题是 API 本身，其目标是：1）支持大多数应用的需要，允许应用程序开发者避免设计他们自己的鉴权机制， 2）允许多种机制与应用服务器更好地整合（例如，GSS API [Linn 1993]和整合 GAA API 生成主题认证和鉴权判定）。

GAA API 不能实现扩展 AC。认证服务进行用户认证并且提供受限制的凭证（以 GAA API 安全内容的形式），通过认证 API 传递给应用系统。

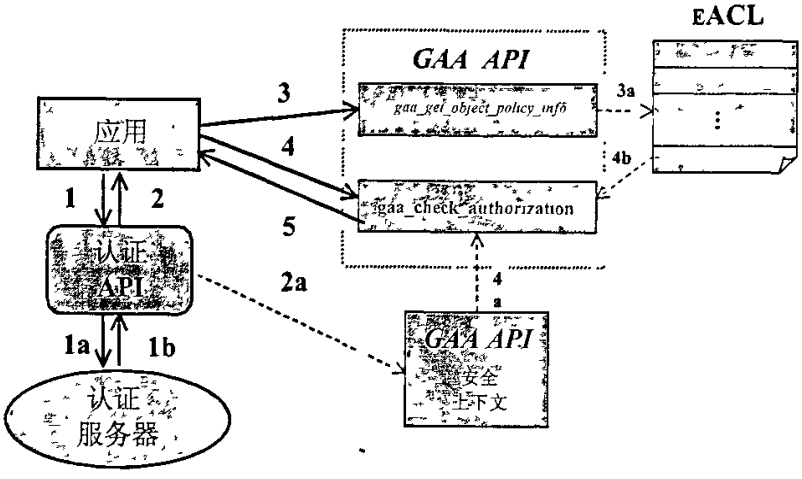


图 2.10 GAA API 模型中事件的顺序

如图 2.10 第 1 和 2 步所示。应用系统调用 GAA API 例程（第 3，4，和 5 步）依照

策略检查鉴权。API 例程获得客户认证后的凭证的主题身份（第 4a 步），和本地文件或者分布式鉴权服务器的策略（第 3a 和 4b 步），或者通过一些其它方法。它们结合本地和分布的鉴权策略和信息。例如，在 GAA API 模型里把机器或者针对应用系统的列表与全球 EACL 结合起来是可能的，这样就使得使用特殊应用策略成为可能。[Ryutov 2000a]里没有定义结合发生的方式，并且这个结合取决于 GAA API 的具体的实施，这表明在编程 API 之后计算环境和策略的变化不能以标准方式支持。

组服务器的使用有显著的不利。首先是通信可测量性问题，因为一些策略可能需要与远程服务器产生不能确定次数的交互，这通常很昂贵，除非服务器和客户协同定位。但是这样的协同定位意味着组服务器例程的数量应该与客户的数量成正比，这是一个明显的性能、维修和管理可量测性问题。第二，当它们都在用户会话期间的时候，几乎不可能获得主题特权属性。

这种方法的另一个缺点是服务，或者是它的代理。代理应该在相同过程中协同定位。API 超过其它模式的主要优势是对非常灵活和强有力的附加条件的支持，其附加条件应该被应用系统强制或者由客户决定。GSS API 提供普通的底层抽象观念，应用程序开发者需要有显著的综合能力才好使用它。

2.6 当前的研究情况

对分布式应用系统资源的访问控制问题有三个主要研究方向。它们是策略代理、接口代理和拦截器、以及大范围的鉴权服务器。

2.6.1 策略代理

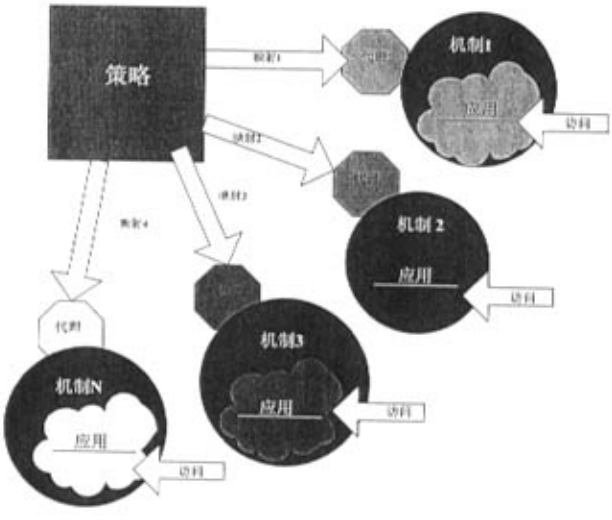


图 2.11 策略代理

“策略代理”这个方法暗示在每个应用系统的计算基础结构里，通过本地机制的方

法实施 AC 策略是有用的，如图 2.11 中所示。它们可能是操作系统 AC 或者是附加包、中间件提供的 AC、DBMS 安全层提供的 AC，或者甚至通过应用系统综合环境提供的 AC 机制。这些方法的主要特征是：以集中 AC 管理的方法，通过把鉴权规则转换成本地机制支持的语言，以及应用系统间规则的分布来取得穿越应用边界的鉴权策略的一致性，这是在策略代理的帮助下取得的。基于这个代理的分布式管理体系结构提供了必要的基础设施，把域范围内的鉴权规则映射到特定机制的规则。

这个方向的所有方法都有下列优势：

• 固有的容错性。一个机制对AC决定和实施失败后，只会影响受机制保护的应用系统，而所有其它系统仍然能受到保护。

• 未经许可的入侵者想要获得一个策略域内全部受保护资源，要推翻全部访问控制机制或策略管理、以及分布式基础设施。因为后者可以使用离线的技术实现，它更安全，且不牺牲时间性能。

• 鉴权判定生成过程的位置。在一个基于策略代理的分布式体系结构里，全部AC决定在本地完成，而且性能降低最小。

• 性能可测量性。因为鉴权过程自然地分布到整个应用系统计算环境，鉴权在本地发行。这样对于大量的应用来说，不会给每个应用客户端带来很长的响应时间。

基于策略代理的方法也有许多固有限制。首先，在策略域方面，AC 策略的粒度和表现能力比较差。其次，策略变化可能非常慢，因为需要重新初始化系统部件，这使得策略变化代价昂贵、经常不一致和频繁的处于停工期。这很容易使基于定期鉴权[Bertino 1996a]的策略不可承受。因为这些挑战和限制，对于新的应用系统采用这个方法很难给出积极的答案。

2.6.2 代理和拦截器

代理和拦截器方法，简称“代理方法”。使用这种方法，现有应用可以具有新的特点，而且不会引起内部变化。为了实施各种各样的安全策略，大多数有能力的安全服务，像 J2EE、CORBA 和 DCOM，都遵循这种方法。

这种概念的依据是：代理应用接口或者拦截交互的应用系统之间的通信。因为鉴权判定是在系统获得控制权之前和/或在发送引用到另一个系统之后做的，所以对应用系统的访问在外部进行控制。为了能实现这个功能，要在通讯层或中间件层拦截这个引用，如图 2.12 中所示。

因为参考监视器能在外部实现，它不用改变应用系统，这是这个方法的主要优势。另一个优势是在当地能够做出所有判定，这是因为它可以在同一个过程空间里部署拦截器和代理。此外，使用本地数据在本地做鉴权判定，这就拥有固有的性能可测量性。

但是，它也有许多限制。首先，AC 粒度不会好于方法及其参数这个级别。也就是说，这个研究方向的方法对于资源的访问控制不能比其上的接口实例、方法和参数具有

更细的粒度。其次，鉴权判定不及时，它们总是在应用系统实行所有控制之前或之后。第三，因为上述原因，在鉴权规则方面不能使用变量，这是因为变量的值在方法调用之后、决定生成之前才能提供。

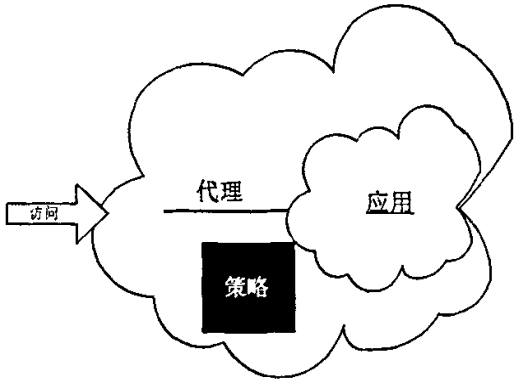


图 2.12 代理和拦截器

主要的缺点是保证实施策略的一致性和用于鉴权判定的数据的一致性成为一个挑战，因为存在很多与应用系统一样的访问控制实例。除非完全自动化，基于代理的 AC 机制的管理将会有非常大的管理开销和很高的人为出错率。

这个研究方向的几种主要方法：对象和 AC 的视角、角色等级，SafeBots，Legion 系统中的 AC 和安全元对象等。

2.6.3 鉴权服务

AC 中，用于分布式应用系统的第三个研究方向是以鉴权服务为基础的。这样的服务在每个策略领域是很有逻辑的。鉴权判定是由服务的实例--鉴权服务器来做的。应用系统执行鉴权服务的判定，但不知道它们是怎样做出的，如图 2.13 中所示。因此应用和鉴权服务都是参考监视器中一部分 [Anderson 1972]。

以鉴权服务器概念为基础的方法的主要优点是：

• AC规则的逻辑集中，它在整个策略领域内具有实施鉴权策略的固有一致性和连贯性。

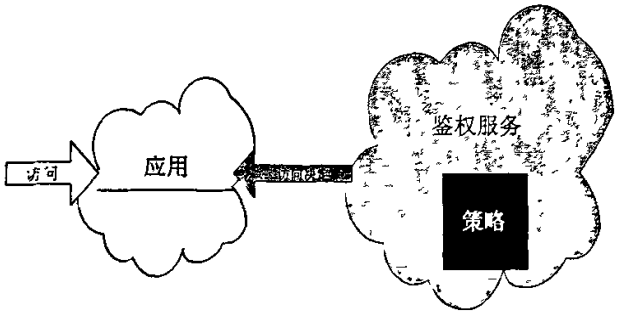


图 2.13 鉴权服务器

• 因为在单个地方鉴权，所以策略的改变和升级比较简单。

• 因为鉴权逻辑是从应用逻辑中分离的，所以替换一个策略而不影响应用系统是可能的。

• 所有属于AC规则的集中的鉴权制度都是独立的、有自然特点的，而且这种制度降低了管理的费用。

• 从应用系统决定从服务器获得鉴权判定开始，当需要一个决定时，服务器就能及时地作出正确判定。

• 可以从服务器获得任意水平粒度资源的鉴权判定。

为了实施这个方法，必须提出几个重要的论点。首先，为了使它在性能方面不会阻塞，设计这样一台服务器比较难。第二，如果服务失败，所有它服务的应用系统将不得不采取一个简化了的和受限制的策略，如“总是否定”或者“总是允许”的策略，这将使系统变得不可操作。因此这样的服务需要高的容错能力。

2.6.3.1 HP的鉴权服务器

Varadharajan 等在 [Varadharajan 1998]里描述的设计原理被用在 Praesidium 鉴权服务的发展过程中[HP1996]，它是一个应用于分布式应用系统的基于规则的鉴权设备。

Varadharajan 等概括了一些用于分布式系统的鉴权的设计原则。他们提议把安全信息分为两类：一类是一般原则，另一个是动态的信息。因此，用它们鉴别三组信息：普通和静止，特殊和静止，以及特殊和动态。应用这种分类，他们建议这样设计分布式安全基础设施：信息储存在一个中心服务器便于被客户“推”到目标地址，接近或者就在目标上，在判定过程中再“拉”回来。这个设计暗示鉴权基础设施有 3 个主要部分： 1)一个储存和管理一般静止安全信息的中心鉴权，2)一个处理特殊静止信息的中心鉴权，3)一个涉及特殊动态信息的目标组件。

这项研究的另一个贡献是鉴权检查的位置和类型。虽然它不是一个主要问题，但是在文献中没遇到过类似的考虑事项。文章建议考虑鉴权检查的 3 个点：如果一个主题可以完全访问这样的一个应用(第 I 层)，然后要在功能类型上执行的一个检查(第 II 层)，以及应用程序里的检查(第 III 层)。这种分类的意义在于在分布式授权系统中建立普通语言和概念。

著者把分布式鉴权服务的功能性区分为二层：管理阶段和“运行-时间”阶段。他们提出两个论点，以保持服务过程中鉴权信息的不同之处。这两个论点是管理阶段捕获信息的存在性，前一个论点，为了实施目的信息，捕获信息可以在访问决定时间之前被编辑，后面的论点中，复制策略用于管理信息和为了访问应用服务器中的评估判定的信息。

运行-时间领域由一台评估引擎和预编译规则的运行-时间数据库联合组成。鉴权判定的内容如下：

第一层 DCE IDL 的接口名字(GSS-API 的进程名)，

第二层 用 DCE IDL 文件(GSS-API 定义的应用)说明的程序的名字，

第三层 鉴权规则。

这个服务设计简略描述了一些元素和方法，它们已经被用在其它相似作品中包括 Adage 在内以及这篇论文描述的工作中。主要要素是把鉴权功能封装到一个服务，这个服务在分布式环境可以提供，也可以在把服务器明确的分为管理和运行-时间域。

主要缺点是缺少文章中提出的设计原则的有效性方面的研究和对鉴权服务的评估。目前还没有出版研究分析这种方法的文章。

2.6.3.2 德克萨斯大学的分布式鉴权服务

德克萨斯大学的 Woo 和 Lam 研究建造一种分布式鉴权服务的理论和实践 [Woo 1993a，Woo 1993 b，Woo 1993 c，Woo 1993d，Woo 1998]。结果，它们设计了这样的一种服务 [Woo 1993 c，Woo 1998]，其关键特征是一种基于语言的方法，这种方法是为了指定鉴权规则。他们的设计以纽曼的早期工作为基础 [Neuman 1993]。

他们观察发现，大多数现有的应用系统进行自己的验证、鉴权和审计 [Woo 1993c]。即使鉴权经常是紧紧的和应用在一起被提取，而且也不容易抽象。Woo 和 Lam 建议：一个更好的解决方法是应该将这些功能分解并作为一套核心服务分别实现。最终这一系列服务能为建造其它一般服务和应用系统的基础。

他们研究的主要内容是在为分布式系统建设一种鉴权服务的过程中出现的两个问题：(1)怎样鉴定在应用系统的鉴权要求中的共性，和怎样设计抽象表现来捕获这些共性，(2)对于从应用系统到鉴权服务器的鉴权和各种各样的企业实体中的相互作用，应该使用什么样的安全协议。显然他们通过用在分布式系统里的验证服务分析来鉴定这些问题 [Burrows 1990，Lampson 1991，Woo 1992]，其中用户证书的普通象征和为认证信息的安全交换所用的相互作用的协议是各种各样的体系结构的区分因素 [IETF 1993，Molva 1992，Neuman 1994b，OMG 1996b，OSF 1996，Schiller 1988，Tardo 1991]。但是，这些因素确实是建造分布式鉴权服务中存在的主要问题。举个例子，一个应用系统可能正好使用 RPC 越过安全(即真实性、机密性)信道获得一个从鉴权服务器来的鉴权判定 [Beznosov 1999b，Varadharajan 1998，Zurko 1998]，如此避免了相互作用协议的发行。

著者声称单独的鉴权服务有下列优势 [Woo1993c]：1)每个应用系统的再实行能力的储蓄，2) 解除应用系统鉴权任务，能导致更高的通过量，3)专业鉴权服务在做 AC 判定时，能提供比个别应用系统更好的方法，4)鉴权服务可以被证实最终是安全的，在证实应用系统的安全过程中降低复杂性，5)匿名能够通过使用信任的鉴权服务来获得，6)统一的鉴权服务有助于审计功能，因此分布式审计业务的建设变得容易。

Woo 和 Lam 设计了一个协议，它使在五个实体中相互作用成为可能。协议的详细描述可以在 [Woo1993c] 里发现。这里，将指出在模型里，为了鉴权的成功所要求的相互作用的关键特征。

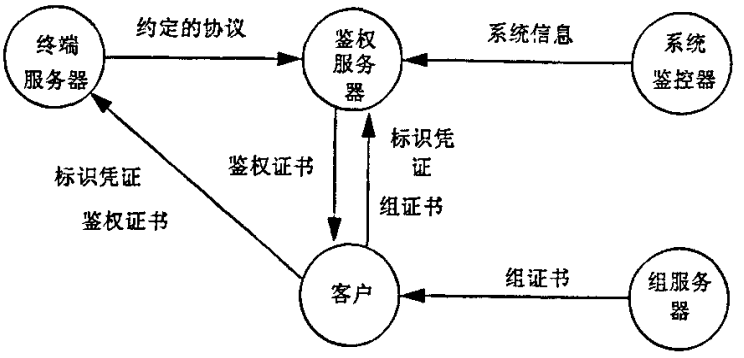


图 2.14　鉴权相关的交互

如图 2.14 所示，通过服务定位器，应用系统 E 定位鉴权服务器 A，然后通过验证服务的相互验证，批准它进入 E。使用签订的协议，在它的末端，A 有一个鉴权说明，这个鉴权说明是鉴权策略的描述，在通用 ACL(GACL)中表示过，它管理进入到 E 的服务。当成功地签约后，E 通知服务定位器：A 被授予为 E 进行鉴权。从现在开始，每位客户都负责从服务定位器获得 A 的参数，并且在 E 服务于客户之前，从 A 获得鉴权证书。当客户从 A 请求鉴权证明书时，启动服务。

Woo 和 Lam 使用的想法(WL 体系结构)非常类似于在 Kerberos 里用到的客户到信任的第三方组织(在 Kerberos 里的 TGS)，为了访问服务而得到一张门票。这个工作是努力去解决 Kerberos 和提供与它相谐调的鉴权服务。Kerberos 缺少主题特权属性的管理并且在适当的位置不能假定任何中间件基础设施。这就使得不同功能和服务，例如验证，特权属性管理，定位发现，鉴权，甚至系统监控，都能混合到 WL 体系结构中。

服务的概念背离了传统模型，其中主题特权属性在验证阶段鉴定，在协商期间决定。组服务允许使用全部特权属性的验证技术。如果客户为了每个应用请求必须与远程组服务器交互，这将潜在地降低鉴权处理的效率。

在 WL 解决办法中，鉴权判定的颗粒性不是很好，因为在联系应用之前，客户需要确切地知道什么样的鉴权能够请求鉴权服务。这使 WL 鉴权服务结构只对像远程登录[Postel 1983]，FTP [Postel 1985]等网络服务有用，在这里所要求的唯一的鉴权将和服务器进行协商。其余的通过操作系统访问控制机制进行控制。

关于 WL 方法的另一个缺点是应用系统和鉴权服务器之间签订的协议。著者认为应用是鉴权策略的来源。不过，在大多数较大的组织中，鉴权策略是与具体组织机构相关的，而与具体应用关系不大。因此，策略不应该保存在应用里。应用最好不要涉及策略管理、层次或者分布状态。

总的来说，接近应用鉴权的 WL 取得了以上所列举的目标。但是，它有明显地局限性：把应用性提供到只有 Kerberos 作为主要安全技术的简单分布式系统，并且对访问控制或者客户特权代表的远程调用也没有什么要求。

2.6.3.3　Adage

Zurko 等对分布式应用和组(Adage) [Zurko 1998]方面进行研究。Adage 体系基于下列原则：用户集中的设计、策略中立、模块化和 RBAC 的使用。他们工作的主要目标是为分布式应用的使用设计一种鉴权服务，其重点是在它的管理接口和工具的可用性上。

Adage 体制由用于管理策略的客户和为了提供鉴权判定的策略判定服务器组成。客户包含 GUI 和鉴权语言(AL)翻译器，以及通过管理 API 与鉴权判定服务器(ADS)之间的相互联系。应用系统希望通过鉴权 API 做出访问 ADS 的鉴权判定。

在用户鉴权数据库(UAD)里，ADS 存储由客户提供的策略信息。ADS 包含一个翻译器，它把 UAD 中的信息改变成更适合快速做出鉴权判定的形式。这个数据库叫做引擎鉴权数据库(EAD)。鉴权引擎是 ADS 的另一个主要部分，它使用 EAD 找出适用于相应判定的规则。

Adage 工程的主要研究目标是设计鉴权服务以适合分布式应用系统，这个系统将使管理和应用接口的使用能够符合心理可接受和可用性的原则，并且能进行系统的可用性研究。这样的一个目标和前面所描述的工作目标有相交处。

2.7 本章小结

鉴权判定与应用逻辑分离的想法已经提出很久了。参考监控器的一个抽象模型[Anderson 1972]是在应用系统外实施鉴权判定的一个典型例子。大多数操作系统在它们的中心安全部分实现鉴权逻辑。还有一些专用安全软件包，把鉴权判定固定到操作系统里[Benantar 1996，CA 1998a，IBM 1976，CA 1998b]。

有两组技术用于获得分布式软件系统服务。一组是不考虑通信技术，仅仅为主题提供验证、通信保护和访问控制技术：Kerberos [IETF 1993，Neuman 1994a]，SESAME [Kaijser 1998，Parker 1995]以及 GAA API [Ryutov 2000a]。这将使使用和混合任何所需要的通信协议成为一种可能，但是为了把安全技术和基础通讯结合起来，开发者将努力承担更多地责任。

另一组是中间件技术，例如 CORBA [OMG 1996 b]，DCE [Gittler 1995]，Java [Lai 1999],以及 DCOM [Microsoft 1998]，它们提供了整合安全子系统的通信基础设施，因此开发者能够享受两者的结合以及和前者更多的无缝应用。

JAAS 提供给用户一种框架和一个标准编程接口，为了验证用户和分配特权。访问控制只在系统资源上实施，而不在 Java 对象和其它应用资源上实施。JAAS 有对不同特权属性通用和可扩展的支持，这个支持容易通过新的类来确定。为了鉴权判定、源码、代码签字人的身份、学科特权属性的值通过类策略接口传给鉴权代码。JAAS 允许鉴权判定的任何颗粒性，而且它不强制鉴权策略实施到任何特别的机制或者用于判定的信息。它也使策略的无缝变化成为可能。不过，这种体系不处理多个应用的鉴权策略的一致性。它也不对取得性能和管理可伸缩性有任何保证。

在 DCE 里，应用系统自己实施和提供管理以访问鉴权策略。应用系统能使用 DCE

访问控制表(ACL)，但是它必须实现大多数访问控制功能，包括 ACL 贮存及其管理。DCE 安全只提供一个有呼叫者的主题和组身份申请。虽然在每个应用基础上进行管理的管理接口已被确定，但是不能直接支持鉴权逻辑之间互相应用的管理，而不可调控。

DCOM的安全模型类似于DCE安全。像DCE一样，ACLs用于编码鉴权策略。DCOM的主要进步是实施策略的能力。鉴权模型被策略的颗粒性显著地妨碍了，在相同的 OS 过程里，不同的对象及其方法没有区别。部件和主机范围的策略含蓄地引入了访问策略领域的想法；既使这样的领域划分是一个管理和功能都成功的解决办法，它仍然不是很清楚。管理部门必须在每个主机或者过程中分别执行，这比在 DCE 里要好，但仍然受限制。虽然 DCOM 安全为应用系统提供了方法，以具体应用方式提供细粒度访问控制，但是仍不能实施特殊应用策略。

SESAME 是一个抽象的通信层，用于安全服务的一个体系结构。因此，它不能控制预先或公布了的远程调用事件。这就是为什么访问控制和其它安全功能必须由应用明确激活的原因。SESAME 鉴权的另一个缺点是缺少把一个策略应用于几个单独主机内的应用系统的功能。鉴权检查的单元是应用系统。所有这些，特别是AC的颗粒性，使SESAME比 JAAS，DCE，DCOM 或者 CORBA 这些用于访问控制资源的技术更缺少吸引力。不过，SESAME 对基础通信协议中立，而且它以自身的特权分布式管理和继承的先进模式而闻名。这使它在建造不同的、多技术和多组织的分布式应用时不可或缺，其中的分布式应用要求鉴权以特权属性为基础，而不是用户身份或使用不同的通信技术。

在 CORBA 安全里，除应用系统外，可以完全实施访问控制。访问控制判定以主题特权属性为基础，要求方法的权利和对象属于其领域范围的访问控制策略。访问控制模型非常好划分，而且没有失去细颗粒性，如果对象位于一个单独的域，或者位于一个能与一个策略领域关联的大对象组，对于判定来说，访问控制模型可能具体到每个对象。与 DCOM 不同，保存在不同计算机的 CORBA 对象能与相同的策略域关联。因为 CS 定义了特权属性的先进概念，这使 AC 策略能够基于角色、组和任何其它有关安全的主题属性。通过特权属性归类的用户，通过策略域归类的对象，和通过所要求的权利概念归类的方法，使访问控制机制高水平的管理和性能可测量性成为可能。不过，在 CORBA 中的访问控制受限的时候，特殊应用策略和具体应用信息都很难实施。

作为一份 IETF 因特网草稿出版的 GAA API，确定了用于应用鉴权的一种框架。API 的目的是为了那些使用 GSSAPI，它缺乏标准定位鉴权接口。这就是 GAA API 的鉴权模型明确适应现有的 GSS API 的原因。如果一个应用使用 GSS API，GSS API 提供了非常通用的、低层的抽象观念，这个应用就需要细粒度的资源保护或者实施复杂的鉴权策略，其中 GAA API 定义了适合大多数应用能力的接口。API 超过当前其它模式的主要优势是，对非常灵活而且强有力的概念的支持，这个概念支持特殊应用策略。API 缺点是它只确定应用和一个鉴权机制之间的接口，而且模型既没有定位管理可伸缩性，也不能定位多应用之间的鉴权策略的一致性。

理想状态是，安全功能应该全部设计在一个应用系统外部。 但是，对于大多数应用系统，它是很难获得的，在这些系统里，通用安全技术所支持的访问控制和其它安全策略都太复杂，或需要更精密的控制。这就是为什么分布式应用资源的细粒度控制是以传统的专门方式来做 [Wilson 1997]，而且现在还没有自动化的方法能保证如此控制的一致性。

研究团体正在向分布式应用系统中控制访问资源的系统方式方向上努力工作。处理这个问题主要有三个研究方向。它们是策略代理、接口代理和拦截器，以及分布式系统范围内的鉴权服务器。

策略代理的方向主要受组织内存在的产品和技术相结合的目标所激励。这个方向的关键之处是，把集中的访问控制管理通过鉴权策略翻译成所支持的语言，并把这些策略在系统中分发。这是在策略代理的帮助下取得的，其中策略代理以应用系统主人身份存在于计算机里。

这个方向上的方法有许多优势：有固定的容错；系统安全自然地隔开而不需要降低运行-时间性能；有高度的运行-时间自治权—对取得性能的可测量性和容错都很必要的一种特性。

这个方法面临的主要挑战是把全球策略的一致性和把全球策略绘制到访问控制机制语言中，以及表现的自动性。这个方法也有很多固有的限制。首先，在一个策略领域里，访问控制策略的粒度和表达性可能与最粗糙、最没有表现力的访问控制机制所支持的策略一样。其次，策略更新的发布可能非常慢，这将很容易使基于定期鉴权的策略无法承受。

另一个是应用接口代理或应用通信拦截之下的一种方法。它对一个应用系统的访问是在外部控制的。在一个应用系统增益控制之前做出鉴权判定，然后，它给另一个系统发送一个远程调用。为了获得鉴权判定，在通信、中间件、或者应用层拦截远程调用。

这个方向的主要优势是它不需要应用系统的任何变化，参考监控器在外部实现，而且开发者可以控制它的尺寸。这样，这个就成为一种好的策略代理方法，用于控制访问已经配置的应用资源。而且，如果一个现有的应用缺少访问控制机制，代理和拦截器就成为了唯一的选择。另一个优势是由对本地应用系统做出全部判定的能力，这使性能可测量性变得非常容易保证。

但是，它也存在很多明显的限制。首先，访问控制的粒度不会比方法及其参数更好。其次，总得在应用系统拥有控制之前或之后做决定。第三，方法调用之后，做出判定之前，可提供变量的值，但是不能在鉴权判定过程中使用变量。第四，因为有很多这样的访问控制实例，实施策略的一致性和用于鉴权判定的数据连贯性是一项挑战。

另一个方向是以鉴权服务为基础，由应用系统来实施服务实例判定。应用系统和鉴权服务器组成参考监控器，它要求信任应用系统能实施访问控制判定。

鉴权服务的目标是在应用系统之外代理通用访问控制决策函数，并且作为一种基础

设施服务分别实现它们。这个方向的主要优势是：固有的一致性和鉴权策略的连贯性；因为认证是在一个合乎逻辑的单个的地方进行的，策略改变和持续升级非常简易；改变策略及其类型并不影响应用系统的能力；访问控制管理相对成本较低；当需要鉴权时，可以获得任意粒度的访问控制。

但是，为分布式鉴权服务建造一个成功的体系，必须处理几个关键问题：性能、容错、可伸缩性、鉴权信息的安全性和鉴权判定的保证能力。

本文期望的成功解决办法是利用代理、拦截器、策略代理和鉴权服务的一种或者它们中的结合。对于与应用紧密结合并存在AC机制的系统来说，策略代理是唯一的选择。在AC机制粗颗粒性的现有系统中，拦截器和代理能够解决这个问题。如果要求细粒度的访问控制、复杂或者动态鉴权策略，应用在不同类型、不同的实施范围，使用鉴权服务器是最好的方法。

第三章 J2EE 安全系统下的 RBAC 支持

在前面章节中本文分析了现存中间件技术的访问控制机制，并发现它们不足以彻底解决应用资源的访问控制问题。尽管如此，有一些机制（比如 J2EE 里的机制）还是允许鉴权策略执行的，另外，它们与相应的服务也能很好地结合，这使得中间件访问控制机制比应用层访问控制更加受欢迎。在一个系统采用应用层访问控制之前，能够最大限度的利用中间件访问控制是非常重要的。这就是为什么说对于中间件访问控制性能的研究在保护应用资源方面是至关重要的。

本章主要做了两个方面的研究。首先，详细地分析了 J2EE 访问控制机制的性能，提出了一种能够正式定义系统状态的 J2EE 保护系统配置。通过该定义，本文提出了在 J2EE 下进行鉴权判定的算法。其次，本文给出了在进行 J2EE 安全服务时，RBAC 模型是如何得到支持的。利用 J2EE 保护系统的配置，本文用 J2EE 安全语言给出了 $RBAC_0$ 和 $RBAC_1$ 两种模型的定义。此外，为了支持 $RBAC_0$-$RBAC_3$ 模型，还分析了实施 J2EE 安全服务所需的必要条件。这个方法在符合 Java 安全规范的前提下支持 $RBAC_0$。为了支持 $RBAC_1$ 和 $RBAC_2$，实现了 J2EE 安全规范范围以外的功能。这项工作加强了对 J2EE 访问控制机制性能的理解，这对于中间件的利用是至关重要的。

3.1 RBAC 概述及应用目的

RBAC[Sandhu 1996]是一个参考模型家族，许可与角色紧密相连，用户被指派给合适的角色。一个角色能够代表资格、权限、职责或者任务分配。RBAC 的一些变量拥有构建角色与角色、许可与角色、用户与角色之间关系的能力。下面有四种现存的 RBAC 的参考模型：不相关角色($RBAC_0$)，角色层次($RBAC_1$)，用户和角色分配约束($RBAC_2$)，以及层次和约束($RBAC_3$)。RBAC 支持三种安全法则：最小权限，权责分离和数据提取。

RBAC主要是为了使访问控制管理和审查更加便捷，它是满足分布式系统需求的最有前景的方法，优于基于格子的强制访问控制（MAC）[Bell 1975]和基于所有者的自主访问控制（DAC）[Lampson 1971]。有一些软件系统可以进行RBAC的建模和实现：Oracle [Notargiacomo 1995], NetWare [Epstein 1995], Java [Giuri 1998], DG/UX [Meyers 1997], object-oriented systems [Barkley 1995], object-oriented databases [Wong 1997], MS Windows NT [Barkley 1998], 和 Enterprise Security Management Systems [Awischus 1997]。

RBAC模型的基本思想是将访问许可权分配给一定的角色，用户通过饰演不同的角色获得角色所拥有的访问许可权。这是因为在很多实际应用中，用户并不是可以访问的客体信息资源的所有者，这样，访问控制应该基于员工的职务而不是基于员工在哪个组，

即访问控制是由各个用户在部门中所担任的角色来确定的。

RBAC从控制主体的角度出发，根据管理中相对稳定的职权和责任来划分角色，将访问权限与角色相联系，这点与传统的MAC和DAC将权限直接授予用户的方式不同；通过给用户分配合适的角色，让用户与访问权限相联系。角色成为访问控制中访问主体和受控对象之间的一座桥梁。不同的用户根据其职能和责任被赋予相应的角色，一旦某个用户成为某角色的成员，则此用户可以完成该角色所具有的职能。

同一个用户可以是多个角色的成员，即同一个用户可以扮演多种角色。同样，一个角色可以拥有多个用户成员，这与现实是一致的，一个人可以在同一部门中担任多种职务，而且担任相同职务的可能不止一人。因此RBAC提供了一种描述用户和权限之间的多对多关系，角色可以划分成不同的等级，通过角色等级关系来反映一个组织的职权和责任关系，这种关系具有反身性、传递性和非对称性特点，通过继承行为形成了一个偏序关系。RBAC中引进了角色的概念，用角色表示访问主体具有的职权和责任，简化了权限设置的管理，从这个角度看，RBAC很好地解决了管理信息系统中用户数量多、变动频繁的问题。

例如，1000个主体访问10000个客体，采用MAC/DAC，须进行1000万次配置。如每次配置需1秒，每天工作8小时，就需10000000/（3600*8）=347.2天。假如，这些用户对权限来说，拥有10个角色，则只需要1000*10+10*10000=11万次配置，节省工作量大约98.9%。

相比较而言，RBAC具有灵活性、方便性和安全性的特点，目前在大型数据库系统的权限管理中得到普遍应用。用户与客体无直接联系，他只有通过角色才享有该角色所对应的权限，从而访问相应的客体。

与此同时，基于J2EE技术的系统扩展已经初具规模。由于它的普遍性，J2EE安全(Java Security，JS)在任何特殊访问控制模型中并不是十分考究的。取而代之的是它还定义了一种机制用来应对大多数情况，并支持各种访问控制模型。例如，它指出了如何利用J2EE鉴权模型[Karjoth 1998]来执行MAC。在接下来的几年内，预计在商业和政府机构中JS范围内展开的重大的财政投资，包括那些利用RBAC理念构建安全策略的机构。如果JS完全支持RBAC模型，这种预见性则变得非常重要。然而，对于正在探究支持支持RBAC参考模型的JS的潜力的研究团体的任何工作，还知之甚少。

3.2 J2EE 访问控制机制

首先，详细了分析了J2EE访问控制机制。然后，为了重新定义RBAC模型，本文定义了J2EE保护状态配置，并详细说明了鉴权算法的设计。在本章的后面，讨论了在J2EE安全策略下对RBAC的支持。

3.2.1 J2EE 访问控制机制分析

上一章我们已经介绍了J2EE鉴权[IBM 2004a]的主要概念。在深入讨论JS访问控制机制之前，先简单的回顾一下JS。简而言之，所有的对象请求都是通过适当的JS AC函数调停的，这些函数强制执行各种不同的安全策略。

JS鉴别体系与CORBA及其相似[范 2004]。用户通过用户代理来鉴别JS环境。用户发起人是客户应用的逻辑部分。它代表用户来鉴别并获取接口实例SecurityLevel2. PrincipalAuthenticator鉴别凭证，如图3.1所示。

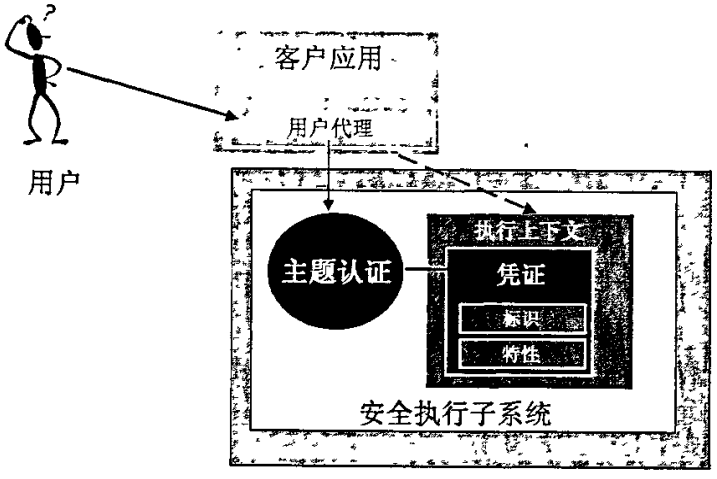


图 3.1 执行上下文的创建

相对于JS 的具体执行所支持的鉴权方法，用户发起人的实例实现了很精确的用户接口。例如，对于密码认证，它提示用户输入用户名和密码。对于智能卡验证，它与智能卡读卡器相连，提示用户插入智能卡。JS 标准不指定任何特殊的认证方法。它所指定的是PrincipalAuthenticator的接口。一个PrincipalAuthenticator实例完成实际认证，并且为一个新主题创建Credentials对象。从用户发起人获得的认证数据和优先安全技术（Kerberos，SESAME，或者其它有可能的技术），以及任何一种策略的基础上，PrincipalAuthenticator用多种信息样例来表示Credentials。Credentials中的信息构成了一种新主题的特性，它代表用户向J2EE目标发出请求。鉴定安全属性是储存在Credentials内信息的一部分。

在JS下的访问控制和其它保护都是基于策略的。有几种策略类型，其中一种是访问控制策略，任何一种策略都与一个在JS中策略域相关。策略域是一种抽象，即允许安全管理员把群中的对象组合起来并且分配策略到群。有相同安全需求的对象被分配到同一策略域。域允许把访问控制策略请求转变成它们的执行对象或接口。图 3.2 举例说明了域和策略概念。它指出策略域与策略是有联系的，如客体(小圆圈)在域里被组织在一起，由策略来管理。不同类型的策略可以与同一策略域关联，每个客体都可能属于多个策略

域。域也可以以联盟、层次或毫不相关的方式组织在一起。

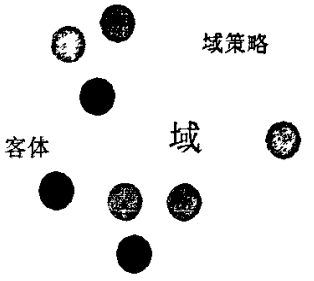


图 3.2 J2EE 安全中的域和策略

策略采用以下三种信息资源：目标对象所属的域策略；来自客户端凭证的信息和指定目标对象的信息；调用的方法名称。下面详细分析JS访问控制机制。

为了阐明所要讨论的内容，用图3.3来说明。在JS模式下principal这个术语相当于传统AC技术中的主题。此处的讨论中将交替地用到这两个术语。Session的概念与principal的概念是无法区别开的。多重主要角色能够对单一用户的权限起作用。它们都拥有一套不同的凭证，因此，在JS中作为一个完全独立的实体存在。在其它数据中，主要角色的凭证还包括安全属性。今后，将属性理解成“安全属性”。就JS AC模式而言，角色只是一个无序的鉴别属性的集合。

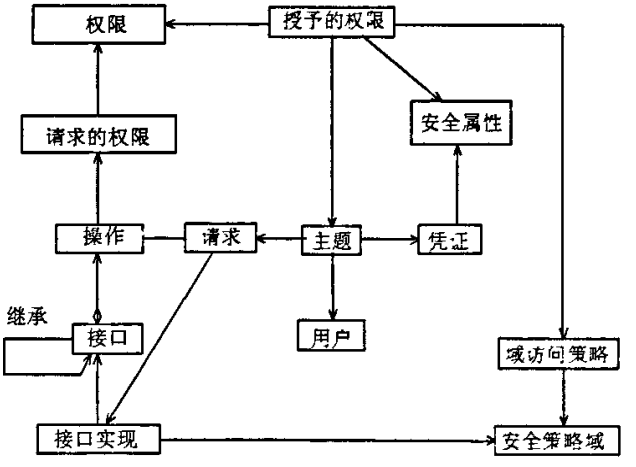


图 3.3 J2EE 访问控制机制中关键元素间的关系

属性是拥有某个类型t、定义了鉴权a、值v和鉴权状态ds的四元集合a = {t, a, v, ds}。ds ∈DS = {i, d}和状态i指出了即时调用所具有的属性，状态d指出了间接调用所具有的属性。这种属性类型被分成两组：特权属性和一般属性。特权属性族列举了一些可识别主要角色特权的属性类型。这些类型包括访问标识、主要角色所属的初级和次级组、性能等。身份属性（如果存在的话）提供了额外的主要角色信息。类型的一些例子如审计

标识、记账标识和认可标识，反映了一个principal可以有多种不同身份来实现不同的目的。Principal credentials可包括任意多个同一类型的属性。表3.1是安全属性赋值给Principal的典型实例。标准J2EE属性类型之一便是角色。由于安全属性定义相应的扩展，JS也可支持J2EE安全标准未定义的属性类型。

表3.1 认证后的主题所具有的安全属性

主题	特性
P1	a1
P2	a2,a6
P3	a2,a3
P4	a4,a5

尽管JS标准化部分没有书面要求如何管理属性，但是这些面向用户的属性的分配却是由用户管理员来执行的。

在J2EE计算模式下的所有主要角色都是响应客体而进行操作调用。在发起请求之前需要弄明白以下两件事情：客体参考（它是客体的唯一标识）和操作名称。J2EE接口能够通过接口继承来继承其它接口。一个接口只对应一个操作名称。因此，任何一步操作都由接口名称及其操作名称所唯一确定。这里，用符号$i_k m_n$来代表在第k个接口的第n个操作。

对于每一个操作都有一套全局的请求权限。这套设置与结合器（所有的或任何一个规则）一起，说明主要角色为调用操作所必需的权限。表3.2例举了i_1, i_2, i_3三个接口操作的必要权限。假设已经定义了必要的权限，并且它们的语义也由熟练掌握每个操作的应用程序开发人员明确地写成了文档。

表3.2 需求权限矩阵

操作	需要的权利	结合器	含义
i_1m_1	r_1	所有	只有拥有r1权限的主题才能调用操作
i_1m_2	r_1 r_2	任意	任何拥有r1或r2权限主题都能调用操作
i_2m_1	r_2 r_3	所有	只有拥有r2和r3权限的主题才能调用操作
i_2m_2	r_2 r_3 r_3	所有	只有拥有r1,r2,r3权限的主题才能调用操作
i_3m_1	r_1, r_2 r_3, r_4	任意	任何拥有r1,r2,r3或r4权限的主题都能调用操作

根据在特定AC策略域的访问策略，主要角色需要根据特权属性来授予不同的权限。每个*DomainAccessPolicy* (DAP)客体都定义了每种安全属性所赋予的权限。表3.3提供了主要角色授权属性和赋予权限的实例。

表3.3 授予权限矩阵

属性	授予的权限	
	域	
	d_1	d_2
a_1	r_1	r_2
a_2	–	r_1
a_3	r_2, r_3	–
a_4	r_3	r_1, r_4
a_5	r_1, r_2, r_3	r_2, r_3 r_4
a_6	r_6	r_1

安全管理程序的任务是定义在每个域的安全属性所授予的权限。当主要角色发出操作请求时，主要角色的有效权限就会通过AccessPolicy.get_effective_rights()计算出来。JS规范没有定义该项操作如何与表3.3中展示的不同特权属性项所赋予的权限相结合。这些详细说明使得JS执行者定义了操作的内部行为。一个最简单的执行语句get_effective_rights()可以将授予主要角色的一组权限转变成主要角色拥有的每--个安全属性的权限集合。举个例子，假设已经正确执行了该操作，采用分配给主要角色 p1, p2, p3 和 p4 (表3.1) 的安全属性的实例、必要权限(表3.2)和授权权限 (表 3.3)实例，由表3.4可得出在每个域中主要角色被授予的权限。

表 3.4　每个主题授予的权限

主题	授予的权限	
	域	
	d_1	d_2
p_1	r_1	r_2
p_2	r_6	r_1
p_3	r_2, r_3	r_1
p_4	r_1, r_2, r_3	r_1, r_2, r_3, r_4

因此，主要角色可以如表3.5中所示来调用操作。值得注意的是，由于主要角色p2在域d1中仅仅被授予权限r6，根据需求权限矩阵（表3.2）可知，由于权限r6不足以调用任何操作，所以它将不能调用任何操作。

表 3.5　每个主题允许的操作

主题	允许的操作	
	域	
	d_1	d_2
p_1	i_1m_1, i_1m_2, i_3m_1	i_1m_2, i_3m_1
p_2	–	i_1m_1, i_1m_2, i_3m_1
p_3	i_1m_2, i_3m_1	i_1m_1, i_1m_2, i_3m_1
p_4	$i_1m_1, i_1m_2, i_3m_1, i_1m_2, i_2m_1$	i_1m_1 i_1m_2 i_3m_1 i_1m_2, i_2m_1 i_2m_2

3.2.2 J2EE 保护状态配置

通过对JS AC模式的分析，在定义3.1中定义了J2EE系统的保护状态配置。如算法3.1所描述的，适应JS的安全服务实现应当产生相同的访问控制判定。

定义3.1 J2EE系统保护状态配置

根据J2EE访问控制机制的能力和RBAC模型，下面给出J2EE保护系统的配置，这个配置从形式上定义了系统的状态。

A 特权属性集合；

IM 接口上定义的操作的集合；

O 不同接口实例的集合；

R 权限集合；

D 访问策略域集合；

C＝{all, any} 权限结合程序的集合；

RRM 需求权限矩阵，每个操作对应一行；第一列是需要的权限，[IM, rights]⊆R，第二列是结合器，[IM, combinator]∈C；

DS={i,d} 授权状态集合；

IDM 接口实例的域成员矩阵，D中的的每个域占一行，O中的每个接口实例占一列。我们用[D,O]标识IDM中的一个元素，有：[D,O]⊆{T,F}，a[d,o]≡T⇒o∈d；

GRM 已授权限集合，A的每个属性占一行；D中的每个访问属性域占一列。[A,D]⊆R；

ER 有效权限，D×2R→2R，在一个域dj∈D中把特权属性a1,a2,...,al (∀i,1<i<l,ai∈A)映射到r1,r2, ..., rp (∀i,1<i<p,ri∈R)上，对于给定的属性，权限有效；

CB 结合，（D→2R）→2R，对于D中的每个域，把从ER返回的权限集合映射到有效权限集合；

IO M×O→IM，把一个操作m和一个接口实例o∈O映射到一个接口操作上。

使用这个定义，本文设计了一个 J2EE 鉴权判定算法（算法 3. 1）。

m：操作名；
o：接口实例，o∈O；
IO(m,o)∈IM；
P=[a_1,...,a_n]，表示主体，a_1,...,a_n ∈A；

DER[]=NULL;
For (对于 IDM[d,o]=true 的每个 d){
    DER[d]=ER(d,p);
    }
CER[d]=CB(DER)
i=IO(m,o);

```
If (RRM[i,combinator]==any){
    for (RRM[i,rights]中的所有 r){
        if (r ∈CER)   return true;
        }
    return false;
    }
Else {
    for (RRM[i,rights]中的所有 r){
        if (r ∉CER)   return false;
        }
    return true;
    }
}
```

函数ER查寻GRM以获取对象o所属域中每个属性的权限。它根据执行任务将这些权限结合在一起，再把有效权限反馈给每个域。

来自有效权限的结果作为函数CB的输入参数来使用，后者根据执行任务把它们连接起来。无论RRM如何处理，由CB返回的权限都需要核对。如果匹配成功的话，访问将被授予；否则，访问将被拒绝。

对于每个域，都会构建起来一个来自[Lampson 1971]的访问矩阵，如表3.6。

表 3.6　访问矩阵

主题	对象		
	i_1	i_2	i_3
P_1	i_1m_1	–	i_3m_3
P_2	i_1m_1,i_1m_2	–	i_3m_3
P_3	i_1m_1,i_1m_2	–	i_3m_3
P_4	i_1m_1,i_1m_2	i_2m_1,i_2m_2	i_3m_3

对于每个JS系统构建所需的访问矩阵，有三点值得注意。

第一，实体不能是对象，例如，J2EE的访问控制就没有主要角色操作的概念，它仅仅有接口操作的概念，按照访问矩阵 [Lampson 1971] 术语来讲，这种就是实体。

第二，由于$i_km_p \equiv i_lm_q <=> k \equiv l \wedge p \equiv q$ (例如仅仅 $p \equiv q$ 对于$i_km_p \equiv i_lm_q$是不充分的)，如表3.6，一般情况下的操作语义可能各不相同。因此，对于每一个实体s和对象o，单元[s,o]的内容对于对象是十分明确的，举例来说，在一个对象上许可的操作在其它对象上操作是不可行的，因为在语义上针对每个接口的操作都不同，除非能够通过继承使得接口相关联。

第三，在给定访问策略域，同一接口的所有实现都是由访问矩阵的同一对象表现出来的。所以，同一接口的实现与访问控制观点几乎没有区别。这就是策略域在J2EE访问控制模型中占有重要地位的原因之一。

3.3 J2EE支持的RBAC系统

3.3.1 访问控制模型

在[Sandhu 1996]定义的四种RBAC参考模型中，$RBAC_0$是最基本的模型。它仅仅要求一个系统要有用户、角色、许可和会话的意图，对面向角色的许可和用户分配没有任何约束。$RBAC_1$除了有$RBAC_0$所有的一切外，还具有角色层次。除$RBAC_0$ 所有的功能以外，$RBAC_2$对面向角色的用户分配和许可都有所限制。$RBAC_3$集合了$RBAC_1$ 和$RBAC_2$所有功能。本节用J2EE保护状态配置的定义（定义3.1）语言来重新定义$RBAC_0$ 和$RBAC_1$。这将指出配置J2EE系统来支持不同RBAC模型的正确性。首先给出RBAC的初始定义。

3.3.2 RBAC模型的初始定义

根据RBAC模型，每一个会话都是一个用户到许多角色的映射。当一个用户建立起一个会话时，他就会激活一个由用户管理员分配给用户的一个角色子集。用户可利用的许可，是会话中所有被激活角色许可的集合。由于它们的语义是可执行且随系统而定的，RBAC把许可作为一种不可译符号。

定义 3.2 $RBAC_0$

根据RBAC模型，$RBAC_0$有这样一些部件：

U，R，P，S(用户集合，角色集合，许可集合，会话集合)；

$PA \subseteq R$(许可分配，多对多关系)；

$UA \subseteq U \times R$(用户分配，多对多关系)；

User：$S \rightarrow U$(每一会话S_i对应唯一用户的映射$User(S_i)$)

Roles：$S \rightarrow 2^R$(每一会话S_i到角色集合的映射)

$Roles(Si) \subseteq (r | User(Si), r) \in UA)$表明一次会话的活跃角色集一定是该会话授权角色集的子集，这一要求是由会话的定义决定的：会话代表用户执行，会话能取得的授权角色集等于用户的角色集。

S_i具有许可 $\bigcup_{r \in Roles(S_i)} \{p | (p,r) \in PA\}$

表明一次会话的许可集是它的活跃角色集中各角色拥有的许可集的并集。

定义 3.3 $RBAC_1$

$RBAC_1$模型有以下一些部件：

U、R，P，S，PA．UA、User(Si)(与RBAC$_0$相同)；

RH：R×R(角色层次，关于R的偏序关系)；

Roles：S→2^R(每一会话S_i到角色集合的映射，并且遵循下面的不变式：)

Roles(Si)⊆{r | ∃(r'≥r)[(User(s_i),r')∈UA]}

表明一次会话的活跃角色集是该会话授权角色及其下级角色集的子集。

S_i具有许可 $\bigcup_{r\in Roles(S_i)}\{p \mid \exists(r' \geq r) \in PA\}$

表明一次会话的许可集是它的活跃角色及其下级角色集中各角色拥有的许可集的并集。

这里重现了[Sandhu 1996]中的RBAC$_0$(定义3.2) 和RBAC$_1$(定义3.3)定义。

3.3.3 RBAC$_0$

对于基本模型RBAC$_0$，本文在JS中描述的的四种标识如下：RBAC用户映射到JS用户；由*role* 类型的特权属性A集代表角色；JS中的许可等同于权限R集；会话等同于仅有安全属性集的主要角色。在定义3.4中，用JS语言正式地对RBAC$_0$下了定义。

定义 3.4 RBAC$_0$

RBAC 中的四个基本概念（用户、角色、许可和会话）与我们在 J2EE 安全服务中定义的概念做如下对应，见表 3.7：

表 3.7 RBAC 与 J2EE 概念对照表

RBAC	J2EE
用户	用户
角色	特权属性（其类型为角色）
许可	权限
会话	Principals

按照表 3.7 的对应关系，重新定义 RBAC$_0$如下：

U，A，R，P（用户，角色属性，权限，Principals）

PA⊆R×A，一个多对多的已授权限与角色安全属性的设定关系

UA⊆U×A，一个多对多的用户到角色安全属性的设定关系

User:P→U，把每个 Principal p_i映射到单个用户 User(pi),在 Principal 生命周期中保持不变；

Roles.P→2^A，把每个 Principal p_i 映射到角色的特权属性集合 roles(pi), roles(pi)⊆{a|((user(pi),a∈A))且 pi 有已授的权限；

$$\bigcup_{a\in roles(p_i)}\{r\mid((r,a)\in A)\}$$

容易看出，这个描述系统的定义与[sandhu1996]中提出的$RBAC_0$定义是一致的。对于此定义，可以看出，由定义3.1配置语言说明的J2EE保护系统可以用来执行与$RBAC_0$定义相一致的安全系统。授权矩阵*GRM*明确说明了*PA*的关系。在JS系统中用户管理员管理着*UA*关系，它详细描述了分配给用户的角色属性值。但是，这样的管理功能超过了JS规定的范畴，也意味着UA关系在功能上是针对具体实现的。根据UA的安全属性，`PrincipalAuthenticator`初始化新的主要角色凭证。表3.1提供了一个实例，即属性a1到a6都有相应得角色。类型*AccessId*的主要角色特权属性值相当于函数*user*的返回值。`PrincipalAuthenticator`的实现应该根据函数*roles*来初始化主要角色凭证。由于RBAC的用户可以激活任意一个指定用户的角色子集，UA的实现就确保了$RBAC_0$的实现。从而可以看出，在定义3.4中应用程序能够直接支持所有关联、函数和具体设置。为了实现JS支持$RBAC_0$，应该具备以下条件：

1. 遵守JS标准
2. 提供一种管理用户-角色分配关系*UA*的一种方法。
3. 为用户通过*user代理*来选择一组角色，并可激活新主要角色的方法。
4. 根据关系*UA*创建包括*role*类型特权属性的主要角色凭证的 `PrincipalAuthenticator`实现。
5. 创建仅包括一个*AccessId*类型特权属性的主要角色凭证的 `PrincipalAuthenticator`实现。

在JS中，$RBAC_0$的一个最直接的实现就是使用*role*的特权属性来构建授予权限矩阵，如表3.3。

3.3.4 $RBAC_1$

$RBAC_1$是具有角色层的$RBAC_0$。定义3.5用JS语言正式的定义了$RBAC_1$。

定义 3.5 J2EE保护系统语言定义的$RBAC_1$

$RBAC_1$在$RBAC_0$的基础上引入角色层次。基于J2EE概念定义的$RBAC_1$模型如下：

U，A，R，P，PA，UA，User（P_i），与 $RBAC_0$ 中一样；

$RH \subseteq A \times A$，R 上的一个偏序关系，角色层次；

$Roles: P \rightarrow 2^A$,修改 $RBAC_0$，其中

$roles(p_i) \subseteq \{a | (\exists a' \geq a)[(user(p_i), a') \in UA]\}$

且 p_i 有已授的权限：

$$\bigcup_{a \in roles(p_i)} \{r | (\exists a'' \leq a)[(r, a'') \in PA]\}$$

函数roles是由PrincipalAuthenticator(图3.1)强制执行的。用户为用户代理提供了一组可激活主要角色的角色。在用户代理验证期间，PrincipalAuthenticator创建新的主要角色凭证。假设它们满足针对$RBAC_1$的函数roles的定义，则可以产生用户所需的角色。

$RBAC_1$的有效实现，允许用户指定任意一个比它低级的角色。这种情况下，PrincipalAuthenticator的实现激活了比指定角色低级的所有角色。

为了JS实现对$RBAC_1$的支持，应该具备以下条件：

1. $RBAC_0$实现
2. 提供一种管理角色层关系RH的方法
3. 根据关系UA和RH，利用函数roles，创建包括role类型特权属性的主要角色凭证的PrincipalAuthenticator实现。

3.3.5 $RBAC_2$

RBAC约束，以及函数user 和 roles [Sandhu 1996]，必须应用于UA和PA关系中，用户管理工具强制执行UA关系的限制。PrincipalAuthenticator需要实现对函数user和roles的约束。安全管理工具强制执行PA关系上的约束。

为了JS可以支持$RBAC_2$，应该具备以下条件：

1. $RBAC_0$实现；
2. 实现对用户管理工具约束的UA关系的支持；
3. 实现函数user和roles约束支持下的PrincipalAuthenticator；
4. 强制执行安全管理工具对PA关系的约束。

3.3.6 $RBAC_3$

$RBAC_3$集合了$RBAC_1$和$RBAC_2$，并在角色层上有一些可能的附加限制，它还可以在JS下执行。很明显，为了JS实现可以支持$RBAC_3$，应该具备以下条件：

1. $RBAC_1$实现；
2. $RBAC_2$实现；
3. 在角色层上，执行可能的附加限制。

上面深入的讨论了J2EE安全服务对$RBAC_1$和$RBAC_2$支持的需求。在$RBAC_1$角色层上附加的静态限制是由用户管理工具来实现的。为了支持动态限制，除管理工具外，PrincipalAuthenticator实现方面的其它功能性也是必要的。

3.4 应用实例

为了更好的说明上面所阐述的内容，下面给出一种实例，即是利用角色层次实现J2EE系统的保护状态（定义3.4定义过的）。

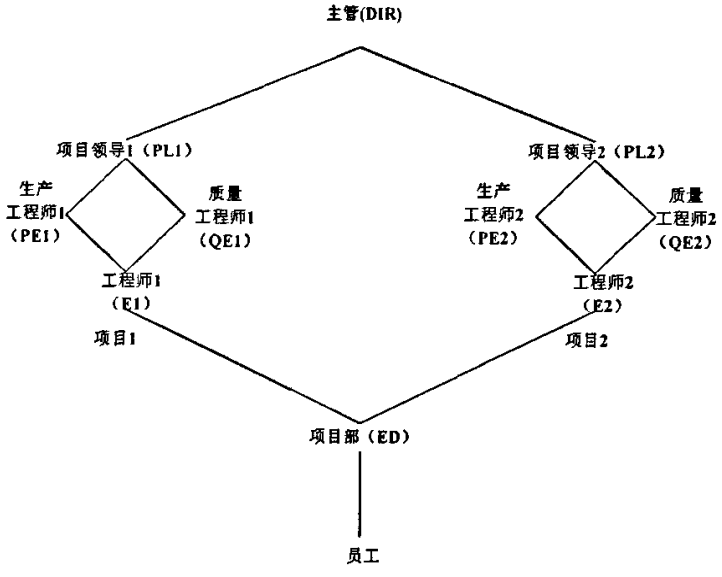


图 3.4 角色层次的例子（[Sandhu 1998b]）

由图3.4[Sandhu 1998b]的实例层可以看出，通过配置基于J2EE的分布式系统来实现对$RBAC_1$的支持，由图3.5和3.6则给出了基于J2EE的分布式系统访问J2EE接口的实现。在RBAC的角色层，一般都是先描述底层的初级角色（很少的许可），然后是高层的高级角色（从初级继承来的许可再加上新的许可）。

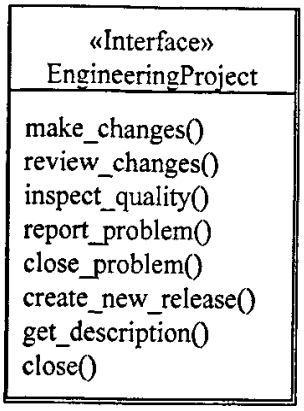


图 3.5 EngineeringProject 接口

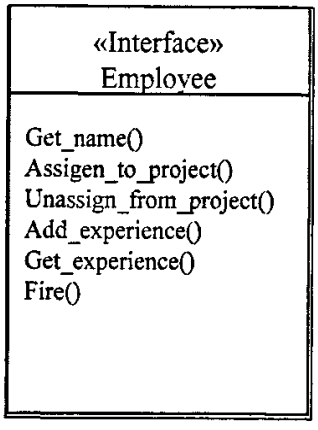


图 3.6 Employee 接口

下面的访问控制策略描述了允许的事件，所有其它事件则被否决。

授权策略

1. 任何人均可查询雇员的姓名和经历；
2. 关于项目，工程部门的所有人都可以得知项目的描述，也能汇报问题；
3. 指定项目的工程师可以改动相关项目并修正涉及项目的改动；
4. 质检工程师检查所指派工程的质量；
5. 生产部工程师创造新的产品；
6. 项目领导者汇集各种问题；
7. 主任管理（指派雇员承担项目，增加雇员的经历记录，开除）雇员和停止工程计划。

函数*ER*返回的是每个属性的授权集合，而CB返回的是每个域的授权集合。

J2EE访问策略域的意向有些不明。为了更明确的理解它，这里提供了两种方法来改进这些策略。一种利用单访问策略域，另一种利用多访问策略域[南 2005b]。

3.4.1 单域解决方案

为了不用访问策略域实现JS中的角色层，这里设计了两种新的接口EngineeringProject1和EngineeringProject2，如图3.7所示。

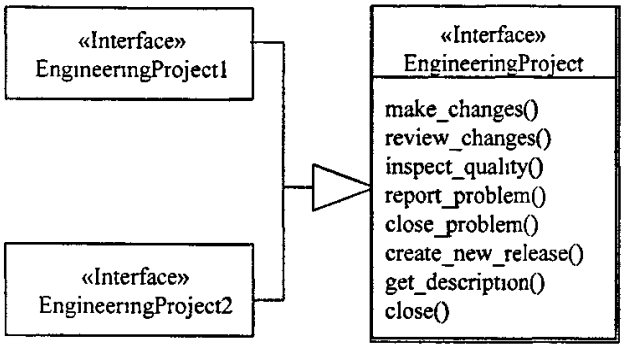


图 3.7 EngineeringProject 接口层次

以下是系统保护态配置：

A = {e, ed, e1, e2, pe1, pe2, qe1, qe2, pl1, pl2, dir}， 所有的属性都有role类型。

IM = {

Employee.get_name,
Employee.assign_to_project,
Employee.unassign_from_project,
Employee.add_experience,
Employee.get_experience,
Employee.fire,
EngineeringProject1.inspect_quality,
EngineeringProject1.make_changes,
EngineeringProject1.report_problem,
EngineeringProject1.review_changes,
EngineeringProject1.close,
EngineeringProject1.close_problem,
EngineeringProject1.create_new_release,
EngineeringProject1.get_description,
EngineeringProject2.inspect_quality,
EngineeringProject2.make_changes,
EngineeringProject2.report_problem,
EngineeringProject2.review_changes,
EngineeringProject2.close,
EngineeringProject2.close_problem,
EngineeringProject2.create_new_release,
EngineeringProject2.get_description

}.

本文不采用任何接口EngineeringProject的实现，仅是使用已分割接口。

• O = {e, ed, e1, e2, pe1,pe2, qe1, qe2, pl1, pl2, dir, prj1, prj2}。 prj1是EngineeringProject1的一个实例，prj2 是 EngineeringProject2的实例。所有O 的其它元素都是接口Employee的实例。

R = {gn, atp, ufp, ae, ge, f, mc1, rc1, iq1, rp1, cp1, cnr1, gd1, c1, mc2, rc2, iq2, rp2,cp2, cnr2, gd2, c2}。

• D = {d1}

• C = {all} –这里仅采用一种结合器。

RRM如表3.7所示。由于所有操作的必要权限都有相同的结合器-"all."，所以可以忽权限结合器群，

• DS = {i, d}

在IDM中，所有接口实例都是单一访问策略域的成员。

GRM如表3.8所示。

表 3.8 单域解决方案的需求权限矩阵

操作	权限
Employee. get_name	gn
Employee. assign_to_ project	atp
Employee. unassign_from_ project	ufp
Employee. add_ experience	ae
Employee. get_ experience	ge
Employee. fire	f
EmployeeringProjedt. get_ description	gdl
EmployeeringProjedt. inspect_ quality	iql
EmployeeringProjedt. make_ changes	mcl
EmployeeringProjedt. review_ changes	rcl
EmployeeringProjedt. report_ problem	rpl
EmployeeringProjedt. close_ problem	cpl
EmployeeringProjedt. create_new_ release	cnrl
EmployeeringProjedt. close	cl
EmployeeringProjedt. get_ description	gd2
EmployeeringProjedt. inspect_ quality	iq2
EmployeeringProjedt. make_ changes	mc2
EmployeeringProjedt. review_ changes	rc2
EmployeeringProjedt. report_ problem	rp2
EmployeeringProjedt. close_ problem	cp2
EmployeeringProjedt. create_new_ release	cnr2
EmployeeringProjedt. close	c2

表 3.9 单域解决方案授予权限的矩阵

特权的属性	授予的权利
e	gn,ge
ed	gd1,gd2,rp1,rp2
e1	mc1,rc1
pe1	cnr1
qe1	iq1
pl1	cp1
e2	mc2,rc2
pe2	cnr1
pe2	iq1
pl2	cp1
dir	atp,ufp,ae,f,c1,c2

$$ER(d_j,a_1,a_2,\cdots,a_l)\subseteq \bigcup_{a_i,1\le i\le l}\{r\mid r\in GRM[a_i,d_j]\}$$

表示每个属性所授权限的集合；

$$CB(r_{1,d_1},r_{2,d_1},\cdots,r_{l,d_1},\cdots,r_{1,d_p},r_{2,d_p},\cdots,r_{m,d_p})\subseteq \bigcup_{每个域d}\{r\mid r\in\{r_{1,d_1},r_{m,d_p}\}\}$$

表示每个域中所授权限的集合。

上述描述的J2EE保护系统配置允许了策略的执行。如，角色pl1的项目1的主管既能够在接口Employee处调用操作*get_name*和*get_experience*，也能在接口EngineeringProject1上关闭大部分操作。

从J2EE保护系统在此方法的配置上看，系统管理开销变的尤为重要。这种开销一般是由于每个项目中分离接口（EngineeringProject(1,2)）不必要的应用。这是因为系统自动地限制了单访问策略域的解决方案。以下阐述了如何用多访问策略域和域的层次来消除保护系统配置数据中不必要的的冗余。

3.4.2 多访问策略域解决方案

一旦每个项目都有一个访问策略域，就可以对所有的项目循环利用EngineeringProject 接口 。也可以利用在不同层级中组合域的JS性能。图3.8展示了一个有限而简单的树型结构层级。以下内容用到了系统保护态的配置：

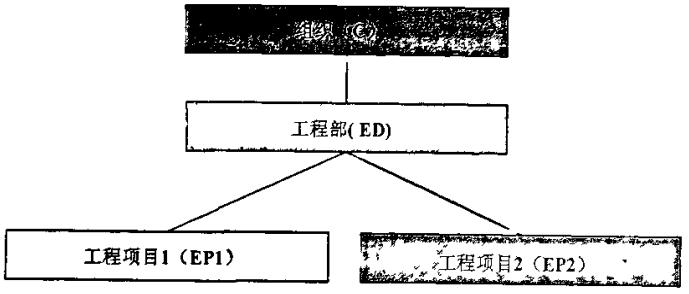


图 3.8　多域解决方案中域的层次

A, *O*, *C*, *DS*, *ER*, 和CM与单域方法中相同，

- *IM* = {
  Employee.get_name,
  Employee.assign_to_project,
  Employee.unassign_from_project,
  Employee.add_experience,
  Employee.get_experience,
  Employee.fire,
  EngineeringProject.inspect_quality,
  EngineeringProject.make_changes,
  EngineeringProject.report_problem,
  EngineeringProject.review_changes,
  EngineeringProject.close,
  EngineeringProject.close_problem,
  EngineeringProject.create_new_release,
  EngineeringProject.get_description
  }.
- *R* = {gn, atp, ufp, ae, ge, f, mc, rc, iq, rp, cp, cnr, gd, c}.
- *D* = {C, ED, EP1, EP2}
- *RRM*如表3.10所示，除接口`EngineeringProject`用来代替两个不同名字的相同接口外，其它与表3.8中的都是相同的。

表 3.10　多访问策略域解决方案需求权限矩阵

操作	权限
Employee. get_name	gn
Employee. assign_to_ project	atp
Employee. unassign_from_ project	ufp
Employee. add_ experience	ae
Employee. get_ experience	ge
Employee. fire	f
EmployeeringProject get_ description	gd
EmployeeringProject inspect_ quality	iq
EmployeeringProject make_ changes	mc
EmployeeringProject review_ changes	rc
EmployeeringProject report_ problem	rp
EmployeeringProject close_ problem	cp
EmployeeringProject create_new_ release	cnr
EmployeeringProject close	c

IDM如表3.11所示。

表 3.11 多访问策略域解决方案接口实例域成员矩阵

接口实例	域			
	C	ED	EP1	EP2
e	X			
ed	X	X		
e1	X	X	X	
pe1	X	X	X	
qe1	X	X	X	
pl1	X	X	X	
e2	X	X		X
pe2	X	X		X
qe2	X	X		X
pl2	X	X		X
dir	X			
prj1	X	X	X	
prj2	X	X		X

正如图3.9的例证，根据如图3.8所示的域层，如果一个对象属于子域，那么它也是所有父域的组成部分。

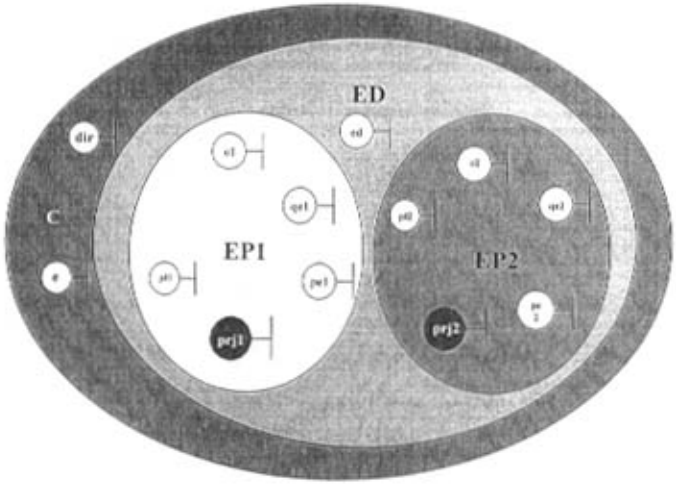


图 3.9 接口实例域成员

GRM如表3.12所示。

表 3.12 多访问策略域解决方案授权矩阵

特权属性	授予的权利			
	域			
	C	ED	EP1	EP2
e	gn	ge	–	–
ed	–	ge,rp	–	–
e1	–	–	mc,rc	–
pe1	–	–	cnr	–
qe1	–	–	iq	–
pl1	–	–	cp,ae	–
e2	–	–	–	mc,rc
pe2	–	–	–	cnr
qe2	–	–	–	iq
pl2	–	–	–	cp, ae
dir	atp,ufp,ae,f,c	–	–	–

前面所述的J2EE保护系统配置允许与单域方法中的配置相同的策略执行。此时，无论是每个项目都有单独的EngineeringProject(1,2) 接口，还是有多余的权限，都没有必要了。而RRM和GRM也变得更加通俗易懂。

鉴于访问策略域的层级结构，这样的系统能够支持更灵活的策略。例如，表3.11中的GRM，支持允许工程项目的领导把工作经历添加到监督管理工作雇员记录里的策略。为了给它授权，只要把雇员指派到一个项目（假设在同一时间每个雇员从事一个项目），表现雇员的接口实现就会把相应这个工程部门的访问策略域分离过去。GRM也强化了仅允许同一部门的同事来查看雇员经历的的细粒度策略。

3.5 本章小结

中间件访问控制机制的理解在企业应用程序的资源保护上是很关键的。本章，不仅详细阐述了最有效的中间件安全技术之一的访问控制机制——J2EE安全——也指出了一种J2EE保护系统的配置。利用配置说明，提出了一种正式说明在JS方面授权的语义的算法。

本文用JS语言定义了$RBAC_0$和$RBAC_1$模型，并描述了$RBAC_0$－$RBAC_3$是如何用JS来实现的。为支持RBAC，除遵守JS标准外，讨论了还需要去实现什么功能。本文利用图例说明了JS保护配置系统的单策略域和多策略域的实例，这些都支持角色层和访问策略。

适应JS规范的实现支持$RBAC_0$－$RBAC_3$。但JS没有明确说明的附加功能却是必要的。PrincipalAuthenticator接口和 Usersponsor的实现必需支持角色和它们的层级($RBAC_1$)。为支持约束($RBAC_2$)， PrincipalAuthentiCator必需的它们进行加强。而用户到角色、角色到权限之间关系的管理工具也是必要的。

本章逐步展开了实现和用JS评估RBAC模型的一个框架。它为JS开发人员提供了认识此系统下RBAC的方向，也给用户提供了一个选择支持$RBAC_0$－$RBAC_3$族模型的标准。这些工作加深了对J2EE AC机制性能的理解，也能充分利用保护应用资源的中间件的效用。

尽管 RBAC 可以代替主要的 AC 模型，但是它的性能却是有限的，鉴权策略将成为这种模型的挑战。另外，J2EE AC 机制的颗粒度将限制在接口操作级别上。这就是为什么 RBAC 和 J2EE 还不能满足所有应用域的需求。下面的内容，即第二个主要研究内容，描述了 RBAC 模型和 J2EE 机制不能满足的情况。

第四章 资源访问判定服务

前面几章阐述了分布式资源的访问控制问题，并且针对相关的工作介绍了一些常用的技术。在那些应用领域中，如果 RBAC 能够支持其鉴权规则，并且 J2EE 的访问控制粒度能够满足的情况下，基于 J2EE 的 RBAC 访问控制机制是足够的。但是如何处理那些不能被 RBAC 支持，或者 RBAC 模型/J2EE 安全访问控制粒度不能完全满足的情况呢？本章研究了一种符合需求的模型——JCRADS（基于 J2EE-CORBA 的资源访问判定服务）。此外，通过带有访问判定策略的例子来演示它的实用性。

资源访问判定服务定义了一个概念上的结构，它概括了在一个鉴权服务中的鉴权逻辑，该鉴权服务在应用程序的外部，也是独立于下面的安全模型和安全规则的。这个结构不但在相当大程度上简化了应用程序和安全系统开发，而且也允许系统始终如一地管理和实施自己的安全规则。

资源访问判定办法满足了保护大多数分布式应用资源的功能和性能需求。对于鉴权判决，可以使用基本证明技术提供的多种类型的主体安全。服务结构允许使用从工作流和其它资源中获得的信息，因此它支持应用领域所特有的策略。它也能够使用在访问控制判定中针对应用的信息。由于鉴权逻辑包装变为一个独立的服务，因此能够容易地实现穿越应用边界时访问控制策略的一致性。此外，这个结构支持多重鉴权模型，并且它有助于安全管理员和应用开发者进行清晰的责任分离。为了实现这种优点，进行设计时，需要在应用层上实施鉴权判定，并且要取得应用系统开发者和使用者之间资源名字在语法意义上的一致性。

尽管当前的设计利用 J2EE 所遵循安全机构，并且需要用非常复杂的技术和体系进行优化，资源访问判定几乎是独立于下层的安全技术的。特别的，取代 J2EE 安全服务标准是没有意义的。然而，资源访问判定办法能够被应用到大多数分布式环境中。

除此之外，本章还展示了在应用和授权服务之间，在没有复杂的交互、并且没有巨大通信开销的情况下，能够完成来自于应用层的鉴权逻辑退藕。使用传统访问矩阵[Lampson 1971]的鉴权系统也能通过一个基本工具来支持针对应用领域的访问判定服务。

4.1 资源访问判定服务体系结构

资源访问判定服务的主要目的是把应用层鉴权逻辑从应用逻辑中分离出来。如同前面讨论的那样，主要的中间件技术是在对中间件客体级提供最好的访问控制粒度。这种鉴权服务是为能访问哪些信息或计算资源做出判定。因此，资源访问判定服务补充了中间件访问控制机制。它依赖并使用中间件安全环境来证明服务和应用之间以及服务各部

分之间通信的正确性（例如：信息可靠性，机密性和完整性保护）。它也假设下层的安全为应用提供一个手段来获得访问主体的安全属性。就像前面所讲述的一样，这些假设对于大多数的中间件安全技术是成立的。

4.1.1 应用系统和资源访问判定服务间的接口

像大多数的这种服务一样，资源访问判定服务为一个应用系统(Application System，AS)提供了鉴权判定。鉴权逻辑包含在AS外部的资源访问判定服务中，它是一个应用程序的传统部分。由于从逻辑上能够把服务进行集中，这种方法允许应用根据鉴权规则的宽度设置来加强AC，自然就保证了策略的一致性。在这个方法中，在启用某个对象的某个方法之后，获得鉴权判定。因此，通过关联资源名字和任何大小、语义的元素，应用系统能够实现任意粒度水平的访问控制。

在图4.1中描述了应用客户、应用系统与鉴权服务之间相互作用的流程。相互作用的序列如下：

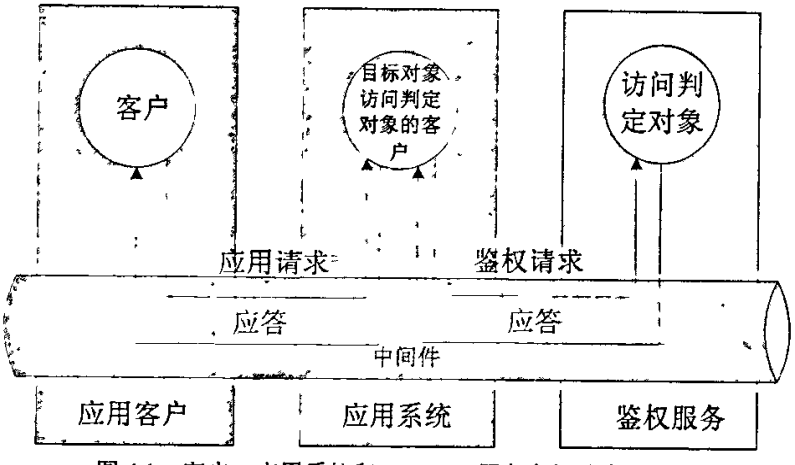


图 4.1 客户、应用系统和 JCRADS 服务之间的交互流程

1. 系统的客户调用应用上的一个操作；
2. 为了处理这个调用，应用系统需要鉴权服务的一个鉴权判定；
3. 服务给出一个判定，并返回给应用系统；
4. 应用系统执行这个判定。如果鉴权服务授权访问，应用系统返回此调用期望的结果；否则，它或者返回部分结果或者抛出一个异常。

应用系统和鉴权判定服务之间使用简单的接口。应用系统开发者为了获得一个鉴权判定只需要编写一个单独的程序调用鉴权服务。每一个鉴权请求由客户主题安全属性、访问的资源名字和这个资源上要完成的操作的名字组成。调用主题的安全属性建议通过应用程序从中间件安全基础设施获得。由应用系统作为它本身的应用逻辑来计算资源和操作名。对于每个鉴权请求，它收到一个二进制的决定（1：yes/0：no）。一个应用系统只从一个JCRADS实例获得一个鉴权判定。为了生成鉴权判定，应用系统和鉴权服务

之间要传输一定数量的数据。生成鉴权请求时，应用传递这样三个参数：表示要访问的资源名的名字一值对序列；访问操作名（例如："creat"，"write"，"use"，"delete"）；该客户的认证安全属性。

这儿的安全属性是当前用户会话的正规属性。客户传递的参数中，头两个（资源名和访问类型）是最值得讨论的。本文提出一个抽象概念，叫做"保护的资源名"，或者就是"资源名"，用于抽象依赖于应用的实体语义，应用系统控制对该实体的访问。资源名可以与应用系统拥有者任何有价值的资源相关联，按照拥有者的兴趣对资源进行控制。

系统把针对应用系统的信息、资源和操作名进行统一编码，然后把它传送给JCRADS服务。比如，向账户存入100元，就可以表示成操作"Store-to-myaccount：500"，资源名携带着帐号。用于操作名和资源名的数据结构很简单也很普通，但是实验证明，这个结构对这类任务有非常强的表示能力。

在应用请求JCRADS服务器进行鉴权判定之前，建议先标识好与这个客户请求相关的资源名和访问操作名。因为对于每个应用系统（至少每个应用域），与抽象资源名相关的实体的方法可能不相同，所以没有定义特别的算法来完成这样的关联。

本方法与大多数基于鉴权服务在客户、应用服务和JCRADS服务器之间的交互方式很相似，但是在它的元素内部构成是不同的。

4.1.2 资源访问判定的逻辑组成

不同的应用场景需要不同的访问控制策略、性能、可测量性和其它的特性，JCRADS体系结构的目标是使它的构件能满足所有这些需求。下图中的构件种类组成了一个JCRADS服务：访问判定对象作为JCRADS客户的接口，并且协调JCRADS构件之间的交互。访问控制策略控制对保护资源进行的访问，它可允许任意多个策略评价程序进行评价判定。判定结合程序通过一定的结合策略，把潜在的多个策略评价程序评价的结果进行结合，然后形成最终的鉴权判定。策略评价定位程序，对于一个给定的到保护资源的访问请求来说，负责跟踪判定结合程序以及多个策略评价程序，并为这些程序提供参考，对于对形成鉴权判定责任。动态属性服务程序基于访问操作和资源名上下文来收集和提供客户的动态属性。

这些构件只是逻辑上进行了划分，实际上它们可以协同定位在一个过程或者主机内。与基于JCRADS体系结构的服务提供的高可用性和容错性一样，这个特征用于提供对动态构件和重配置的更进一步的支持。图4.2展现了鉴权服务构件之间的交互。如下：

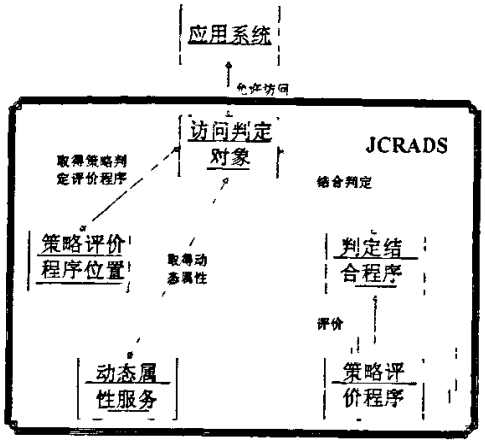


图 4.2 JCRADS 构件之间的交互

1．鉴权服务通过访问判定对象接口接受一个请求；

2．访问判定对象获得与请求的资源名相关的策略评价程序和判定结合程序的对象参考；

3．访问判定程序根据资源名和想要执行的访问操作上下文，通过动态属性服务来获得主题的动态属性；

4．因为可能用到多个策略评价程序来形成鉴权判定。访问判定对象委托判定结合程序的一个实例轮流检测策略评价程序（第一步选择的），并且把多个策略评价程序形成的多个评价结果封装成一个最终的判定。

5．判定结合程序从 PE 获得判定且根据结合策略进行结合。判定转到访问判定对象，它一次性把判定返回给应用程序。

JCRADS体系结构的所有构件都可以动态地由遵从接口规范的不同实现来替换。这使得它能支持应用的插入和删除、策略的变化和计算环境的变化等。例如，如果应用插入后，需要保护新的资源，就需要动态地增加一个新的PE（或者是一套PE），为能使用这些新的PE，PEL也要重新配置。图4.3表现了鉴权策略的一个交互流程的例子。下面展示JCRADS怎么样支持鉴权策略及对策略变化的支持。

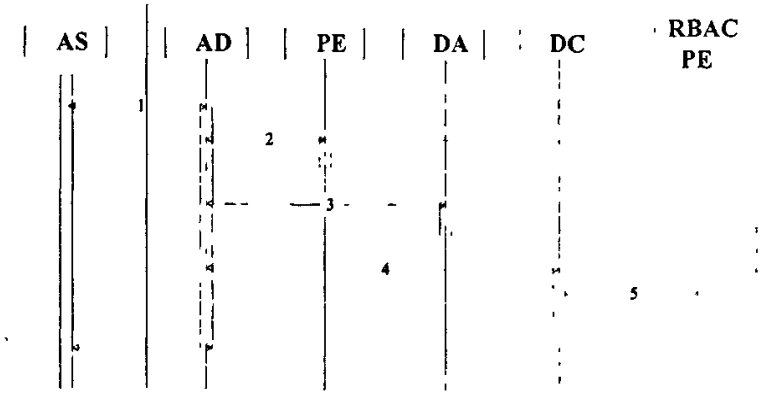


图 4.3 一个交互流程的例子

与大多数鉴权判定服务（services Simon 1997, Varadharajan 1998, Woo 1993c）不一样，JCRADS体系结构支持不同的鉴权判定策略的实现。如下图（图4.4）所示，可以看出鉴权策略表示的范围与JCRADS体系结构的范围不同。每一个PE由不同的接口管理。这样设计使得JCRADS可以很好的支持现存的策略引擎（并且支持将来的一些引擎），这些引擎最初可能不是为PE开发的。

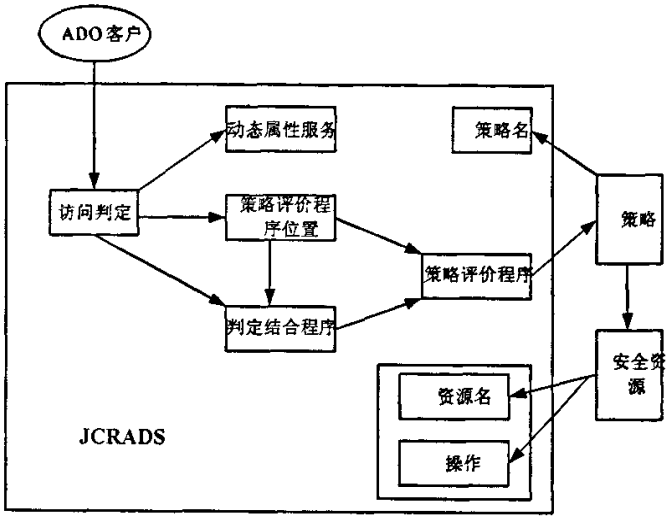


图 4.4 主要元素及其附件

每个请求使用一个鉴权引擎来评价请求的访问，这个鉴权引擎支持一个特定的策略。JCRADS中引入了多个评价程序和一个结合程序，对于同一个请求，这就为多个策略（甚至可能是不同类型的策略）管理鉴权判定提供了方法。这个与[Bertino1996b]相似，Bertino定义了一个带冲突解决和最高规则的外部鉴权模型。在JCRADS中，这样的规则由特定的判定结合程序来实现。

JCRADS体系结构元素上的区别之一是使用了动态属性服务。它支持这样的策略：一些要素的值从一个请求传到另一个请求时发生变化或者由部门的工作流状态来确定。这些要素就是动态属性，语义上与主题安全属性一致。在它们传递请求到相应的DC或者PE之前，ADO从DAS获得它们。动态属性是这样的一些属性，它们的值只能在鉴权判定请求发生时才能被确定。这样，它们就是针对问题中的请求的。JCRADS体系结构中DAS的引入增加了可用的鉴权判定生成信息的变化，并且使得使用传统的访问矩阵[Lampson 1971]来支持复杂和动态的访问控制策略成为可能。

所有的JCRADS构件，加上上面介绍的运行接口，都有接口来管理它们。这些接口构成了JCRADS的管理模型，它的范围和主要元素如图4.5所示。

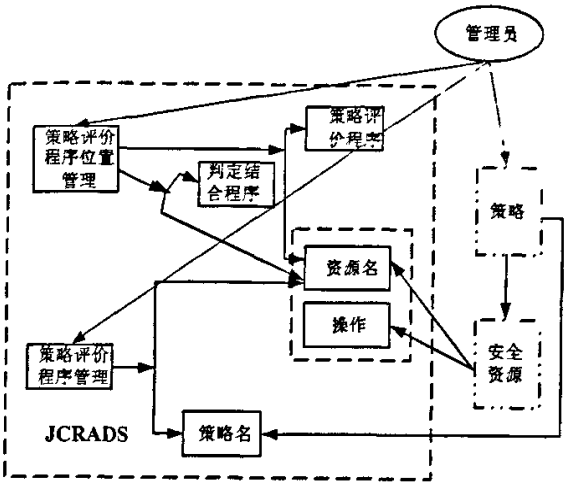


图 4.5 管理元素及其附件

JCRADS不指定鉴权策略及其表示手段，它允许JCRADS管理程序对要访问的资源使用针对PE接口定义的策略，这是通过策略名的概念和可管理的PE和PEL接口来实现的。对于那些能评价多个策略的PE，策略名用来关联策略和一个资源名。通过对策略命名而不提供策略表示的定义，就可以使得JCRADS能适用多数已有的和将来的鉴权语言。

运行和管理接口、支持的数据结构、J2EE IDL中的所有定义以及相关语义描述，构成了JCRADS的体系结构。如下图（图4.6）所示。

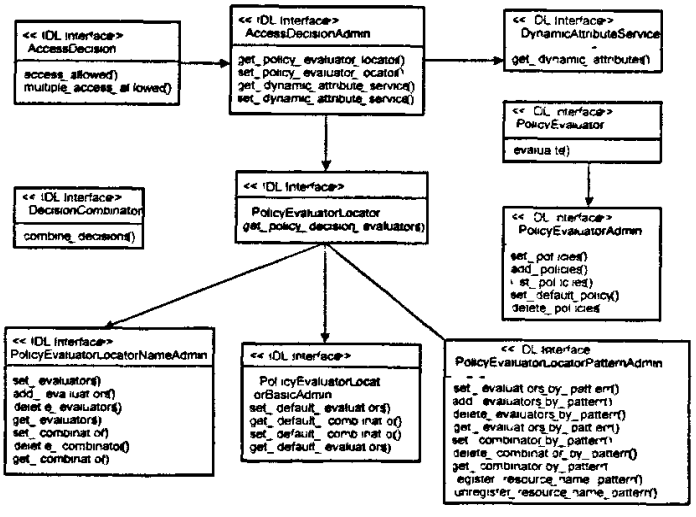


图 4.6 JCRADS 体系结构的计算部分

JCRADS体系结构的管理部分在一个实现内允许JCRADS对象的替换，例如，AccessDecisionAdmin接口包含检测和描述PEL参考，set_policy_evaluator_locator()操作允许管理程序把ADO指向不同的PEL实例，这个操作改变后，ADO将使用新的PEL。因此，通过这个方式，JCRADS就可以实现对策略和环境改变的支持。

4.2 JCRADS 对 JAVA-CORBA 的支持

JavaIDL是Java 2开发平台中的CORBA功能扩展。在Java 2中引入JavaIDL，使得利用OMG IDL能够定义服务对象的基本功能，并且将IDL根据CORBA规范的要求，映射到Java语言，并以此开发出标准的具有互操作性和可连接性的分布式应用。JavaIDL使分布式、支持Web的Java应用可以基于IIOP协议透明地调用远程服务。

JavaIDL运行-时间（Runtime）组件包括一个全兼容的对象请求代理——Java ORB，用于基于IIOP协议实现分布式对象之间的通信。该ORB支持瞬态CORBA对象和瞬态名字服务器，并且ORB生存期受运行ORB进程生存期的限制。

在程序设计中，首先对要实现的服务对象功能进行系统分析，并创建IDL接口描述文件对功能进行描述。然后利用JavaIDL提供的IDL到Java语言的映射工具将IDL文件映射为客户端桩（Stub）文件和服务器骨架（Skeleton）文件。

在实现的客户端应用程序中，包括对远程对象的引用、服务功能请求的发送以及服务对象返回结果的解析处理等功能。通常，客户端应用程序利用命名服务实现对远程对象的绑定，并通过客户端ORB将客户端与服务对象联系起来，实现方法的远程调用。

在服务器端，ORB利用服务对象骨架将调用请求和参数的数据格式进行转换，把远程调用转换为对本地对象中方法的调用。当方法返回时，骨架对计算结果进行转换和封装，通过ORB把结果返回给客户机。

4.2.1 建立 CORBA 应用程序的过程

分布式应用程序设计的主要问题是确定建立在对象级上的客户与服务对象的关系，从其最根本的功能来讲，服务对象提供远程接口，客户对象调用远程接口，客户对象不需要了解远程CORBA对象的位置以及实现细节，也不需要了解哪个ORB 用于对象之间的交互。

按照实现的基本过程，CORBA对象服务的实现方式分为两种：对象的命名引用方式和字符串化对象引用方式。CORBA创建分布式应用程序的过程大体如下：

- 进行系统分析，确定服务对象需要实现的功能；
- 根据系统分析结果，编写IDL接口说明文件；
- 编译接口说明文件，产生服务对象的骨架与客户对象的桩（可选）；
- 基于客户对象的桩，编写客户对象程序；
- 基于服务对象的骨架或者动态请求实现，编写服务对象程序；
- 分别编译客户对象和服务对象程序；
- 启动服务对象程序；
- 启动客户对象程序。

4.2.2 应用程序示例

以下用一个例程说明建立应用程序的过程：

1、对象功能描述和系统简要设计

在服务对象端将一个字符串对象赋值，客户端通过调用服务对象方法获取该字符串的值。根据对象功能的说明，用UML描述出服务对象需要实现的功能：

getIt() : String[]

2、服务对象接口定义

根据系统分析结果，用IDL编写出服务对象方法描述程序getMessage.idl：

```
module getMessage
{ interface getIt
    { string returnObject();
    };
  };
```

3、编译getMessage.idl

idltojava -fno-cpp getMessage.idl。

4、编写客户端程序

```
//引入相关类库
import org.omg.CosNaming.*;
import org.omg.CORBA.*;
//客户端对象方法
public class client
{ public static void main(String args[])
    { // 创建和初始化ORB
    ORB orb = ORB.init(args, null);
    // 获取根命名服务上下文对象
   org.omg.CORBA.Object naming =
   orb.resolve_initial_references("NameService");
    NamingContext namingContext = NamingContextHelper.narrow(naming);
        //解析命名中的对象引用
        NameComponent nc = new NameComponent("getMessage", "");
        NameComponent path[] = {nc};
    getMessage.getIt method =getMessage-
Helper.narrow(namingContext.resolve(path));
    // 调用服务对象方法
```

```
        String result=method.returnObject();
        }
    }
```

5、编写服务对象程序

```
    // 引入相关类库
    import org.omg.CosNaming.*;
    import org.omg.CosNaming.NamingContext
    Package.*;
    import org.omg.CORBA.*;
    //服务方法
    class returnMethod extends _getMessage-
    ImplBase
    { public String getIt()
      { String result ="How about it";
        return result;
      }
    }
    //服务器端方法
    public class server
    { public static void main(String args[])
      { // 创建和初始化ORB
        ORB orb = ORB.init(args, null);
        // 创建服务对象并将其向ORB注册
        returnMethod obj=new returnMethod();
        orb.connect(returnMethod);
        // 获取根命名上下文
        org.omg.CORBA.Object objRef =
        orb.resolve_initial_references
         ("NameService");
        NamingContext ncRef = Naming
        ContextHelper.narrow(objRef);
        // 绑定命名中的对象引用
        NameComponent nc = new NameComponent("getMessage", "");
        NameComponent path[] = {nc};
        ncRef.rebind(path, objRef);
```

```
        // 等待来自客户机的调用
        java.lang.Object sync=new java.lang
        .Object();
        synchronized (sync)
            { sync.wait();
            }
        }
    }
```

6、分别编译服务器端和客户端程序

（1）编译服务器端程序:

```
javac getMessage\server.java
```

（2）编译客户端程序:

```
javac getMessage\client.java
```

7、运行

（1）打开一个仿真终端窗口，启动命名服务，其中3388为通信端口号：

```
tnameserv -ORBInitialPort 3388
```

（2）在另一个窗口中输入以下命令，运行服务端程序:

```
java server -ORBInitialPort 3388
```

（3）在另一个窗口中输入以下命令，运行客户端程序：

```
java client -ORBInitialPort 3388
```

4.3 应用实例

JCRADS概念上的结构是非常广泛的，并且JCRADS各组件的作用以及它们之间的相互作用是很难理解的。为了阐述JCRADS结构和它的性能，本节举出了一个详细的例子，这个例子进一步澄清了JCRADS的概念，它也展示了JCRADS对基于角色和关系这种策略的支持方法。

考虑一套典型的科技资源访问控制领域的规则。科技资源基础条件平台由许多分布式系统构成，这些分布式系统被用来：

1）收集、登记、整理信息和应用资源；

2）管理大平台用户、其它子平台用户以及自己已有和新建用户等；

3）控制对大平台、其它子平台以及本身的资源访问。

4.3.1 初始化策略

平台采用表4.1所列规则来控制科技人员对科技资源的访问，资源的所有记录由表4.2所示部分组成。

这个规则粗略地分配用户的等级，它允许任何用户都能看到平台的资源目录列表；资源项目的录入人员能够修改和维护相应的资源项目；特定领域的一般用户能够读取本领域相应的资源信息；管理人员能够设定当前的规则；用户可以浏览相应领域内的资源。除此之外，这个规则没有反映资源的创建者、拥有者、提供者，而他们有资格知道关于相应资源的一些信息。

表 4.1 访问控制规则

规则编号	规则定义
P1.1	任何用户都能看到资源目录列表
P1.2	资源项目的录入人员能够修改和维护相应的资源项目
P1.3	特定领域的一般用户能够读取本领域相应的资源信息
P1.4	二级门户特权用户可以浏览本门户内的所有资源
P1.5	大门户用户可以浏览大门户、各相关二级门户内的所有资源
P1.6	大门户特权用户可以浏览系统所有资源
P1.7	系统管理员能够修改系统当前的规则

表 4.2 资源记录条目

项目名称	标识/缩写
题名	Tit
主题	Sub
描述	Des
来源	Sur
创建者	Cre
出版者	Pub
内容	Con
权限	Rig
日期	Dat
类型	Typ
格式	For
标识	ID

4.3.2 模型策略

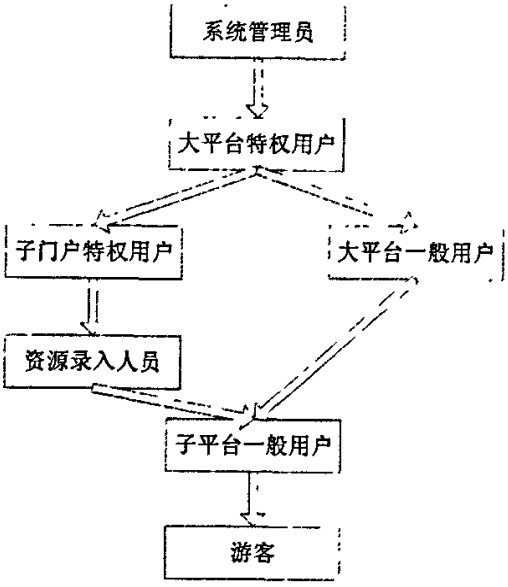


图 4.7 角色层次（角色、层次间的关系，RH）

表 4.3 用户—角色的分配关系（UA）

角色	用户						
	a	b	c	d	e	f	g
系统管理员	+						
大平台特权用户		+					
大平台一般用户			+				
子门户特权用户				+			
资源录入人员					+		
子平台一般用户				+		+	
游客							+

使用带有角色等级－$RBAC_1$ [Sandhu 1996]的 RBAC 模型能够实现策略 1。为了定义 $RBAC_1$ 系统的构成，需要制定基于角色等级、用户到对象和特权到角色的关系，还有用户和角色的功能。在图 4.7 中定义角色等级(RH)。根据这个等级，例如，资源录入这个对象，除了具有和子平台一般用户相同的特权之外还有它自己的特权。表 4.3 显示了用户到角色的分配关系(UA)，在表中可以看到用户 g 被分配给游客用户，用户 d 被分配给子门户特权用户，这意味着，当用户 d 登录进入系统时，他能够激活对象资源录入人员、子平台一般用户或游客，然而用户 g 只能够激活角色游客，这是因为，根据角色等级一个用户能够操作任何比分配给他的角色等级低的对象。如果用户 d 激活角色资源录入人员，那么他将拥有分配给其所能用的特权。

表格 4.4 显示了许可到角色的分配关系(PA)。

表 4.4 许可—角色的分配关系

角色	资源											
	Tit	Sub	Des	Sur	Cre	Pub	Con	Rig	Dat	Typ	For	ID
系统管理员	R	R	R	R	R	R	R	R	R	R	R	R
大平台特权用户	R	R	R				R		R	R		
大平台一般用户	R	R	R				R					
子门户特权用户	R	R	R	R			R					
资源录入人员	R	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
子平台一般用户	R						R					
游客	R											

JCRADS 按照由上面 PA、UA 和 RH 关系定义的 RBAC 系统，实施鉴权，图 4.8 给出了它的结构。

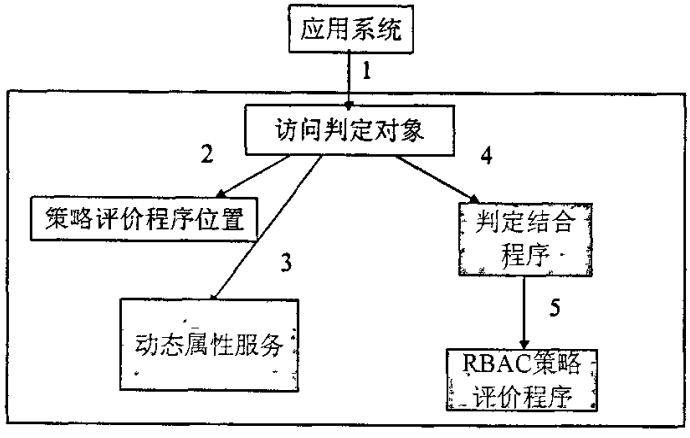


图 4.8 JCRADS 的基于角色策略的配置

1. 访问允许；
2. 取得策略判定评价程序；
3. 取得动态属性；
4. 结合判定；
5. 评价。

ADO 从 PEL 为 DC 和单个 PE(RBAC PE)获得一个参数，PEL 总是返回相同的参数。DAS 返回它从 ADO 收到的同样的安全属性列表。RBAC PE 应用 PA 关系评价鉴权请求。如果 PE 返回“unknown”作为评估结果，DC 就拒绝访问(例如，如果资源名在 PA 表中没有找到)，否则它就返回和 PE 一样的值。

本文设计了一个分布式安全环境，这个环境支持在验证过程中用户对角色的激活，这是通过实施 UA 和 RH 关系并且使用用户和角色实现的。这意味着角色由下层的安全环境配置。现有安全技术能够实现这种假设。例如，在上一章中研究的 J2EE 安全服务对 $RBAC_0$- $RBAC_3$ 模型的支持。在这个例子中，一个向 JCRADS 做出鉴权请求的应用要提供一个基本安全属性的列表，这些属性包括用户激活的所有角色。

使用 RBAC 给资源访问判定规则建模的另一种方法是把判定用户角色的任务分配给 DAS 或者 RBAC PE 本身。在这两种任务分配方法中，本文选择的是第一种方法，即把判定用户角色的任务分配给 DAS，这是因为激活的角色是用户管理员管理的安全属性。这种方法也支持任务的动态分离，这是通常情况下 RBAC 的基本特征。

4.4 本章小结

本章提出了一种为分布式应用系统分离鉴权和应用逻辑的方法，并证明了退藕是一种实现对分布式应用资源进行控制访问的一种非常实用的方法。

本章设计了一个基于J2EE-CORBA的鉴权服务结构－JCRADS。JCRADS非常简单、通用，并且还能够支持许多应用领域的鉴权判定。当使用JCRADS方法时，鉴权和应用逻辑分离，应用鉴权给基于JCRADS的鉴权服务进行鉴权判定。JCRADS结构能够提供任何层次的保护资源粒度，因为类所属的数据结构代表一个资源名，访问相应资源时使用该资源名。与其它鉴权服务结构[VaRADharajan 1998, Zurko 1998]相反，这个结构是与策略无关的，它允许使用各种类型的策略。例如，本文例示的JCRADS对基于角色的策略的支持。

JCRADS结构也与作出鉴权判定的原始信息无关，这只要求该信息能够以主要安全属性的形式正确地表示。这个特点允许基于JCRADS的服务支持多种类别的鉴权信息。此外，动态属性服务的信息不强求针对请求的信息采用标准的方式。因为对服务鉴权的请求从应用的内部发起的，只有当应用系统处理客户请求时，它才能够向服务提供可用信息，例如，通过[Woo 1998]，用户请求不予支持。JCRADS能够把鉴权逻辑包装成一个服务，这个服务能够为许多应用系统服务。

新的应用能够很容易地添加到分布是应用系统环境中，或者从分布式计算环境中剔除，而不影响JCRADS体系结构。就像本文所展示的，鉴权策略的改变会造成JCRADS各组件或者它们之间的重新组合或者配置，并且可能替换掉其中的一部分，从理论上讲，这样能够动态地实现策略的变化而无需关闭服务器。鉴权逻辑的变化在同一点上实现，这使得管理非常稳定。上面所述证实了以前的说法，即JCRADS方法能够在策略、应用、计算环境和用户之间频繁的切换。这个模式与Adage的设计[Zurko 1998]模式相同。在分布式环境中鉴权判定服务的管理和鉴权接口被封装成一个独立的实体，通过J2EE接口提供给管理客户和应用服务。在每个对ADS的鉴权请求中，应用指定访问主体的名字、资源名(取自Adage的术语)和对资源操作的行为。

在设计上JCRADS与Adage也有许多差异，最主要的不同在于JCRADS把鉴权服务作为其内在的部分。在Adage中，一个RBAC鉴权服务器、两个策略数据库和一个译码器被预定义并且建立起ADS。JCRADS结构允许不同的评价机制，同时有自己的规划语法和管理接口。本文定义了JCRADS客户和管理员的接口，而且定义了策略评估、判定结合以及其它JCRADS的内部构件。JCRADS内部接口允许在JCRADS服务中定义，

对第三方兼容 JCRADS 的组件的可以动态安装。此外，Adage 鉴权服务能用作 JCRADS 策略评价程序。

JCRADS 重新利用 J2EE 安全服务基础，它依靠服务来提供所有其它安全功能，诸如用户安全管理(成员关系、角色分配等)、安全性、通信完整性和保密性、审计以及不可抵赖性。然而，在 Adage 中，鉴权服务器和 ADS 管理工具与用户管理和安全基础的保证部分紧紧地整合在一起，以便当用户进入或者离开角色时评估所用的策略，如果所实施的策略要求实体的静态和动态分离[Gligor 1986]，需要保持每个角色的状态和每个主体的当前标号。此外，服务器执行部分用户管理工作(实现实体的静态分离)和保证工作(实体的动态分离)。另一个差别是在 Adage ADS 中存在两个逻辑上有区别的数据库。一个用来储存译码器定义的 Adage 策略对象，另一个储存编译形式，这个定义支持鉴权服务器作出的评价。

无论这个方法如何可靠，建立它的基础性能可行性是重要的，这就是为什么本文要按照 JCRADS 结构开发一个原型鉴权服务。后面的章节将描述这个服务及研究的结果。

第五章 JCRADS的原型工具研究与设计

前一章提出了一个方法来解决对分布式资源的访问控制问题－JCRADS，并且展示了这一结构的主要优点：它能使应用和鉴权逻辑上分离；它支持细粒度资源的AC；可以通过配置来支持不同的AC模型，特别是RBAC；它支持使用应用领域所特有的因素，诸如用户和资源所有者之间的关系等；它能够使用由不同开发者创建并由不同的鉴权机构管理的鉴权服务器；它的分布式特性能使鉴权判定保持一致等。

然而，存在的问题是如何设计一个灵敏的(即：对策略和条件的变化敏感)、可扩展的(即：能够兼容新的功能)并且轻便的鉴权服务，以及使用这个方法会产生什么样的性能。这些问题非常关键，以便理解这个方法的有效性和这个领域其它方法的有效性。

为了研究这些问题，本文设计并配置一个测试床－基于应用鉴权服务的JAS。它附着于JCRADS服务，并且作为对JCRADS方法研究的一个框架。除了把JAS开发成一个测试床之外，本文还获得了对配置应用鉴权服务的一些经验。

这章是专门设计和配置JAS的，主要的设计要求是灵敏度、扩展性和可配置性。针对于通用、面向对象、基于组件的分布式安全服务，实际利用的设计模式提供了简单并且精确的解决方法。服务基于J2EE技术，并利用J2EE的命名服务。

本文定义的计算模型是正确的，除了可行性证明以外，本章更多地解释如何为分布式应用设计和配置鉴权服务。

这章如下安排：5.1节给出JAS设计的一个大体概况，解释它的组成成分的主要元素；通过在5.2 节和5.3节详细描述DC和PE的设计来阐述5.1节的观点；在5.4节讨论设计和开发JAS的结果，并给出结论。

5.1 JAS的概述

像前面涉及到的，JAS结构的主要目的是证明JCRADS方法的可行性，并且为科技基础条件平台门户系统开发一个实验框架，支持针对应用、细粒度、复杂和动态的访问控制策略，提供必要的不可抵赖性、容错性、有效性和实用性。除了使JAS设计确认JCRADS结构之外，本章力求实现它的可配置性、工具支持能力、便捷性、灵敏度，还有对于当前和将来设计具有充分的扩展性。本节给出JAS主要设计元素的概述。

5.1.1 中间件技术

为了使得JAS工具便捷和可扩展，本文尽量使用标准技术。本章选择J2EE中间件技术作为开发环境和工具，它的安全服务提供必要的功能，用于给不同的鉴权策略建模。J2EE命名服务允许JAS的分布式组件以平台无关的方式相互查找。对于每一个JAS组件，

可以自由地选择工具语言。如图5.1所示。下一个设计判定是关于JAS构件提供的接口。

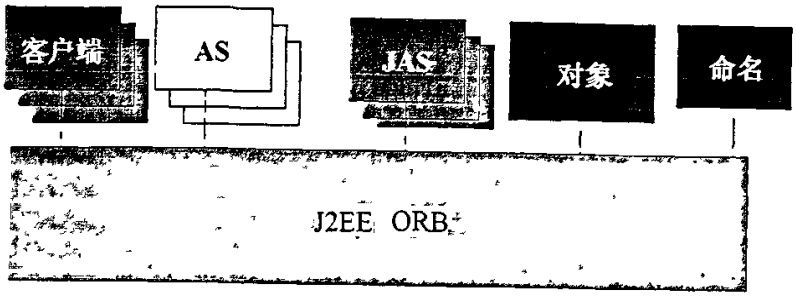


图 5.1 JAS 主要组成部分

表 5.1 JAS 和 JCRADS 扩展的 IDL 间的关系

JCRADS构成	JCRADS体系结构定义的IDL接口	JAS设计定义的扩展IDL接口
ADO	AccessDecision	AccessDecisionExt
	AccessDecisionAdmin	AccessDecisionAdminExt
PEL	PolicyEvaluatorLocator PolicyEvaluatorLocatorAdmin	
	PolicyEvaluatorLocatorBasicAdmin	PolicyEvaluatorLocatorAdminExt
	PolicyEvaluatorLocatorNameAdmin	
	PolicyEvaluatorLocatorPatternAdmin	
DAS	DynamicAttributeService	DynamicAttributeServiceExt
		DynamicAttributeServiceAdminExt
DC	DecisionCombinator	
PE	PolicyEvaluator	PolicyEvaluatorExt
	PolicyEvaluatorAdmin	PolicyEvaluatorAdminExt

5.1.2 组件接口

在JCRADS结构中定义的IDL接口[IBM 2004b][范 2004][SUN 2004]表现出了所有基于这种结构的服务所共有的功能。要求JAS设计通过接口提供额外的功能，这个功能应该允许运行-时间接口获得管理接口的参数并且能够停止组件工作[龚 2004]。因此，这里给表5.1所列的运行-时间接口和管理接口引入扩展，这些扩展允许使用额外的功能，而不改变JCRADS接口。由于J2EE IDL接口的继承能力，如果不使用额外定义的操作和属性，新定义的接口被JAS客户视为基本的JCRADS接口。

5.1.3 工具语言

有两个要求影响工具语言的选择－工具便捷性和编码的简单性。根据分析和比较，本文使用Java作为工具语言。Java虚拟机(JVM)对于大多数操作系统是可用的，并且这个语言给快速开发提供了一个优势，诸如面向对象和无用存储单元收集。Java还提供了类的动态载入，这使得在建立和运行-时间和下层的ORB中间件相容的Java类的载入中，配置和改变JAS特性时具有非常大的灵敏度。

然而，Java有几个约束。大部分JAS组件提供多重IDL接口－运行-时间和管理接口，运行-时间接口用在鉴权判定的计算中，管理接口定义能够配置JAS组件特性的操作。考虑到这种情况，本文决定每个JAS组件配置两个类型的IDL接口，仅使用一个Java类，如图5.2所示。例如，Java类“动态属性服务”配置两个IDL接口：“动态属性服务-Ext”和“动态属性服务管理-Ext”。在Java中，使用定义了与IDL接口符合的操作和属性的类来配置一个IDL接口[柴2003]。然而，因为Java不支持多重类继承，不能使用继承来配置运行-时间和管理IDL接口。

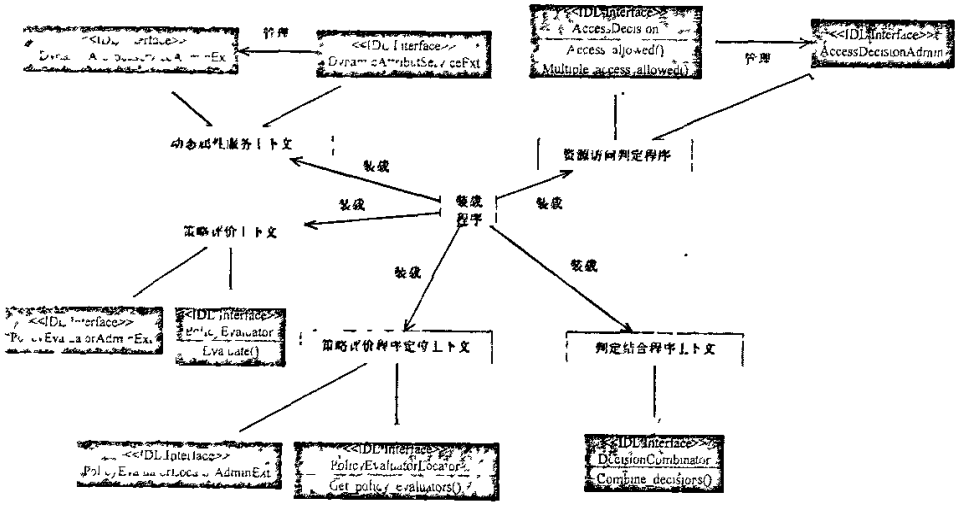


图 5.2 JAS 结构体系

为了绕过Java单一继承的限制，本文使用一种以Tie方法[Pedrick 1998]为基础的、著名的代理机制来配置组件。然而，Tie只能使用与ORB环境相互作用的最小的机制。在“鉴权”代理类配置的“组件操作”接口中实际配置组件的操作，如图5.3所示。用这个方法，因为代理类可以从任何类继承，唯一的要求是代理类使用“组件操作”接口，在创建客体时可以获得更大的灵敏度。

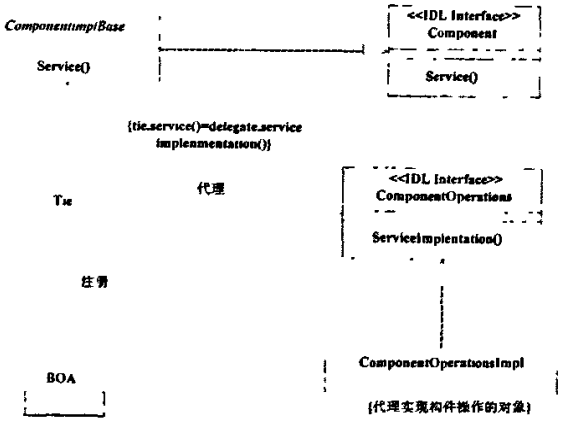


图 5.3 J2EE 对象的实现

Java ORBs的当前版本通过执行更多线程的请求来支持并发调用[IBM 2004b]。这种实现方式虽然对性能有利，但是增加了开发的难度和不确定性，它要求开发者在改变客体状态时要非常小心。为了说明这种情况，本文决定使用完全同步的方法作为JAS的工具。虽然这个性质不能保证系统没有大的错误，诸如死机和资源匮乏，但是它确实能在对象层保证性能的稳定。然而这样做也引入了不必要的同步，它会影响整个运行-时间性能，这是因为同步方法的需求比不同步的需求开销更大。对JAS组件的同步操作粒度很粗，且能造成线程堵塞。

5.1.4 设计扩展性

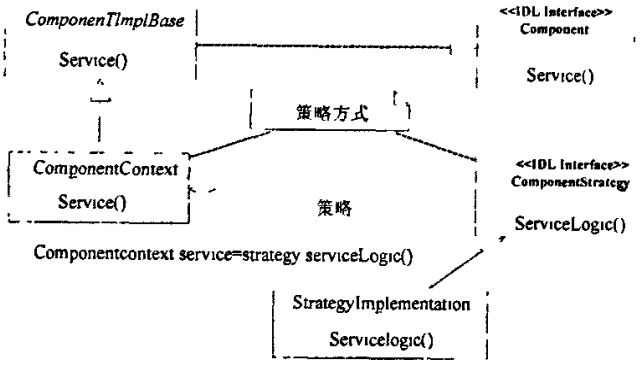


图 5.4 用策略方式实现服务器

在设计过程中，对同一JAS组件的不同请求是明显的，诸如DC和PE使用不同的逻辑。例如，DC以多种方式从多重PEs中合并出结果。一个解决方法是每个组成特性应用一个类。然而，这会创建许多相关类，它们只在功能上略微不同。本文选择基于策略（Strategy）[Gamma 1995]的设计模式。

在Strategy模式中，context类使用所有工具共有的逻辑，并且Strategy类提供具体工具所特有的性能，如图5.4所描述的。这个模式允许使用与每一个JAS组件有关的算法系列和基本功能(context类)。由于Java是选定的工具语言，这里把Strategy定义为Java接口，在这种情况下，组成元素所构成的环境和Java类执行由Strategy接口表示的服务，有了DC和PE组件的Strategy工具，再深入一步：它们的工具是基于称为模版方法（Template Method） [Gamma 1995]的设计模式。模式根本的思想是在一个基类中，定义一个算法的轮廓和框架，而详细的内容在下级类中定义(如图5.5所示)。

经过仔细分析，本系统中所用组件的配置大都趋向于共享一个公用的功能，鉴于此，在DC和PE的设计中，本文使用了模版模式。例如，DC的配置需要解决PE客体从ADO收到的参数，不管是否使用判定结合策略。类似地，PE的配置需要依据资源名来维持策略的结合，而不用管策略的储存方式和评估方法。这个公用功能可以在抽象的Strategy类中使用(图5.5)。之后，就可以通过改变这个类来获得特殊的配置(在图5.5的例子中的策略A、B、C)。

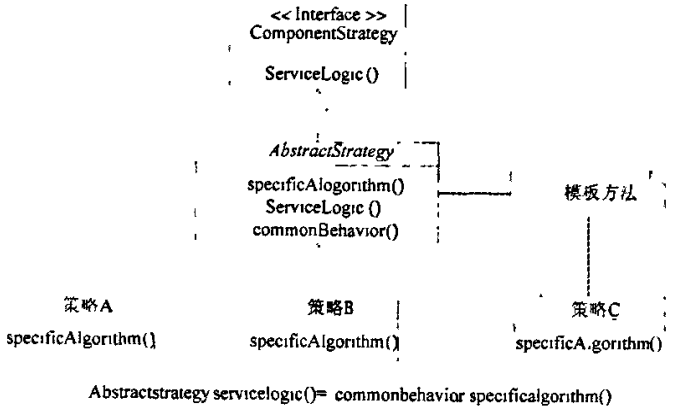


图 5.5 模板的应用

5.1.5 通用组成结构

如图5.6所示，以相同的方式配置所有的组成成分，这使得设计和编码更快，因为组件能够重复利用，并且为了理解每一个组件工作的规律，开发人员只需学习一种结构[龚2004]。这也更容易看到区别，例如，PEL没有对它的管理接口的扩展，然而由于它的简单性，DC缺乏一个管理接口。

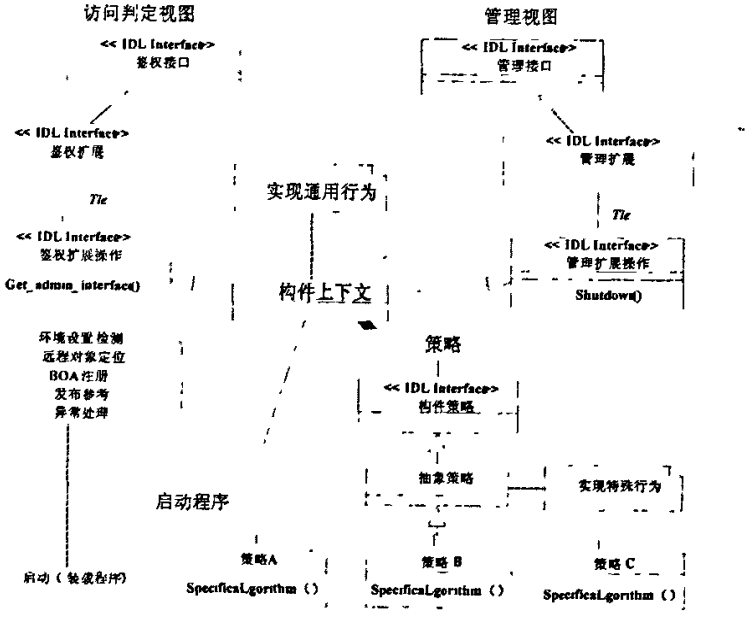


图 5.6 JAS 组件的通用结构

5.1.6 组件预置和发现

为了在不同的配置和负荷下研究JAS的性能，这样就要求JAS提供这些性能：

1、使用不同的策略评估和/或判定结合；

2、允许在相同的过程或者群体中寻找组件，允许在网络和系统不同位置中对不同组件进行展开；

3、容易改变配置并且快速重复相同的实验。

例如，如图5.7所示，它要完成在一个过程中合并应用和鉴权逻辑，或者在一个独立的过程中装入每个JAS组件(图5.8)。此外，还要在不重新编译源代码的情况下，能够在不同的配置中定位服务。

为了在不同的配置中简化引导JAS组件的过程，本章引入两种技术[南 2005b]。第一个如图5.2所示，是组件定位器的使用，它能在一个过程中定位同一组件的实例。定位器所需要的所有信息通过配置文件或命令行参数提供。然而，一旦定位组件，就有必要让它们彼此发现，即获得相应的对象参数，无论它们是否在一个过程中、在一个机器上还是在不同的网路结点上。第二种技术一字母化的组件参数表示。这个技术允许通过改变易读的符号的参数代表来选择传递和获得组件参数的方式，特别在配置文件和命令行，并且当客户和目标都在同一个过程时，可以避免过多地使用中间件。

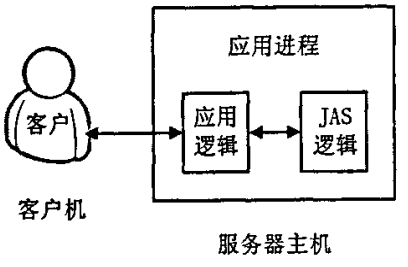


图 5.7　参数配置

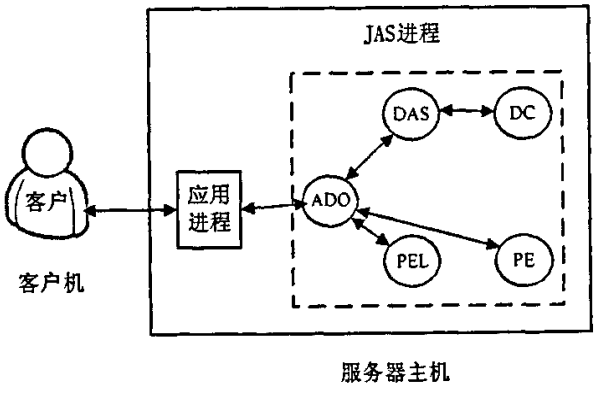


图 5.8　JAS 配置

为了在不同的过程中发现定位的组件，可以使用以字符串形式储存在一个文本文件中的、可由双方操作的对象参数。定位程序的主要好处是在需要定位命名服务时，能十分便捷地进行服务发现，它通过发送一个UDP广播和一个预定义端口实现。如果定位程序事件在网络中可用，它将为命名服务响应一串对象参数，通过在命名等级中的名字，

发现一个组件对象参数。

为了阐述上面讨论的JAS设计，在下两节中讨论DC和PE。其它组件的设计与DC和PE类似。

5.2 判定结合程序

DC概括了“判定结合”逻辑，该逻辑给一个对象鉴权时，判定鉴权接口生效(图5.9)。在当前的JAS中DC只有一个运行-时间接口[南 2005b]，用带有“判定结合程序上下文”的判定结合程序激活IDL接口。“判定结合程序上下文”同样使用Tie方法协调要引入的管理接口。

DC展示了所用JAS组件的简单设计。然而，DC的对象能够表现不同的行为。例如，DC能以多种方式从多重PEs中合并结果，即一种类型的DC能够使用逻辑与合并原则合并多重结果，而另一个类型的DC能够使用多数投票原则合并多重结果。但是，原则的这两种形式，不必改变DC合并PEs结果的方式：在这两种情况下DC可能不需要考虑所有的结果。考虑到这些情况，这里使用策略模式设计DC。授权给一个对象不同的判定合并策略来实施“判定结合程序策略”Java接口。接口工具之一是“抽象/结合程序”类。这个类被进一步提取成两个类，开放式或者封闭式策略。使用开放式策略，如果没有PE对象拒绝访问，DC授予访问；使用封闭式策略，它执行一个严密的合并原则－仅当PE对象也同意时DC才授予访问。

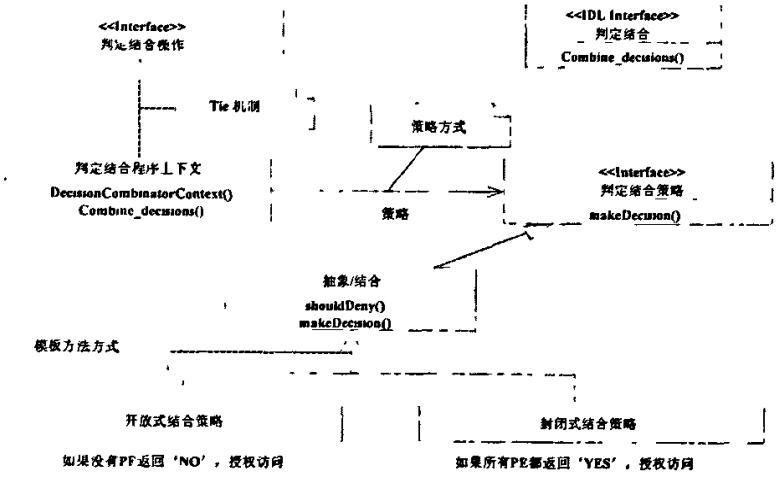


图 5.9 DC 的设计

5.3 策略评价程序

PE的功能是评估鉴权策略，这些策略是关于具有安全属性、资源名和操作名的资源。PE有运行-时间和管理IDL接口“策略评价程序”和 “策略评价程序管理”，这两个被

“策略评价程序扩展”和“策略评价程序管理扩展” IDL 接口扩展[南 2005b](见图5.10)。本文使用一个单独的Java类“策略评价程序上下文”来对它们进行配置。

像5.1节所描述的，在Java中，一个IDL接口有一个工具类配置[Sun 2004b][范 2004]，因此，不能用继承来配置多重IDL接口。由于这个限制，本文使用Tie方法来配置“策略评价程序扩展”和“策略评价程序管理扩展” IDL接口。在这种情况下，“策略评价程序上下文”把操作功能鉴权给配置“策略评价程序扩展”和“策略评价程序管理扩展”Jave接口的对象。(见图5.10)。

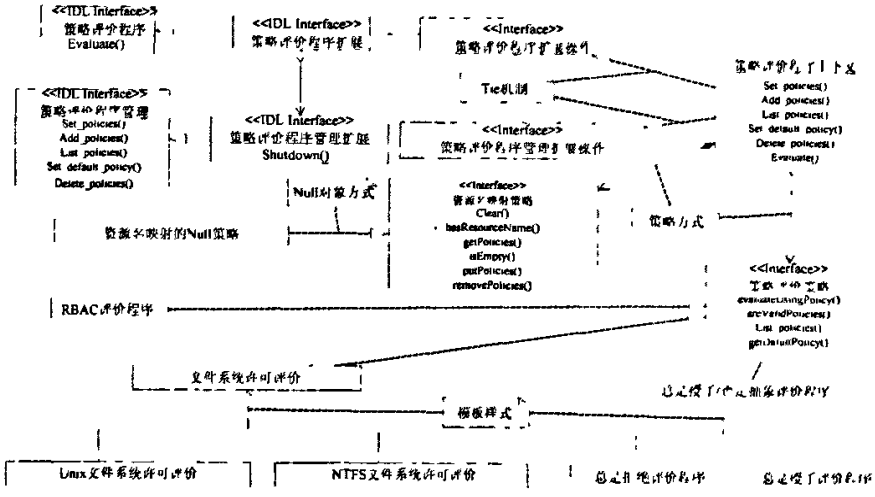


图 5.10 PE 的设计

PE的不同实例表示不同的行为。例如，JAS服务可以使用PE组件配置策略评价机制。然而，大多数PE的这些实例使用相同的机制，把资源名和访问控制策略结合在一起。

有些PE类仅在评价策略上不同，为了避免这些相关类的引入，本文使用一个基于策略“Strategy”形式的解决方法[南 2005b]。用这个方法，“策略评价程序扩展”执行PE的多数工具所共有的功能。例如，鉴权原则的加法和除法不会在PE实例之间改变。不同的评价策略给一个对象配置“策略评价程序”Strategy Java接口(见5.10)。类似地，与访问策略相关的资源名的管理可以在PE实例之间变化，因此，“策略评价程序上下文”把这个功能的工具鉴权给对象来配置资源名映射策略。通过使用这个实例，可以使用任何适合于当前需要又独立于“策略评价程序” Strategy的工具来配置这种结合。

“策略评价程序”Strategy接口的工具用样本方法模式进一步限定，如图5.10所示，这种模式比较容易地扩展和修改策略评价机制。

在JAS的设计中本文使用的另一个模式是零对象(Null Object)模式[南 2005b]。用这种模式，能够提供“空操作”的类，对于这种类在执行期间没有特殊的工具可用，在设计PE时，用它来定义“资源名映射的Null策略”类的接口(见图5.10)。

5.4 本章小结

在前一章中本文要求JCRADS结构具有下面的性质：简易性、灵活性和通用性。简易性使用简单的接口来实现，通过AS对JCRADS服务请求做出简单的操作，并且通过用简单的结构在请求和JCRADS服务之间交换信息。简易性也通过在JCRADS结构和JAS设计中使用包装规则来实现，对于个别的策略，做出鉴权判定的程序复杂性被包装在PEL、DAS和PE对象中。当构建JAS时，本文发现DC对于简化JAS的设计贡献很大，这是因为DC包含了判定结合策略，这些策略能完全改变JAS的整个鉴权逻辑。只有当使用复杂的访问策略时，复杂性增大，而这种复杂性仍然包含在私有的组件内部。在PE工具的内部增大复杂性不会增大PEL的复杂性，反之亦然。然而，在这些情况，更加复杂的策略评价机制的引入可能增大判定结合程序的复杂性。

灵活性是存在于JAS中的另一个特性。在JAS中的改变可以表现为在访问控制策略、策略评价和动态属性中的改变。例如，像在前一章例子中展示的，新的访问控制原则能通过改变或者替换现存的PE和DC对象而改变。为了演示JCRADS的扩展性和灵活性，设计JAS来支持关机、重新初始化或者替换其组件。例如，使用不同版本的DC，并且版本能够自由替换。

JCRADS结构的灵活性的更多表现是支持JAS组件的不同结构。为了使用JAS的不同结构指导性能实验，设计下面的这几个结构来支持组件的展开，而不用改变源代码，(见图5.11)。这种配置允许JAS获得最大性能(避免使用ORB中间件和下面的网络)、可用性或者灵活性。

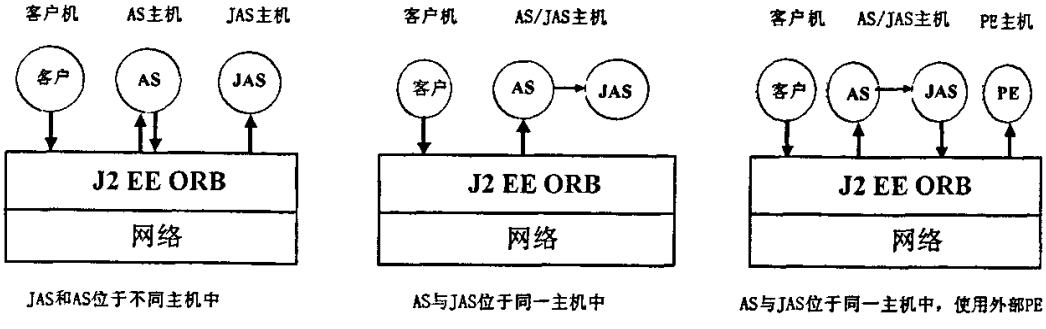


图 5.11 JAS 体系结构

通过上面的设计和配置JAS之后，可以看出JCRADS结构能保持足够的一致性，对于不同的环境，能够用不同的请求和设计策略对其进行配置。这可以用几行代码实现而不用设计模式。另一方面，一个JCRADS服务能使用一个复杂的设计配置来实现容错、高性能和稳定性。本文设计JAS工具的期望是最终得到一个简单的设计，这个设计灵活、可扩展、可配置。

本文使用标准的技术开发了一个具体的JCRADS结构工具－JAS。这个工具是灵活的、可配置的、可扩展的、轻便的。

这章的工作的主要贡献是JCRADS结构原型(JAS)的一个具体设计，应用不同粒度和复杂性的不同鉴权机制实验，从JAS可以得出JCRADS结构是可行的，并且在IDL中对它定义的计算模型是正确的。

第六章 JAS的性能测量方法

对于JCRADS的鉴权服务来说，主要关注的方面就是全面的系统性能。不管方法有多么好，如果执行能力受到性能的约束，这种方法对开发者来说是没有什么太大意义的。这一章介绍关于JAS性能的研究。

基于JCRADS系统的鉴权服务性能方面的最主要问题不在于是否要支付性能开销，而是支付多少的问题。人们往往最希望用中间件和信息管理开销来加快应用程序的响应时间。然而，这些信息管理开销是有数量限制的，因为JCRADS系统体系可以在网络环境中定义同一个进程中的多个组成部分，也可以定义相同主机或不同主机中的多个组成部分，而这些不同的环境会不同程度上影响整个的运行-时间和性能。

这一章的结构如下所述。6.1讨论了性能的测量模型；6.2描述了本文用于试验的JAS配置；6.3和6.4分别说明了测试的环境和试验的程序；6.5给出了试验结果并对结果进行了解释；基于试验数据，6.6给出了充分实现JCRADS服务性能的方法。6.7得出了本文的结论。

6.1 测量模型

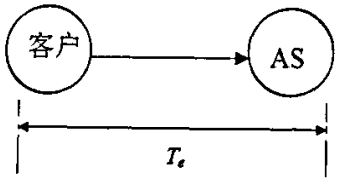
在试验中定义一个模型是至关重要的。测量模型决定试验的范围、会得出什么样的结论以及应该如何来解释。这个测量模型也决定了试验的复杂度和可行性。既然本文研究的主要着眼点不在于性能的研究，所以我们就选择了一种最底线的方法，也就是使用一个最简单、大部分都可以得到的试验框架来获得必要的性能测量法。

当定义结构框架的时候，首先要说明的是本文要使用的是绝对的还是相对的性能测量方法。绝对性能测量方法只有在统一基准下才能恰当地解释出来。这个基准包括执行平台、语言、中间件以及其它因素。因为不知道适合论文目标的同一基准，因此，本文决定使用相同的设计、通讯技术以及运行平台的参考模型相关的测量方法。

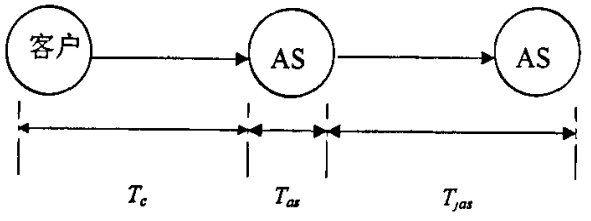
本文选择的参考模型是应用系统（AS）。在应用软件的计算复杂度与实验结构的复杂度相同的进程中，这个AS就把应用软件和鉴权功能联系起来，从而通过这个参考模型与试验系统相比较，就可以估计它们在性能上的差异。JAS中所有的鉴权机构都是在试验系统中模拟的。

系统性能有很多不同意义，也存在着多重的方面。系统性能的测量是依据性能是怎么被定义的。如果把性能定义成由每一个时间单位提供的鉴权请求数量，或者每一个请求都需要等待的时间，那么我们就可以利用时间 T_{JAS}（图 6.1b）作为 JAS 性能的估计值。此时，JAS 的进程是通过鉴权的要求来实现的。然而，因为参考模型与它没有关系，也没有什么价值，就没有必要引出参考模型了。当一个应用软件提出要求，而 AS 要完成

进程时，就用不着定义使用时间 T_{as}，因为这个进程已经依次包含时间 T_{JAS}。但是得根据两端都有响应时间来估计响应时间 T_c。在观察性能的角度来说，当通过 JAS 来计算出鉴权结果时，这一点是非常重要的。这就是 AS 性能为什么就是端到端的响应时间的原因。这一点客户在与 AS 进行连接时也能够发现。



(a)参考模型



(b)JAS配置的响应时间

图 6.1 测量性能时间

通过性能和时间的定义，就可以（如图 6.1a 所示）决定参考模型。利用标准时间 T_c 和 T_e，在每一个 JAS 结构外部鉴权的情况下，考虑响应时间增长的百分比 I，控制方法可以由下面公式来计算。

$$I = \left(\frac{T_c}{T_e} - 1\right) \times 100 \tag{1}$$

本章前面说过，JAS的最初设计不是当前存在方法中最优设计，那就可以用性能的可测量性拓宽它的范围，所以，即使系统只存在一个用户时，也要进行运行-时间的测量。如图6.1b所示，用户连续地向单个应用软件系统发送请求。用户只有收到发出请求的回应之后，才能发送新的请求。

如果把复杂度相同的鉴权逻辑细微的调动一下，使得每一个鉴权请求都会影响整个系统。这样一来，鉴权请求的数量就与JAS的组成部分和位置相关。也正因为这样，本文使用了不同的JAS结构来观察JAS的组成部分是怎样影响的响应时间。

6.2 JAS 的结构

因为由JAS的组成部分可以构成很多不同种结构，这就需要从中选择试验所需要的。JAS的结构决定计算鉴权请求时由发送的信息所交叉的界线。其界线有三种：对象，进程和主机。信息进入进程界线时，它不可避免地进入ORB层。每当信息进入主机界线时，

也进到了网络。所以把“进入主机界线”视为进入到了网络。

在适当选取JAS的结构时，另一个重要问题可以由图6.2看出。信息在JAS成分之间传输的方法共有三种：

1）在一个进程中从一个对象传到另一个对象；

2）从一个进程的对象传到另一个进程的对象；

3）在不同进程中的对象之间依次的传递，这些进程也是在不同主机之间发生的。

界线分为对象层、进程层和主机层。当一个界线被交叉的时候，所有那些划分到层次以下的部分也被交叉了。

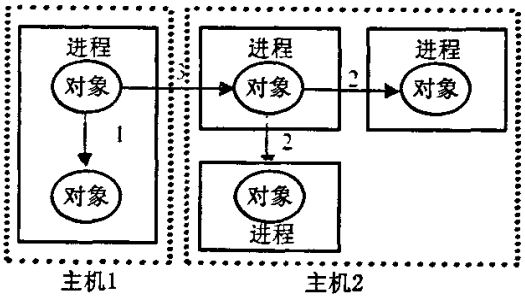
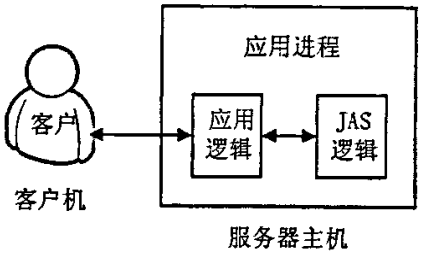


图 6.2 信息穿过的界线

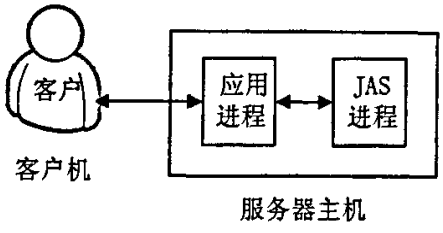
当构件都在同一个地址空间定位的时候，信息就会交叉到目标界线，这可以通过JVM的直接调用来通信。当相互通信的构件位于相同主机内，并且在它们自己的进程中进行信息交换的时候，信息就会交叉到进程界线。即使这样，通信是通过中间件ORB来实现的。所以这些界线也称为中间件界限。然而，这种形式的通信方式，也可以用象IPC[Nutt 1997, Stevens 1993]的其它机构来实现。最后，当构件在不同主机时，信息就会交叉到主机界线。这个就包含了上面提到的中间件和通信子系统。

JAS可以展开成不同的结构，结构的范围需要测量。当所有JAS的构件分配在同一个进程，而且信息仅是在对象界线中穿越时，这个边界处就是结构的配置。这是一个很有效、但不能得到一个定量的回答的结构配置。为了突出这一点，通信的JAS结构（图6.3中的A、B、D所示）都是以“对象”结束。在另一个边界处就是一个构件。在这个构件中，所有JAS才能在不同主机中进行。从这可以看出构件非常高的适应性。

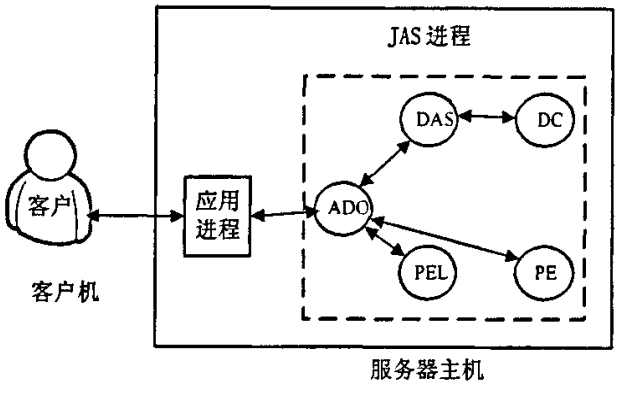
这里测试的是当所有的构件分别在各自不同进程的情况，如C、E和G所示。剩下的策略当作一个PE（F和G结构）使用，最好能够利用分布在单独的主机里的PE。



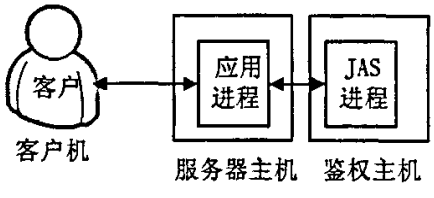
A）参考模型



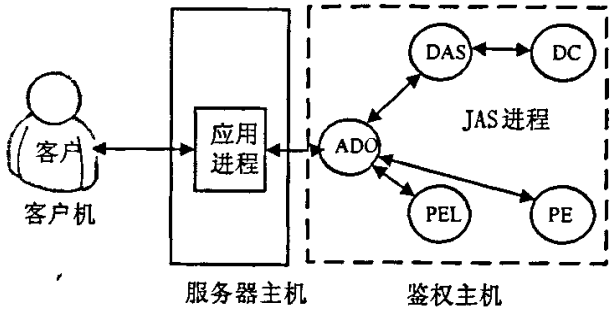
B）进程/对象



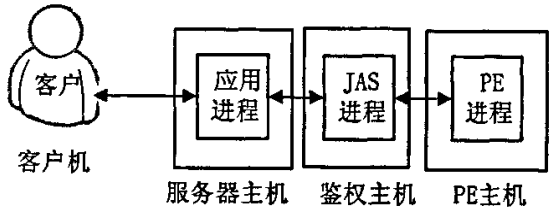
C)进程/进程



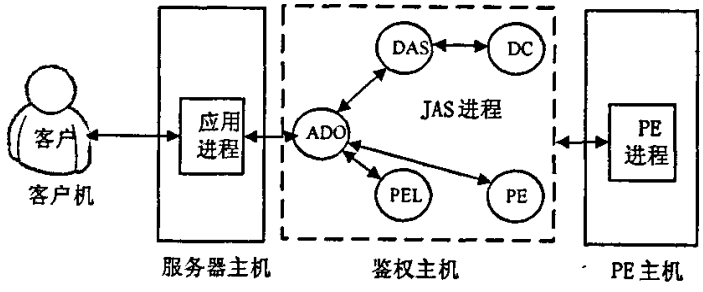
D）主机/对象



E）主机/进程



F）主机/对象/PE主机



G）主机/进程/PE主机

图 6.3 相关模型和 JAS 配置

应用软件的性能受两个方面的影响。一个是受信息在JAS构件之间跨越边界类型的影响，另一个就是受上一层通信和AS与JAS之间信息的影响。这就需要测量结构的性能时既考虑与应用软件相同主机的（B和C），也考虑不同主机的（D、E、F、G）。为了强调这一区别，与其相应的名字以“进程” 开头，或者以“主机”开头。

为了得出性能的相对测量，需要一个有鉴权逻辑的参考结构，而这个结构与其它结构的计算复杂度相当，且加上应用逻辑。为此，以应用进程内的所有JAS交叉定位模型作为模拟参考模型，如图6.3A所示。它允许每一个计算机业务去辨别到底是服务于应用软件还是服务于上一级的鉴权服务。一旦辨别出来，就可以把鉴权结果封装进单个的鉴权模块中。

分析了JAS配置的基本原理后，接下来解释每个部分。通过进程/对象的结构，AS和JAS作为独立的进程，并存于同一个服务器主机内，JAS的构件也可以并存于同一个进程中，如图6.3B所示。AS与JAS之间的信息是通过ORB中间件（进程界线）来传输的，

然而，JAS构件内部的信息传输则是调用本身自带的、叫做JVM的方法。图6.3C说明了进程/进程的结构。JAS构件在它自己的进程中展开。如图6.3D中的主机/对象结构中，JAS构件都并存在同一个进程当中，然而，AS和JAS却在不同的主机中。那就是说，AS和JAS之间的信息是通过ORB中间件和通信子系统来传递，而JAS成分只是在对象界线中传递。

在主机/进程（图6.3E）中，AS和JAS存在不同的主机中，而 JAS的成分存在于同一主机中各自的进程中。图6.3F以图解的形式说明了主机/对象/PE主机的结构。除了PE的成分可以访问不同主机之外，这种结构与主机/对象结构是相似的。JAS构件之间的通信发生在对象与主机界线内。最后，在主机/进程/PE主机的结构，如图6.3G。PE位于鉴权主机之外的某一个主机之内，而可以访问不同进程的JAS成分都存于鉴权主机内。注意到两个成分通过进程界线来交换信息的时候，信息的经由途径包括主机中共存的中间件和上下文交换。另一方面，主机界线中没有这样的上下文交换，因为交互的双方不会为了执行时间而彼此竞争。

由于这种可配置性，开发者与管理员们可以以最高性能（避免使用ORB中间件和网络）和最高灵活性（所有的成分都存在于网络中的各个系统中）的方法实施JAS。例如，管理者可以在实施JAS的时候，通过使用主机/对象结构来避免使用中间件和网络。然而，在几个PE零散地分布（也许在不同子网中）的组织中，JAS可以以主机/对象/PE或者主机/进程/PE对象结构实施。主机/对象或主机/进程结构可以用来实施JAS成分。本文希望的是所有的成分可以在同一个进程中或同一个主机中共存，而一个或多个PE就可以分布式的实施。

在定义了测试模型、参考模型以及试验模型之后，下面讨论试验时需要的环境。

6.3 试验环境

我们的测试环境是由4个Windows 2000服务器组成。每一个服务器有512MB的内存储器，装备一个100MB网卡。这些服务器连接在一个100MB的三层交换机下。此外，在测试期间，使用JDK 1.4.2、J2EE ORB和所有的Java类，Jar文档存在本地硬盘里面。只有在网络的利用率小于1%时，完成性能的测试，为的是减少不相干网络负荷的影响。

6.4 试验程序

这个试验需要一个客户机，一个 AS 和一个 JAS。这个 JAS 中包括一个 ADO、PEL、DAS、DC 和 PE。性能测量的目的就是估计最差的情况，就是当 JAS 响应鉴权请求时，客户机产生的性能缺陷。测量客户机产生的响应时间 T_c，这是客户机通过 JAS 来实现外部访问控制时产生的，然后再测出响应时间 T_e。有了这两个数字，就可以通过方程 1 来计算响应时间增长的百分比。当 PE 对象总是允许访问时，DC 对象就会实现逻辑与的结合策略。

重复执行 6.2 节中介绍的六个结构，性能测试所需要的其它参数有应用程序的处理时间 B 和鉴权请求的数量 N。N 是由每个客户机的请求决定。应用程序的处理时间表示 AS 在服务于客户请求时产生的延迟和由 ADO 返回来的鉴权结果的执行时间。这个时间不包括 JAS 的处理时间。虽然试验了使用一个客户机，但一个客户机的请求就能够由 AS 引起许多鉴权请求，这可以由可变的多个数量的鉴权请求来模拟，每一个客户机可以提供若干个要求。

对于性能的试验，既然目的是测量出相对于参考模型的最差的情况，那就选用开销最低的结构和评价策略。这是因为越复杂的鉴权方针，在不增加中间件和通信开销的情况下，会增加计算量。假设没有新的信息进入组件，计算开销的增加对于参考模型来说发生在鉴权逻辑。在方程 2 中，△是与鉴权服务的额外计算复杂性有关的增量。这就意味着 $I'<I$,因为 $T_c>T_e$, $\Delta>0$。

$$I' = \left(\frac{T_c + \Delta}{T_e + \Delta} - 1\right) \times 100 \qquad (2)$$

做这个性能试验的时候，没有使用缓存技术。因为缓存技术会减少通信开销，从而也降低时间I的增加。远程调用没有使用安全通信方式，是因为由于通信安全的存在使得有些开销是无法控制的。此外，信息的保护是由应用软件与执行部分决定的。不同的应用软件需要不同层次的保护，不同层次的保护有不同的性能。所以，也对信息进行保护，只是严格估计最差情况下响应时间的增加。

6.5 测量结果

根据图6.3中给出的JAS配置的测量方法，图6.4阐述了结果。计算出响应时间的增长量，用它作为每个鉴权请求的时间函数。例如，在主机/对象配置的情况下（图6.3 D），响应时间比参考模型增加了31%，在作出一个鉴权请求之前，它需要10ms时间处理这个申请。

观察这两组JAS结构，性能最好的那一组是由这些结构组成的：这个组中全部或者大多数JAS组件是协同定位的。即使在一个主机上配置PE，并且它与所有其它JAS组件分离，JAS的性能仍然比第二组中的任何结构都好。这个组中包括了JAS的配置，当所有的组件通过传输信息而互相作用时，在同一台主机上是否使用ORB在进程间通信是完全没有意义的。

鉴权的应用处理时间（ms） / 各结构响应时间增长%	1	10	100	1000	10000
B 过程/对象	33	11	2	1	0
D 主机/对象	76	31	4	1	0
F 主机/对象/PE主机	140	52	8	2	0
E 主机/过程	534	187	26	3	0
C 过程/过程	528	201	28	3	1
G 主机/过程/PE主机	635	213	30	4	1

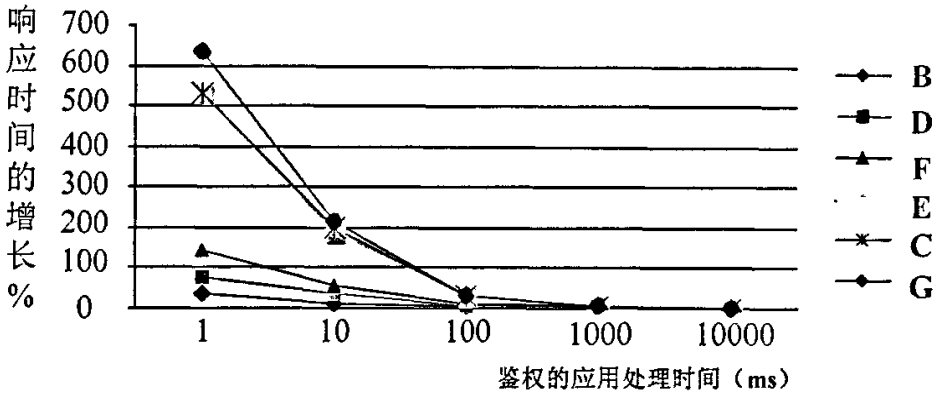


图 6.4 各种 JAS 配置的响应时间增长%

结果表明，每一次处理鉴权请求的应用逻辑所需的总时间对应用客户的相对性能有很大的影响。在两个组中，时间的增长会导致性能在2-10倍范围增长。结果还表明，没有使用鉴权服务的应用，对于JAS配置是不敏感的。这些条件对使用JAS是有利的。但是如果更多的应用系统使用JAS鉴权服务，所有的组件是协作定位的，应用系统拥有者能负担性能上10%的损失，JCRADS方法的性能将会得到全面的提高。

6.6 性能

性能试验表明，为了开发和成功的利用基于JCRADS的鉴权服务，建议在整个系统中使用中间技术。表6.1中给出了比较结果。

首先，必须研究在应用程序中使用鉴权服务模版。正如表6.1中显示的那样，如果使用不频繁（每10秒钟不多于一个鉴权申请），这种应用就能以任何可能的方式使用基于JCRADS服务的配置。其次，加强了应用中的性能约束，它需要了解应用和服务的协作定位、服务构成等。另一方面，应用系统集中的使用了每秒处理100或者更多鉴权请求的服务，具有严格的性能约束，它只允许不多于5%的性能损失。

表 6.1 应用请求与 JAS 推荐配置的关系

响应迁时增长的限制	鉴权服务用法（每个请求的应用逻辑时间）		
	强 10ms或者更少	中等 100ms至1s	弱 10s或者更多
严格（5%或者更少）	位于一个过程中的授权和应用功能	一个过程中的JAS构件	任何配置或者位置
中等（10%和30%之间	1.一个过程的JAS构件 2.同一个主机上的JAS和应用	任何配置或者位置	任何配置或者位置
不严格（30%以上）	任何配置或者位置	任何配置或者位置	任何配置或者位置

一些分布式系统能使用一些技术来增加性能，这样对配置和位置的要求就会相对宽松。例如，通过使用ORBs方法对位于同一主机上的对象实行通信层最优化，通信开销将会降到最少。另一种方法是隐藏从以前的服务组件中获得的结果。这个技术对于重复的鉴权请求是很有帮助的。

基于JCRADS的鉴权服务的部署和实现应该考虑组件之间的交互。这就是说，应该应用最优化技术来描述高交互率的组件。例如，需要多个PE的策略评价，可以在同一个过程中用适当的DC从协作定位的多个PE中进行优化。

并发请求的性能指标是本文要考虑的另外一个方面。虽然并发请求的处理并不包含在性能测试中，但是它是保证更深入研究的一个重要的因素，并发性是基于组件系统的重要问题。当出现并发访问的时候，需要避免读/写和写/读的矛盾，安全保存系统中所有对象维持有效状态的安全保障[Lea 1996]。为了定位这个问题，JAS实现完全的同步。尽管这个方法不能保证系统不出现故障（例如死锁和资源缺乏等），它确实能在Java语言层次上保证一致性。当前的设置允许同步对象实例[Lea 1996]，但这也会引入不必要的同步，因为同步方法比常规方法的开销要高，所以它会影响整个系统的运行性能。如果JAS组件间的运行是同步的，就要仔细地考虑同步粒度，否则将会导致线程阻塞。

6.7 本章小结

关于JCRADS方法可行性的主要问题，在功能足够之后，就是基于JCRADS鉴权服务是否能够达到必需的性能。用JAS来研究JCRADS体系的性能，发现对这个问题没有一个简单的回答。试验表明了两个主要影响性能的因素是应用执行时间与鉴权请求数量和JAS配置的比率。由于这些因素的变化，应用对象的响应时间增长能高达600%，也能降到1%。对于每一种应用，每当使用基于JCRADS的鉴权服务时，要决定为了保证足够的性能需要什么。

使用优化分布式计算机系统技术，这一服务性能可以进一步得到改进。例如，ORBs的利用对于提高响应时间是有很大潜能的。ORBs的利用是指优化在相同主机内的对象

之间的通信。当没有这种条件的情况下，所有JAS组件共存在同一地址空间的时候也可以提高其性能。它们共存在一个地址空间的原因是为了避免中间件带来的开销增加。同样，从不同的JAS组件得到的结果进行缓存也可以提高性能。最优化技术主要集中在高交互率的组件，像DC和PE。进程 / 对象和主机 / 对象结构（图6.3B, 图6.3D）的性能结果通常用于估计鉴权服务响应时间的增加。

第七章 非安全中间件的安全封装

随着科技基础条件平台项目的深入开发，采用的中间件种类越来越多，它们以复杂的方式相互作用[南 2004a]。这些中间件中，有一些是从网络上下载的，不是十分的可信。所以在使用前要了解这些中间件拥有的安全特性，比如，机密性数据不会在网络上泄漏，但是，很难证实这些中间件有很好的安全特性。本章设计的封装器，可以让这些中间件在安全的环境下执行，它可以为中间件之间、中间件和其它系统资源之间提供细粒度的控制[南 2004b]。

本部分重点研究表示封装器的方法，通过这种方法来阐述及严格证明安全性。文章分析并设计了一种编程语言模型，即 box-π calculus，这种语言模型支持中间件描述，并加强了安全策略。本文使用 box-π calculus 描述了几种封装器，探讨了各种封装器能够保证的安全性。

7.1 问题描述

软件系统不断发展。越来越多的单一的应用正在被来自于不同来源的软件中间件所代替。现在，大范围、分布式的应用系统都是由一些小的中间件进行架构，它们以复杂的方式相互作用，执行各种各样的信息处理任务。此外，虽然中间件库没有什么变化，并且系统管理员经常控制着中间件库，但是，在网络上还是很容易就能下载得到源代码；有些技术（比如：Java）甚至允许在程序运行的过程中，使用新中间件对应用系统进行扩展。

在这样可变的操作环境下，传统的安全机制和策略显得非常落后，虽然口令和访问控制机制适合保护系统的完整性，但是不能解决用户免受从他们的帐户上下载正在运行的代码的问题[IAJR97, GWTB96, NL98]。有些方法（如 Java 沙盒）通过隔离来承诺安全性，但是，这些方法也无法令人满意，因为中间件会自由地相互作用，或者根本彼此就不相互作用[VB99, Gon97]。需要一种非常好的保护机制，能够考虑软件中间件之间的相互连接。

虽然不容易分析和修改大型的、第三方软件包，但是可以阻止软件包和系统中其它部分的通信，可以提取出不同软件中间件边界的代码[Jon99,GRPA97, FHL+96, GWTB96]。因此，可以监控这些中间件调用的操作以及中间件之间交换的数据。本文把能够封装不可信中间件的代码框架称为安全封装器，或者简称为封装器。

显然，对封装器进行写入不应该只是留给终端用户去做。本文期望封装器是可以重新使用的中间件，用户只需要选择最合适的封装器、定制其参数和安装就可以了。所有

这些过程都是动态的：封装器和新的应用程序相比较而言，应该更容易加入到运行系统中去。用户需要封装器能够保证的安全性的一个清晰的描述。

本文主要研究封装器这样的安全环境，重点讨论怎样表示封装器，并能够严格定义和证明其安全性。显然，如果没有这样严格的定义和证明，设计者很难深入研究。虽然封装器很重要，但是，它可能就是一个小软件，因此很容易证明其属性。

为了表示和解释封装器，需要一个小的编程语言，它有定义好的语义，允许对软件中间件的构成进行直接表示，也支持对安全策略的加强。本文根据 π *calculus* 的初始定义重新设计了 box-π *calculus*。先看一个简单的例子：用 calculus 写的封装器 W_1。它封装了一个中间件，并且控制了它与环境的交互，限制了这些交互作用仅仅通过通道 *in* 和 *out* 来实现。W_1 定义为一个一元的关系：

$$W_1[\_]=(\nu\, a)(a[\_]$$

$$|\, !\, in^{\uparrow}\, y.\overline{in}^{a}\, y$$

$$|\, !\, out^{a} y.\overline{out}^{\uparrow}\, y$$

这就生成了一个名字为 *a* 的盒子，它有两个传送器，一个是通过通道 *in* 从环境接收消息，并且把消息送到封装程序；另一个是通过通道 *out* 从封装程序中接收消息，并且把消息送到环境中去。任何程序 *P*(也可能是恶意的程序)都可以封装成 $W_1[P]$；*calculus* 和 W_1 的设计保证了无论 *P* 的特性如何，封装程序 $W_1[P]$ 都只能在 *in* 和 *out* 两个通道上与环境进行交互。这可以通过阻止 *P* 和外界的所有交互来实现，W_1 是一个不能令人满意的封装器——它太“忠诚”，它总是诚实地在 *in* 和 *out* 两个通道上传送消息。此外，如果 *P* 的性能很好，那么封装对它的行为没有任何作用。在下面的命题中，对这个不正规的性质会有详细的阐述。W_1 是反常的，因为除了传送合法消息，它没有其它行为——其它的合理的一元封装器都可以执行某种日志功能，或者有一个控制接口。任何合理的封装器所具有的诚实特性因此显得有些脆弱；为了说明这一点（还有其它的安全特性），本文对 *calculus* 广泛使用了标记转换语义。

W_1 控制单个中间件和环境的交互。第二个例子要深入解决的问题，是允许控制中间件之间的交互。W_2（在下面定义和详细介绍）是一个二元封装器，它封装了两个中间件 *P* 和 *Q*，即 W_2（*P*，*Q*），两个中间件都可以以一种受限制的方式与环境交互，也允许信息沿着有向信道从 *P* 流向 *Q*，反之不可以。

W_1 和 W_2 都选择的尽可能简单，它们的中间件都有彼此交互和与外界交互的接口。对任意接口和有任何数目的封装器的直接推广会使符号更为复杂。

7.2 Box-π 定义与设计

用于研究封装特性的 *box-*π 演算语言必须由可交互的中间件构成。这些中间件并发执行，引入了非确定性机制。并且这个语言是基于演算过程的。*box-*π 演算基于巨大的分布式演算设计空间，该演算依靠的是 Milner，Parrow 和 Walker 的 π 演算。相关的演算曾被很多人使用，例如[AFG98，Ama97，AP94，CG98，CG99，FGL+96，HR98C，HR98b，RH98，Sew97，Sew98，SWP98a，SWP98b，VC98，VC99]等。关于设计空间的简要观点在[Sew99]中能找到，这里突出选择 *box-*π 的主要设计。

box-π 演算是基于异步信息传输，其内部中间件通过无序的异步信息进行交互。*box-*π 把 π 演算作为子演算，这样的演算包括著名的[HT91，Bou92，ACS96]。它们都非常有表现力，支持许多编程习语（包括函数和对象），并且完全遵循图灵机机制，因此 *box-*π 过程可以执行专用的内部计算。

在标准的 π 演算中，对于 π 必须加入一个原语来限制传输。如果一个进程发送信息到另一个进程，那么防止信息反向传输的唯一的方法就是加入一个并不适合这里的类型系统来观测上述问题。因此要加入一个盒式的原语，提供分层保护域的盒可以被嵌套，信息传输通过盒边界时严格受限。演算设计里有一个原则就是每个盒都能够控制该盒内部子盒的任何交互，包括子盒和外部的以及和子盒之间的交互。因此传输是在盒和它的父盒之间或者在同一个父盒的子盒之间进行，特别地，没有父盒协助时，两个同胞盒不能交互。为了使一个盒和它的一个子盒交互，盒需要命名，这和 π 演算中的通道命名类似。封装器的安全属性依赖于创建新盒名字的能力。

传输通信的值采用专用的数组名，这是非常方便的。注意进程处理期间不允许数据传输。而且，不提供盒移动的原语。因此，*box-*π 演算完全是最初的顺序，这对易处理的行为理论是非常重要，用这个理论来描述和证明安全属性。

7.2.1 语法

box-π 演算的语法如下：

名称 定义一个名称为无穷集 N，用小写的罗马字母 n,m,x,y,z（除了 i,j,k,o,p,u,v 以外）等表示。命名所有的盒和传输通道，并且名字作为 π 演算中的变量而出现。

值和模式 进程之间是通过传输值来进行交互，并且进程通过模式匹配解析它们收到的值。值 u,v 可以是名字也可以是数组，由模式 p 来决定对应数组结构：

u,v	::=	x	名字
		$< v_1 \cdots v_k >$	数组 ($k \geq 0$)
P	::=	–	通配符
		x	名字匹配
		$(p_1 \cdots p_k)$	数组匹配（$k \geq 0$, 没有重名)

进程 主要的句法类别就是进程，包括 P，Q。下面分三方面来介绍原语。

盒 盒 n[P]名字是 n，它可以包含一个任意进程 P。盒的名字不需要是唯一的，进程 n[0]| n[0]有两个完全不同的盒，名字都为 n，都包含一个空的进程，它们是并行的。

P ::= $n[P]$ 名字为n的盒包含进程P
P| P′ P和P′是并行的
0 空进程

传输 标准的π演算异步传输原语是$\bar{x}v$ 和$xp.P$， 表示在名为x的通道上有一个输出的值v；$xp.P$是一个在名叫 x 的信道里接收一个输出值，在 P 中把它封装成 p 的进程。在此，用标识来表示盒层次的传输的方向。一个输入标识是ι，如果是盒中的输入可以用*作为输入标识，如果是父盒中的输入可以用↑作为输入标识，如果是名为 n 的子盒输入可以用 n 作为标识。输出标识o可以是上面情况的任意一种输出。由于技术方面的原因，用$\bar{\uparrow}$或$\bar{n}$标识符来表示输出标识。$\bar{\uparrow}$表示的是来自父盒的、尚且还没有连接到输入的输出；$\bar{n}$表示的是来自子盒 n 的、尚且还没有连接到输入的输出。传输原语如下：

P ::= ...
$\bar{x}^{o}v$ 从通道x到o的输出v
$x^{\iota}p.P$ 从ι 输入到通道x
$!x^{\iota}p.P$ 复制输入
...

这个复制输入（$!\ x^{\iota}p.P$）的本质就是无限次的并行执行$x^{\iota}p.P$。这就提高了计算能力，例如，可以很容易的编写递归程序，而且其理论也非常地好理解。在$x^{\iota}p.P$和$!x^{\iota}p.P$中以模式 p 出现的名字都在进程 P 进行并合。

创建新名 通道名和盒名都可以用一个标准的π演算$(\nu x)P$算子重新命名。这表明了在进程 P 中任何情况下，x 都赋予新的名字。

P ::= ...
$(\nu x)P$ 创建新名

在$(\nu x)P$ 中，把 x 组合到进程 P。通过写一个自由名函数 fn（-），逐步把遍及组合的名字达到α转换、定义值、标识和进程。

7.2.2 化简

算法的最简单语义定义是化简语义，一个 P → P′的化简步骤，表示 P 能通过一个内

部计算化简成为P'。用公式$\overline{*}=*$或者$\overline{\overline{\iota}}=\iota$，首先定义一个标识$\iota$的补$\overline{\iota}$。再定义一个局部函数$\{-/_\}$,用来把一个模式、值、被定义的量或一个局部函数的名字转换成具体的值。

$$\{^{v}/_{\_}\}=\{\}$$

$$\{^{v}/_{x}\}=\{x \mapsto v\}$$

$$\{^{<v_1..v_k>}/_{(p_1..p_k)}\}=\{^{v_1}/_{p_1}\}\cup\cdots\cup\{^{v_k}/_{p_k}\}$$ 如果这些被定义并且k = k',否则不执行

置换σ（从名字到具体值）到一个进程术语 P 的运用是一个普通的定义，写作σP。它也是一个局部操作，因为语法规定任意的值不能出现在自由名出现的地方。用$\{^{v}/_{p}\}P$来表示应用置换$\{^{v}/_{p}\}$的结果。这个可能是没被定义得的，或因为$\{^{v}/_{p}\}$是没被定义的，或是因为$\{^{v}/_{p}\}$是一个置换，但置换到 P 中的应用是未被定义的。我们注意到名——名置换$\{^{y}/_{x}\}P$结果应用通常是已被定义的。定义结构上的全等“≡”，比如规则那样最小的全等关系。这样就部分的允许了可以在语句结构上临近。

$$P|0\equiv P \qquad (vx)(vy)P\equiv(vy)(vx)P$$

$$P|Q\equiv Q|P \qquad (vx)(P|Q)\equiv P|(vx)Q \quad x\notin f_n(P)$$

$$(P|Q)|R\equiv P|(Q|R) \qquad (vx)n[P]\equiv n[(vx)P] \quad x\neq n$$

化简关系式现在就是进程间满足下面公理规则的最小关系。(Red Comm)和（Red Repl）公理是受$\{^{v}/_{p}\}P$是否定义明确的客观条件的限制。

$$n[\overline{x}^{\uparrow}v|Q]\rightarrow\overline{x}^{n}v|n[Q] \qquad \text{(Red Up)}$$

$$\overline{x}^{n}v|n[Q]\rightarrow n[\overline{x}^{\downarrow}v|Q] \qquad \text{(Red Down)}$$

$$\overline{x}^{\tau}v|x^{\iota}p.P\rightarrow\{^{v}/_{p}\}P \qquad \text{(Red Comm)}$$

$$\overline{x}^{\tau}v|!x^{\iota}p.P\rightarrow!x^{\iota}p.P\{^{v}/_{p}\} \qquad \text{(Red Repl)}$$

$$P\rightarrow Q\Rightarrow P|R\rightarrow Q|R \qquad \text{(Red Par)}$$

$$P\rightarrow Q\Rightarrow(vx)P\rightarrow(vx)Q \qquad \text{(Red Res)}$$

$$P\rightarrow Q\Rightarrow n[P]\rightarrow n[Q] \qquad \text{(Red Box)}$$

$$P\equiv P'\rightarrow Q'\equiv Q\Rightarrow P\rightarrow Q \qquad \text{(Red Struct)}$$

Red Up 公理允许给父盒的输入可以穿过封闭的边界。同样，Red Down 公理允许给第 n 个子盒的输入可以穿过 n 的边界。Red Comm 公理允许实现同一个盒内的相应的输入和输出同步。同样，Red Repl 公理也是相似的，只不过是把复制的结果保存在结果状态中。

通过盒边界的传输有两个化简步骤，比如，下面所讲的向下和向上的传输

$$n[\overline{x}^{\uparrow}v]\,|\,x^{n}p.P \rightarrow n[0]\,|\,\overline{x}^{\overline{n}}v\,|\,x^{n}p.P$$
$$\rightarrow n[0]\,|\,\{v/p\}P$$
$$\overline{x}^{n}v\,|\,n[x^{\uparrow}p.P] \rightarrow n[\overline{x}^{\overline{\uparrow}}v\,|\,x^{\uparrow}p.P]$$
$$\rightarrow n[\{v/p\}P]$$

这样就消除了三方（盒、输入、输出）同步的需要，也在语义和执行模式上得到了简化。

7.2.3 加标识的变换

语义上的化简仅仅定义为进程内部计算。描述安全属性时必须涉及到进程与它们的环境的交互，需要更多的结构：一个被标识的变换关系表现为一个进程具有更潜在的输入输出的特征。

给 *box-π* 演算以明确索引形式的方式加一个标签原语，在进程结构中被归纳定义。

$\ell ::= \tau$　　内部行为

$\overline{x}^{o}v$　　输出行为

$x^{\gamma}v$　　输入行为

γ 是除了 $\overline{\uparrow}$ 以外的所有输出标识。加标识变换可以分为两种，一种是与越过盒边界时传送信息有关的，另一种是与输入、输出之间的通信有关的。移动标签是：

$\overline{x}^{n}v$ (输出到子盒n)　　$x^{\overline{n}}v$ (从父盒输入到子盒n)

$\overline{x}^{\uparrow}v$ (输出到父盒)

如果 o 是 n 或者 ↑ 形式那么 $mv(o)$ 是真。传输标签是

$\overline{x}^{*}v$ (本地输出)　　$x^{*}v$ (本地输入)

$\overline{x}^{\overline{n}}v$ (从子盒n的输出)　　$x^{n}v$(输入从子盒n的输出)

$\overline{x}^{\overline{\uparrow}}v$(父盒的输出)　　$x^{\uparrow}v$(输入从父盒的输出)

标识将使给定的变换对实现同步。当 A 是一个有限名集，并且 $fn(P) \subseteq A$ 时，加标识的变换关系可以用：

$$A \mid\!- \; P \xrightarrow{\ell} Q$$

来表示。可以说成 P 和 P 的环境名字在集合 A 中（A 为有线集），P 可以通过加标识变换 l 变为 Q。这个关系是在满足规则的最小单位。假设 x 不属于集合 A，用 A, x 来表示 $A \cup \{x\}$ 集合，假设 A 和 p 不相交，用 A, p 来表示 A 与 p 模式中出现名字的合并。在这里，一些辅助符号是很有用的。 序列标签 $l_1 \ldots l_k$,可以写成

$$A \mid\!- \; P \xrightarrow{l_1} \ldots \xrightarrow{l_k} P_{k+1}$$

$$\frac{}{\bar{x}^{o}v \xrightarrow{\bar{x}^{o}v} o} \quad (out) \qquad \frac{}{x^{r}p.P \xrightarrow{x^{r}v} \{v/p\}P} \quad (In)$$

$$\frac{}{!x^{r}p.P \xrightarrow{x^{r}v} !x^{r}p.P \mid \{v/p\}P} \quad (Repl)$$

$$\frac{}{n[P] \xrightarrow{x^{\delta}v} n[\bar{x}^{\bar{\delta}}v \mid P]} \quad (Box-2) \qquad \frac{}{A \mid -n[P] \xrightarrow{\tau} (\nu fn(x,v)-A)(\bar{x}^{\bar{n}}v \mid n[P'])} \quad (Box-1)$$

$$\frac{P \xrightarrow{\tau} P'}{n[P] \xrightarrow{\tau} n[P']} \quad (Box-3)$$

$$\frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q} \quad (Bar) \qquad \frac{A \mid - P \xrightarrow{\bar{x}^{r}v} P' \quad A \mid -Q \xrightarrow{x^{r}v} Q'}{A \mid -P \mid Q \xrightarrow{\gamma} (\nu fn(x,v)-A)(P' \mid Q')} \quad (Comm)$$

$$\frac{A, x \mid - P \xrightarrow{\ell} P'}{A \mid - (\nu x)P \xrightarrow{\ell} (\nu x)P'} \quad (Res-1) \qquad \frac{A, x \mid - P \xrightarrow{\bar{y}^{o}v} P'}{A \mid - (\nu x)P \xrightarrow{\bar{y}^{o}v} (\nu x)P'} \quad (Res-2)$$

$$\frac{P \xrightarrow{\ell} P' \quad P' \equiv P''}{P \xrightarrow{\ell} P''} \quad (Struct\ Right)$$

Res-1 的前提是 $x \notin fn(\ell)$，Res-2 的前提是如果 $\neg mv(o)$ 则 $x \in fn(v)-fn(y,o)$，否则 $x \in fn(y,v)-fn(o)$。索引 $A \mid -$ 在规则中被省略。在所有的规则里结论公式 $A \mid - P \xrightarrow{\ell} Q$ 的隐含条件是 $fn(P) \subseteq A$。在（In）和（Repl）公理里，隐含条件 $\{v/p\}P$ 是限定的。

表示在 $A_i = AU \bigcup_{j \in 1..i} fn(\ell_j)$ 中，存在 $P_2 \ldots P_k$，并且任意的 $i \in 1 \ldots k$，如果 $\ell \neq \tau$，这里把 $A \mid - P \xrightarrow{\tau}{}^{*} \xrightarrow{\ell} \xrightarrow{\tau}{}^{*} P'$ 写成 $A \mid - P \overset{\vec{l}}{\Rightarrow} P'$，如果 $\ell = \tau$，那么 $A \mid - P \overset{\vec{l}}{\Rightarrow} P'$ 被定义成 $A \mid - P \xrightarrow{\tau}{}^{*} P'$。

这两句原语在下面的定理中具有一致性。

定理 1 如果 $fn(P) \subseteq A$，当且仅当 $P \xrightarrow{\ell} Q$ 时，$A \mid - P \xrightarrow{\tau} Q$。

这让我们明确了标识原语能够传递足够的信息。

7.2.4 互模拟

为了精确描述“一个封装器不用改变正常操作的程序”的行为，需要一个操作上相等的关系。当 π 演算异步的时候，可以基于弱异步互模拟来得到一个恰当的概念。我们认为 S 中的每一个 S_A 在 $\{P \mid fn(P) \subseteq A\}$ 集合中就是一个均衡关系。在这里，关系集合 S

是由有限名集所表示的。当如下表示的时候，就说 S 是一个弱异步互模拟：

- $PS_AQ, A|- P\xrightarrow{\ell}P'$ 并且 ℓ 是一个输出或者 τ 转变，那么 $\exists Q'$，$A|- Q\overset{\bar{\ell}}{\Rightarrow}Q'\wedge P'S_{A\cup fn(\ell)}Q'$ 并且
- $PS_AQ, A|- P\xrightarrow{x^r v}P'$ 意味着 $\exists Q'$，$A|- Q\xrightarrow{x^r v}Q'\wedge P'S_{A\cup fn(x^r v)}Q'$ 或者 $\exists Q'$，$A|- Q\Rightarrow Q'\wedge P'S_{A\cup fn(x^r v)}(Q'|\bar{x}^r v)$

用≈来表示所有弱异步互模拟的并集中。

7.3 安全封装器

这一部分给出 4 个封装器的例子。第一个例子是封装了一个个别中间件，限制它与外界的交互作用，并且只有遵循特定的协议才能与外界进程通信。第二个例子与第一个很相似，只不过记录了所有这样通信的日志。第三个封装器封装了 2 个中间件，它允许每一个中间件都可以通过限定的方式和外界交互，并且第一个中间件的信息可以传输到第二个中间件（但反方向不就不可以）。第四个封装器封装了三个中间件，并且控制了它与环境的交互，限制了这些交互作用仅仅通过通道 *in* 和 *out* 来实现，达到了安全性目的。

一个封装器的设计必须和一些关于中间件和环境或者中间件间进行传输的协议有关。对于前两个封装器的例子，这里固定了 2 个通道名，为 in 和 out。它们在接收和发送信息时都是独立的。并且，这里假设中间件可在若干个盒中执行，也可以与父盒进行通讯。我们把接收到的值 v，进行复制，把一对 $\langle y\ y\rangle$ 送到输出端的一个很小成分可以写成

$$!in^{\uparrow}y.\overline{out}^{\uparrow}\ <y\ y>$$

一个有问题的中间件也可以向一非法的输出通道输入数据。例如：

$$!in^{\uparrow}y.(\overline{net}^{\uparrow}y\,|\,\overline{out}^{\uparrow}\ <y\ y>)$$

或者监听一个在该系统的其它部分的传输,例如：

$$!c^{*}y.(\overline{net}^{\uparrow}c\,|\,\bar{c}^{*}y)$$

要描述一个中间件是否遵循加标签变换语义时，可以说成对于一元封装器 P 可正常运行，当且仅当 A|— $P\xrightarrow{l_1\ldots l_k}Q$ 时候，那么 l_j 就是 $in^{\uparrow}v$，$\overline{out}^{\uparrow}v$ 或者 τ 形式。

过滤封装器 过滤器是指单纯地限制进程通信能力的封装器。考虑一个只在 in 和 out 两个通道进行交互的静态过滤器。

$$W_1[\_]\overset{def}{=}(va)(\ a[\_]\ |!in^{\uparrow}y.\overline{in}^{a}y\ |out^{a}y.\overline{out}^{\uparrow}y)$$

安装一个传信者，它可以越过边界移动合法信息。通过这一过程W_1在一个新命名的盒中执行它的中间件。注意这个和深层的封装器不是绑定关系，而是相等的，假设，无论在哪里把 W1 应用到一个进程 P 时，在 P 中不会自由产生一个新绑定 a。不考虑 P 的性能，W1[P]一定遵循一种协议，这个协议用标识传输原语就可以清楚描述的。

命题 2 对任何程序 P，且$a \notin fn(P)$，如果 A|— $W_1[P] \xrightarrow{l_1 \ldots l_n} Q$ 那么l_j是$in^{\uparrow}v$，$\overline{out}^{\uparrow}v$或者$\tau$形式。

可以通过可获得状态的直接描述来证明。而这种状态可由加标识变换$^{W_1}[P]$得到。一元封装器的这个特性特别抽象。

日志封装器 过滤器通过沿 log 通道把副本发送到周围，来保持进程中所有通信的日志。

$$L[\_] \stackrel{def}{=} (va)(\ a[\_] \mid !in^{\uparrow}y.(\overline{\log}^{\uparrow}y \mid \overline{in}^{a}y) \mid out^{a}y.(\overline{\log}^{\uparrow}y \mid \overline{out}^{\uparrow}y)$$

一个封装程序$L[P]$也可以再次通过受限方式进行交互。

命题 3 对于所有的程序 P 且$a \notin fn(P)$，如果 A|— $L[P] \xrightarrow{l_1 \ldots l_n} Q$，那么$l_j$是$in^{\uparrow}v$，$\overline{out}^{\uparrow}v$，$\overline{\log}^{\uparrow}v$或者$\tau$形式。

管道封装器 一个管道封装器能控制两个中间件之间的受约束的信息流。这里给出一个二元封装器 W2，它包含 2 个进程。$W_2[Q_1,Q_2]$的执行中两个被封装的进程Q_i，可以在通道 in_i 和 out_i 与周围环境可以交互，这是如上所说的是一样的。另外，Q_1能够通过 mid 通道发送消息到Q_2。这里的管道执行是无序的。

$$\begin{aligned} W_2[\_{}_1,\_{}_2] \stackrel{def}{=} (va_1,a_2)(\ & a_1[\_{}_1] \mid a_2[\_{}_2] \\ & \mid !in_1^{\uparrow}y.\overline{in_1}^{a_1}y \\ & \mid !in_2^{\uparrow}y.\overline{in_2}^{a_2}y \\ & \mid out_1^{a_1}y.\overline{out_1}^{\uparrow}y) \\ & \mid out_2^{a_2}y.\overline{out_2}^{\uparrow}y) \\ & \mid mid_1^{a_1}y.\overline{mid_1}^{a_2}y) \end{aligned}$$

与上面的一样，W2 没有绑定的情况下，总把 W2 应用到进程P_1,P_2，还假设

$\{a_1,a_2\}\cap fn(P_1,P_2)=\phi$。当且仅当对于任何进程$P_1,P_2$满足适当的自由名时，也就是$\{a_1,a_2\}\cap fn(P_1,P_2)=\phi$时，才认为二元封装器为真，如果A|— $L[P_1,P_2]\xrightarrow{l_1\ l_2}Q$，那么$l_j$是$in_i^{\uparrow}v$，$\overline{out_i}^{\uparrow}v$或者$\tau$形式。

命题4　W2为真。

比如说，假设$P_2=\overline{mid}^{\uparrow}v$，第二个封闭的进程把一个数据发送到第一个进程。有

$$W_2[P_1,\overline{mid}^{\uparrow}v] = (va_1,a_2)(\ a_1[P_1]\,|\,a_2[\overline{mid}^{\uparrow}v]\,|\,R)$$
$$\rightarrow (va_1,a_2)(\ a_1[P_1]\,|\,a_2[0]\,|\,\overline{mid}^{\overline{a_2}}v\,|\,R)$$

这里R是一个并行传输的合成。输出$\overline{mid}^{\overline{a_2}}v$在决定性状态下不能进行更深层交互，也就是当$a_2$受限时，不能与周围环境更深交互，当$a_1\neq a_2$时，不能与传输$!mid^{a_1}y.\overline{mid}^{a_2}y$。

这些封装器都假定了一个简单、固定的协议。它直接产生一系列任意通道来代替in,out和mid。也可以直接允许n元封装器，来封装很多中间件，让信息按照给定的顺序在中间件间传输。

三元中间件封装器

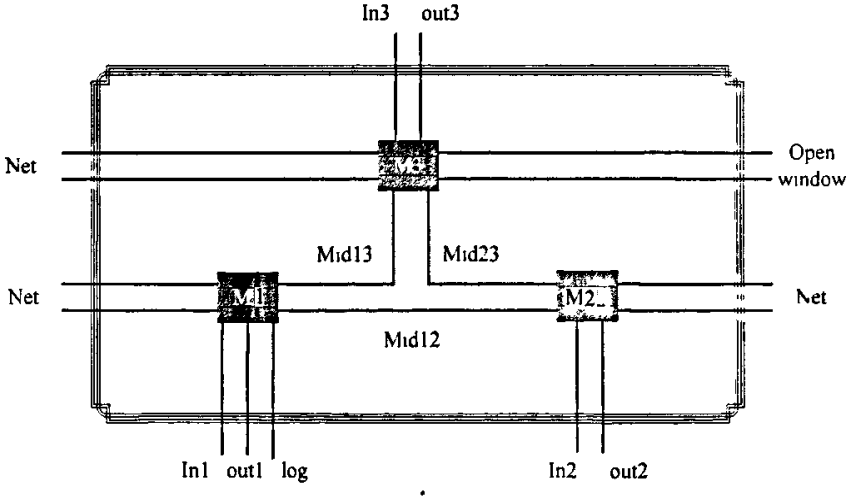


图7.1　三元中间件封装器

这个图表示了一个封装器W封装了三个中间件M_1,M_2和M_3，并且控制了它与环境的交互，限制了这些交互作用仅仅通过通道*in*和*out*来实现，达到了安全性目的。其中M_1,M_2和M_3都与net相连，M_3还和open window相连。

下面就解释一下用box-π calculus来表示的过程。

$W[-1,-2,-3]\Longrightarrow V(a_1,a_2,a_3)\ \ (a_1[-1]\,|\,a_2[-2]\,|\,a_3[-3])$

定义一个三元封装器W，封装了三个中间键，分别为M_1,M_2和M_3,这样就生成了

个名字为 a_1,a_2,a_3 的盒子，它们分别有两个传送器——个是通过通道 *in* 从环境接收信息，并且把信息送到封装程序；另一个是通过通道 *out* 从封装程序中接收信息，并且把信息送到环境中去，既 in 和 out 。M_1,还有一个 log 通道。

$!\ in_1^{\ y}\cdot(\overline{in}^{a_1}y\,|\,\overline{\log}^{\uparrow}y)$

传感器通过 in_1 通道从环境中接收信息，并把信息传到 a_1 中的封装程序，且通过 log 通道把通信的日志记录下来。

$!\ out_{\ 1}^{\ a_1}y\cdot(\overline{out_{\ 1}}^{\uparrow}y\,|\,\overline{\log}^{\uparrow}y)$

通过 out_1 通道从 a_1 中的封装程序中接收信息，并把信息传送到环境中去，且通过 log 通道把通信的日志记录下来。

$!\ in_2^{\uparrow}\cdot\overline{in}^{a_2}y$

传感器通过 in_2 通道从环境中接收信息，并把信息传到 a_2 中的封装程序。

$!\ out_2^{a_2}y\cdot\overline{out_2}^{\uparrow}y$

通过 out_2 通道从中 a_2 的封装程序中接收信息，并把信息传送到环境中去。

$!\ in_3^{\uparrow}\cdot\overline{in_3}^{a_3}y$

传感器通过 in_3 通道从环境中接收信息，并把信息传到 a_3 中的封装程序。

$!\ out_3^{a_3}y\cdot\overline{out_3}^{\uparrow}y$

通过 out_3 通道从中 a_3 的封装程序中接收信息，并把信息传送到环境中去。

$!\ net\_in_1^{\uparrow}y\cdot(\overline{net\_in_1}^{a_1}y\,|\,\overline{\log}^{\uparrow}y)$

传感器通过 in_1 通道从 net 中接收信息，并把信息传到 a_1 中的封装程序，且通过 log 通道把通信的日志记录下来。

$!\ net\_out_1^{a_1}y\cdot\ (if\ \ y不包含个人信息如email等\ \ then\ \ \overline{net\_out_1}^{\uparrow}y\ \ else\ \ 0\,|\,\overline{\log}^{\uparrow}y)$

通过 out_1 通道从 a_1 中的封装程序中接收信息，并当信息 y 中不包含个人信息如 e-mail 等，就把信息传送到 net 中去，且通过 log 通道把通信的日志记录下来，否则就记录成空进程。

$!\ net\_in_2^{\uparrow}y\cdot\ if\ \ y不是来自某域或某IP\ \ then\ \ \overline{net\_in_2}^{a_2}y\ \ else\ \ 0$

传感器通过 in_2 通道从 net 中接收信息，当信息 y 不是来自某域或某 IP 时，就把信息传到 a_2 中的封装程序，否则记录成空进程。

$!\ net\_out_2^{a_2}y\cdot\overline{net\_out_2}^{\uparrow}y$

通过 out_3 通道从中 a_3 的封装程序中接收信息，并把信息传送到 net 中去。

$!\ net\_in_3^{\uparrow}y\cdot\ if\ \ y\in\{update,install\}\ \ then\ \ \ \overline{net\_in_3}^{a3}y\ \ else\ \ 0$

传感器通过 in_3 通道从 net 中接收信息，当 y 更新或安装信息时，信息传到 a_3 中的封装程序，否则记录为空进程。

$!\ net\_out_3^{a_3} y \cdot \overline{net\_out_3}^{\uparrow} y$

通过 out_3 通道从中 a_3 的封装程序中接收信息，并把信息传送到 net 中去。

$!\ mid_{13}{}^{a_1} y \cdot\ (if\ \ y \in \{0,1\}\ \ then\ \ \overline{mid_{13}}^{a_3} y\ \ else\ \ 0 \mid \overline{\log}^{\uparrow} y)$

M_1 通过 mid_{13} 通道可以把 a_1 中封装程序中的信息 y 发送到 M_3 中，当 y 是 0 或 1 时，M_3 中封装程序中的信息发送到 M_1，否则记录为空进程。

$!\ mid_{13}{}^{a_3} y \cdot (\overline{mid_{13}}^{a_1} y \mid \overline{\log}^{\uparrow} y)$

通过 mid_{13} 通道进行 M_1 和 M_3 中的信息的交换，并且通过 log 通道把通信的日志记录下来。

$!\ mid_{12}{}^{a_1} y \cdot (\overline{mid_{\ 1}}^{a_2} y \mid \overline{\log}^{\uparrow} y)$

通过 mid_{12} 进行 M_1 和 M_2 的信息的交换，并且通过 log 通道把通信的日志记录下来。

$!\ mid_{23}{}^{a_2} y \cdot \overline{mid_{23}}^{a_3} y$

通过 mid_{23} 行 M_2 和 M_3 之间的信息的交换。

$!\ openwindow^{a_3}(s\ \ x) \cdot \overline{openwindow}^{\uparrow} < s\ \ x >$

$\mid\ x^{\uparrow}(getc\ \ putc\ \ close) \cdot \bar{x}^{a_3} < getc\ \ putc\ \ close >$

$!\ getc^{a_3} y \cdot (\overline{getc}^{\uparrow} < c\ \ y > \mid y^{\uparrow} \bar{y}^{a_3} c)$

$!\ putc^{a_3}(c\ \ y) \cdot (\overline{putc}^{\uparrow} < c\ \ y > \mid y^{\uparrow} \bar{y} \cdot^{a_3})$

$!\ close^{a_3} y \cdot (\overline{close}^{\uparrow} y \mid y^{\uparrow} \cdot \bar{y}^{a_3})$

传感器通过 s 来自任意子盒的信息 x 输入，并且把子盒的名字绑定到 a_3，也可以将 a_3 中的信息 x 输出到 openwindow。

来自 openwindow 信息 x 可以读取、写入和关闭到 M_3 中，并且可以读取、写入和关闭到 a_3 中。

读取 a_3 中的信息 y，并且读到 openwindow，在把 c 中得到的 y 传到 a_3 中。

把 c 中得到的 y 写入到 a_3 中。把 c 中得到的 y 写进来，并把 y 信息传送到盒 a_3。

关闭 y 信息到 a_3。把 y 信息关闭，然后传送到 a_3。

这个模型表示了一个封装了三个中间件的封装器，它控制了任意中间件之间、各个中间件与环境之间的交互，限制了各中间件与 net 的交互作用仅仅通过通道 *in* 和 *out* 来实现，并事例了 M_1 还和 Log 相连，M_3 和 open window 相连，达到了安全性目的。

7.4 本章小结

这一章提出了一套用于中间件安全封装器的理论。共设计并证明四个封装器：过滤封装器、日志封装器、管道封装器和三元中间件封装器等。根据三元封装器，能很容易的构造出 n 元封装器的模型。这些封装器能保证不可信的中间件与其它中间件、网络、

操作系统、运行环境以及日志等之间的安全信息交互，并能实施动态、灵活的安全策略。通过最小并发的编程语言（*box-π* 演算）来研究这个问题，并证明了一个基本的理论结果：化简和标签变换的语义上的一致性。

这部分内容开辟了值得我们深入研究的很多方向，很多推论和结论等待进一步的发掘和探索。而且对于二进制封装器的理解上，也要做进一步的努力。目前设计的四个封装器还没有在国家科技基础条件平台上进行实际应用和部署，还处于理论上的演练阶段。另外，为封装器设计更丰富的接口，甚至，支持移动原语。

第八章 总结和展望

现有的中间技术是必需的，但不能有效的保护大范围、分布式资源。针对一些实际应用中比较关键的问题，本文进行了深入的分析和研究，得出了相应的结果。并根据结果在国家科技基础条件平台项目中进行了部署，或者进行了模拟和推演。

8.1 研究成果

第一，本文定义了一个 J2EE 保护系统状态的配置，设计了 J2EE 中的鉴权判定算法，这个配置和鉴权判定算法一起在数学上定义了 J2EE 安全鉴权系统的状态和行为。利用 J2EE 保护系统状态的配置，本文进行了 J2EE 安全服务对 RBAC 模型支持的设计。首先，本文在 J2EE 安全语言中提供了 $RBAC_0$ 和 $RBAC_1$ 的实现；然后，为了支持 $RBAC_0$—$RBAC_3$ 模型，分析了实现 J2EE 安全服务的需求。为了支持 RBAC 模型实现，本文设计了兼容 J2EE 安全规范的方法。这个工作通过最大化 J2EE 访问控制机制的效用来更好的理解它的能力，这一点对于使用中间件来保护系统资源是至关重要的。

第二，本文设计了基于 J2EE 的资源访问判定服务（JCRADS）体系结构，这是一种新颖的配置鉴权机制的体系结构方法，在功能上，这个鉴权机制满足了在鉴权判定中使用针对应用的信息来保护更细粒度的应用资源的需求，它允许应用与鉴权逻辑的分离，这样就使得应用开发、部署和管理更加有效。而且它能保证多个应用间策略的一致性。本文通过模型化鉴权策略的手段展示了其功能和性能，表明鉴权策略需要使用针对应用的信息作为用户和资源拥有者之间的联系。

无论是使用J2EE/JAAS鉴权机制还是基于JCRADS的服务，鉴权和应用逻辑的分离使得分布式系统及其安全性能简单化，这样更容易提高它们的质量。同样重要的，它为保持鉴权策略的一致性和集中安全鉴权及管理、降低时间开销、减少误差等方面都铺平了道路。

第三，通过实现基于 J2EE 应用鉴权服务的原型系统（JAS，这个原型系统是根据 JCRADS 构建的），本文获得了一些基于 JCRADS 的鉴权服务设计的心得。另外，还给出了 JCRADS 体系结构的主要特征，比如灵活性、可配置能力和可扩展能力等，以及使用 J2EE 中间件技术对这些性能进行了实例化。给出了开发 JAS 的经验，这为基于 JCRADS 服务的设计提供了指导性的方针。

通过 JAS 测试床，本文获得了 JAS 构件的不同组件性能的定量评估。本文发现，合理地使用执行时间、鉴权请求的数量以及性能约束，JAS 配置可以提供需要的性能。

最后，本文研究了封装器的表示方法，并使用 calculus 语言设计了四种封装器：过滤封装器、日志封装器、封装器和 n 元封装器，探讨了各种封装器能够保证的安全性。

8.2 进一步的工作

虽然 JCRADS 方法能够满足大多数的需求，对于分布式企业资源的访问控制还远远没有解决。对于将来的研究有很多问题和机遇。

本文中提出了两种方法对分布式资源的访问控制问题进行解决。一个重要的问题是如果两种系统一起使用，如何统一的管理AC。一种方法是使用基于JCRADS的服务来完成鉴权判定，这不仅仅是对于应用系统，还包括中间件层、网络、DBM和操作系统。另外一种方法是鉴权策略通过代理传送给中间件和AC机制。与其它的方法相比，这种方法是不是更好呢？

当鉴权判定可以代表一个服务的时候，一个潜在的性能和可用性问题就出现了。需要新的研究来理解基于JCRADS结构鉴权服务的分布式技术是怎样实现性能的稳定性和有效性的。

JCRADS方法对于基于大多数中间技术的问题是有效的。本文用基于J2EE的原型来实施和管理试验的性能。因为分布式系统通常使用多种中间件技术，研究鉴权服务多重技术设计的复杂性是很有意思的。

与任何复杂的软件系统一样，基于JCRADS服务的应用系统的组成不仅能协调各组成要素一起工作，还要保证各组件作为一个整体连贯的工作，保证端对端的传输。虽然这已经超出了这个论题的范围，但是这是非常重要的。

JCRADS方法已经在国家科技基础条件平台项目中进行了应用。而且这个方法也能被用到其它的安全性能中，例如审计和通信保护等。问题是，这个模型能在没有重要改变的情况下被重用吗？如果不能将做哪些调整呢？有没有标准的方法来统一和综合这些相关的决策，使得不仅是AC还有大多数安全性能都可以在应用中被重用。

参考文献

1. [Abrams 1991] M. Abrams, J. Heaney, O. King, L. J. LaPadula, M. Lazear, and I. Olson,"A Generalized Framework for Access Control: Towards Prototypingthe Orgcon Policy," In Proceedings of National Computer Security Conference,1991, pp. 257-266.
2. [Abrams 1990a] M. D. Abrams, K. W. Eggers, L. J. LaPadula, and I. W. Olson, "A GeneralizedFramework for Access Control: an Informal Description," TheMITRE Corporation, McLean, Virginia, USA MP-90W00043, August1990.
3. [Abrams 1989] M. D. Abrams, A. B. Jeng, and I. M. Olson, "Generalized Framwork forAccess Control: An Informal Description," The MITRE Corporation,Springfield, VA, USA MTR-89W00230, September 1989.
4. [Abrams 1990b] M. D. Abrams, L. J. LaPadula, and I. M. Olson, "Building GeneralizedAccess Control on UNIX," In Proceedings of USENIX workshop onUNIX Security, Portland, Oregon, USA, 1990, pp. 65-70.
5. [Amoroso 1994] E. Amoroso, *Fundamentals of Computer Security Technology.* PrenticeHall, 1994.
6. [Anderson 1972] J. Anderson, "Computer Security Technology Planning Study," AirForce Electronic Systems Division ESD-TR-73-51, Vols. I and II, 1972.
7. [Ashley 1997] P. Ashley, "Authorization for a Large Heterogeneous Multi-DomainSystem," In Proceedings of Australian Unix and Open Systems GorupNational Conference, 1997.
8. [Awischus 1997] R. Awischus, "Role Based Access Control with Security AdministrationManager (SAM)," In Proceedings of the Second ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1997, pp. 61-68.
9. [Barkley 1995] J. Barkley, "Implementing Role-based Access Control Using Object Technology," In Proceedings of The First ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1995, pp. 93-98.202
10. [Barkley 1999] J. Barkley, K. Beznosov, and J. Uppal, "Supporting Relationships in Access Control Using Role Based Access Control," In Proceedings of ACM Role-based Access Control Workshop, Fairfax, Virginia, USA,1999, pp. 55-65.
11. [Barkley 1998] J. Barkley and A. Cincotta, "Managing Role/Permission Relationships Using Object Access Types," In Proceedings of The Third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1998, pp.73-80.
12. [Bell 1975] D. E. Bell and L. J. LaPadula, "Secure Computer Systems: Unified Exposition and Multics Interpretation," MITRE, Bedford, MA, USA, Technical Report ESD-TR-75-306, March 1975.
13. [Benantar 1996] M. Benantar, R. Guski, and K. M. Troidle, "Access control systems: From host-centric to network-centric computing," *IBM Systems Journal*, vol. 35(1), pp. 94-112, 1996.
14. [Bertino 1996a] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "Supporting Periodic Authorizations and Temporal Reasoning in Database Access Control," In Proceedings of

22th International Conference on Very Large Data Bases, Mumbai (Bombay), India, 1996, pp. 472-483.

15. [Bertino 1996b] E. Bertino, S. Jajodia, and P. Samarati, "Supporting Multiple Access Control Policies in Database Systems," In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, California, 1996.
16. [Beznosov 1997] K. Beznosov, "Applicability of CORBA Security to the Healthcare Problem Domain," Object Management Group corbamed/97-09-11, September 1997.
17. [Beznosov 1998a] K. Beznosov, "Issues in the Security Architecture of the Computerized Patient Record Enterprise," In Proceedings of Second Workshop on Distributed Object Computing Security, Baltimore, Maryland, USA, 1998.
18. [Beznosov 1998b] K. Beznosov, "Requirements for Access Control: US Healthcare Domain," In Proceedings of Third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1998, pp. 43.
19. [Beznosov 2000] K. Beznosov, "Information Enterprise Architectures: Problems and Perspectives," School of Computer Science, Florida International University, Miami technical report 2000-06, June 2000. 203
20. [Beznosov 1999a] K. Beznosov and Y. Deng, "A Framework for Implementing Role-based Access Control Using CORBA Security Service," In Proceedings of Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1999, pp. 19-30.
21. [Beznosov 1999b] K. Beznosov, Y. Deng, B. Blakley, C. Burt, and J. Barkley, "A Resource Access Decision Service for CORBA-based Distributed Systems," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 310-319.
22. [Blakley 1999] B. Blakley, *CORBA Security: an Introduction to Safe Computing with Objects*, First ed. Addison-Wesley, 1999.
23. [Bloomer 1992] J. Bloomer, *Power Programming with RPC*. O'Reilly & Associates, 1992.
24. [BullSoft 1995] BullSoft, "AccessMaster," Bull Soft, 1995. [Burrows 1990] M. Burrows, M. Abadi, and R. Needham, "A Logic of Authentication," *ACM Transaction on Computer Systems*, vol. 8(1), pp. 18-36, 1990..
25. [CA 1998a] CA, "CA-ACF2 for OS/390," Computer Associates International, 1998.
26. [CA 1998b] CA, "CA-Top Secret for OS/390," Computer Associates International, 1998.
27. [CA 1999] CA, "Unicenter TNG: Product Information," Computer Associates International, 1999.
28. [柴 2003] 柴晓路，用J2EE架构企业级应用，开放系统世界—赛迪网，2003年02月25日。
29. [Caswell 1995] D. L. Caswell, "An Evolution of DCE Authorization Services," *Hewlett-Packard Journal: technical information from the laboratories of Hewlett-Packard Company*, vol. 46(6), pp. 49--54, 1995.
30. [CIST-NRC 1999]CIST-NRC, *Trust in Cyberspace*. Committee on Information Systems Trustworthiness, National Research Council. National Academy Press, 1999.
31. [Curry 1992] D. A. Curry, *UNIX System Security: A Guide for Users and System Administrators*. Addison-Wesley, 1992.

32. [DeBoever 1997] L. R. DeBoever, "Concept of "Highly Adaptive" Enterprise Architecture," In Proceedings of Enterprise Architecture Conference, 1997.
33. [DEC 1989] DEC, "Guide to VAX/VMS System Security --- Version 5.2," Digital Equipment Corporation, 1989. 204
34. [DHHS 1998] DHHS, "Security and Electronic Signature Standards; Proposed Rule," 45 CFR Part 142, Department of Health and Human Services, 1998.
35. [DHHS 1999] DHHS, "Standards for Privacy of Individually Identifiable Health Information; Proposed Rule," Department of Health and Human Services, 1999.
36. [Eddon 1999] G. Eddon, "The COM+ Security Model Gets You out of the Security Programming Business," *Microsoft Systems Journal*, vol. 1999(11), 1999.
37. [Epstein 1995] J. Epstein and R. Sandhu, "NetWare 4 as an Example of Role-Based Access Control," In Proceedings of Proceedings of the First ACM Workshop on Role-Based Access Control, Gaithersburg, Maryland, USA, 1995, pp. 71-82.
38. [Espinal 2000] L. Espinal, K. Beznosov, and Y. Deng, "Design and Implementation of Resource Access Decision Server," Center for Advanced Distributed Systems Engineering (CADSE) - Florida International University, Miami technical report 2000-01, January 2000.
39. [范 2004] 范逖，CORBA 及Java IDL应用编程，http://www.fanqiang.com，2004
40. [Filman 1996a] R. Filman and T. Linden, "Communicating Security Agents," In Proceedings of The Fifth Workshop on Enabling Technologies: Infrastructure for Collabarative Enterprises, Stanford, CA, USA, 1996, pp. 86-91.
41. [Filman 1996b] R. Filman and T. Linden, "SafeBots: a Paradigm for Software Security Controls," In Proceedings of New Security Paradigms Workshop, Lake Arrowhead, CA USA, 1996, pp. 45-51.
42. [Fowler 1997] M. Fowler, *Analysis Patterns: Reusable Object Models*, First ed. Addison Wesley Longman, 1997.
43. [Gamma 1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Element of Reusable Object-Oriented Design*. Addison-Wesley, 1995.
44. [Gittler 1995] F. Gittler and A. C. Hopkins, "The DCE Security Service," *Hewlett-Packard Journal*, vol. 46(6), pp. 41-48, 1995.
45. [Giuri 1998] L. Giuri, "Role-Based Access Control in Java," In Proceedings of Proceedings of the Third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1998, pp. 91-99.
46. [Gligor 1986] V. Gligor, C. Burch, R. Chandersekaran, L. Chanpman, M. Hecht, W. Jiang, G. Luckenbaugh, and N. Vasudevan, "On the Design and the Implementation of Secure Xenix Workstations," In Proceedings of 205IEEE Symposium on Security and Privacy, Oakland, CA, 1986, pp. 102-117.
47. [Gong 1997] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers, "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2," In Proceedings of The USENIX Symposium on Internet Technologies and Systems, Monterey, California, 1997, pp. 103-112.
48. [龚 2004] 龚永生，通用验证系统，http://www-900.ibm.com/developerWorks/cn/index.shtml，2004。
49. [Grampp 1984] F. T. Grampp and R. H. Morris, "UNIX Operating System Security,"

AT& Bell Laboratories Technical Journal, vol. 63(8), pp. 1649-1672, 1984.
50. [Grand 1998] M. Grand, *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, vol. 1. Wiley Computer Publishing, 1998.
51. [Grimes 1997] R. Grimes, *Professional DCOM Programming*. Wrox Press Inc., 1997.
52. [Grimm 1999] R. Grimm and B. Bershad, "Providing Policy-Neutral and Transparent Access Control in Extensible Systems," *Lecture Notes in Computer Science*, pp. 317-338, 1999.
53. [Grimshaw 1998] A. S. Grimshaw, M. J. Lewis, A. J. Ferrari, and J. F. Karpovich, "Architectural Support for Extensibility and Autonomy in Wide-Area Distributed Object Systems," Department of Computer Science, University of Virginia CS-98-12, 1998.
54. [Grimshaw 1997] A. S. Grimshaw and W. A. Wulf, "The Legion Vision of a Worldwide Virtual Computer," *Communications of the ACM*, vol. 40(1), pp. 39-45, 1997.
55. [Grimson 2000] J. Grimson, W. Grimson, and W. Hasselbring, "The System Integration Challange in Health Care," *Communications of the ACM*, vol. 43(6), pp. 48-55, 2000.
56. [Hailpern 1990] B. Hailpern and H. Ossher, "Extending Objects to Support Multiple Interfaces and Access Control," *IEEE Transactions on Software Engineering*, vol. 16(11), pp. 1247-1257, 1990.
57. [Hale 1999] J. Hale, P. Galiasso, M. Papa, and S. Shenoi, "Security Policy Coordination for Heterogeneous Information Systems," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 219-228.
58. [Heydon 1994] A. Heydon and J. D. Tygar, "Specifying and Checking UNIX Security Constraints," *Computing Systems*, vol. 7(1), pp. 9-12, 1994. 206
59. [Hommes 1990] R. Hommes, "VMS Security Architecture," In Proceedings of DECUS Europe Symposium, Cannes, France, 1990.
60. [HP 1996] HP, "HP Adds Value to DCE Security Framework with Praesidium Authorization Server," *DCE application development trends Magazine*, 1996.
61. [IBM 1976] IBM, *Resource Access Control Facility (RACF). General Information.* IBM Red Books, 1976.
62. [IBM 2004] IBM, Java 授权内, http://www-900.ibm.com/developerWorks/cn/index.shtml.
63. [IEEE] IEEE, *IEEE P1003.6.1 Standard for Information Technology: Portable Operating System Interface (POSIX): Protection, Audit, and Control Interfaces.* IEEE Computer Society Press.
64. [IETF 1993] IETF, "RFC 1510, The Kerberos Network Authentication Service, V5," Internet Engineering Task Force, 1993.
65. [Jonscher 1995] D. Jonscher and K. R. Dittrich, "Argos -- A Configurable Access Control System for Interoperable Environments," In Proceedings of IFIP WG11.3 Ninth Annual Working Conference on Database Security, Rensselaerville, NY, 1995, pp. 39-66.
66. [Kaijser 1998] P. Kaijser, "A Review of the SESAME Development," *Lecture Notes in Computer Science*, vol. 1438, pp. 1-8, 1998.
67. [Karger 1991] P. A. Karger, M. E. Zurko, D. W. Bonin, A. H. Mason, and C. E. Kahn, "A Retrospective on the VAX VMM Security Kernel," *IEEE Transactions on Software Engineering*, vol. 17(11), pp. 1147-1165, 1991.
68. [Karjoth 1998] G. Karjoth, "Authorization in CORBA Security," In Proceedings of Fifth European Symposium on Research in Computer Security (ESORICS), 1998, pp. 143-158.

69. [南 2004a] 南凯等，“国家科技基础条件平台门户系统”需求设计，2004.1。
70. [南 2004b] 南凯等，“国家科技基础条件平台门户系统”技术方案，2004.4。
71. [南 2005a] 南凯等，“国家科技基础条件平台门户系统”技术规范2.0，2005.3。
72. [南 2004b] 南凯等，“国家科技基础条件平台门户系统”设计文档，2004.3。
73. [Kleinoder 1996] J. Kleinoder and M. Golm, "MetaJava: An Efficient Run-Time Mete Architecture for Java," In Proceedings of Fifth IEEE International Workshop on Object-Orientation in Operating Systems, Seattle, WA, USA, 1996.
74. [Kong 1995] M. M. Kong, "DCE: An Environment for Secure Client/Server Computing," *Hewlett-Packard Journal*, vol. 46(6), pp. 6-15, 1995.
75. [Lai 1999] C. Lai, L. Gong, L. Koved, A. Nadalin, and R. Schemers, "User Authentication And Authorization In The Java Platform," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, USA, 1999, pp. 285-290. 207
76. [Lampson 1991] B. Lampson, M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practics," In Proceedings of ACM Symposium on Operating Systems Principles, Asilomar Conference Center, Pacific Grove, California, 1991, pp. 165-182.
77. [Lampson 1971] B. W. Lampson, "Protection," In Proceedings of 5th Princeton Conference on Information Sciences and Systems, Princeton, 1971, pp. 437.
78. [LaPadula 1990] L. J. LaPadula, "Formal modeling in a Generalized Framework for Access Control," In Proceedings of Computer Security Foundation Workshop III, 1990, pp. 100-109.
79. [Lea 1996] D. Lea, *Concurrent Programming in Java: Design Principles and Patterns*. Addison-Wesley, 1996.
80. [Linn 1993] J. Linn, "Generic Security Service Application Program Interface," Internet Engineering Task Force, Internet Draft RFC 1508, September 1993.
81. [Linn 1997] J. Linn, "Generic Security Service Application Program Interface," IETF RFC 2078, January 1997.
82. [Luckenbaugh 1986]G. L. Luckenbaugh, V. D. Gligor, L. J. Dotterer, and C. S. Chandersekaran, "Interpretation of the Bell-LaPadula Model in Secure Xenix," In Proceedings of DoD-NBS Conference on Computer Security, 1986.
83. [Maes 1987] P. Maes, "Computational Reflection," in *Artificial Intelligence Laboratory*. Vrije Universiteit Brussel, 1987.
84. [McCauley 1979] E. J. McCauley and P. J. Drongowski, "KSOS -- The Design of a Secure Operating System," In Proceedings of National Computer Conference, 1979.
85. [McInerney 1999] M. J. McInerney, *Windows NT Security*. Prentice Hall, 1999.
86. [McMahon 1995] P. McMahon, "Making the Internet Safe for Business," *ICL Systems Journal*, vol. 10(2), 1995.
87. [Meyers 1997] W. J. Meyers, "RBAC Emulation on Trusted DG/UX," In Proceedings of Proceedings of the Second ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1997, pp. 55-60.
88. [Microsoft 1998] Microsoft, "DCOM Architecture," Microsoft, 1998.
89. [Molva 1992] R. Molva, G. Tsudik, E. V. Herreweghen, and S. Zatti, "KryptoKnight Authentication and Key Distribution System," In Proceedings of Euro208 pean

Symposium on Research in Computer Security, Toulouse, France, 1992.
90. [Mowbray 1997] T. J. Mowbray and R. C. Malveau, *CORBA Design Patterns*. Wiley Computer Publishing, 1997.
91. [Mowbray 1995] T. J. Mowbray and R. Zahavi, *The Essential CORBA: Systems Integration Using Distributed Objects*. Wiley Computer Publishing, 1995.
92. [Mullender 1990] S. J. Mullender, G. v. Rossum, A. S. Tanenbaum, R. v. Renesse, and H. v. Staveren, "Amoeba: A Distributed Operating System for the 1990s," *Computer*, vol. 23(5), pp. 44-53, 1990.
93. [NCSC 1987] NCSC, "A Guide to Understanding Discretionary Access Control in Trusted Systems," National Computer Security Center NCSC-TG-003, September 30 1987.
94. [Neuman 1993] B. C. Neuman, "Proxy-Based Authorization and Accounting for Distributed Systems," In Proceedings of International Conference on Distributed Computing Systems, Pittsburgh, Pennsylvania, 1993.
95. [Neuman 1994a] B. C. Neuman and T. Ts'o, "Kerberos: an Authentication Service for Computer Networks," *IEEE Communications Magazine*, vol. 32(9), pp. 33-38, 1994.
96. [Neuman 1994b] B. C. Neuman and T. Y. Ts'o, "Kerberos: an Authentication Service for Computer Networks," University of Southern California, Information Sciences Institute ISI/RS-94-399, 1994.
97. [Notargiacomo 1995]L. Notargiacomo, "Role-Based Access Control in ORACLE7 and Trusted ORACLE7," In Proceedings of the First ACM Workshop on Role-Based Access Control, Gaithersburg, Maryland, USA, 1995, pp. 65-69.
98. [NSF 1999] NSF, "Information Technology Research Program Requirements," National Science Foundation, 1999.
99. [Nutt 1997] G. Nutt, *Operating Systems: A Modern Perspective*. Addison-Wesley, 1997.
100. [OMG 1996a] OMG, "CORBAservices: Common Object Services Specification," Object Management Group, 1996.
101. [OMG 1996b] OMG, "Security Service Specification," Object Management Group, 1996. 209
102. [OMG 1997] OMG, "Clinical Observations Access Service RFP," Object Management Group December 1997.
103. [OMG 1998a] OMG, "Interoperable Naming Service, Joint Revised Submission," Object Management Group, document orbos/98-10-11, October 1998.
104. [OMG 1998b] OMG, "Person Identification Service," Object Management Group, specification corbamed/98-02-29, February 1998.
105. [OMG 1999a] OMG, "The Common Object Request Broker: Architecture and Specification," Object Management Group, Specification formal/99-10-08, 1999.
106. [OMG 1999b] OMG, "IDL to Java Language Mapping," Object Management Group, Specification formal/99-07-53, 1999.
107. [OMG 1999c] OMG, "Resource Access Decision Facility," Object Management Group OMG document number: corbamed/99-05-04, May 1999.
108. [OSF 1996] OSF, "Authentication and Security Services," Open Software Foundation, 1996.
109. [Parker 1995] T. Parker and D. Pinkas, "SESAME V4 - Overview," SESAME December

1995.
110.[Pedrick 1998] D. Pedrick, J. Weedon, J. Goldberg, and E. Bleifield, *Programming with VisiBroker: A Developer's Guide to Visibroker for Java.* Wiley Computer Publishing, 1998.
111.[Pfleeger 1989] C. P. Pfleeger, *Security in Computing.* Prentice-Hall, 1989.
112.[Postel 1982] J. B. Postel, "RFC 821: Simple Mail Transfer Protocol," University of Southern California Information Sciences Institute RFC 821, August 1982.
113.[Postel 1983] J. B. Postel, "TELNET Protocol Specification," DDN Network Information Center, Request for Comments 854, May 1983.
114.[Postel 1985] J. B. Postel, "File Transfer Protocol," DDN Network Information Center, Request for Comments 959, October 1985.
115.[Quarterman 1985]J. S. Quarterman, A. Silberschatz, and J. L. Peterson, "4.2BSD and 4.3BSD as Examples of the UNIX System," *ACM Computing Surveys*, vol. 17(4), pp. 379-418, 1985. 210
116.[Riechmann 1997]T. Riechmann and F. J. Hauck, "Meta Objects for Access Control: Extending Capability-based Security," In Proceedings of New Security Paradigms Workshop, Langdale, Cumbria, UK, 1997, pp. 17-22.
117.[Riechmann 1998]T. Riechmann and F. J. Hauck, "Meta Objects for Access Control: A Formal Model for Role-based Principals," In Proceedings of New Security Paradigms Workshop, Charlottesville, VA USA, 1998, pp. 30-38.
118.[Rubin 1999] W. Rubin and M. Brain, *Understanding DCOM.* P T R Prentice Hall, 1999.
119.[Ryutov 2000a] T. Ryutov and C. Neuman, "Access Control Framework for Distributed Applications (Work in Progress)," Internet Engineering Task Force, Internet Draft draft-ietf-cat-acc-cntrl-frmw-03, March 9 2000.
120.[Ryutov 2000b] T. Ryutov and C. Neuman, "Generic Authorization and Access control Application Program Interface: C-bindings," Internet Engineering Task Force, Internet Draft draft-ietf-cat-gaa-bind-03, March 9 2000.
121.[Ryutov 2000c] T. Ryutov and C. Neuman, "Representation and Evaluation of Security Policies for Distributed System Services," In Proceedings of DARPA Information Servability Conference Exposition, Healton Head, South Carolina, 2000.
122.[Saltzer 1974] J. H. Saltzer, "Protection and the Control of Information Sharing in Multics," *Communications of the ACM*, vol. 17(7), pp. 388-402, 1974.
123.[Sandhu 1996] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29(2), pp. 38-47, 1996.
124.[Sandhu 1998a] R. Sandhu and Q. Munawer, "How to Do Discretionary Access Control Using Roles," In Proceedings of ACM Workshop on Role-based Access Control, Fairfax, Virginia, USA, 1998, pp. 47-54.
125.[Sandhu 1998b] R. Sandhu and J. S. Park, "Decentralized User-Role Assignment for Web-based Intranets," In Proceedings of the Third ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1998, pp. 1-12.
126.[Sandhu 1994] R. Sandhu and P. Samarati, "Access Control: Principles and Practice," *IEEE Communications Magazine*, vol. 32(9), pp. 40-48, 1994.
127.[Schiller 1988] J. I. Schiller, S. P. Miller, B. C. Neuman, and J. H. Salzer, "Project Athena Technical Plan - Kerberos Authentication and Authorization System," 1988. 211

128.[Schmidt 1999] D. C. Schmidt, "Dove: A Distributed Object Visualization Environment," *C++ Report*, vol. 11(3), 1999.
129.[Schmidt 1998] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design of the TAO Real-time Object Request Broker," *Computer Communications*, vol. 21(4), 1998.
130.[Simon 1997] R. Simon and M. E. Zurko, "Adage: An Architecture for Distributed Authorization," OSF Research Institute, Cambridge 1997.
131.[Soley 1996] R. M. Soley and C. M. Stone, *Object Management Architecture Guide*, 3 ed. John Wiley & Sons, 1996.
132.[Stevens 1993] W. R. Stevens, *Advanced Programming in the UNIX Environment.* Addison-Wesley, 1993.
133.[Sumner 1999] M. Sumner, "Critical Success Factors in Enterprise Wide Information Management Systems Projects," In Proceedings of ACM SIGCPR Conference on Computer Personnel Research, 1999, pp. 297 - 303.
134.[Sun 2004a] SUN, Java™ IDL, http://www.sun.com,2004.
135.[Sun 2004a] SUN, 基于Java IDL的分布式程序设计, http://www.sun.com,2004.
136.[Tardo 1991] J. J. Tardo and K. Alagappan, "SPX: Global Authentication Using Public Key Certificates," In Proceedings of IEEE Symposium on Research in Security and Privacy, Oakland, California, USA, 1991, pp. 232-244.
137.[Thomas 1994] R. K. Thomas and R. S. Sandhu, "Conceptual Foundations for a Model of Task-based Authorizations," In Proceedings of IEEE Computer Security Foundations Workshop, Franconia, NH, USA, 1994, pp. 66-79.
138.[USA 1996] USA, "Health Insurance Portability and Accountability Act, Public Law 104–191," US Goverment, 1996.
139.[Varadharajan 1998]V. Varadharajan, C. Crall, and J. Pato, "Authorization in Enterprisewide Distributed System: A Practical Design and Application," In Proceedings of 14th Annual Computer Security Applications Conference, 1998.
140.[Walker 1980] B. J. Walker, R. A. Kemmerer, and G. J. Popek, "Specification and Verification of the UCLA Unix Security Kernel," *Communications of the ACM*, vol. 23(2), pp. 118, 1980.
141.[Weiderhold 1992]G. Weiderhold, "Mediators in the Architecture of Future Information Systems: A New Approach," *IEEE Computer*, vol. 25(3), pp. 38-49, 1992. 212
142.[Wilson 1997] W. Wilson and K. Beznosov, "CORBAmed Security White Paper," Object Management Group corbamed/97-11-03, November 1997.
143.[Wong 1997] R. K. Wong, "RBAC Support in Object-Oriented Role Databases," In Proceedings of the Second ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, 1997, pp. 109-120.
144.[Woo 1992] T. Y. C. Woo and S. S. Lam, "Authentication for Distributed Systems," *Computer*, vol. 25(1), pp. 39-52, 1992.
145.[Woo 1993a] T. Y. C. Woo and S. S. Lam, "Authorizations in Distributed Systems: A Formal Approach," In Proceedings of The 13th IEEE Symposium on Research in Security and Privacy, Oakland, CA, USA, 1993, pp. 33-50.
146.[Woo 1993b] T. Y. C. Woo and S. S. Lam, "Authorizations in Distributed Systems: A New Approach," *Journal of Computer Security*, vol. 2(3), pp. 107-136, 1993.
147.[Woo 1993c] T. Y. C. Woo and S. S. Lam, "Designing a Distributed Authorization

Service," University of Texas at Austin, Computer Sciences Department TR93-29, September 1993.

148.[Woo 1993d] T. Y. C. Woo and S. S. Lam, "A Framework for Distributed Authorization," In Proceedings of Conference on Computer and Communications Security, Fairfax, Virginia, USA, 1993, pp. 112-118.

149.[Woo 1998] T. Y. C. Woo and S. S. Lam, "Designing a Distributed Authorization Service," In Proceedings of IEEE INFOCOM, San Francisco, 1998.

150.[Wreder 1998] K. Wreder, K. Beznosov, A. Bramblett, E. Butler, A. D'Empaire, E. Hernandez, E. Navarro, A. Romano, M. Tortolini-Taylor, E. Urzais, and R. Ventura, "Architecting a Computerized Patient Record with Distributed Objects," In Proceedings of Health Information Systems Society Conference, 1998, pp. 149-158.

151.[Wulf 1996] W. A. Wulf, C. Wang, and D. Kienzle, "A New Model of Security for Distributed Systems," In Proceedings of New Security Paradigms Workshop, Lake Arrowhead, CA USA, 1996, pp. 34-43.

152.[Yoder 1997] J. W. Yoder and J. Barcalow, "Archictectural Patterns for Enabling Application Security," In Proceedings of Pattern Languages of Programming, Monticello, Illinois, USA, 1997.

153.[Zachman 1997] J. A. Zachman, "Enterprise Architecture: The Issue of the Century," *Database Programming and Design*, pp. 44-53, 1997. 213

154. [Zurko 1998] M. E. Zurko, R. Simon, and T. Sanfilippo, "A User-Centered, Modular Authorization Service Built on an RBAC Foundation," In Proceedings of Annual Computer Security Applications Conference, Phoenix, Arizona, 1998.

155.[Aba97] Martin Abadi. "Secrecy by typing in security protocols". In TACS'97 (open lecture) LNCS1281, pages611-638 September 1997.

156.[ACS96] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. In Ugo Montanari and Vladimiro Sassone, editors, CONCUR'96 volume 1119 of Lecture Notes in Computer Science, pages 146-162 Springer-Verlag,1996.

157.[AFG98] Martin Abadi, Cedric Fournet, and Georges Gonthier. Secure implementation of channel abstractions. In LICS 98 (Indiana), pages 105-116 IEEE,Computer Society Press, July 1998

158. [AG97] Martin Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zurich, pages 36-47 ACM Press, April 1997

159.[BTS98] Godmar Back, Patrick Tullmann, Leigh Stoller, Wilson C. Hsieh, and Jay Lepreau. Java operating systems: Design and implementation. Technical Report UUCS-98-015 University of Utah Department of Computer Science, August 6,1998.

160.[Gon04] Li Gong. Java security architecture (JDK 1.4) Technical report, JavaSoft,July 2004.

161.[HR98a] Nevin Heintze and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In Proceedings of the 25th POPL, January 1998.

162.[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In Pierre America, editor, Proceedings of ECOOP '91, LNCS512, pages 133-147, July 1991.

163.[IAJR97] Nayeem Islam, Rangachari Anand, Trent Jaeger, and Josyula R. Rao. A flexible security system for using Internet content. IEEE Software, 14(5):52-59 September / October 1997.

164.[Mye99] Andrew C. Myers. Jflow: Practical static information flow control. In Proceedings of the 26th ACM Symposium on Principles of Programming Languages (POPL99), 1999.

165.[Sew97] Peter Sewell. Global local subtyping for a distributed π-calculus. Technical Report 435, University of Cambridge, August 1997. Available from http://www.cl.cam.ac.uk/users/pes20/.

166.[Sew98] Peter Sewell. Global local subtyping and capability inference for a distributed π-calculus. In Proceedings of ICALP'98, LNCS 1443, pages 695-706,1998.

167.[Sew99] Peter Sewell. A brief introduction to applied π, January 1999. Lecture notes for the Mathfit Instructional Meeting on Recent Advances in Semantics and Types for Concurrency: Theory and Practice, July 1998.

168.[VIS96] D. Volpano, C. Irvine, and G. Smith. A sound type system for secure flow analysis. Journal of Computer Security, 4:167-187, May 1996

致 谢

撰写博士论文是一个痛苦而充满思考乐趣的过程。值此停笔之时，我希望能以文字表达我对师长、同学、朋友以及家人感激之情之万一。

首先感谢我的导师宋成研究员。宋老师无论从学术上、生活上都给予我细致入微的关怀。学习上，从课题的研究方向到论文完成，导师精心指导，一丝不苟；生活上导师关心我的困难，给了我无微不至的照顾与帮助，使得我能够顺利毕业。导师以自己渊博的知识、远卓的见识、开阔的思路、深厚的理论功底和奋力进取的精神，给我以深深的启迪和教益。我从导师身上学到的和感受到的不仅仅是知识和学问，更重要的是工作方法、工作精神和做人的道理，这一切无疑都将使我受益终身。

感谢网络中心的阎保平主任、RD室的南凯主任、许海燕、徐浩等几位老师，他们在工作、学业和生活上给予我关怀和指导。感谢我所在的RD室的全体员工，特别是马昭、杨凯、王金一、陈薇等几位同志，为我创造了良好的科研环境。

还要感谢我的同学们，包括龚淑瑶、王东安、秦刚、杨德婷、董科军等同班同学，和你们的讨论让我受益非浅。感谢师兄冯彦军、叶润国、吴宇、张鸿、罗泽、马永征等，谢谢你们的悉心指导。感谢安全组的杨宏伟、李生、毕业、刘艳、李林琳等师弟妹们，和你们在一起生活充满乐趣。也要感谢网络中心的游军玲、赵晖、秦欣等其它师弟师妹们，能和你们结识就是一种缘分。

另外，感谢Nokia研究中心的马建老师，谢谢你给予我获取知识的机会。感谢我有幸在Nokia 认识并成为朋友的袁满、阚志刚、罗军、何锐、谷秭、王岚、苑宏川、求钦龙、赵宁，感谢德国朋友多米尼克，和你们的探讨使我获益良多。

感谢网络中心的佟钊老师，乔建伟老师，计算所研究生处李林老师、宋守礼老师、张晓辉老师以及周世佳老师等给予的关怀和帮助。

感谢我的爱人和我的儿子，你们的爱给我了极大的支持和信心。

感谢所有关心我、支持我和帮助过我的亲人和朋友们，是你们的亲情和友情伴我渡过了博士研究期间这段终生难忘的日子，我当永远铭记！

作者简历

姓名：张方舟， 性别：男， 出生日期：1973.07.04， 籍贯：河南永城

2002.9 -- 2005.12 中科院计算所计算机网络专业博士生

2000.4 -- 2002.7 日本国，东京，Toshiba IT Solution System，FOMA组

1995.9 -- 1998.3 大庆石油学院计算机科学专业硕士生

1991.9 -- 1995.7 大庆石油学院计算机科学专业本科生

【攻读博士学位期间发表的论文】

[1]、3G接入技术中认证鉴权的安全性研究 张方舟 叶润国 冯彦君 宋成 微电子学与计算机

[2]、WPKI技术应用在3G安全体系中的研究与实现 张方舟 谷秭 冯彦君 叶润国 秦刚 “21世纪 计算机科学与技术”第八届研究生学术研讨会

[3]、A Security Scheme of SMS System Zhang Fangzhou APOC2004, SPIE

[4]、一种基于公钥技术的3G安全体系结构及其实现 张方舟 王东安 叶润国 徐 浩 宋 成 计算机工程与应用

[5]、基于J2EE的访问控制决定服务的研究与实现 张方舟 王东安 李生 秦刚 宋成 计算机工程

[6]、J2EE安全机制对RBAC模型的支持的研究 张方舟 王东安 计算机应用研究

[7]、非安全中间件的安全保障机制的研究（已投稿）

[8]、MANET中一种端到端的启发式TCP改进机制 冯彦君 张方舟 叶润国 宋成 王东安 “21世纪 计算机科学与技术”第八届研究生学术研讨会

[9]、无线传感器网络安全协议的研究，王东安，张方舟，秦刚，南凯，阎保平，计算机工程。

[10]、网格计算中基于信任的访问控制研究，王东安，张方舟，南凯，阎保平，计算机应用研究。

【攻读博士学位期间申请专利和著作权】

[1] Nokia 发明专利：Zhang Fangzhou, Liu Qing, etc, “The Secure SMS”，已获NOKIA批准，并正在申请美国专利

[2] 《JSP应用开发技术》 ISBN：7-115-13724-2，人民邮电出版社

【攻读博士学位期间参加的科研项目】

[1] 《国家科技基础条件平台门户应用系统》统一身份认证和资源访问控制系统

[2] 《科技基础条件平台网络规划与建设》的科技基础条件平台网络环境规划方案研究

[3] Nokia 研究中心项目“The Secure SMS”