面向方面和容器安全实现机制的对比

熊峻锋　符云清　吴中福
(重庆大学计算机技术和应用学院　重庆400044)

摘　要　本文主要的目标是通过对比说明:使用面向方面技术来提供软件的安全特性,比现在主流软件体系结构(J2EE、MS.NET)所采用的通过容器来提供软件的安全特性有诸多的优点。为了形象化的说明问题,本文主要结合J2EE体系结构和JBOSS应用服务器[6]来进行讨论。使用AspectJ1.1[3]进行面向方面编程,采用了Java认证和授权服务API(JAAS)[4]。首先,讨论基于组件系统的安全特性。然后,详细地介绍基于容器和基于面向方面的安全特性的实现,并使用虚拟的银行交易系统演示、对比这两种不同的实现机制(使用EJB编写)。

关键词　面向方面,容器,安全

Comparison Study of Aspect-Oriented and Container Implementing Security

XIONG Jun-Feng　FU Yun-Qing　WU Zhong-Fu
(College of Computer Technology and Application,Chongqing University,Chongqing 400044)

Abstract　The major aim of this article is to explain that the security property provided by the AOP has more advantages than that provided by container which is used by main stream software architecture(J2EF.MS.NET). In order to present a vivid explanation,this article mainly illustrates the security property in terms of J2EF architecture and JBOSS application server. Do the aspect-oriented programming by utilizing AspectJ1.1 and also use JAAS. First, discuss the security property on the base of componene system,and then introduce in detail the implement of security property on the base of container and AOP. The process will be illustrated with a virtual bank transaction system, thus pretenting the comparison between the two implementing machnisim(useing EJB).

Keywords　Aop,AspjectJ,J2EE,Container,Security

1　介绍

随着软件工程技术的发展和软件体系结构的出现,基于组件的软件系统已显示出来越来越大的优势,特别是在分布式系统的开发中,取得了巨大的成功。基于组件的系统具有组件功能明确,组件接口定义规范,组件间耦合松散的特点。

由于组件主要是以软件的功能特性为导向来划分的,因此软件的其它特性的实现代码就有可能散布在各个组件中,包括软件的安全特性。这样就造成虽然软件系统的功能特性划分虽然清晰,但是其它特性却杂乱无章。这就不能满足现在的软件的使用者和开发者对软件系统越来越多的特性要求(如使用者的安全性、易维护性、可扩展性等要求;开发者的可复用性等要求)。

1.1　容器提供的安全

在J2EE的环境中,组件的安全主要是由各自的容器来负责的。容器提供了组件所需的常见的安全特性的基础实现,而组件本身不必为了安全特性付出过多代价。但这同时也意味着组件所实现的安全特性仅限于容器所提供的。J2EE为应用程序开发提供了两种形式的基于容器的安全性:说明性的安全性和可编程的安全性。

1.说明性的安全性　其通过安全结构描述的方式来代表应用程序的安全特性需求,安全结构一般包括安全角色,访问控制和验证要求等。在J2EE平台中部署描述符充当了说明的安全性的主要工具。部署描述符是组件开发者和应用程序部署者或应用程序组装者之间的交流工具。应用程序的开发者用它来表示应用中的安全需求,应用程序部署者或应用程序组装者将安全角色与部署环境中的用户和组映射起来。

在程序运行时容器从部署描述符中提取出相应的安全策略,然后容器根据安全策略执行安全验证。说明的安全性不需要开发人员编写任何安全相关的代码,一切都是通过配置部署描述符来完成的。下面举例演示如何添加EJB容器的约束,使只有合法用户且具有相应的角色才能存取组件BankEJB。而组件BankEJB却不须改变或重新编译。

```
〈session〉
    〈ejb-name〉Bank/ejb-name〉
    〈ejb-class〉BankEJB/ejb-class〉
    〈security-role-ref〉
        〈role-name〉Client〈/role-name〉
    〈/security-role-ref〉
〈/session〉
```

图1　应用程序描述文件提供了说明性的安全

2.可编程的安全性　其在说明性的安全性的基础上,使安全敏感的应用可以通过调用被容器提供的API来对安全作出决断。这可以在一定程度上弥补说明性的安全性不足以满足企业的安全模型的情况。这些方法允许组件可以从容器获得安全信息,但是J2EE标准的API使容器提供的信息仅局限于主体和它的安全角色。直接在组件内部实现安全需求不仅费时,而且要求容器商业逻辑提供者要对安全机制、策略有深刻的了解。一些J2EE产品供应商在他们的应用服务器

熊峻锋　硕士研究生,主要研究方向:网络安全;吴中福　博士生导师,主要研究方向:神经网络、网络安全。

中提供了额外的安全功能 但是这又导致了不兼容性和不可重用性。

由于 J2EE 只是一个规范，而各 J2EE 产品供应商的实现并不完全相同，这就导致了容器提供的安全有一个致命的问题，就是不能有效地支持几个应用服务器之间的交互。

1.2 面向方面的安全

安全很难用传统的编程技术来很好地实现。面向对象技术允许分离的需求主要是可以被映射到具体的对象的需求，它很难分离更抽象和更复杂的系统属性，因为这些属性将横跨多个模块。而安全就是具有这样一种特点的系统属性。AOP 却提供了一种新的编程方法学来解决这个问题。AOP 可以把面向对象技术横跨多个模块的系统属性(包括系统的安全属性)分离出来并把它打包到一个独立的模块中，称其为方面。AspectJ 就是对 Java 进行了面向方面的扩展，它使用面向方面的特色加强 Java 语言。在 AspectJ 有许多概念[6]，现只介绍三个主要的概念。

1. 连接点(joinpoint)：程序执行过程中明确定义的点。在 Aspect 中包括以下可用连接点：

1)方法的调用和执行；2)构造器的调用和执行；3)对属性的读/写访问；4)异常处理的执行；5)对象和类的初始化执行。

2. 切入点(pointcut)：用来截获连接点的一个程序构造，一个切入点可以通过使用通配符指定多个连接点。

3. 通知(advise)：当程序运行到指定切入点处要执行的代码；有如下三种执行方式。

1) before：在连接点前面执行。2) after ：在连接点后面执行。可指定是正常返回时执行，还是抛出异常时执行。3) around：包在连接点外面，还可决定是否运行此连接点，并修改该连接点上下文环境。

AspectJ 也允许通过增加新的类成员和修改类之间的继承关系来修改类和它的层次。面向方面可以影响源代码和编译过的类。第二种情况十分重要，如果我们想把面向方面技术应用到已编译、打包的组件上，这种解决方案给我们提供了一种不用修改源代码，就可以增强现有或者添加全新的安全模块的可能性。这样就可以在不影响功能特性的情况下就可以添加安全特性。在使用 AspectJ 给基于 J2EE 的程序添加安全特性时，我们也可以使用容器或外部安全库函数提供的安全功能。下面是一个示例演示了 AspecJ 的基本用法和概念。

```
//一个普通的 JAVA 类
public class HelloWorld{
  public static void say(String msg){
   System.out.println(msg);
  }
  public static void sayToPerson(String msg,String name) {
   System.out.println(name + "," + msg);
  }
}
//一个 AspectJ 中的方面，用以使上述的类礼貌些，让它在打印消//
息前打印"Good day!"，打印消息后打印"Thank you!".
public aspect MannerAspect{
//以下两行定义了一个名为 callSayMsg()的切入点。它会截获对类
//HelloWorld 所有的以"say"开始的公共静态方法的调用，就是上
//面 JAVA 类的两个函数。
  pointcut callSayMsg();
  call(public static void HelloWorld.say*(..));
//下面定义了两个通知，分别在上述定义的切入点前后进行不同的
//打印操作。
  before() : callSayMsg(){
  System.out.println("Good day!");
  }
  after() : callSayMsg(){
  System.out.println("Thank you!");
  }
}
```

图2 一个 AspectJ 的示例

2 安全实现技术的对比

对一个软件系统来说，存在多种安全特性[1~3]。本文主要讨论有代表性(因为认证用容器可以实现，但不完美，而记录和审计是容器所不能实现的)的两种安全特性：

·身份认证：建立和验证用户所宣称的身份的过程。

·记录和审计：把程序或用户对系统的某些操作记录到日志文件中，并通过对日志的分析以期发现是否有违规的操作或潜在的威胁。

下面，我们将展示在基于组件的系统中，如何用容器和 AspectJ 解决上述提到安全问题。

2.1 身份认证

我们使用 JAAS 来进行认证。为了对客户端程序进行认证，我们必须初始化安全管理器：配置登录模块，产生一个登录上下文的实例，然后进行登录[7]。图2显示了使用纯 java 进行认证的过程。在初始化客户端程序时，就进行认证操作。

```
class BankClient{
  LoginContext lc =null;
  public static void main(String [] args){
    //Callback to get username and password. Required by
    //LoginContext
    AppCallbackHandler handler = new
AppCallbackHandler( "xyz","ejf");
      try{
          lc = new LoginContext("Bank",handler);
          lc.login();
      }catch( LoginException e ){
      }
      BankHome homeBank = (BankHome) ctx.lookup ( " ejb/
      Bank");
      Bank bank = homeBank.create();
      System.out.println(bank.getAccountInfo("bill"));
      try {
         lc.logout();
      }catch(LoginException e){
    }
  }
}
```

图3 使用纯 java 对客户端程序进行认证

下面的代码是通过使用 AspectJ，实现类似的安全特性

```
//普通开发人员书写功能代码
class BankClient{
   public static void main(String [] args) {
       BankHome homeBank = (BankHome)
ctx.lookup( "ejb/Bank" );
       Bank bank = homeBank.create();
       System.out.println( bank.getAccountInfo("bill" ) );
   }
}
//安全专家书写登录代码
aspect BankAspect{
   LoginContext lc = null
   pointcut mainExecution():
       execution(public static void main( .. ));
   //运行 main()函数前进行登录操作
   before(): mainExecution(){
       AppCallbackHandler handler=new
AppCallbackHandler("xyz","ejf");
       try {
          lc = new LoginContext( "Bank", handler );
          lc.login();
       }catch(LoginException e){
       }
   }
   //运行 main()函数后进行注销操作
   after() returning:mainExecution(){
       try {
          lc.logout();
       }catch(LoginException e){
       }
   }
}
```

图4 使用 AspectJ 对客户端程序进行认证

程序提供的认证机制部分被置于方面中。这样实现认证

的代码就与其它代码分离开来。

2.2 记录和审计

记录和审计服务可以提供收集和分析信息系统的活动情况。使用它们的主要目的就是找到系统潜在的软件漏洞并发现原因。虽然记录和审计是安全中很重要的一环，但目前为止J2EE却没有提供标准的解决方案。目前通常的做法是，把程序的某些操作记录在日志文件中而一般的系统都不提供审计功能。虽然在J2EE中可以用编程性的安全来解决这个问题，但这引出了如下两个问题：1)要修改组件的源代码，这将极大地破坏组件的结构。记录和审计属于这种关注点的范畴，即可以很好地通过面向方面来实现。下面的程序清单演示了记录是怎样通过面向方面技术实现的。如果某个方法抛出给定的任何异常，就将这种情况记入日志。

```
aspect BankAspect{
    pointcut bankMethods();
    execution(public * bank.*(..)) && this(SessionBean);
    after() throwing (BankSecurityException e);
bankMethods(){
        Log log1 = Log.getInstance();
        log1.write(e);
    }
}
```

图5 使用AspectJ实现记录功能

审计功能也可以通过类似的方式实现，只要把对程序执行操作进行记录改为进行审计即可。下面就对容器和AOP实现安全特性的方法进行一个简要的对比：

表1 容器管理与AOP在实现安全方面的对比

	容器管理的安全		面向方面的安全
安全机制	说明性的安全性	可编程的安全性	
使用方式	被动式	宣告式	被动式
身份认证	局限于容器提供的功能	局限于容器提供的功能，如果使用第三方提供的服务则降低了通用性。	不用修改组件代码
记录和审计	不支持	需要修改组件代码	
结论	只能实现固定的几种安全特性。	宣告式并不能完全的解决模块分离问题。	因为不使用第三方服务，所以不存在兼容性，但各种特性的实现机制没有行业规范，所以通用性不好

注：使用方式说明和对比

使用方式	释义	优缺点
被动式	服务提供者在服务使用者不知情的情况下把服务强加给服务使用者	1. 可以让使用者完全不必关注与己无关的事情。 2. 有些服务与使用者关系密切，不能使用这种方式。
宣告式	服务使用者必须显示的调用服务提供者提供的服务	2. 使用者对服务如果不甚了解，可能会对同类的多种服务无从选择。 3. 如果要使用的服务有很多种，会造成使用者内部代码混乱，并可能忘记调用某些服务。

结论 在安全体系结构的建立方面，容器和AOP各有利弊。但是在实现第二类需求时，容器却无能为力了，而AOP可以较好地完成任务。容器所提供安全实现机制可以让开发人员不必介入安全方面的事务，而把它留给安全专家来进行处理，应用程序安全属性是在应用程序的描述文件中进行定义的。另一方面我们只能利用容器提供的有限的安全特性，它不具备足够的灵活性来实现更复杂和更具动态性的安全策略。我们也可以使用基于程序的安全实现方法，但是这却导致了安全逻辑和功能逻辑互相交叉不能很好分离。最后形成的代码就极度混乱并且难以维护。而使用AspectJ解决安全问题却具有更加灵活和可扩展的特性，这就可以产生独立的并具有灵活性的安全模块并且可以把它很方便地置入未考虑安全的应用程序中。这种方法就让应用程序不必为了引入安全特性而修改源代码。综上所述，目前为止最为完美的解决方案就是把二者进行有机的结合。

参考文献

1 Security Functionality Requirements. National Institute of Standards and Technology, 1992 97. dustry, tark. Integrate security infrastructures with JbossSX, JavaWorld, April 2001
2 Information Technology Security Evaluation Criteria (ITSEC). Department of Trade and InLondon 1991
3 http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/aspectj-home
4 王妍. J2EE中的安全 [EB/OL]. http://www-900.ibm.com/developerWorks/cn/java/l-j2eeSecurity/index.shtml
5 Stark S. Integrate security infrastructures with JbossSX, JavaWorld, April 2001
6 JAAS page at Sun web site. http://java.sun.com/products/jaas
7 Resource Access Decision Facility Specification, OMG, 2001. http:omg.org/docs/ formal/01-04-01.pdf
8 徐迎晓. Java安全性编程实例[M]北京. 清华大学出版社

(上接第198页)

参考文献

1 The emerging Technologies That Will Change the World. MIT Technology Review, Jan./Feb. 2001 issue
2 Workshop on "Aspect-oriented Modeling with UML". http://lglwww.epfl.ch/workshops/aosd-uml/
3 Suzuki J, Yamamoto Y. Extending UML with Aspects: Aspect Support in the Design Phase. In: AOP Workshop at ECOOP'99, Portugal, 1999
4 Clarke S, Walker RJ. Composition Patterns: An Approach to Designing Reusable Aspects. In: Proc. of ICSE, 2001
5 Aldawud O, Bader A, Elrad T. Weaving with statecharts. http://www2.umassd.edu/swsoc/workshops/
6 Lions J M, Simoneau D, Pitette G. Extending OpenTool/ UML Using Metamodeling: An Aspect Oriented Programming Case Study. In: Second Intl. Workshop on AOM, 2002
7 Aldawud O, Elrad T, Bader A. A UML Profile for Aspect Oriented Modeling. In: OOPSLA 2001 workshop on AOP
8 OMG. Unified Modeling Language Specification. Version 1.3, 1999
9 Rational Rose 2000e, Rose Extensibility User's Guide. Copyright (c) 1998-2000 Rational Software Corporation
10 Laddad R. I want my AOP. http://www.javaworld.com/ javaworld/jw-01-2002/. 2002