第六章 操作系统与访问控制

在学习了最主要的反入侵技术之后，接下来在第 6 章和第 7 章集中阐述访问控制问题。从本质上讲，访问控制所要解决的核心问题是抑制对计算机系统中的资源对象的非法存取与访问，而安全策略则是实现访问控制的依据，因此对两者的讨论密不可分。在目前分布式系统和移动代码(ActiveX、JavaApplet、Java 的远程加载类代码等)广泛应用的情况下，访问控制与安全策略问题又被进一步延伸到网络环境中并且更为复杂。我们首先在这一章概要讨论操作系统中的本地访问控制以及针对网络移动代码的访问控制机制，包括常用的安全策略模型及体系结构，下一章继续讨论分布式应用系统中更为复杂的安全策略与访问控制问题。

6.1 概 念

访问控制所要解决的根本问题，也就是安全策略要达到的目的，用一句话来表达就是：限定系统中的主体仅能以明确授权的方式访问特定的对象。这里主体的涵义包括用户、线程或进程；权限的涵义包括读、写、删除、执行、输入、输出、启动、终止等等，随着所作用于其上的对象不同而不同。对象的例子包括文件系统、目录、文件、I/O 流、内存区域、socket、管道、全局同步对象(互斥锁、信号量、事件/信号)、线程/进程(例如进程通讯时)、动态库/库函数等等。

系统的访问控制机制必须满足以下性质：

强制性(*mandatory*)，即访问控制机制不可被绕过；

最小授权原则(*least-priviledged*)：主体在任何时刻仅被授予完成当前任务所必须的最小权限；

最小范围原则(*need-to-konw*)：主体在任何时刻仅被允许访问完成当前任务所必须访问的那些对象。

当然，就现实的系统而论，这些理想的准则未必总能圆满实现，甚至在具体应用中有互相矛盾的可能，这就需要在实际软件实现时合理权衡折衷，但这一章不沿这一方向深入讨论。

前面已经指出，安全策略是在计算机系统中实现访问控制的依据，虽然具体表达形式随应用问题而不同，但都可以抽象为一个统一的概念模型，这就是访问控制矩阵(*Access*

Matrix)。在形式上，访问控制矩阵是一个 2-维表，行表示主体，列表示对象，矩阵元为访问权限，如下面这个例子，它表达系统中的三个主体 D1、D2 和 D3 对四个对象 O1～O4 所被赋予的访问权限，其中 D1 对 O1 和 O3 具有(且仅具有)读权限，对 O2 和 O4 没有任何访问权限；D2 对 O1 具有读和写权限但除此而外没有其他权限，对 O2 仅有写权限，对 O3 具有读和打印输出权限但除此而外没有任何其它权限，对 O4 仅具有执行权限。读者不妨自己解释 D3 对 O1 到 O4 所具有的权限。这样一个系统的访问控制就是要保证 D1～D3 对 O1～O4 的访问严格遵循以上策略或规则。

	O1	**O2**	**O3**	**O4**
D1	read		read	
D2	read,write	write	read,print	execute
D3		read, write	read	read,execute

下面的访问控制矩阵是一个更复杂的例子，其中的 D1～D3 不再表示单一一个主体，而是一组以相同的权限履行访问功能的主体的集合，称做主体域或简称域。权限“transition”表示允许一个域中的主体迁移到另一个域，例如表的第一行表明域 D1 中的任何主体可以迁移到域 D2，即以 D2 被赋予的权限实施对系统对象的访问，但不能迁移到域 D3。

	O1	**O2**	**O3**	**D1**	**D2**	**D3**
D1	read		read		transition	
D2	read,write	write	read,print			transition
D3		read, write	read	transition	transition	

6.2 访问控制策略的实例

安全策略的访问控制矩阵模型统一而直观，但针对实际应用则未必总能最有效率。根据具体应用的特点，已经从访问控制矩阵模型派生出了一系列更具体的策略模型，这里列举几个最常用的实例。

第一类例子是基于域的策略。这里域的概念和上面关于域的概念类似，系统中的每个主

体都属于特定的保护域(可以属于多个域)；域是一组规则的集合，每个规则表达对某个对象所允许的一组访问权限；当一个主体属于某个域时，该主体仅能以该域中的规则所限定的方式访问特定的对象。例如，下面是一个有 3 个保护域 D_1～D_3 的系统，以域 D_1 为例，凡属于该域的主体都对对象 O1 具有读/写/执行权限、对 O2 具有写/执行权限、对 O3 具有读/删除权限，但除此而外没有任何其它权限。对属于域 D_2 和 D_3 的主体，它们对 O2 都具有写权限、对 O4 都具有执行权限，但对 O1 和 O3 则具有不同的权限。

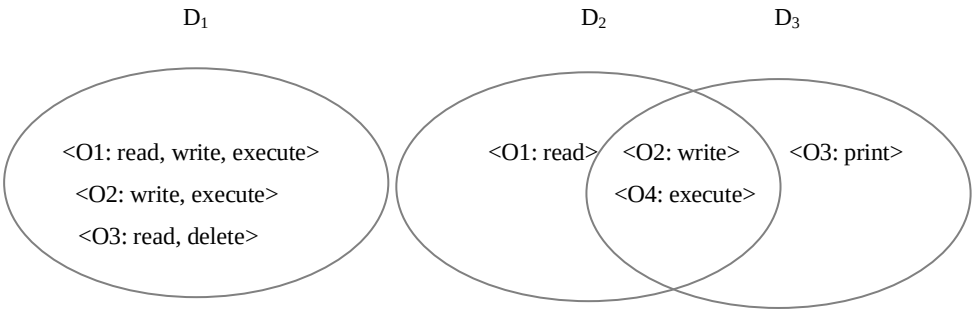


图 6-1 基于域的策略模型的例子

基于域的策略模型的一个最典型的访问控制机制就是经典的 *Unix/Linux* 文件系统的访问控制。在那里主体有两类：用户和用户组，每个用户(组)确定一个保护域。主体对文件对象的访问权限由文件的存取属性表达，存取属性永远和文件对象关联在一起，它指明文件的创建者(用 owner-id 标识，这总是一个用户)及创建者的权限(读/写/执行，分别以 r-w-x 三位为 1 来表达)、所允许的用户组(group-id 表达)及指派给组成员用户的权限 r-w-x、其它(不属于该组的)用户(用 others 泛指)的访问权限 r-w-x。

第二类例子是基于域-型的策略模型及其访问控制机制(*Domain-Type Enforcement*，简称 *DTE*)。这里域(*Domain*)是进程的集合，每个进程在任一时刻隶属于且仅属于一个域，但在指定的条件下进程可以从一个域迁移到另一域；型(*Type*)是系统对象的集合，例如文件等；常用的访问权限有[1]{read, write, execute, create, directory-descend, send-signal, transition}。在 DTE 策略模型中，当域 A 中的一个进程 P_1 执行一个程序文件 P_2 且 P_2 被指定为域 B 的入口点时，进程 P_1 就由 A 迁移(transition)到域 B。下面是 DTE 策略的一个例子[2]，粗体为关键字。

[1] 这些权限以 UNIX/LINX 上的实现为例，象 send-signal 这样的权限显然不适用于 WINDOWS。
[2] 这里选取的是一个 LINUX 系统中的例子。不同的系统有不同的表达形式，但概念大同而小异，在此读者根据给出的注释(#后面的文字)就足以理解这些脚本配置语言的涵义。

```
types    root_t    log_t    #型声明
domains    common_d    log_d    #域声明
default_rtype root_t    #  文件系统根目录的型
default_domain common_d #进程树根(init: pid=1)所属的域
  #  访问控制策略：域  (入口点) (对型的访问权限) (对域的访问权限) (对信号的访问权限)
spec_domain    log_d (/sbin/syslog) (rdx->root_t rwxcd->log_t) () ()
                                        #以上策略项中对域和信号的访问权限为空
spec_domain common_d () (rwxcd->root_t r->log_t) (auto->log_d) ()
          # auto->log_d 表示每当 execve 被调用时，检查被执行的程序是否为域 log_d
           的入口点，如果是则当前进程从原来的域迁入域 log_d

assign –r /var/adm/log log_t #对目录/var/adm/log 中的所有文件都指派其具有型 log_t
```

以上策略中定义了两个型,即两个对象集合,分别明明为 root_t 和 log_t，其中 root_t 包含系统根目录下的全体文件对象。系统还定义了两个域，分别命名为 common_d 和 log_d，其中 common_d 包含所有本地进程 init[3]的子进程作为成员，log_d 开始则仅包含一个进程成员/sbin/syslog[4]。log_d 对 root_t 中的成员具有读(r)/删除(d)/执行(x)三种访问权限、对 log_t 中的成员对象具有读(r)/写(w)/修改(c)/执行(x)/删除(d)五种访问权限，读者还可以类似解释该策略所定义的域 common_d 对 root_t 和 log_t 中的成员对象的访问权限。

上面这些例子都是针对系统层次的安全问题，但实际上访问控制策略也是许多复杂应用系统的核心，例如下面这个著名的中国长城(*Chinese-Wall*)策略就是一类十分现实的例子，它源出于金融交易。考虑一组上市公司，它们被划分为一些集团，每个公司仅属于一个集团，同一集团中的各公司之间的利益是冲突的。系统的主体，即金融分析家，可以查询任何对象(即公司)的财务信息，但必须受以下策略的约束：

一旦一个主体访问了某家公司的财务信息，他/她将不再能访问该公司所属集团中的任何其它公司的财务信息。

另一类复杂的安全策略是所谓竞标审计(*Sealed-Bid Auction*)策略，它也同样出于实际应用，是对诸如投标、拍卖等活动规则的概括，基本规则如下：

任何投标都限制于一个特定的时间区间内，否则无效；

投标一旦提交则不可抵赖；

中标方一定是出价最高的一方；

投标方的名字不得泄露给任何一方，包括审计方；

[3] 这是 UNIX/LINUX 上的一个用户模式的进程，进程标识号 PID 永远为 1，它在操作系统内核完成初始化之后、但在启动任何其它用户模式的进程之前被(自动)启动。

[4] 这也是 UNIX/LINUX 上的一个用户模式的守护进程(daemon)，用以写日志消息。

中标方的帐户及资金必须可核实。

与前面两类策略不同，后两类策略的复杂程度超出了访问控制矩阵的能力，需要更复杂的模型，我们将在下一章详细讨论。

6.3 安全操作系统的通用架构: Flask 体系结构

安全操作系统与访问控制联系紧密，后者是实现前者安全性的最主要的机制。另一方面，有很多网络安全机制实际上与操作系统的安全机制关系紧密，例如第 3 部分将要重点阐述的网络安全协议全部都需要实现为操作系统内核模式或用户模式的可信任的组件，下一节专门讨论的针对移动代码的安全访问机制本质上也需要实现为操作系统上的一个可信任的执行环境，甚至有很多早期的网络安全机制目前已完全演变成为操作系统内核安全机制的一部分，因此这一节专门对安全操作系统进行一些概要的阐述。

所谓安全操作系统指的是可信任的操作系统，而非具有一定安全功能的普遍使用的商用普通操作系统如 Windows、Linux、FreeBSD、OpenBSD、Solaris、AIX、HP-UX 等。安全操作系统是针对严格的安全规范和需求专门开发的一类操作系统，这类系统在编程接口(API)层次上与现行普通操作系统基本一致(这样的目的是为了使软件移植的代价最小，但实际上很难做到完全一致)，但其内核事务的处理机制则变动较大，其安全程度也需要经历严格的测试和评估。这类系统与普通系统比较还有一个重大差异，就是普通系统在执行安全机制时所依赖的信息或属性很少，例如 Unix/Linux 或 Windows 在履行对文件对象的访问控制时仅仅依据用户(主体)的身份标识和文件对象类型与标识来决定授权与否，而安全操作系统在履行访问控制功能时所依据的信息和属性往往复杂得多。在实践中，目前几乎所有主流操作系统的开发商都在其普通操作系统之上通过对内核的安全性增强来实现这类系统，典型的如 TrustedBSD、TrustedSolaris 等。这一节阐述构造这类安全操作系统的一种经过实践验证的良好的统一架构，即 Flask 体系结构。

从本质上说，安全操作系统需要做的就是强制性地保证该系统范围内的一切活动都严格符合给定的安全策略或规则，而访问控制机制是实现安全操作系统的最重要的技术途径。除了访问控制机制必须满足第 6.1 节的基本原则之外，安全操作系统还需要满足通用性或称策略独立性原则，即安全操作系统的具体构造不依赖于任何特定的安全策略，而是容许(用户根据自身需要)实施任意类型的安全策略及相应的访问控制机制。通用性原则的缘由不言而喻：操作系统作为一个通用环境必须满足绝大多数计算事务的安全需要，而不能束缚在任

何单一的安全策略上。换句话说，一个安全操作系统应该是一个安全框架，在这一框架内可以根据实际需要灵活地嵌入使用者配置的安全策略及相应的访问控制。于是问题归结为：应该如何构造这一能够适应任何安全要求的“万能”系统？

Flask 体系结构是构造通用的安全操作系统的一种经过实践验证的良好的统一架构，其原理表示在图 6-2 之中。

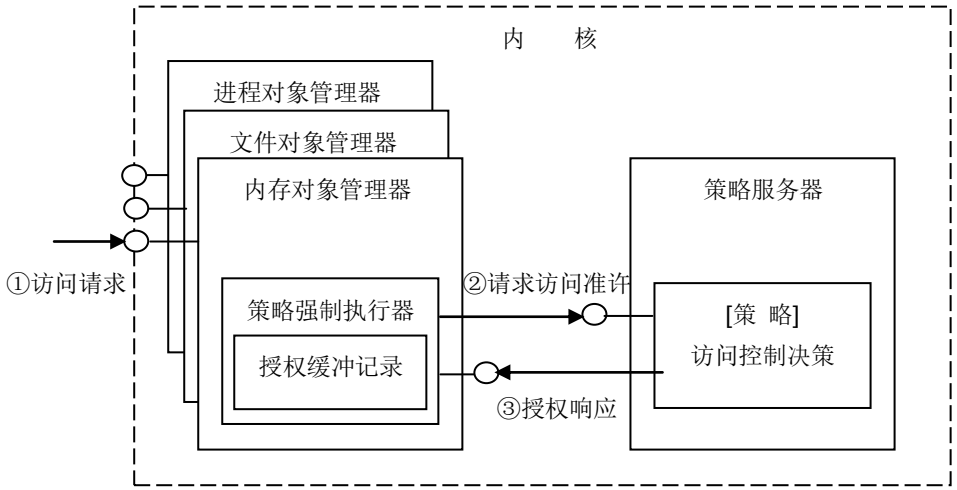


图 6-2 *Flask* 体系结构的客户/服务器模型

回顾在操作系统课程中学习过的系统内核，与这类内核比较，基于 Flask 体系结构的安全操作系统多出一类组件，这就是策略服务器[5]，以及加入了安全性代码的对象管理器。

每个策略服务器执行特定的安全策略，如果系统需要同时实施多个安全策略，甚至完全不同类型的安全策略[6]，则内核可以同时加载和运行多个策略服务器，但从本质上讲，整个系统只需要一个策略服务器就可以工作。如果系统在运行期间需要改变安全策略，只要替换策略服务器组件即可，系统其它部分保持不变。为此，策略服务器可以实现为可信任的共享库或动态链接库。

读者还记得，当代操作系统内核结构都遵循面向对象原则，相应地存在一批各种类型的对象管理器，负责创建、撤消/回收、读/写存取相应的对象，例如图 6-2 中出现的进程对象管理器、文件对象管理器、内存对象管理器等，正是通过请求或调用这些对象管理器上的接口函数，用户模式的进程或其它内核线程才得以访问特定类型的操作系统资源。Flask 体系结构要求对这些对象管理器加入了安全性代码。如图 6-2 所示(其中数字表示调用顺序)，

[5] 英文原文是 security server, 这里按照其涵义意译成策略服务器。
[6] 这里不深入讨论保证多策略一致性的问题, 多数情况下这是一个较复杂的问题, 但与 Flask 体系结构本身无关。

当接收到一个外部访问请求①时，对象管理器除了完成在普通操作系统中需要履行的所有功能之外，一个重要的补充功能是根据当前的请求的上下文访问策略服务器，即图中对策略服务器接口的调用②，其中传递给策略服务器的参数包括请求方的安全标识号、被请求对象的安全标识号、被请求对象的类型和被请求的访问权限，最后这一参数是一个位矢量，对每种类型的对象 Flask 模型都规定了一组特定访问权限，例如对进程(或线程)对象规定的权限包括执行、改变宿主身份、发送信号、创建新进程、跟踪另一个进程、改变进程优先级等；策略服务器接收到授权请求②后，根据其策略(在逻辑上系统的每个安全策略都由一个特定的策略服务器执行，但在实现时可以由一个策略服务器履行多个策略)决定是否应该对当前的访问请求予以授权，并将决策结果返回对象管理器，即图 6-2 中的消息③；最后，对象管理器根据策略服务器的授权结果决定如何完成访问请求①或完全拒绝该访问请求。

图 6-2 中各个对象管理器的接口可以理解为熟知的系统调用函数，例如标准的 POSIX API。策略服务器的接口函数如下，其中的数据类型由 Flask 模型及其具体实现决定，这里就不再继续深入了：

```
int security_compute_av(
        security_id_t    ssid; /*请求方的安全标识号*/
        security_id_t    tsid; /*被请求对象的安全标识号*/
        security_class_t    tclass; /*被请求对象的类型*/
        access_vector_t    requested; /*表达被请求的访问权限的位矢量*/
        access_vector_t*    allowed; /*表达被授权的位矢量*/
        ……
);
```

因为对象管理器是访问操作系统资源对象的必经之地，因此加入了安全性代码的对象管理器形象地说就成为安全操作系统中安全策略的强制执行点，Flask 模型正是据此保证安全策略不可被绕过，同时使安全扩展导致的功能代码增加得最少。另一方面，注意对象管理器如何强制访问控制与将要被强制履行的安全策略本身无关，如何履行安全策略，即如何根据当前上下文决定授权完全在策略服务器内部完成，即使替换另一个策略服务器，图 6-2 仍然有效，安全增强的对象管理器和策略服务器之间仅仅通过预先定义的、与策略无关的接口相互作用。因此，Flask 体系结构巧妙地利用面向对象原理实现了策略独立性。

细心的读者回想到，如果每次访问对象都需要经过策略服务器明确授权，则系统的性能会由此下降，特别对那些复杂的授权决策计算更是这样。Flask 解决这一问题的方法简单而通俗，就是在每个对象管理器中建立授权缓冲记录，将策略服务器返回的授权矢量做有限寿命缓存，使得同一主体针对同一对象的同类访问不必每次经由策略服务器计算授权，而是

一次请求/授权、多次有效。

目前基于 Flask 体系结构实现的安全操作系统实例包括 *Linux* 对 Flask 的实现即 SELinux/SLM(Secure Linux Module)，以及 *BSD/Unix* 对 Flask 的实现即 BSD Flask，具体的技术细节可以参考章末所列举的参考文献。

6.4 针对移动代码的安全机制：*Java*2 访问控制

在分布式系统中，可移动代码(ActiveX、JavaApplet、Java 的远程加载类等)带来一类新的访问控制问题，这一节以 Java 为例概要阐述针对这类问题的解决方案。

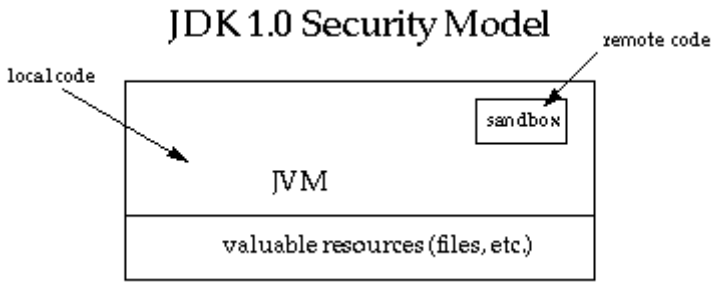
作为一种针对分布式系统的面向对象式编程语言，Java 程序由类和类的实例即对象组成，特别是 Java 类的代码可以驻留于任何机器上而并不限定一定要事先存在于运行机器中，这意味着某些 Java 类的代码将在运行期间从远程迁移到本地机器、然后在本地虚拟机 JVM 的环境中运行。如何验证这些远程类是否可信任，并且保证它们以符合本地安全策略的方式访问本地资源对象，是 Java 安全机制必须解决的核心问题。

图 6-3 是 Java 安全模型的演变历程，从早期的沙箱模型直到最近的基于策略的模型。*a*)中的沙箱模型(*sandbox*)的思想实际上是从安全角度将 Java 类分为两大类：代码存储于本地的类和代码从远程迁移而来的类，本地虚拟机 JVM 对前者完全信任而对后者完全不信任，因此对后者的运行、特别是后者能够访问的本地对象加以严格的监控和限制，这种监控与限制的机制就是沙箱。*b*)中的 Java/JDK 1.1 安全模型进一步扩展了 Java/JDK 1.0 的沙箱模型，引进了所谓签字类(*signed applet*)，这实际上是 JVM 所信任的某些实体对 Java 类代码的数字签名，以 JAR(*Java Archive*)格式伴随类代码一起在 JVM 之间迁移。一旦 JVM 验证了类的数字签名，则把该类作为完全可信任的类与本地可信任的类完全同等看待，而那些没有数字签名或 JVM 不信任其签字方的远程类则依照沙箱模型在运行期间加以控制。

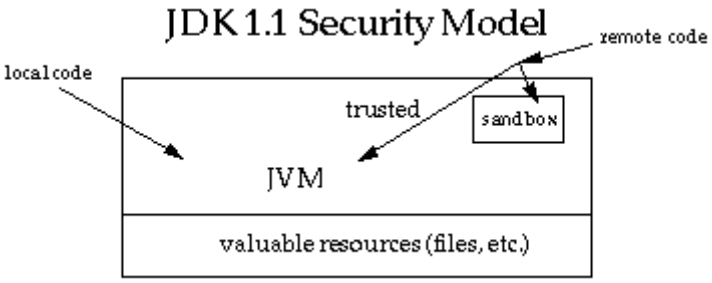
Java 2 的安全模型超越以上这种内置可信任类的做法，类加载器(*classloader*)在调入类代码时验证类的安全属性，接下来 JVM 完全依照本地安全策略决定如何对各个类代码的运行实施访问控制。Java 2 的安全模型可以集成各种安全策略，非常灵活而通用。

Java 2 安全模型的一个重要概念是保护域，所谓保护域是这样一组 Java 对象的集合，它们在相同的安全策略之下接收访问控制。Java 对象至少可以划分为两个保护域：应用保护域和系统保护域，后者包含典型的系统资源对象如文件系统、通讯端口、输入/输出设备等。保护域的具体划分是 Java 安全策略的一部分内容，类加载器在创建或启动一个(无论本

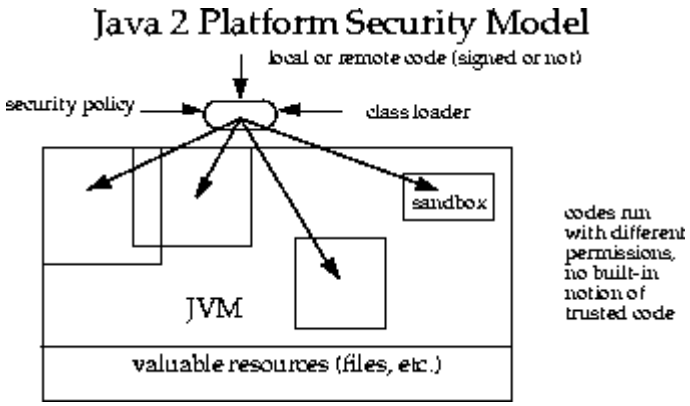
地或远程)对象时将该对象放入特定的保护域，JVM 据此在运行期间建立一个从 Java 类/对象到其所属的保护域、再从保护域到该域上的允许访问权限的映射关系。当域中的任何对象被访问时[7]，JVM 请求 Java 安全管理器(*security manager*)执行安全策略以决定是否授权。细心的读者甚至可能注意到，这一过程在逻辑上与图 6-2 的 *Flask* 模型有相似之处。



a）Java/DK 1.0 安全模型



b）Java/JDK 1.1 安全模型



c）Java 2 安全模型

图 6-3 Java 安全模型的演变

Java 2 安全模型的第二个重要概念是访问权限，这是一个抽象类 **java.security.Permission**，用来使程序设计者(作为其继承类)定义所需要的任何访问权限并把该权限关联

[7] 根据面向对象的概念，精确地说就是对象上的某个方法被调用。

到特定的对象。例如

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```

创建了一个针对名为"/tmp/abc"的特定文件对象的“只读”权限。这里的 java.io.FilePermission 是 Java 2 内置的、java.security.Permission 的继承类，属于程序包 java.io。java.security.Permission 的多数继承类都象 java.io.FilePermission 这样属于某个特定的程序包，这些重要的继承类有 java.net.SocketPermission、java.lang.RuntimePermission、java.util.PropertyPermission、java.awt.AWTPermission、java.net.NetPermission 等。

创建权限的进一步的例子还有：

```
import java.io.FilePermission;
FilePermission p = new FilePermission("catfile", "read,write");
FilePermission p = new FilePermission("/tmp/cattmp", "read,delete");
FilePermission p = new FilePermission("/bin/*", "execute");
FilePermission p = new FilePermission("*", "read"); /*对当前目录下的所有文件关联只读权限*/
FilePermission p = new FilePermission("<<ALL FILES>>", "read");
                    /*对文件系统中的所有文件关联只读权限*/
import java.net.SocketPermission;
SocketPermission p = new SocketPermission("java.sun.com","accept");
p = new SocketPermission("204.160.241.99","accept");
p = new SocketPermission("*.com","connect");
p = new SocketPermission("*.sun.com:80","accept");
p = new SocketPermission("*.sun.com:-1023","accept");
p = new SocketPermission("*.sun.com:1024-","connect");
```

在所建立的每个权限类上必须实现的最重要的方法之一是 implies，“a implies b”在 Java 2 安全模型中的涵义是“持有授权 a 则自然地持有授权 b”。

Java 2 安全模型的第三个重要概念是安全策略，这也是实现为抽象类 **java.security.Policy** 的继承类。策略配置文件具体表达对一组特定主体的授权，下面是一些典型的例子：

```
grant signedBy "Roland"{ permission a.b.Foo; }; /*对主体"Roland"签字的所有类代码授权 a.b.Foo */
grant signedBy "Roland, Li" {
        permission java.io.FilePermission "/tmp/*", "read";
        permission java.util.PropertyPermission "user.*", “read,write”;
}; /*对主体"Roland"或"Li"签字的所有类代码授权只读访问"/tmp/下的所有文件对象、
   且读/写访问所有名字形如"user.*"的文件对象。*/
```

关于策略配置文件的详细格式见章末列举的参考文献，这里就不再继续深入讨论了。

Java 2 安全模型的第四个重要概念是安全管理器，这是一个 SecurityManager 类的对象，当任何 Java 对象被访问时，JVM 调用 SecurityManager 的方法 checkPermission 执行安全策

略以获取相应的授权，例如：

```
SecurityManager security = System.getSecurityManager();
if (security != null) {
        FilePermission perm = new FilePermission("path/file", "read");
        security.checkPermission(perm);
}
```

SecurityManager::checkPermission 遍历当前调用者的堆栈，检查调用链上的每个方法所属的类和这些类所属的保护域。每个保护域在当前所访问的对象上的权限由策略所定义，安全管理器的授权决策规则之一是：如果当前所请求的权限落在当前调用链上伴随的所有这些权限之中，即属于这些权限的交集，则允许授权。

最后，类加载器也是 Java 2 安全模型中的重要环节。类加载器负责在线加载 Java 类代码到本地 JVM 中运行，它一般是以 C 语言编写的一个本地操作系统组件，从本地文件系统启动。实际上存在几种不同类型的加载器，例如扩展类加载器加载程序包中的类，应用程序加载器加载用户定义的类。加载器读取 Java 类的字节码文件，将加载的 Java 类关联到特定的保护域。加载器还为其加载的类提供唯一的命名空间。

6.5 小结与进一步学习的指南

访问控制与安全策略是计算机安全领域永恒的主题，至今已经发展成为内容非常丰富并且应用广阔的领域，特别是对随着网络应用而出现的当代大规模分布式计算系统，其中的相关问题一直以来都是研究的丰富源泉。相对而言，我们在本章和下一章仅就其中很专门的一部分进行了概要讨论，本章重点讨论的是安全操作系统的通用架构和 Java 平台对移动代码安全问题的解决方案，其中 Flask 体系结构以面向对象的风格建立起一个能集成任何安全策略的安全操作系统的通用架构，而 Java 平台也是基于面向对象技术解决远程类代码对本地资源对象的安全访问控制问题。

关于第 6.1-6.2 节所列举的传统的访问控制技术有很多教科书予以讨论，甚至很多阐述 Unix/Linux 及少量阐述 Windows/Win32 API 编程的书籍也详细讨论了这些平台上的访问控制规则，特别是文件系统的访问控制规则。注意这些普通平台上的访问控制属于所谓随意性访问控制(*DAC: Discretionary Access Control*)，与真正强制性的访问控制机制(*MAC: Mandatory Access Control*)有很大差别，后者目前只有在专用的安全操作系统上才给予了精确的实现，

有大量各种不同类型的安全策略，其中著名的 *DTE* 策略模型由下面这篇论文建立：

S.Hallyn P.Kearns *Domain and Types Enforcement in Linux*, Proc. 4th Ann. Linux Conf. 2000.

其它类型的策略如著名的 RABC 策略、基于能力的策略等因为并非与网络安全特别有关，本章没有讨论，感兴趣的读者可以参考 M.Bishop 所著的非常全面的教科书 *Computer Security: The Art and Science*, Addison-Wesley Inc. 2002，其影印版由清华大学出版社出版。

安全操作系统是计算机安全领域的一个重要分支，对此可以写出一整本书加以阐述。*Flask* 体系结构由美国国家安全局(NSA)委托商业公司 SCC(Secure Computing Cooperation)设计和实现，详细讨论可以参考开发方的技术报告：R.Spencer et al *The Flask Security Architecture: System Support for Diverse Security Policies*, Proc. 8th USENIX Security Symposium, 1999:123-139 及总结报告 P.Loscocco, S.Smalley *Integrating Flexibale Suport for Security Policies into Linux Operating System*, Proc. USENIX Annual Tech. Conf, 2001。在 NSA 的网站上还可以下载基于 *Flask* 实现的安全 Linux(称为 SELinux)源代码。以类似于 *Flask* 的体系结构所实现的安全操作系统还有著名的 TrustedBSD，它是 FreeBDS 的安全版本，详细技术可以研读其开发者的论文 R.Watson et al. *Design and Implementation of the TrustedBSD MAC Framework*, Symposium on UNIX Security, 2003。注意这篇论文在引言部分详细对比讨论和分析了各种典型的实现安全操作系统的技术途径，值得一读。TrustedBDS 后来进一步被改进为基于 Flask 架构的安全操作系统：

R.Watson *Security Enhanced BSD*, Tech. Report, NAL, July 2003.

第一个具有比较全面的安全功能(但仍然不属于安全操作系统)的 Windows 系统是 Windows 2000 Server，特别是在它上面集成了带访问控制的目录服务系统 ADS，使它成为安全的分布式应用的比较理想的平台。

关于 Java/J2SE 的内在安全体系结构最详细而清晰的阐述见 Java 技术的权威网站 http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-specTOC.fm.html 或者该技术的主要开发者 L.Gong 撰写的名著 *Inside Java2 Platform Security*, Addison-Wesley Reading, 1999。.NET 也存在类似的针对移动代码的安全问题，完整的讨论可以参考 La Macchia 和 A.Brian 的著作 *.NET Framework Security*, Addison-Wesley Professional, 2002。

关于安全系统有一个著名的安全内核或称访问监控器的理论，6.3 节的 *Flask* 模型和 6.4 *Java* 2 安全模型都可以看做这一理论的特殊应用和扩展，其中对应的安全内核分别就是 *Flask* 模型的策略服务器和 *Java 2* 的安全管理器。安全内核理论还可以应用于硬件，关于这一理论可以参考 1.4 节列举过的 D.Gollman 的著作第 5 章。

最后要指出本教程完全没有涉及的一个重要领域，就是安全系统的规范与评估，这是关于安全系统特别是安全操作系统的详细的功能表达及评测方法，并且根据安全功能的多少与程度划分为不同的安全等级(根据这一划分，绝大多数普通操作系统都属于比较低的安全等级)。目前这些安全标准中最有影响的是美国联邦政府信息处理标准和欧盟制订的信息安全系统评估标准，其中包括一整套测试与评估技术，感兴趣的读者可以阅读上面列举的 Bishop 的教科书，该书对此有详细的专门阐述。

习 题

6-1 查看你的计算机上任何一个目录下的文件的访问属性，然后以访问控制矩阵重新表达该目录的安全策略。

6-2 用访问控制矩阵表达图 6-1 的安全策略。

6-3 考虑一个任意的访问控制矩阵，矩阵元可以表达至多四种权限：读、写、创建和删除。试设计一种数据结构来实现这一策略模型。

6-4 考虑一个帐户文件对象，它由一条一条的记录组成，每个记录包含客户名字、客户类型(普通客户、职员、经理三种类型之一)、帐户号、当前余额、信用等级这些字段。

1) 用访问控制矩阵表达出以下策略：

任何客户可以读他们自己的记录的所有字段；

职员可以读任何记录的除信用等级以外的任何字段；

经理可以创建新记录、读文件中的任何记录的任何字段、更新任何记录中的余额；

记录一经创建则不得删除并且客户名字不得被修改。

2) 设计一种数据结构来实现以上策略。

6-5 设计安全策略必须考虑全面，否则表面上的安全规则实际上起不到所期望的作用。例如这样一个员工信息文件，它由一条一条的记录组成，每个记录包含员工名字、年龄、职位、收入等这些字段。对这一文件对象设计了以下安全策略：

任何员工允许读他们自己的记录的所有字段；

任何员工允许读任何记录的除收入以外的任何字段；

任何员工允许查询至少 100 条记录的收入字段值的平均值(假定实际记录数总多于 100 条。这里规定样本数的下限是为了使平均值能够充分掩盖各个记录的收入值)。

1) 用访问控制矩阵表达出以上策略；

2) 如果你想获取某位员工“X”的收入，以上策略真能阻止你吗？为什么？

3）如果上一小题的答案是“否”，你应该怎样修改以上策略？对修改后的策略，你能设计一种数据结构加以实现吗？