位地址前缀。沙盒的尺寸可以选取到足够容纳最大的Applet而不浪费太多的地址空间。如果页面调用满足的话，物理内存不会成为问题。每个Applet拥有两个沙盒，一个放置代码，另一个放置数据，如图9-37a所示的16个16MB的沙盒。

沙盒的用意在于保证每个Applet不能跳转到或引用其他的代码沙盒或数据沙盒。提供两个沙盒的目的是为了避免Applet在运行时超越限制修改代码。通过抑制把所有的Applet放入代码沙盒，我们减少了自我修改代码的危险。只要Applet通过这种方法受到限制，它就不能损害浏览器或其他的Applet，也不能在内存里培植病毒或者对内存造成损失。

只要Applet被装入，它就被重新分配到沙盒的开头，然后系统检查代码和数据的引用是否已被限制在相应的沙盒里。在下面的讨论中，我们将看一下代码引用（如JMP和CALL指令），数据引用也是如此。使用直接寻址的静态JMP指令很容易检查：目标地址是否仍旧在代码沙盒里？同样，相对JMP指令也很容易检查。如果Applet含有要试图离开代码沙盒的代码，它就会被拒绝并不予执行。同样，试图接触外界数据的Applet也会被拒绝。

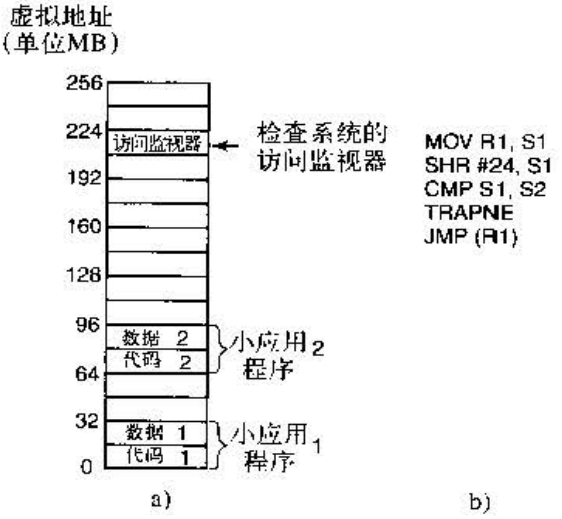

图9-37 a) 内存被划分为16 MB的沙盒；
b) 检查指令有效性的一种方法

最困难的是动态JMP。大多数计算机都有这样一条指令，该指令中要跳转的目标地址在运行的时候计算，该地址被存入一寄存器，然后间接跳转。例如，通过JMP（R1）跳转到寄存器1里存放的地址。这种指令的有效性必须在运行时检查。检查时，系统直接在间接跳转之前插入代码，以便测试目标地址。这样测试的一个例子如图9-37b所示。请记住，所有的有效地址都有同样的高k位地址，所以该地址前缀被存放在临时寄存器里，如说S2。这样的寄存器不能被Applet自身使用，因为Applet有可能要求重写寄存器以避免受该寄存器限制。

有关代码是按如下工作的：首先把被检查的目标地址复制到临时寄存器S1中。然后该寄存器向右移位正好将S1中的地址前缀隔离出来。第二步将隔离出的前缀同原先装入S2寄存器里的正确前缀进行比较。如果不匹配就激活陷阱程序杀死进程。这段代码序列需要四条指令和两个临时寄存器。

对运行中的二进制程序打补丁需要一些工作，但却是可行的。如果Applet是以源代码形式出现，工作就容易得多。随后在本地的编译器对Applet进行编译，自动查看静态地址并插入代码来校验运行中的动态地址。同样也需要一些运行时间的开销以便进行动态校验。Wahbe等人（1993）估计这方面的时间大约占4%，这一般是可接受的。

另一个要解决的问题是当Applet试图进行系统调用时会发生什么？解决方法是很直接的。系统调用的指令被一个叫做基准监视器的特殊模块所替代，这一模块采用了与动态地址校验相同的检查方式（或者，如果有源代码，可以链接一个调用基准监视器的库文件，而不是执行系统调用）。在这两个方法中，基准监视器检查每一个调用企图，并决定该调用是否可以安全执行。如果认为该调用是可接受的，如在指定的暂存目录中写临时文件，这种调用就可以执行。如果调用被认为是危险的或者基准监视器无法判断，Applet就被终止。若基准监视器可以判断是哪一个Applet执行的调用，内存里的一个基准监视器就能处理所有这样Applet的请求。基准监视器通常从配置文件中获知是否允许执行。

2. 解释

第二种运行不安全Applet的方法是解释运行并阻止它们获得对硬件的控制。Web浏览器使用的就是这种方法。网页上的Applet通常是用Java写的，Java可以是一种普通的编程语言，也可以是高级脚本语言，如安全TCL语言或Javascript。Java Applet首先被编译成一种叫做JVM（Java虚拟机，Java Virtual Machine）的面向栈的机器语言。正是这些JVM Applet被放在网页上，当它们被下载时就插入到浏览器内置的JVM解释器中，如图9-38所示。

使用解释运行的代码比编译运行的代码好处在于，每一条指令在执行前都由解释器进行检查。这就