给了解释器识别校验地址是否有效的机会。另外，系统调用也可以被捕捉并解释。这些调用的处理方式与安全策略有关。例如，如果Applet是可信任的（如来自本地磁盘的Applet），它的系统调用就可以毫无疑问会被执行。但是如果Applet不受信任（如来自Internet的Applet），它就会被放入沙盒来限制自身的行为。

高级脚本语言也能够被解释执行。这里，解释执行不需要机器地址，所以也就不存在脚本以不允许的方式访问内存所带来的危险。解释运行的缺点是：它与编译运行的代码相比十分缓慢。

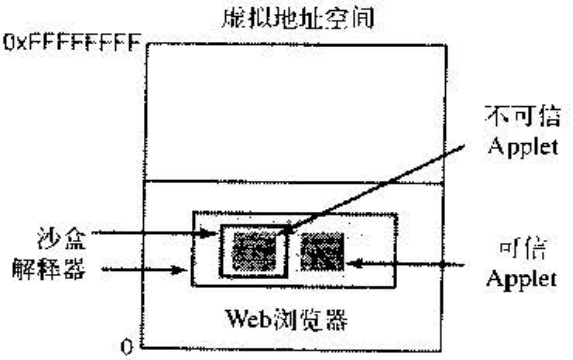

图9-38 Applet可以被Web浏览器以解释方式执行

9.8.7 Java安全性

人们设计了Java编程语言和相关的运行时系统，是为了一次编写并编译后就能够在Interent上以二进制代码的形式运行在所有支持Java的机器上。从一开始设计Java语言开始，安全性就成为其重要的一部分。在这一小节，我们来看看它的工作原理。

Java是一种在类型上安全的编程语言，也就是说编译器会拒绝任何与自身类型不一致的变量使用。而C语言正好相反，请看下面的代码：

```
naughty_func()
{
   char *p;
   p = rand();
   *p = 0;
}
```

代码把产生的随机数放在指针p中。然后把0字节存储在p所包含的地址中，覆盖了地址里原先的任何代码和数据。而在Java中，混合使用类型的语句是被语法所禁止的。而且，Java没有指针变量、类型转换、用户控制的存储单元分配（如malloc和free），并且所有的数组引用都要在运行时进行校验。

Java程序被编译成一种叫做JVM（Java Virtual Machine）字节码的中间形态二进制代码。JVM有大约100个指令，大多数指令是把不同类型的对象压入栈、弹出栈或是用算术合并栈里的对象。这些JVM程序通常是解释执行程序，虽然在某些情况下它们可以被编译成机器语言以便执行得更快。在Java模式中，通过Internet发送到远程计算机上运行的Applet是JVM程序。

当Applet到达远程计算机时，首先由JVM字节码校验器查看Applet是否符合规则。正确编译的Applet会自动符合规则，但无法阻止一个恶意的用户用汇编语言写JVM格式的Applet。校验的规则包括：

1) Applet是否伪造了指针？
2) 是否违背了私有类成员的访问限制？
3) 是否试图把某种类型的变量用作其他类型？
4) 是否产生栈上溢或下溢？
5) 是否非法地将变量从一种类型转换为另一种类型？

如果Applet通过了所有的测试，它就能被安全地执行并且不用担心它会访问非自己所有内存空间。

但是Applet也可以通过调用Java方法（过程）来执行系统调用。Java处理这种调用的方法也在不断在进步。在最初的Java版本JDK（Java Development Kit）1.0里，Applet被分为两类：可信的与不可信的。从本地磁盘取出的Applet是可信的并被允许执行任何所需要的系统调用。相反，从Internet获取的Applet是不可信的。它们被限制在沙盒里运行，如图9-38所示，实际上并不能做什么事。

在从这一模式中取得了些经验后，Sun公司认为对Applet的限制太大了。在JDK 1.1版本里，引入了版本标注。当Applet从Internet传递过来后，系统首先查看Applet是否有用户信任的个人或组织标注（通过用户所信任的标注者列表来定义）。如果是，Applet就被允许做任何操作，否则就必须在沙盒里运行并且受到很强的限制。