## 1 为什么需要沙箱机制？

默认情况下，一个应用程序是可以访问机器上的所有资源的，比如CPU、内存、文件系统、网络等等。

但是这是不安全的，如果随意操作资源，有可能破坏其他应用程序正在使用的资源，或者造成数据泄漏。为了解决这个问题，一般有下面两种解决方案：

(1) 为程序分配一个限定权限的账号：利用操作系统的权限管理机制进行限制

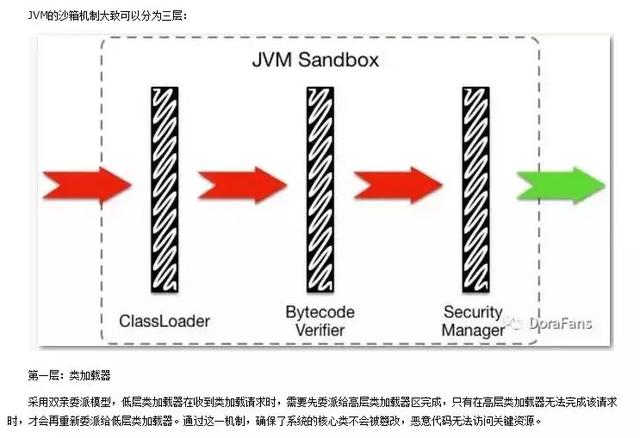
(2) 为程序提供一个受限的运行环境：这就是沙箱机制

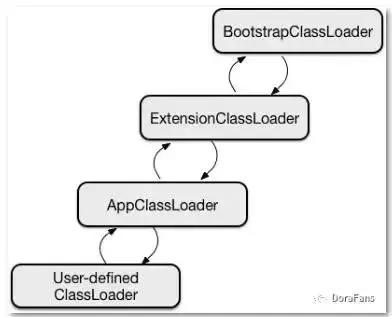
## 2 什么是沙箱机制？

如上所述，沙箱就是一个限制应用程序对系统资源的访问的运行环境。

沙箱很多情况下都是实现在虚拟机（VM）中，比如Java的虚拟机JVM、Javascript的虚拟机V8引擎、Android中的虚拟机Dalvik/ART，以及以太坊的虚拟机EVM等等。具体的实现方式各有不同，本文重点分析一下JVM和EVM的沙箱机制实现。

## 3 JVM的沙箱机制





****第二层：字节码校验器****

类字节码被加载后，需要在字节码层面进行检验，具体包括以下内容：

(1) 变量要在使用之前进行初始化

(2) 方法调用与对象引用类型之前要匹配

(3) 访问私有数据和方法的规则没有被违背

(4) 对本地变量的访问落在运行时堆栈内

(5) 运行时堆栈没有溢出通过这一机制，可以确保字节码符合Java语言规范，避免访问无效访问或越界访问。

****第三层：安全管理器****

这一层是交由应用开发者来维护的，开发者可以根据自身需求，制定对应的安全策略。默认的安全策略配置位于以下两个配置文件中：

· $JAVA\_HOME/conf/security/java.security

· $JAVA\_HOME/conf/security/java.policy

默认情况下安全管理器是不开启的，可以在命令行增加一个参数来开启它：

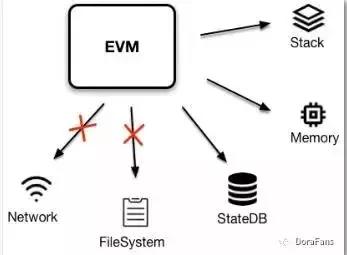
java -Djava.security.manager SandboxTest

默认的安全策略配置不允许应用程序对文件的读写操作，所以如果你试图写一个文件，运行时会报如下错误：

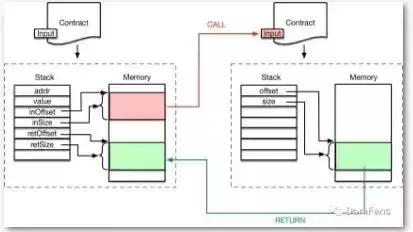


## 4.EVM的沙箱机制

EVM本身是一个相对封闭的环境，不支持对网络和文件系统的直接访问，从这个层面来说已经起到一部分沙箱的功能了：



另外，在智能合约之间互相调用时，EVM会重新分配堆栈和内存空间，在一个全新的环境中运行新合约，即使出现问题也不会破坏原有执行环境，从而为智能合约的执行提供了一个沙箱环境。



最后，每个智能合约的存储也是相互独立的，开发者可以根据需求，限制合约可以访问的存储空间，避免未授权的访问或修改。举例来说，下图就描述了CALL和CALLCODE指令对存储空间访问的区别：

