链滴

Java 安全管理器 SecurityManager

作者：zwxbest

原文链接：https://ld246.com/article/1543414159148

来源网站：链滴

一、文章的目的

这是一篇对Java安全管理器入门的文章，目的是简单了解什么是SecurityManager，对管理器进行简配置，解决简单问题。

比如在阅读源码的时候，发现这样的代码，想了解是做什么的：

```
SecurityManager security = System.getSecurityManager(); if (security != null) {
security.checkWrite(name);
}
```

亦或者在本机运行正常，在服务器运行报错，想解决问题：

```
Exception in thread "main" java.security.AccessControlException: access denied (java.lang.Runt
mePermission createSecurityManager)
    at java.security.AccessControlContext.checkPermission(AccessControlContext.java:374)
    at java.security.AccessController.checkPermission(AccessController.java:549)
    at java.lang.SecurityManager.checkPermission(SecurityManager.java:532)
    at java.lang.SecurityManager.(SecurityManager.java:282)
    at xia.study._01Thread.ThreadTest.creatThread1(ThreadTest.java:18)
    at xia.study._01Thread.ThreadTest.main(ThreadTest.java:13)
```

这时候具备一些SecurityManager的基础知识还是有必要的。

二、SecurityManager应用场景

当运行未知的Java程序的时候，该程序可能有恶意代码（删除系统文件、重启系统等），为了防运行恶意代码对系统产生影响，需要对运行的代码的权限进行控制，这时候就要启用Java安全管理器。

三、管理器配置文件

3.1 默认配置文件

默认的安全管理器配置文件是 $JAVA_HOME/jre/lib/security/java.policy，即当未指定配置文时，将会使用该配置。内容如下：

```
// Standard extensions get all permissions by default

grant codeBase "file:${{java.ext.dirs}}/*" {
    permission java.security.AllPermission; };  // default permissions granted to all domains

grant {
    // Allows any thread to stop itself using the java.lang.Thread.stop()
    // method that takes no argument.
    // Note that this permission is granted by default only to remain
    // backwards compatible.
    // It is strongly recommended that you either remove this permission
    // from this policy file or further restrict it to code sources
    // that you specify, because Thread.stop() is potentially unsafe.
    // See the API specification of java.lang.Thread.stop() for more
```

```
        // information.
    permission java.lang.RuntimePermission "stopThread";  // allows anyone to listen on un-pri
ileged ports
    permission java.net.SocketPermission "localhost:1024-", "listen";  // "standard" properies th
t can be read by anyone

    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
    permission java.util.PropertyPermission "java.vendor.url", "read";
    permission java.util.PropertyPermission "java.class.version", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";
    permission java.util.PropertyPermission "os.arch", "read";
    permission java.util.PropertyPermission "file.separator", "read";
    permission java.util.PropertyPermission "path.separator", "read";
    permission java.util.PropertyPermission "line.separator", "read";  permission java.util.Proper
yPermission "java.specification.version", "read";
    permission java.util.PropertyPermission "java.specification.vendor", "read";
    permission java.util.PropertyPermission "java.specification.name", "read";  permission java.u
il.PropertyPermission "java.vm.specification.version", "read";
    permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
    permission java.util.PropertyPermission "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read"; };
```

3.2 配置文件详解

详解见第五部分，此处知道有这个配置文件即可。

四、启动安全管理器

启动安全管理有两种方式，建议使用启动参数方式。

4.1 启动参数方式

启动程序的时候通过附加参数启动安全管理器：

```
-Djava.security.manager
```

若要同时指定配置文件的位置那么示例如下：

```
-Djava.security.manager -Djava.security.policy="E:/java.policy"
```

4.2 编码方式启动

也可以通过编码方式启动，不过不建议：

```
System.setSecurityManager(new SecurityManager());
```

通过参数启动，本质上也是通过编码启动，不过参数启动使用灵活，项目启动源码如下（sun.misc

Launcher）：

```
// Finally, install a security manager if requested
String s = System.getProperty("java.security.manager"); if (s != null) {
    SecurityManager sm = null; if ("".equals(s) || "default".equals(s)) {
        sm = new java.lang.SecurityManager();
    } else { try {
            sm = (SecurityManager)loader.loadClass(s).newInstance();
        } catch (IllegalAccessException e) {
        } catch (InstantiationException e) {
        } catch (ClassNotFoundException e) {
        } catch (ClassCastException e) {
        }
    } if (sm != null) {
        System.setSecurityManager(sm);
    } else { throw new InternalError( "Could not create SecurityManager: " + s);
    }
}
```

可以发现将会创建一个默认的SecurityManager；

五、配置文件简单解释

5.1 配置基本原则

在启用安全管理器的时候，配置遵循以下基本原则：

1. 没有配置的权限表示没有。
2. 只能配置有什么权限，不能配置禁止做什么。
3. 同一种权限可多次配置，取并集。
4. 统一资源的多种权限可用逗号分割。

5.2 默认配置文件解释

第一部分授权：

```
grant codeBase "file:${{java.ext.dirs}}/*" {
    permission java.security.AllPermission; };
```

授权基于路径在"file:${{java.ext.dirs}}/*"的class和jar包，所有权限。

第二部分授权：

```
grant {
    permission java.lang.RuntimePermission "stopThread";
 ......
}
```

这是细粒度的授权，对某些资源的操作进行授权。具体不再解释，可以查看javadoc

5.3 可配置项详解

当批量配置的时候，有三种模式：

- directory/ 表示directory目录下的所有.class文件，不包括.jar文件
- directory/* 表示directory目录下的所有的.class及.jar文件
- directory/- 表示directory目录下的所有的.class及.jar文件，包括子目录

可以通过${}来引用系统属性，如：

"file:${{java.ext.dirs}}/*"

六、问题解决

当出现关于安全管理的报错的时候，基本有两种方式来解决。

6.1 取消安全管理器

一般情况下都是无意启动安全管理器，所以这时候只需要把安全管理器进行关闭，去掉启动参数可。

6.2 增加相应权限

若因为没有权限报错，则报错信息中会有请求的权限和请求什么权限，如下：

```
Exception in thread "main" java.security.AccessControlException: access denied (java.io.FilePermission E:\pack\a\a.txt write)
```

上面例子，请求资源E:\pack\a\a.txt，的FilePermission的写权限没有，因此被拒绝。

也可以开放所有权限：

```
grant {permission java.security.AllPermission; };
```