## 解决故障的思路：

* 重视报错提示信息：每个错误的出现，都是给出错误提示信息，一般情况下这个提示基本定位了问题的所在，因此一定要重视这个报错信息，如果对这些错误信息视而不见，问题永远得不到解决。
* 查阅日志文件：有时候报错信息只是给出了问题的表面现象，要想更深入的了解问题，必须查看相应的日志文件，而日志文件又分为系统日志文件（/var/log）和应用的日志文件，结合这两个日志文件，一般就能定位问题所在。
* 分析、定位问题：这个过程是比较复杂的，根据报错信息，结合日志文件，同时还要考虑其它相关情况，最终找到引起问题的原因。
* 解决问题：找到了问题出现的原因，解决问题就是很简单的事情了。
* 从这个流程可以看出，解决问题的过程就是分析、查找问题的过程，一旦确定问题产生的原因，故障也就随之解决了。

### 问题 1：文件系统破坏导致系统无法启动

操作系统分区文件系统出现了问题，这个问题发生的机率很高，通常引起这个问题的原因主要是系统突然断电，引起文件系统结构不一致，一般情况下，解决此问题的方法是采用 fsck 命令，进行强制修复。

# umount /dev/sda\*

# fsck.ext4 -y /dev/sda\*

### 问题 2：“no space left on device” 错误与解决方法

# crontab -e

编辑完后保存退出后，报错 no space left on device

根据上面的报错了解到是磁盘空间满了，那么首先是检查磁盘空间。

# df -h

查看到是 / var 磁盘分区空间已经达到 100%，至此定位了问题所在。是 / var 磁盘空间饱满导致，因为 crontab 会在保存时将文件信息写到 / var 目录下面，然而这个磁盘没有空间了，所以报错。

接着通过命令 du –sh \* 命令检查 / var 目录下面的所有文件或者目录的大小，发现 / var/spool/clientmqueue 目录占用了 / var 整个分区大小的 90%，那么 / var/spool/clientmqueue 目录下的文件都是怎么产生的，能否删除，基本上都是邮件信息，可以删除。

### 问题 3：文件已经删除，但是空间没有释放的原因

运维监控系统发来通知，报告一台服务器空间满了，登陆服务器查看，根分区确实满了，这里先说一下服务器的一些删除策略，由于 linux 没有回收站功能，所以线上服务器上所有要删除的文件都会先移到系统 / tmp 目录下，然后定期清除 / tmp 目录下的数据。这个策略本身没有什么问题，但是通过检查发现这台服务器的系统分区中并没有单独划分 / tmp 分区，这样 / tmp 下的数据其实占用根分区的空间，既然找到了问题，那么删除 / tmp 目录下一些占用空间较大的数据文件即可。

# du -sh /tmp/\* | sort -nr |head -3

通过命令发现在 / tmp 目录下有个 66G 大小的文件 access\_log，这个文件应该是 apache 产生的访问日志文件，从日志大小来看，应该是很久没有清理的 apache 日志文件了，基本判定是这个文件导致的根空间爆满，在确认此文件可以删除后，执行如下删除命令，

# rm /tmp/access\_Iog

# df -h

从输出来看，根分区空间仍然没有释放，这是怎么回事？

一般来说不会出现删除文件后空间不释放的情况，但是也存在例外，比如文件进程锁定，或者有进程一直在向这个文件写数据，要理解这个问题，就需要知道 linux 下文件的存储机制和存储结构。

一个文件在文件系统中存放分为两个部分：数据部分和指针部分，指针位于文件系统的 meta-data 中，在将数据删除后，这个指针就从 meta-data 中清除了，而数据部分存储在磁盘中。在将数据对应的指针从 meta-data 中清除后，文件数据部分占用的空间就可以被覆盖并写入新的内容，之所以出现删除 access\_log 文件后，空间还没有释放，就是因为 httpd 进程还在一直向这个文件写入内容，导致虽然删除了 access\_Ilog 文件，但是由于进程锁定，文件对应的指针部分并未从 meta-data 中清除，而由于指针并未删除，系统内核就认为文件并未被删除，因此通过 df 命令查询空间并未释放。

问题排查：

既然有了解决思路，那么接下来看看是否有进程一直在向 access\_log 文件中写入数据，这里需要用到 linux 下的 lsof 命令，通过这个命令可以获取一个仍然被应用程序占用的已删除文件列表。

# lsof | grep delete

从输出可以看出，/tmp/access\_log 文件被进程 httpd 锁定，而 httpd 进程还一直向这个文件写入日志数据，最后一列的‘deleted’状态说明这个日志文件已经被删除，但是由于进程还在一直向此文件写入数据，因此空间并未释放。

解决问题：

到这里问题就基本排查清楚了，解决这一类问题的方法有很多，最简单的方法就是关闭或者重启 httpd 进程，当然重启操作系统也可以。不过这些并不是最好的办法，对待这种进程不停对文件写日志的操作，要释放文件占用的磁盘空间，最好的方法是在线清空这个文件，具体可以通过如下命令完成：

# echo “”>/tmp/access\_log

通过这种方法，磁盘空间不但可以马上释放，也可以保障进城继续向文件写入日志，这种方法经常用于在线清理 apache /tomcat/nginx 等 web 服务产生的日志文件。

### 问题 4："too many open files" 错误与解决方法

问题现象：这是一个基于 java 的 web 应用系统，在后台添加数据时提示无法添加，于是登陆服务器查看 tomcat 日志，发现如下异常信息：java.io.IOException: Too many open files

通过这个报错信息，基本判断是系统可以用的文件描述符不够了，由于 tomcat 服务室系统 www 用户启动的，于是以 www 用户登陆系统，通过 ulimit –n 命令查看系统可以打开最大文件描述符的数量，输出如下：

$ ulimit -n

65535

可以看到这台服务器设置的最大可以打开的文件描述符已经是 65535 了，这么大的值应该够用了，但是为什么提示这样的错误呢？

解决思路，这个案例涉及 ulimit 命令的使用。

在使用 ulimit 时，有以下几种使用方法：

1、 在用户环境变量中加入

如果用户使用的是 bash，那么可以在用户目录的环境变量文件. bashrc 或者. bash\_profile 中加入 “ulimit –u128” 来限制用户最多可以使用 128 个进程

2、 在应用程序的启动脚本中加入

如果应用程序是 tomcat，那么可以再 tomcat 的启动脚本 startup.sh 中加入‘ulimit -n 65535’来限制用户最多可以使用 65535 个文件描述符

3、 直接在 shell 命令终端执行 ulimit 命令

这种方法的资源限制仅仅在执行命令的终端生效，在退出或者和关闭终端后，设置失效，并且这个设置不影响其他 shell 终端

解决问题：

在了解 ulimit 知识后，接着上面的案例，既然 ulimit 设置没有问题，那么一定是设置没有生效导致的，接下来检查下启动 tomcat 的 www 用户环境变量是否添加 ulimit 限制，检查后发现，www 用户并无 ulimit 限制。于是继续检查 tomcat 启动脚本 startup.sh 文件是否添加了 ulimit 限制，检查后发现也没有添加。最后考略是否将限制加到了 limits.conf 文件中，于是检查 limits.conf 文件，操作如下：

# cat /etc/security/limits.conf | grep www

www soft nofile 65535

www hard nofile 65535

从输出可知，ulimit 限制加在 limits.conf 文件中，既然限制已经添加了，配置也没有什么错，为何还会报错，经过思考，判断只有一种可能，那就是 tomcat 的启动时间早于 ulimit 资源限制的添加时间，于是首先查看下 tomcat 启动时间，操作如下：

# uptime

Up 283 days

# pgrep -f tomcat

4667

# ps -eo pid,lstart,etime|grep 4667

4667 Sat Jul 6 09;33:39 2013 77-05:26:02

从输出可以看出，这台服务器已经有 283 没有重启了，而 tomcat 是在 2013 年 7 月 6 日 9 点启动的，启动了将近 77 天，接着继续看看 limits.conf 文件的修改时间，

# stat /etc/security/limits.conf

通过 stat 命令清除的看到，limits.conf 文件最后的修改时间是 2013 年 7 月 12，晚于 tomcat 启动时间，清楚问题后，解决问题的方法很简单，重启一下 tomcat 就可以了。

### 问题 5：Read-only file system 错误与解决方法

解析：出现这个问题的原因有很多种，可能是文件系统数据块出现不一致导致的，也可能是磁盘故障造成的，主流 ext3/ext4 文件系统都有很强的自我修复机制，对于简单的错误，文件系统一般都可以自行修复，当遇到致命错误无法修复的时候，文件系统为了保证数据一致性和安全，会暂时屏蔽文件系统的写操作，讲文件系统 变为只读，今儿出现了上面的 “read-only file system” 现象。

手工修复文件系统错误的命令式 fsck，在修复文件系统前，最好卸载文件系统所在的磁盘分区。

# umount /www/data

Umount : /www/data: device is busy

提示无法卸载，可能是这个磁盘中还有文件对应的进程在运行，检查如下：

# fuser -m /dev/sdb1

/dev/sdb1: 8800

接着检查一下 8800 端口对应的什么进程，

# ps -ef |grep 8800

检查后发现时 apache 没有关闭，停止 apache

# /usr/local/apache2/bin/apachectl stop

# umount /www/data

# fsck -V -a /dev/sdb1

# mount /dev/sdb1 /www/data

### 访问频率限制解决一些攻击

Nginx来处理访问控制的方法有多种，实现的效果也有多种，访问IP段，访问内容限制，访问频率限制等。

用Nginx+Lua来做访问限制主要是考虑到高并发环境下快速访问控制的需求。

Nginx处理请求的过程一共划分为11个阶段，分别是：

post-read、server-rewrite、find-config、rewrite、post-rewrite、 preaccess、access、post-access、try-files、content、log.

* post-read:   
  读取请求内容阶段   
  Nginx读取并解析完请求头之后就立即开始运行
* server-rewrite   
  Server请求地址重写阶段
* find-config   
  配置查找阶段
* rewrite   
  Location请求地址重写阶段
* post-rewrite   
  请求地址重写提交阶段 。
* preaccess   
  访问权限检查准备阶段
* access   
  访问权限检查阶段
* post-access   
  访问权限检查提交阶段 。
* try-files   
  配置项try\_files处理阶段
* content   
  内容产生阶段   
  log   
  日志模块处理阶段   
  在openresty（一款基于 NGINX 和 Lua 的 Web 平台）中，可以找到：

set\_by\_lua，access\_by\_lua，content\_by\_lua，rewrite\_by\_lua等方法。

那么访问控制应该是，access阶段。

**解决：**

按照正常的逻辑思维，我们会想到的访问控制方案如下：

1.检测是否被forbidden？

=》是，forbidden是否到期：是，清除记录，返回200，正常访问；否，返回403；

=》否，返回200，正常访问

2.每次访问，访问用户的访问频率+1处理

3.检测访问频率是否超过限制，超过即添加forbidden记录，返回403