详解如何在 docker 容器中捕获信号

更新时间：2017年09月27日 10:29:48 作者：sparkdev

> 本篇文章主要介绍了如何在 docker 容器中捕获信号，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧

我们可能都使用过 docker stop 命令来停止正在运行的容器，有时可能会使用 docker kill 命令强行关闭容器或者把某个信号传递给容器中的进程。这些操作的本质都是通过从主机向容器发送信号实现主机与容器中程序的交互。比如我们可以向容器中的应用发送一个重新加载信号，容器中的应用程序在接到信号后执行相应的处理程序完成重新加载配置文件的任务。本文将介绍在 docker 容器中捕获信号的基本知识。

信号(linux)

信号是一种进程间通信的形式。一个信号就是内核发送给进程的一个消息，告诉进程发生了某种事件。当一个信号被发送给一个进程后，进程会立即中断当前的执行流并开始执行信号的处理程序。如果没有为这个信号指定处理程序，就执行默认的处理程序。

进程需要为自己感兴趣的信号注册处理程序，比如为了能让程序优雅的退出(接到退出的请求后能够对资源进行清理)一般程序都会处理 SIGTERM 信号。与 SIGTERM 信号不同，SIGKILL 信号会粗暴的结束一个进程。因此我们的应用应该实现这样的目录：捕获并处理 SIGTERM 信号，从而优雅的退出程序。如果我们失败了，用户就只能通过 SIGKILL 信号这一终极手段了。除了 SIGTERM 和 SIGKILL ，还有像 SIGUSR1 这样的专门支持用户自定义行为的信号。下面的代码简单的说明在 nodejs 中如何为一个信号注册处理程序：

```
process.on('SIGTERM', function() {
 console.log('shutting down...');
});
```

关于信号的更多信息，笔者在《linux kill 命令》一文中有所提及，这里不再赘述。

容器中的信号

Docker 的 stop 和 kill 命令都是用来向容器发送信号的。注意，只有容器中的 1 号进程能够收到信号，这一点非常关键!

stop 命令会首先发送 SIGTERM 信号，并等待应用优雅的结束。如果发现应用没有结束(用户可以指定等待的时间)，就再发送一个 SIGKILL 信号强行结束程序。

kill 命令默认发送的是 SIGKILL 信号，当然你可以通过 -s 选项指定任何信号。

下面我们通过一个 nodejs 应用演示信号在容器中的工作过程。创建 app.js 文件，内容如下：

```
'use strict';

var http = require('http');

var server = http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/plain'});
 res.end('Hello World\n');
}).listen(3000, '0.0.0.0');

console.log('server started');

var signals = {
```

大家感兴趣的内容

1 Docker获取镜像报错do
2 Docker 给运行中的容器
3 docker.service启动失败
4 Docker 清理命令集锦
5 浅谈docker-compose网
6 ubuntu14.04+docker的
7 docker容器如何优雅的
8 Docker容器访问宿主机
9 详解如何使用Docker部
10 详解docker国内镜像拉

最近更新的内容

一文快速入门Docker推荐
Docker拉取镜像的完整步
Docker 清理命令集锦
使用docker搭建gitlab详
解决docker加载新的镜像
Docker 教程之Docker Hu
Docker Compose引用环
Docker集群的创建与管理
win7下docker安装与报错
docker 如何搭建私有仓库

常用在线小工具

CSS代码工具
JavaScript代码格式化工具
在线XML格式化/压缩工具
php代码在线格式化美化工
sql代码在线格式化美化工
在线HTML转义/反转义工
在线JSON代码检验/检验/
JavaScript正则在线测试工
在线生成二维码工具(加强
更多在线工具

```
  'SIGINT': 2,
  'SIGTERM': 15
};

function shutdown(signal, value) {
 server.close(function () {
  console.log('server stopped by ' + signal);
  process.exit(128 + value);
 });
}

Object.keys(signals).forEach(function (signal) {
 process.on(signal, function () {
  shutdown(signal, signals[signal]);
 });
});
```

这个应用是一个 http 服务器，监听端口 3000，为 SIGINT 和 SIGTERM 信号注册了处理程序。接下来我们将介绍以不同的方式在容器中运行程序时信号的处理情况。

应用程序作为容器中的 1 号进程

创建 Dockerfile 文件，把上面的应用打包到镜像中：

```
FROM iojs:onbuild
COPY ./app.js ./app.js
COPY ./package.json ./package.json
EXPOSE 3000
ENTRYPOINT ["node", "app"]
```

请注意 ENTRYPOINT 指令的写法，这种写法会让 node 在容器中以 1 号进程的身份运行。

接下来创建镜像：

```
$ docker build --no-cache -t signal-app -f Dockerfile .
```

然后启动容器运行应用程序：

```
$ docker run -it --rm -p 3000:3000 --name="my-app" signal-app
```

此时 node 应用在容器中的进程号为 1：

```
nick@u16os:~$ docker container exec my-app ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.1  0.5 766724 21816 ?        Ssl+ 10:53   0:00 node app
root         17  0.0  0.0  17496  2044 ?        Rs   10:54   0:00 ps aux
```

现在我们让程序退出，执行命令：

```
$ docker container kill --signal="SIGTERM" my-app
```

此时应用会以我们期望的方式退出：

```
nick@u16os:~$ docker run -it --rm -p 3000:3000 --name="my-app" signal-app
server started
server stopped by SIGTERM
```

应用程序不是容器中的 1 号进程

创建一个启动应用程序的脚本文件 app1.sh，内容如下：

```
#!/usr/bin/env bash
node app
```

然后创建 Dockerfile1 文件，内容如下：

```
FROM iojs:onbuild
COPY ./app.js ./app.js
COPY ./app1.sh ./app1.sh
COPY ./package.json ./package.json
RUN chmod +x ./app1.sh
EXPOSE 3000
ENTRYPOINT ["./app1.sh"]
```

接下来创建镜像：

```
$ docker build --no-cache -t signal-app1 -f Dockerfile1 .
```

然后启动容器运行应用程序：

```
$ docker run -it --rm -p 3000:3000 --name="my-app1" signal-app1
```

此时 node 应用在容器中的进程号不再是 1：

```
nick@u16os:~$ docker container exec my-app1 ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0  20040  2804 ?        Ss+  11:00   0:00 bash ./app1.sh
root          6  0.3  0.5 766724 22028 ?        Sl+  11:00   0:00 node app
root         11  0.0  0.0  17496  2116 ?        Rs   11:01   0:00 ps aux
```

现在给 my-app1 发送 SIGTERM 信号试试，已经无法退出程序了！在这个场景中，应用程序由 bash 脚本启动，bash 作为容器中的 1 号进程收到了 SIGTERM 信号，但是它没有做出任何的响应动作。

我们可以通过：

```
$ docker container stop my-app1
# or
$ docker container kill --signal="SIGKILL" my-app1
```

退出应用，它们最终都是向容器中的 1 号进程发送了 SIGKILL 信号。很显然这不是我们期望的，我们希望程序能够收到 SIGTERM 信号优雅的退出。

在脚本中捕获信号

创建另外一个启动应用程序的脚本文件 app2.sh，内容如下：

```
#!/usr/bin/env bash
set -x

pid=0

# SIGUSR1-handler
my_handler() {
 echo "my_handler"
}

# SIGTERM-handler
term_handler() {
 if [ $pid -ne 0 ]; then
  kill -SIGTERM "$pid"
  wait "$pid"
 fi
 exit 143; # 128 + 15 -- SIGTERM
}
# setup handlers
# on callback, kill the last background process, which is `tail -f /dev/null` and ex
trap 'kill ${!}; my_handler' SIGUSR1
trap 'kill ${!}; term_handler' SIGTERM

# run application
node app &
pid="$!"

# wait forever
while true
do
 tail -f /dev/null & wait ${!}
done
```

这个脚本文件在启动应用程序的同时可以捕获发送给它的 SIGTERM 和 SIGUSR1 信号，并为它们添加了处理程序。其中 SIGTERM 信号的处理程序就是向我们的 node 应用程序发送 SIGTERM 信号。

然后创建 Dockerfile2 文件，内容如下：

```
FROM iojs:onbuild
COPY ./app.js ./app.js
COPY ./app2.sh ./app2.sh
COPY ./package.json ./package.json
RUN chmod +x ./app2.sh
EXPOSE 3000
ENTRYPOINT ["./app2.sh"]
```

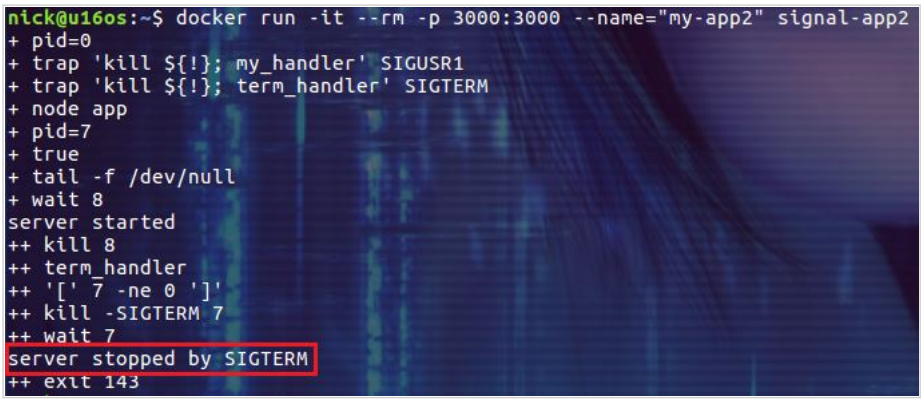
接下来创建镜像：

```
$ docker build --no-cache -t signal-app2 -f Dockerfile2 .
```

然后启动容器运行应用程序：

```
$ docker run -it --rm -p 3000:3000 --name="my-app2" signal-app2
```

此时 node 应用在容器中的进程号也不是 1，但是它却可以接收到 SIGTERM 信号并优雅的退出了：



结论

容器中的 1 号进程是非常重要的，如果它不能正确的处理相关的信号，那么应用程序退出的方式几乎总是被强制杀死而不是优雅的退出。究竟谁是 1 号进程则主要由 EntryPoint, CMD, RUN 等指令的写法决定，所以这些指令的使用是很有讲究的。

以上就是本文的全部内容，希望对大家的学习有所帮助，也希望大家多多支持脚本之家。

您可能感兴趣的文章:

docker容器如何优雅的终止详解

Docker 容器操作退出后进入解决办法

Docker常用的清除容器镜像命令小结

Docker 解决容器时间与主机时间不一致的问题三种解决方案

Docker为网络bridge模式指定容器ip的方法

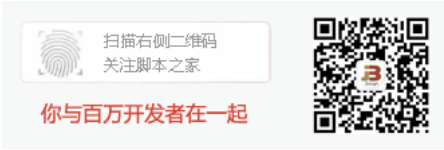
如何在Docker容器内外互相拷贝数据

Docker容器中文乱码(修改docker容器编码格式)的解决方案

Docker 容器内存监控原理及应用

详解挂载运行的docker容器中如何挂载文件系统

两种方式创建docker镜像的启动容器时区别介绍(总结篇)



微信公众号搜索“脚本之家”，选择关注
程序猿的那些事、送书等活动等着你

原文链接：http://www.cnblogs.com/sparkdev/p/7598590.html?utm_source=tuicool&utm_medium=referral

docker 容器 捕获信号

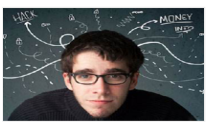
相关文章

解决docker pull镜像速度慢的问题的方法

本篇文章主要介绍了解决docker pull镜像速度慢的问题的方法，小编觉得挺不错的，现在分享给大家，也给大家做个参考。一起跟随小编过来看看吧 2017-11-11

浅谈Docker run 容器处于created状态问题

这篇文章主要介绍了解决Docker run 容器处于created状态问题，具有很好的参考价值，希望对大家有所帮助。一起跟随小编过来看看吧 2021-03-03



本地使用docker打包部署镜像的方法

这篇文章主要介绍了本地使用docker打包部署镜像的方法，文中通过示例代码介绍的非常详细，对大家的学习或者工作具有一定的参考学习价值，需要的朋友们下面随着小编来一起... 2020-12-12