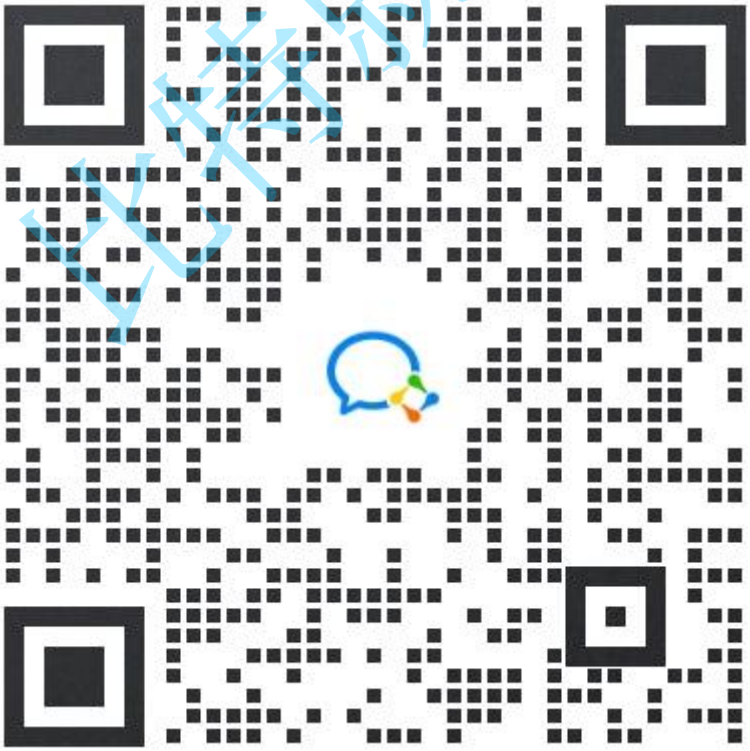
多阶段构建

版权说明

对比特项目感兴趣，可以联系这个微信。



基础知识

构建 docker 镜像可以有下面两种方式

- 将全部组件及其依赖库的编译、测试、打包等流程封装进一个 `docker` 镜像中。但是这种方式存在一些问题， 比如 `Dockefile` 特别长，可维护性降低；镜像的层次多，体积大，部署时间长等问题
- 将每个阶段分散到多个 `Dockerfile`。一个 `Dockerfile` 负责将项目及其依赖库编译测试打包好后，然后将运行文件拷贝到运行环境中，这种方式需要我们编写多个 `Dockerfile` 以及一些自动化脚本才能将其两个阶段自动整合起来
- 为了解决以上的两个问题，`Docker 17.05` 版本开始支持多镜像阶段构建。只需要编写一个 `Dockerfile` 即可解决上述问题。

实战目的

掌握多阶段构建，能够尽量减小 Docker 镜像的大小。

实战步骤

构建 docker 镜像可以有下面两种方式

- 将全部组件及其依赖库的编译、测试、打包等流程封装进一个 `docker` 镜像中。但是这种方式存在一些问题， 比如 `Dockefile` 特别长，可维护性降低；镜像的层次多，体积大，部署时间长等问题
- 将每个阶段分散到多个 `Dockerfile`。一个 `Dockerfile` 负责将项目及其依赖库编译测试打包好后，然后将运行文件拷贝到运行环境中，这种方式需要我们编写多个 `Dockerfile` 以及一些自动化脚本才能将其两个阶段自动整合起来
- 为了解决以上的两个问题，`Docker 17.05` 版本开始支持多镜像阶段构建。只需要编写一个 `Dockerfile` 即可解决上述问题。

下面我们做个实验来验证一下：假如现在有一个 C 语言程序，我们想用 Docker 帮我们编译成可执行文件，然后执行该可执行文件。

```
C
#include <stdio.h>

int main()
{
    printf("hello docker!\n");
    return 0;
}
```

编写 Dockerfile，因为要有 C 语言的编译环境，我们需要安装 gcc，这次我们选用

centos:7

```
Dockerfile
FROM centos:7
# 设置版本
ENV VERSION 1.0
#设置国内源
RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g' \
         -e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://mirro
rs.ustc.edu.cn/centos|g' \
         -i.bak \
         /etc/yum.repos.d/CentOS-Base.repo
RUN yum makecache
# 设置工作目录
WORKDIR /src
# 拷贝源文件
COPY demo.c .
# 安装 gcc
RUN yum install -y gcc && \
    gcc -v
# 编译源文件
RUN gcc demo.c -o demo && \
    rm -f demo.c && \
    yum erase -y gcc
# 运行可执行文件
CMD ["/src/demo"]
```

构建这个 Docker 镜像, 并且创建容器测试结果

```
Shell
# 通过 Dockerfile 生成镜像 1.0
[zsc@VM-8-12-centos c_demo]$ docker build -t multi:v1.0 .

# 创建容器，  测试结果
[zsc@VM-8-12-centos c_demo]$ docker container run --name multi1 --
rm -it multi:v1.0
hello docker!

# 查看镜像信息
[zsc@VM-8-12-centos c_demo]$ docker image ls |grep multi
REPOSITORY    TAG         IMAGE ID       CREATED          SIZE
cpp           v0.2      9097d5ddcbc3   About a minute ago   904MB
```

从结果可以看到生成镜像非常的大，达到 900MB+。

实际上当我们把 `test.c` 编译完以后，并不需要一个大的 `GCC` 编译环境，一个小的运行环境镜像即可。这时候我们就可以使用多阶段构建解决这个问题。

```
Dockerfile
# 第一阶段构建
# AS 给第一阶段镜像取别名
FROM centos:7 as basebuilder
# 设置版本
ENV VERSION 1.0
#设置国内源
RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g' \
         -e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://mirro
rs.ustc.edu.cn/centos|g' \
         -i.bak \
         /etc/yum.repos.d/CentOS-Base.repo
RUN yum makecache
# 设置工作目录
WORKDIR /src
# 拷贝源文件
COPY demo.c .
# 安装 gcc
RUN yum install -y gcc && \
    gcc -v
# 编译源文件
RUN gcc demo.c -o demo && \
    rm -f demo.c && \
    yum erase -y gcc
# 运行可执行文件
CMD ["/src/demo"]

# 第二阶段构建
FROM centos:7
# 拷贝第一阶段生成的可执行程序
COPY --from=basebuilder /src/demo /src/demo
# 运行可执行程序
CMD ["/src/demo"]
```

构建镜像并创建容器测试结果

```
Shell
```

```
# 通过 Dockerfile 生成镜像 2.0
[zsc@VM-8-12-centos c_demo]$ docker image build -t multi:2.0 .

# 通过 Dockerfile 生成镜像 2.0
[zsc@VM-8-12-centos c_demo]$ docker run --name multi2 --rm
multi:v2.0
hello docker!

# 查看镜像信息
[zsc@VM-8-12-centos c_demo]$ docker image ls
REPOSITORY    TAG         IMAGE ID       CREATED          SIZE
multi         v2.0      4596e8b0755d   About a minute ago   204MB
```

可以看到这个生成的镜像是 204MB。

我们进一步优化可以选取较小的基础镜像像 alpine，busybox，debian 都有很小的体积，我们以 busybox 为例再次构建，Dockerfile 如下

```
C
# 第一阶段构建
# AS 给第一阶段镜像取别名
FROM centos:7 as basebuilder
# 设置版本
ENV VERSION 1.0
#设置国内源
RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g' \
         -e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://mirro
rs.ustc.edu.cn/centos|g' \
         -i.bak \
         /etc/yum.repos.d/CentOS-Base.repo
RUN yum makecache
# 设置工作目录
WORKDIR /src
# 拷贝源文件
COPY demo.c .
# 安装 gcc
RUN yum install -y gcc && \
    gcc -v
# 编译源文件
RUN gcc demo.c -o demo && \
    rm -f demo.c && \
    yum erase -y gcc
```

```
# 运行可执行文件
CMD ["/src/demo"]

# 第二阶段构建
FROM busybox
# 拷贝第一阶段生成的可执行程序
COPY --from=basebuilder /src/demo /src/demo
# 运行可执行程序
CMD ["/src/demo"]
```

构建镜像并创建容器测试结果,可以看到我们将镜像已经缩小到了不到 **5MB**

```
C
# 通过 Dockerfile 生成镜像 3.0
[zsc@VM-8-12-centos c_demo]$ docker image build -t multi:3.0 .

# 通过 Dockerfile 生成镜像 3.0
[zsc@VM-8-12-centos c_demo]$ docker run --name multi3 --rm
multi:v3.0
hello docker!

# 查看镜像信息
[zsc@VM-8-12-centos c_demo]$ docker image ls
REPOSITORY    TAG         IMAGE ID      CREATED          SIZE
multi         v3.0      a09ad0559775   2 minutes ago       4.87MB
```

多阶段构建可以很好的解决镜像的**层次多，体积大，部署时间长、维护性低**等问题。

对多阶构建为什么节省了空间?

因为我们编译使用的软件，都没有打到我们的运行态的软件里面，所以可以变得更小。