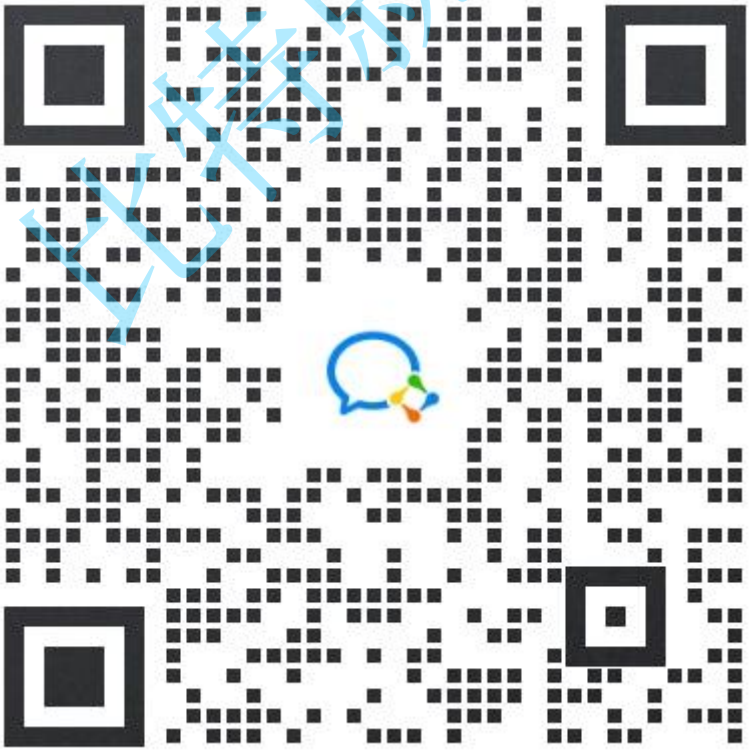
合理使用缓存

版权说明

对比特项目感兴趣，可以联系这个微信。



基础知识

- 在镜像的构建过程中，Docker 会根据 Dockerfile 指定的顺序执行每个指令。在执行每条指令之前，Docker 都会在缓存中查找是否已经存在可重用的镜像，如果有就使用现存的镜像，不会再重复创建。
- 在上边提到 Dockerfile 中的每一条指令都会产生一层 `image layer`。当某一个 layer 修改后，后面的所有 layer 我们都不能使用缓存， 这一点一定要注意。
- 如果不想在构建过程中使用缓存，你可以在 docker build 命令中使用 `--no-cache=true` 选项。但是我们建议最好合理使用缓存， 这样可以加快构建镜像的效率。

实战目的

掌握如何编写 Dockerfile 可以更好的利用缓存加速.

实战步骤

还是之前 C 语言的例子， 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。

- 编写 C 语言源代码

```C
#include <stdio.h>

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

- 编写 Dockerfile

```Dockerfile
# 选择基础镜像
FROM centos:7
# 设置工作目录
WORKDIR /src
# 拷贝源文件
COPY demo.c .
#设置国内源
RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g' \
         -e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://m
```

```
irrors.ustc.edu.cn/centos|g' \
         -i.bak \
         /etc/yum.repos.d/CentOS-Base.repo
RUN yum makecache
# 安装 gcc
RUN yum install -y gcc
# 编译源文件并删除源文件及 gcc 工具
RUN gcc demo.c -o demo && \
    rm -f demo.c && \
    yum erase -y gcc
# 运行可执行文件
CMD ["/src/demo"]
```

- 构建镜像

```
Shell
root@139-159-150-152:/data/myworkdir/dockerfile/cache#  docker
image build -t cache:1.0 .
[+] Building 80.3s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 584B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/centos:7
1.8s
 => [auth] library/centos:pull token for registry-1.docker.io
0.0s
 => [1/7] FROM
docker.io/library/centos:7@sha256:be65f488b7764ad3638f236b7b51
5b3678369a5124c47b8d32916d6487418ea4
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 27B
0.0s
 => CACHED [2/7] WORKDIR /src
0.0s
 => [3/7] COPY demo.c .
0.1s
 => [4/7] RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g'
```

```
-e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://m
irrors.ustc.edu.cn/centos|g'           -i.bak          0.4s
 => [5/7] RUN yum makecache
49.0s
 => [6/7] RUN yum install -y gcc
21.9s
 => [7/7] RUN gcc demo.c -o demo &&     rm -f demo.c &&
yum erase -y gcc
1.9s
 => exporting to image
4.9s
 => => exporting layers
4.9s
 => => writing image
sha256:4aca2a80d5302830d3ca55e6fbb6f0ddf99820a19cbe75c0af371bc
c62107559
0.0s
 => => naming to docker.io/library/cache:1.0
```

可以发现由于是我们第一次构建镜像，并没有使用到缓存，构建了整整 80s

- 改变源代码文件重新创建镜像

```
Shell
# 修改源文件输出 hello bit
[zsc@VM-8-12-centos cache_demo]$ cat demo.c
#include <stdio.h>

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

# 重新构建镜像
root@139-159-150-152:/data/myworkdir/dockerfile/cache# docker
image build -t cache:2.0 .
[+] Building 85.6s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 584B
0.0s
 => [internal] load .dockerignore
```

```
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/centos:7
0.8s
 => [1/7] FROM
docker.io/library/centos:7@sha256:be65f488b7764ad3638f236b7b51
5b3678369a5124c47b8d32916d6487418ea4
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 111B
0.0s
 => CACHED [2/7] WORKDIR /src
0.0s
 => [3/7] COPY demo.c .
0.1s
 => [4/7] RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g'
-e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://m
irrors.ustc.edu.cn/centos|g'          -i.bak          0.5s
 => [5/7] RUN yum makecache
27.8s
 => [6/7] RUN yum install -y gcc
50.6s
 => [7/7] RUN gcc demo.c -o demo &&     rm -f demo.c &&
yum erase -y gcc
1.4s
 => exporting to image
4.1s
 => => exporting layers
4.1s
 => => writing image
sha256:c8ca737f7ee80f215c3dd700db1c0cff6e0ac077f8ac9890a2c0aee
10ffc6eaf
0.0s
 => => naming to docker.io/library/cache:2.0
```

从上边的结果看起来这次依然没有用到缓存,时间还变长了变成了 85s， 还是要重新安装 gcc。那不对呀， 不是说在构建之前会查找可用的缓存吗， 按道理我们之前构建过， 这次应该能用到上次的缓存， 怎么没用到呢？这是因为我们修改了源文件 demo.c， 下面的所有指令都需要去重新构建。 那我们怎么避免这种情况呢？

可以调整 Dockerfile 命令执行的顺序，建议将不经常修改的内容放在 Dockerfile

的前边， 经常修改的内容放在后边。

- 调整 Dockerfile 的顺序

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

- 构建镜像

```
Shell
# 修改 dockerfile 之后第一次构建镜像依然不会使用到缓存
[zsc@VM-8-12-centos cache_demo]$ root@139-159-150-
152:/data/myworkdir/dockerfile/cache# docker image build -t
cache:3.0 .
[+] Building 37.8s (13/13) FINISHED
 => [internal] load build definition from Dockerfile
0.1s
 => => transferring dockerfile: 563B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/centos:7
```

```
1.7s
 => [auth] library/centos:pull token for registry-1.docker.io
0.0s
 => [1/7] FROM
docker.io/library/centos:7@sha256:be65f488b7764ad3638f236b7b51
5b3678369a5124c47b8d32916d6487418ea4
0.0s
 => [internal] load build context
0.1s
 => => transferring context: 27B
0.0s
 => CACHED [2/7] RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g'
-e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://m
irrors.ustc.edu.cn/centos|g'          -i.bak    0.0s
 => CACHED [3/7] RUN yum makecache
0.0s
 => [4/7] RUN yum install -y gcc
33.0s
 => [5/7] WORKDIR /src
0.1s
 => [6/7] COPY demo.c .
0.1s
 => [7/7] RUN gcc demo.c -o demo &&     rm -f demo.c &&
yum erase -y gcc
1.4s
 => exporting to image
1.6s
 => => exporting layers
1.5s
 => => writing image
sha256:c872644e087919176cc9d68940a0ba63c0f674f9dde2641f3dbf67c
8def8dbf1
0.0s
 => => naming to docker.io/library/cache:3.0
# 修改源文件 demo.c
[zsc@VM-8-12-centos cache_demo]$ cat demo.c
#include <stdio.h>

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

```
# 重新构建镜像
[zsc@VM-8-12-centos cache_demo]$ root@139-159-150-
152:/data/myworkdir/dockerfile/cache# docker image build -t
cache:4.0 .
[+] Building 2.5s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
0.0s
 => => transferring dockerfile: 563B
0.0s
 => [internal] load .dockerignore
0.0s
 => => transferring context: 2B
0.0s
 => [internal] load metadata for docker.io/library/centos:7
0.8s
 => [1/7] FROM
docker.io/library/centos:7@sha256:be65f488b7764ad3638f236b7b51
5b3678369a5124c47b8d32916d6487418ea4
0.0s
 => [internal] load build context
0.0s
 => => transferring context: 111B
0.0s
 => CACHED [2/7] RUN  sed -e 's|^mirrorlist=|#mirrorlist=|g'
-e
's|^#baseurl=http://mirror.centos.org/centos|baseurl=https://m
irrors.ustc.edu.cn/centos|g'           -i.bak   0.0s
 => CACHED [3/7] RUN yum makecache
0.0s
 => CACHED [4/7] RUN yum install -y gcc
0.0s
 => CACHED [5/7] WORKDIR /src
0.0s
 => [6/7] COPY demo.c .
0.1s
 => [7/7] RUN gcc demo.c -o demo &&     rm -f demo.c &&
yum erase -y gcc
1.3s
 => exporting to image
0.2s
 => => exporting layers
0.2s
 => => writing image
```

```
sha256:bd3d0f355dd035805c613d6f0a0649a565b679015bfb34757600a52
fb7cb18a0
0.0s
 => => naming to docker.io/library/cache:4.0
```

在重新构建镜像时发现此时已经可以使用缓存， 构建镜像的效率变得非常之高,我们只需要不到 **3s 就可以完成构建了**。