啥都不说了，先上一个看看

```
FROM registry.tongbanjie.com/library/tomcat:6.0.39_jdk6_test
MAINTAINER APPMD5:8c31576ba2b11a3ea028ebd07fbf9810
ENV APPNAME=tz-transfer
COPY docker-init/ /home/tomcat/docker-init/
RUN chmod 777 docker-init/init
COPY tz-transfer.war /data/www/ROOT
RUN unzip -o /data/www/ROOT/tz-transfer.war -d /data/www/ROOT/tz-transfer/
ENTRYPOINT ["/home/tomcat/docker-init/init"]
```

什么是dockerfile?

Dockerfile是一个包含用于组合映像的命令的文本文档。可以使用在命令行中调用任何命令。 Docker通过读取Dockerfile中的指令自动生成映像。

docker build命令用于从Dockerfile构建映像。可以在docker build命令中使用-f标志指向文件系统中任何位置的Dockerfile。

例：

```
docker build -f /path/to/a/Dockerfile
```

Dockerfile的基本结构

Dockerfile 一般分为四部分：基础镜像信息、维护者信息、镜像操作指令和容器启动时执行指令，'#' 为 Dockerfile 中的注释。

Dockerfile文件说明

Docker以从上到下的顺序运行Dockerfile的指令。为了指定基本映像，第一条指令必须是*FROM*。一个声明以 # 字符开头则被视为注释。可以在Docker文件中使用RUN，CMD，FROM，EXPOSE，ENV等指令。

在这里列出了一些常用的指令。

FROM： **指定基础镜像，必须为第一个命令**

格式：

```
FROM <image>
FROM <image>:<tag>
FROM <image>@<digest>
```

示例：

```
FROM mysql:5.6
```

注：

tag或digest是可选的，如果不使用这两个值时，会使用latest版本的基础镜像

MAINTAINER： 维护者信息

格式：

```
MAINTAINER <name>
```

示例：

```
MAINTAINER Jasper Xu
MAINTAINER sorex@163.com
MAINTAINER Jasper Xu <sorex@163.com>
```

RUN： 构建镜像时执行的命令

RUN用于在镜像容器中执行命令，其有以下两种命令执行方式：

shell执行

格式：

```
RUN <command>
```

exec执行

格式：

```
RUN ["executable", "param1", "param2"]
```

示例：

```
RUN ["executable", "param1", "param2"]
RUN apk update
RUN ["/etc/execfile", "arg1", "arg1"]
```

注：

RUN指令创建的中间镜像会被缓存，并会在下次构建中使用。如果不想使用这些缓存镜像，可以在构建时指定--no-cache参数，如：docker build --no-cache

ADD：将本地文件添加到容器中，tar类型文件会自动解压(网络压缩资源不会被解

压），可以访问网络资源，类似wget

格式：

```
    ADD <src>... <dest>
    ADD ["<src>",... "<dest>"] 用于支持包含空格的路径
```

示例：

```
    ADD hom* /mydir/          # 添加所有以"hom"开头的文件
    ADD hom?.txt /mydir/      # ? 替代一个单字符,例如："home.txt"
    ADD test relativeDir/     # 添加 "test" 到 `WORKDIR`/relativeDir/
    ADD test /absoluteDir/    # 添加 "test" 到 /absoluteDir/
```

COPY：功能类似ADD，但是是不会自动解压文件，也不能访问网络资源

CMD：构建容器后调用，也就是在容器启动时才进行调用。

格式：

```
    CMD ["executable","param1","param2"] (执行可执行文件，优先)
    CMD ["param1","param2"] (设置了ENTRYPOINT，则直接调用ENTRYPOINT添加参数)
    CMD command param1 param2 (执行shell内部命令)
```

示例：

```
    CMD echo "This is a test." | wc -
    CMD ["/usr/bin/wc","--help"]
```

注：

CMD不同于RUN，CMD用于指定在容器启动时所要执行的命令，而RUN用于指定镜像构建时所要执行的命令。

ENTRYPOINT：配置容器，使其可执行化。配合CMD可省去"application"，只使用参数。

格式：

```
    ENTRYPOINT ["executable", "param1", "param2"] (可执行文件，优先)
    ENTRYPOINT command param1 param2 (shell内部命令)
```

示例：

```
    FROM ubuntu
    ENTRYPOINT ["top", "-b"]
    CMD ["-c"]
```

注：

ENTRYPOINT与CMD非常类似，不同的是通过docker run执行的命令不会覆盖ENTRYPOINT，而docker run命令中指定的任何参数，都会被当做参数再次传递给ENTRYPOINT。Dockerfile中只允许有一个ENTRYPOINT命令，多指定时会覆盖前面的设置，而只执行最后的ENTRYPOINT指令。

LABEL： 用于为镜像添加元数据

格式：

```
    LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

示例：

```
   LABEL version="1.0" description="这是一个Web服务器" by="IT笔录"
```

注：

使用LABEL指定元数据时，一条LABEL指定可以指定一或多条元数据，指定多条元数据时不同元数据之间通过空格分隔。推荐将所有的元数据通过一条LABEL指令指定，以免生成过多的中间镜像。

ENV： 设置环境变量

格式：

```
    ENV <key> <value>  #<key>之后的所有内容均会被视为其<value>的组成部分，因此，一次只能设置一个变量
    ENV <key>=<value> ...  #可以设置多个变量，每个变量为一个"<key>=<value>"的键值对，如果<key>中包含空格，可以使用\来进行转义，也可以通过""来进行标示；另外，反斜线也可以用于续行
```

示例：

```
    ENV myName John Doe
    ENV myDog Rex The Dog
    ENV myCat=fluffy
```

EXPOSE：指定于外界交互的端口

格式：

```
EXPOSE <port> [<port>...]
```

示例：

```
EXPOSE 80 443
EXPOSE 8080
EXPOSE 11211/tcp 11211/udp
```

注：

EXPOSE并不会让容器的端口访问到主机。要使其可访问，需要在docker run运行容器时通过-p来发布这些端口，或通过-P参数来发布EXPOSE导出的所有端口

VOLUME：用于指定持久化目录

格式：

```
VOLUME ["/path/to/dir"]
```

示例：

```
VOLUME ["/data"]
VOLUME ["/var/www", "/var/log/apache2", "/etc/apache2"
```

注：

一个卷可以存在于一个或多个容器的指定目录，该目录可以绕过联合文件系统，并具有以下功能：

1 卷可以容器间共享和重用
2 容器并不一定要和其它容器共享卷
3 修改卷后会立即生效
4 对卷的修改不会对镜像产生影响
5 卷会一直存在，直到没有任何容器在使用它

WORKDIR：工作目录，类似于cd命令

格式：

```
    WORKDIR /path/to/workdir
```

示例：

```
    WORKDIR /a  (这时工作目录为/a)
    WORKDIR b  (这时工作目录为/a/b)
    WORKDIR c  (这时工作目录为/a/b/c)
```

注：

通过WORKDIR设置工作目录后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT、ADD、COPY等命令都会在该目录下执行。在使用docker run运行容器时，可以通过-w参数覆盖构建时所设置的工作目录。

USER：指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户。使用USER指定用户时，可以使用用户名、UID或GID，或是两者的组合。当服务不需要管理员权限时，可以通过该命令指定运行用户。并且可以在之前创建所需要的用户

格式：

```
    USER user
    USER user:group
    USER uid
    USER uid:gid
    USER user:gid
    USER uid:group
```

示例：

```
    USER www
```

注：

使用USER指定用户后，Dockerfile中其后的命令RUN、CMD、ENTRYPOINT都将使用该用户。镜像构建完成后，通过docker run运行容器时，可以通过-u参数来覆盖所指定的用户。

ARG： 用于指定传递给构建运行时的变量

格式：

```
    ARG <name>[=<default value>]
```

示例：

```
    ARG site
```

```
ARG build_user=www
```

ONBUILD： 用于设置镜像触发器

格式：

```
ONBUILD [INSTRUCTION]
```

示例：

```
ONBUILD ADD . /app/src
ONBUILD RUN /usr/local/bin/python-build --dir /app/src
```

注：

当所构建的镜像被用做其它镜像的基础镜像，该镜像中的触发器将会被钥触发

以下是一个小例子：

```
# This my first nginx Dockerfile
# Version 1.0

# Base images 基础镜像
FROM centos

#MAINTAINER 维护者信息
MAINTAINER tianfeiyu

#ENV 设置环境变量
ENV PATH /usr/local/nginx/sbin:$PATH

#ADD  文件放在当前目录下，拷过去会自动解压
ADD nginx-1.8.0.tar.gz /usr/local/
ADD epel-release-latest-7.noarch.rpm /usr/local/

#RUN 执行以下命令
RUN rpm -ivh /usr/local/epel-release-latest-7.noarch.rpm
RUN yum install -y wget lftp gcc gcc-c++ make openssl-devel pcre-
devel pcre && yum clean all
RUN useradd -s /sbin/nologin -M www
```

```
#WORKDIR 相当于cd
WORKDIR /usr/local/nginx-1.8.0

RUN ./configure --prefix=/usr/local/nginx --user=www --group=www --
with-http_ssl_module --with-pcre && make && make install

RUN echo "daemon off;" >> /etc/nginx.conf

#EXPOSE 映射端口
EXPOSE 80

#CMD 运行以下命令
CMD ["nginx"]
```

最后用一张图解释常用指令的意义^-^