具体不懂请查看：

https://yeasy.gitbooks.io/docker_practice/image/dockerfile/entrypoint.html

可以理解为dockerFile里面几乎什么都可以做 dockerFile的基本操作。

DockerFile可以给我们构建定制的镜像

```
FROM nginx
RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
```

这个 Dockerfile 很简单，一共就两行。涉及到了两条指令， `FROM` 和 `RUN` 。

基本指令：

1.COPY package.json /usr/src/app，将上下文目录中的文件复制到镜像内的位置

2.ADD ubuntu-xenial-core-cloudimg-amd64-root.tar.gz / add：的源路径可以为一个URL，docker引擎会试图去下载这个链接中的文件，然后将文件权限自动设置为600

3.

如果使用 `shell` 格式的话，实际的命令会被包装为 `sh -c` 的参数的形式进行执行。比如：

```
CMD echo $HOME
```

在实际执行中，会将其变更为：

```
CMD [ "sh", "-c", "echo $HOME" ]
```

这就是为什么我们可以使用环境变量的原因，因为这些环境变量会被 shell 进行解析处理。

4.ENTRYPOINT 入口点

1.利用CMD 命令不能接收到传递进来的参数，而利用ENTRYPOINT可以

```
FROM ubuntu:16.04
RUN apt-get update \
    && apt-get install -y curl \
    && rm -rf /var/lib/apt/lists/*
ENTRYPOINT [ "curl", "-s", "http://ip.cn" ]
```

这次我们再来尝试直接使用 `docker run myip -i` ：

会将-i这个参数传递给ENTRYPOIN，变成curl -s http://ip.cn -i

2.想事先启动某个脚本

```
FROM alpine:3.4
...
RUN addgroup -S redis && adduser -S -G redis redis
...
ENTRYPOINT ["docker-entrypoint.sh"]

EXPOSE 6379
CMD [ "redis-server" ]
```

可以看到其中为了 redis 服务创建了 redis 用户，并在最后指定了 `ENTRYPOINT` 为 `docker-entrypoint.sh` 脚本。

```
#!/bin/sh
...
# allow the container to be started with `--user`
if [ "$1" = 'redis-server' -a "$(id -u)" = '0' ]; then
    chown -R redis .
    exec su-exec redis "$0" "$@"
fi

exec "$@"
```

5. ENV 设置环境变量

ENV 设置环境变量

格式有两种：

- `ENV <key> <value>`
- `ENV <key1>=<value1> <key2>=<value2>...`

这个指令很简单，就是设置环境变量而已，无论是后面的其它指令，如 `RUN`，还是运行时的应用，都可以直接使用这里定义的环境变量。

```
ENV VERSION=1.0 DEBUG=on \
    NAME="Happy Feet"
```

这个例子中演示了如何换行，以及对含有空格的值用双引号括起来的办法，这和 Shell 下的行为是一致的。

定义了环境变量，那么在后续的指令中，就可以使用这个环境变量。比如在官方 `node` 镜像 `Dockerfile` 中，就有类似这样的代码：

```
ENV NODE_VERSION 7.2.0

RUN curl -SLO "https://nodejs.org/dist/v$NODE_VERSION/node-v$NODE_VERSION-linux-x64.tar.xz
  && curl -SLO "https://nodejs.org/dist/v$NODE_VERSION/SHASUMS256.txt.asc" \
  && gpg --batch --decrypt --output SHASUMS256.txt SHASUMS256.txt.asc \
  && grep " node-v$NODE_VERSION-linux-x64.tar.xz\$" SHASUMS256.txt | sha256sum -c - \
  && tar -xJf "node-v$NODE_VERSION-linux-x64.tar.xz" -C /usr/local --strip-components=1 \
  && rm "node-v$NODE_VERSION-linux-x64.tar.xz" SHASUMS256.txt.asc SHASUMS256.txt \
  && ln -s /usr/local/bin/node /usr/local/bin/nodejs
```

在这里先定义了环境变量 `NODE_VERSION`，其后的 `RUN` 这层里，多次使用 `$NODE_VERSION` 来进行操作定制。可以看到，将来升级镜像构建版本的时候，只需要更新 `7.2.0` 即可，`Dockerfile` 构建维护变得更轻松了。

下列指令可以支持环境变量展开：
`ADD`、`COPY`、`ENV`、`EXPOSE`、`LABEL`、`USER`、`WORKDIR`、`VOLUME`、`STOPSIGNAL`、`ONBUILD`。

6.VULUME 定义匿名卷

```
docker run -d -v mydata:/data xxxx
```

在这行命令中，就使用了 `mydata` 这个命名卷挂载到了 `/data` 这个位置，替代了 `Dockerfile` 中定义的匿名卷的挂载配置。

7.EXPOSE 声明端口

对于docker容器默认运行于桥接网络中，因此所有容器互相之间都可以直接访问，这样存在安全性的问题。

因此当不为桥接模式时，links的容器才能互通，并且只有在EXPOSE所声明的端口才可以被访问。

格式：EXPOSE <端口1> [<端口2>...]

8.WORKDIR 指定工作目录

WORKDIR 指定工作目录

格式为 `WORKDIR <工作目录路径>` 。

使用 `WORKDIR` 指令可以来指定工作目录（或者称为当前目录），以后各层的当前目录就被改为指定的目录，如该目录不存在， `WORKDIR` 会帮你建立目录。

之前提到一些初学者常犯的错误是把 `Dockerfile` 等同于 Shell 脚本来书写，这种错误的理解还可能会导致出现下面这样的错误：

```
RUN cd /app
RUN echo "hello" > world.txt
```

如果将这个 `Dockerfile` 进行构建镜像运行后，会发现找不到 `/app/world.txt` 文件，或者其内容不是 `hello` 。原因其实很简单，在 Shell 中，连续两行是同一个进程执行环境，因此前一个命令修改的内存状态，会直接影响后一个命令；而在 `Dockerfile` 中，这两行 `RUN` 命令的执行环境根本不同，是两个完全不同的容器。这就是对 `Dockerfile` 构建分层存储的概念不了解所导致的错误。

之前说过每一个 `RUN` 都是启动一个容器、执行命令、然后提交存储层文件变更。第一层 `RUN cd /app` 的执行仅仅是当前进程的工作目录变更，一个内存上的变化而已，其结果不会造成任何文件变更。而到第二层的时候，启动的是一个全新的容器，跟第一层的容器更完全没关系，自然不可能继承前一层构建过程中的内存变化。

因此如果需要改变以后各层的工作目录的位置，那么应该使用 `WORKDIR` 指令。