

目录

1. 一键生成镜像脚本	1
1.1. 基础镜像脚本	1
1.2. QT4 运行时镜像脚本	1
1.3. Qt Creator（包含 QT4）运行时镜像脚本	1
2. QT4 以及 Qt Creator 镜像的运行	1
2.1. 基础镜像运行	1
2.2. QT4 镜像运行	2
2.2.1. Dockerfile 编写	2
2.2.2. 打包 Qt 应用程序创建镜像	3
2.2.3. 运行 Qt 应用程序	3
2.3. Qt Creator 镜像运行	3

1. 一键生成镜像脚本

1.1. 基础镜像脚本

此脚本的目的是一键生成包含 vim、net-tools、procps、zip、unzip、ping、less、ls 等基础命令的最小镜像；脚本包括四个文件：mini.sh、slimify-excludes、slimify-includes、tar-exclude，使用 sudo 执行 mini.sh，以镜像名字作为参数。

```
sudo bash mini.sh baseimage
```

此脚本执行后，会自动把 baseimage 镜像导入到 docker 内，并在本地以 tar.gz 格式保存。该脚本生成的基础镜像导入 docker 的大小为 93M，tar.gz 文件为 35M。

1.2. QT4 运行时镜像脚本

此脚本的目的是一键生成包含 make、g++、qt4 等 qt 应用编译及运行必需的包；脚本包括四个文件：qt4.sh、slimify-excludes、slimify-includes、tar-exclude，使用 sudo 执行 qt4.sh，以镜像名字作为参数。

```
sudo bash qt4.sh qt4-dev
```

此脚本执行后，会自动把 qt4-dev 镜像导入到 docker 内，并在本地以 tar.gz 格式保存。该脚本生成的 Qt4 镜像导入 docker 的大小为 860M，tar.gz 文件为 485M。

1.3. Qt Creator（包含 QT4）运行时镜像脚本

此脚本的目的是一键生成包含 make、g++、qt4、qtcreator 等 qtcreator 开发 qt4 应用所必需的基础包；脚本包括四个文件：qtcreator.sh、slimify-excludes、slimify-includes、tar-exclude，使用 sudo 执行 qtcreator.sh，以镜像名字作为参数。

```
sudo bash qtcreator.sh qtcreator-dev
```

此脚本执行后，会自动把 qtcreator 镜像导入到 docker 内，并在本地以 tar.gz 格式保存。该脚本生成的 Qt Creator 镜像导入 docker 的大小为 1.79G，tar.gz 文件为 884M。

2. QT4 以及 Qt Creator 镜像的运行

2.1. 基础镜像运行

使用 docker images 来列出本地主机上的镜像，找到基础镜像 baseimage。使用 docker run 命令创建容器并运行，命令如下。

```
docker run -it baseimage:latest bash
```

```

user@user-F300-G30:~/build/sh$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
qtcreator-0.1       latest             95b6f42500e7       4 minutes ago      1.79GB
qt4-0.9             latest             cf685f7b9109       22 minutes ago      860MB
qt4-demo            latest             3fe0d5374b04       28 minutes ago      861MB
qt4-0.3             latest             37a19a05a1bb       About an hour ago    860MB
qt4-0.1             latest             c14da14efa38       About an hour ago    1.18GB
baseimage           latest             6c13566e7094       About an hour ago    93.1MB
user@user-F300-G30:~/build/sh$ docker run -it baseimage:latest bash
root@5ee6b26e291b:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 172.17.0.3  netmask 255.255.0.0  broadcast 172.17.255.255
    ether 02:42:ac:11:00:03  txqueuelen 0  (Ethernet)
    RX packets 19  bytes 3086 (3.0 KiB)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    loop txqueuelen 1  (Local Loopback)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

root@5ee6b26e291b:/#

```

2.2. QT4 镜像运行

2.2.1. Dockerfile 编写

Dockerfile 文件与 Qt 应用源码在同一目录（非必要），目录结构如下。

```

├── Dockerfile
├── helloworld
│   ├── helloworld.pro
│   ├── helloworld.pro.user
│   ├── main.cpp
│   ├── mainwindow.cpp
│   ├── mainwindow.h
│   └── mainwindow.ui

```

Dockerfile 内容如下。

```

FROM qt4-dev:latest

COPY ./helloworld/ /home/helloworld

WORKDIR /home/helloworld

RUN ls /home/helloworld

RUN qmake helloworld.pro; make

RUN chmod +x /home/helloworld/helloworld; ls /home/helloworld/

CMD /home/helloworld/helloworld

```

2.2.2.打包 Qt 应用程序创建镜像

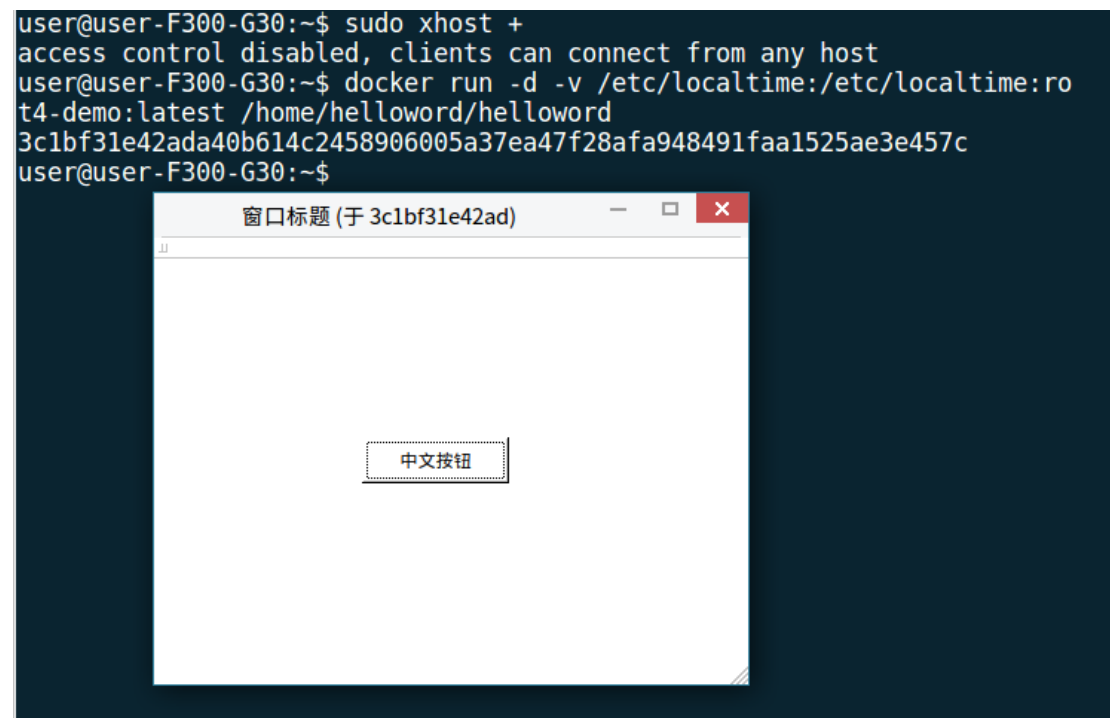
在 Dockerfile 文件的目录下，执行 `docker build` 命令，将 Qt 应用程序打包并创建镜像，命令如下。

```
docker build -t qt4-demo:new .
```

2.2.3.运行 Qt 应用程序

上一步命令会在 `docker` 内生成一个 `qt4-demo` 镜像，`tag` 为 `new`，使用命令创建容器并对导入的 Qt 应用程序编译运行，命令如下。

```
sudo xhost +
docker run -d -v /etc/localtime:/etc/localtime:ro -v /tmp/.X11-unix:/tmp/.X11-unix -d -e LANG=C.UTF-8 -e DISPLAY=unix$DISPLAY -e GDK_SCALE -e GDK_DPI_SCALE qt4-demo:new /home/helloworld/helloworld
```



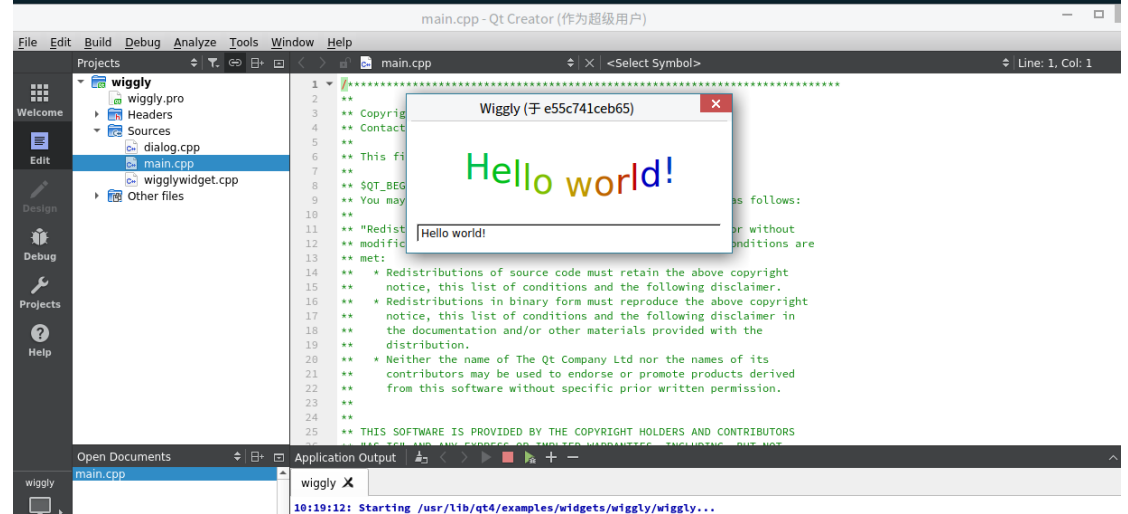
程序关闭后，容器会停止运行，需要再次启动该程序，使用 `docker start` 启动容器即可。

2.3. Qt Creator 镜像运行

直接使用 `docker run` 命令创建 Qt Creator 容器并启动，命令如下。

```
sudo xhost +
docker run -d -v /etc/localtime:/etc/localtime:ro -v /tmp/.X11-unix:/tmp/.X11-unix -d -e LANG=C.UTF-8 -e DISPLAY=unix$DISPLAY -e GDK_SCALE -e GDK_DPI_SCALE qtcreator:latest qtcreator
```

```
user@user-F300-G30:~$ sudo xhost +
access control disabled, clients can connect from any host
user@user-F300-G30:~$ docker run -d -v /etc/localtime:/etc/localtime:ro -v /tmp/.X11-unix:/tmp/.X11-unix -d -e LANG=
UTF-8 -e DISPLAY=unix$DISPLAY -e GDK_SCALE -e GDK DPI SCALE qtcreator-0.1:latest qtcreator
38388e44ab00104c434cffff9dd309099c8cc63e506c32ce74bed28657e91157
user@user-F300-G30:~$
```



程序关闭后，容器会停止运行，需要再次启动该程序，使用 `docker start` 启动容器即可。