参与 Docker 项...

本篇是第八部分“生态篇”的第四篇。在这个部分，我会为你介绍 Docker 生态中的相关项目，以及如何参与到 Docker 项目中，最后会聊聊 Docker 未来的走向。上一篇，我为你介绍了更底层的容器运行时 runc。本篇，我将为你介绍如何参与到 Docker 项目的开发中。

回顾

Docker 到现在已经 7 岁了，其代码组织和结构均已发生了不少的变化。

最初 Docker 只是一个独立的二进制文件，而不是像现在这样有 Docker CLI 和 Docker Daemon 这样的拆分。并且在更早之前 Docker Daemon 模式是直接选择监听在 127.0.0.1:4242 的，而不是像现在默认使用 UNIX Socket。

基本准则

在聊到具体的项目之前，我先来为你介绍下 Docker 社区的规范和基本准则吧。要参与到社区，就必须要遵循社区的规范。

- **保持友善。**Docker 社区是一个开放的社区，请对任何社区成员保持礼貌和尊重。
- **鼓励多样性参与。**Docker 社区鼓励大家积极参与贡献，无论他的背景如何。
- **保持合法。**不要在社区中发布私人或者敏感信息，并且不要违反法律。
- **紧贴主题。**在你发送消息的时候，请注意选择到正确的位置发布。因为你在发布消息的时候，可能会有很多人收到提醒，无论如何，大家都不喜欢垃圾邮件。
- **不要直接给维护者发送邮件。**这里其实不单是指邮件，而是指任何私聊的功能。请尽可能的使用 GitHub 上的 @ 功能，而不要直接去发邮件打扰维护者。

编码风格

Docker 项目的代码是使用 Go 语言编写的，所以 Go 社区的编码指南也同样适用于 Docker 项目中。但是，记住一个 Docker 社区的额外准则，就是使代码库更容易被人类浏览和理解。

最基本的规则如下：

- 所有的代码应该都通过 `gofmt -s`；
- 所有的代码都通过默认级别的 `golint`；
- 代码中应该添加注释，以便大家知道其添加的原因和背景；
- 所有的代码应该符合 Go 社区 Effective Go 和 CodeReviewComments 的规范；

你可能已经发现了，在编码方面 Docker 社区是推荐遵循 Go 社区的编码规范的，并且上述的规则基本符合大多数开源社区的规则。

Docker 项目的组织方式

docker-ce

当前我们使用的 Docker 版本，其项目代码和 release 均存储在 https://github.com/docker/docker-ce 仓库中。

项目结构很直观：

复制

```
(MoeLove) ➜  docker-ce git:(master) tree -L 2
.
├── CHANGELOG.md
├── components
│   ├── cli
│   ├── engine
│   └── packaging
├── components.conf
├── CONTRIBUTING.md
├── Makefile
├── README.md
└── VERSION

4 directories, 6 files
```

components 目录下包含三个主要的组件：

- cli 是 Docker CLI 项目的源代码
- engine 是 Docker Engine 项目的源代码
- packaging 是用于 Docker CE 打包构建所用的配置

并且，其中的 components.conf 文件中，配置了各组件所用的仓库地址和所使用的分支：

复制

```
[component "packaging"]
    url = git@github.com:docker/docker-ce-packaging
    branch = master
[component "engine"]
    url = git@github.com:moby/moby
    branch = master
[component "cli"]
    url = git@github.com:docker/cli
    branch = master
```

此仓库中的代码是自动从各组件的上游仓库同步过来的，如果有遇到什么问题，则需要到对应的上游项目中去提交 Issue 或者 Pull Request。

Docker Engine

Docker Engine 也就是我们平时提到的 Docker Daemon 及其相关的组件，之前使用的代码仓库是 https://github.com/docker/engine，但现在经过讨论，决定还是直接使用 https://github.com/moby/moby，最终目的是将 docker/engine 项目归档。

此项目中的包含一个特殊的目录：rootless。它包含着 Docker Engine 的 rootless 模式，并且其二进制也是独立进行分发的。

如果想要为此项目做贡献，当修改代码后，可直接在代码目录中执行 `make BIND_DIR=. shell`，这样便可以启动一个容器，作为 Docker 的开发环境。

这个容器中，包含了构建 Docker 二进制相关的依赖，你可以在此环境中完成开发、构建及测试等。

Docker CLI

Docker CLI 就是我们平时所使用的 Docker 工具，代码仓库是

> https://github.com/docker/cli

其入口是 cmd/docker/docker.go，我们使用的各命令则在 cli 目录下。

在你修改代码后，可通过执行 `make -f docker.Makefile binary` 来构建二进制文件，以进行相关功能的测试等。

其他组件

除了上述几个 Docker 的核心仓库外，还有不少很重要的组件的仓库：

- containerd：https://github.com/containerd/containerd/ Docker 中容器生命周期管理相关的功能组件
- runc：https://github.com/opencontainers/runc Docker 底层的容器运行时
- libnetwork：https://github.com/docker/libnetwork Docker network 的核心代码库
- buildkit：https://github.com/moby/buildkit Docker 下一代的构建系统

总结

本篇，我为你介绍了 Docker 相关项目的项目结构和组织方式。

如果想要参与 Docker 上游项目的开发，只需按照我在本文中的介绍，定位到自己想要修改的上游项目，并修改代码，提交 Pull Request 即可。

另外，在提交 Pull Request 的时候，需要签署 DCO（Developer Certificate of Origin），具体说明请参考 https://developercertificate.org/。在进行 `git commit` 操作时，可直接使用 `-s` 选项进行签署。

下一篇，便是本专栏的最后一篇了，我将会和你谈谈 Docker 生态与未来走向。