Fabric 从2.0 开始支持三种chaincode 运行方式，下面我们就分别介绍这三种chaincode 运行方式。

1：chaincode 以docker 的方式运行，这种方式是chaincode 最原始也是最基本的运行方式。当application 端实例化链码，或者是invoke chaincode 的时候，peer节点会运行一个chaincode docker.此时启动的chaincode docker 会作为一个client 端主动去peer 端完成chaincode 注册。

Fabric1.x 的阶段使用的便是这种运行方式。但是这种方法也是存在一些问题：

1）那就是启动的chaincode docker 与peer 节点必须是要运行在同一台服务器上，如果当我们的chaincode 数量较多的时候，对节点服务器资源的要求比较大。

3）此时peer 节点就类似一个chaincode 编排工具，需要维护chaincodes 的整个生命周期，以及connection 管理，这无形之中也是给peer 节点代码维护带来很多困扰。

3）不利于对chaincode的统一管理，同时一个chaincode 也只能连接一个peer 节点这在资源上也是一种浪费。

4）无法对chaincode 进行横向扩容。

而且当我们以docker的方式运行链码，

2：fabric 2.0 使用脚本编译启动chaincode。 此时chaincode 是一个运行在peer 宿主机上的二进制文件。并且chaincode 同样需要以client的身份就行运行。虽然当前的chaincoed不在是以docker的形式运行。但是peer节点同样是需要维护和管理chaincode的整个生命周期。并且由于chaincode 是直接运行在宿主机上（当然也可以和peer 运行在同一个docker 中）peer docker 直接在宿主机编译和启动chaincode 这会引出许多新的问题。当然如果peer 不是以docker的形式进行启动，问题就会解决。

3：fabric 2.0 支持的另一种脚本链接chaincode 方式，大家在这里应该注意到我在这里使用的是链接，而不是前面说的编译启动。为什么这样说呢，因为第三种方式chaincode 是以server 的形式启动。当我们invoke or init chaincode的时候Peer节点将作为一个client端去主动连接chaincode server.

当我们使用这种方式来运行chaincode 的时候，这会解决很多困扰我们的问题，如；此时peer 节点就不用去维护chiancode 的生命周期，我们可以将维护chaincode的功能交给k8s等更加专业的程序编排工具。可以大大减少peer节点维护chaincode 的压力。同时chaincode 也可以进行扩容，实现更多的业务功能。

注：第二种peer 以docker的形式直接在宿主机上编译和启动chaincode 。测试失败。网上成功的例子也都是peer以可执行程序的方式直接运行在宿主机上，然后编译启动chaincode.

这里强烈推荐使用第三种chaincode 启动方式。