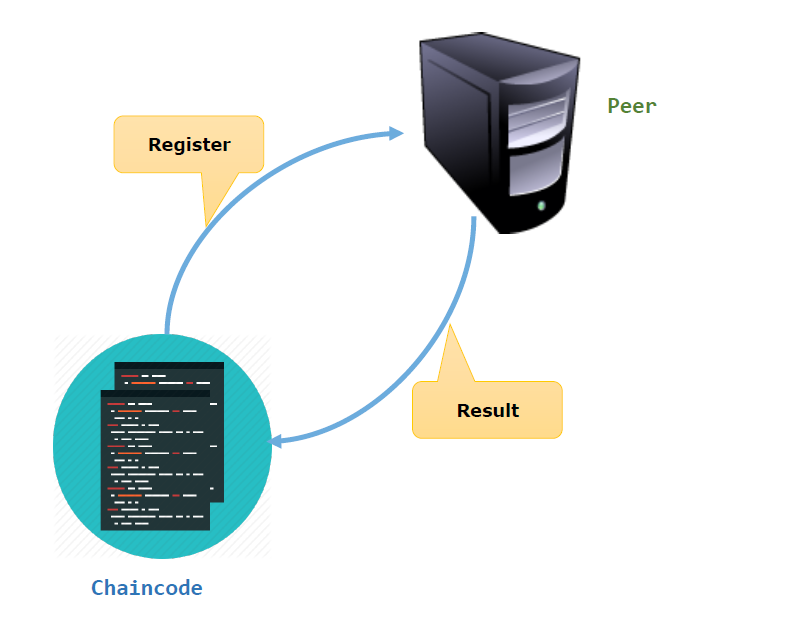
在 fabric 1.0中，chaincode存放在peer中，chaincode本质上是预先定义的fabric接口的实现。

|  |
| --- |
| type Chaincode interface {  //，**对应于deploy transaction时触发**  Init(stub ChaincodeStubInterface) pb.Response  // 查询或更更新world state**对应于Invoke transaction 和Query transaction时触发**  Invoke(stub ChaincodeStubInterface) pb.Response  } |

例如： 编写输出hello world 的chaincode

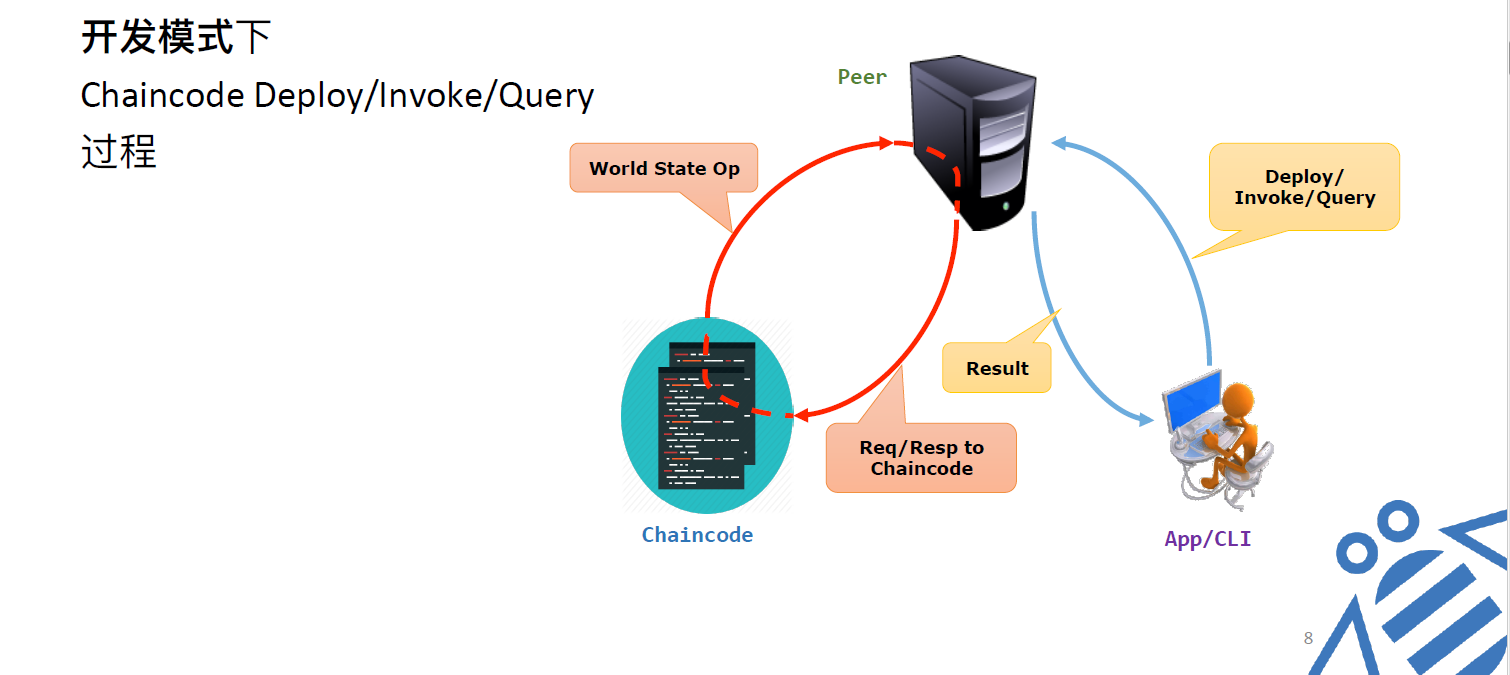
|  |
| --- |
| package main  import (  "errors"  "fmt"  "strconv"  "github.com/hyperledger/fabric/core/chaincode/shim"  )  type HelloWorldChaincode struct {  }  func (t \*HelloWorldChaincode) **Init(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {**  **fmt.Printf("HelloWorld - Init called with function %s!\n", function)**  **return nil, nil**  **}**  func (t \*HelloWorldChaincode) **Invoke(stub shim.ChaincodeStubInterface, function string, args []string) ([]byte, error) {**  **fmt.Printf("HelloWorld - Invoke called with function %s!\n", function)**  **return nil, nil**  **}**  func main() {  err := shim.Start(new(HelloWorldChaincode))  if err != nil {  fmt.Printf("Error starting Hello World chaincode: %s", err)  }  } |

Chaincode的注册过程：



假设一个用户想要创建一个新的智能合约，他在本地编写好代码后将相关程序（chaincode）提交各peer（通常要提交给4个peer节点）。

Chaincode的调用过程：



假设，一个是当前chaincode的一个参与人员，他发起一个popsal transaction给背书节点。调用的方法是Query。调用的chaincode hash值是0x3214。背书节点收到这个请求后，调用他本地存储的相应chaincode，进行模拟运行，如果运行合法，就将运行结果返回给客户端，客户端将结构转发给order node并由进行共识验证，之后order node排序打包成Block发布出来，最后由各个节点进行检验。