



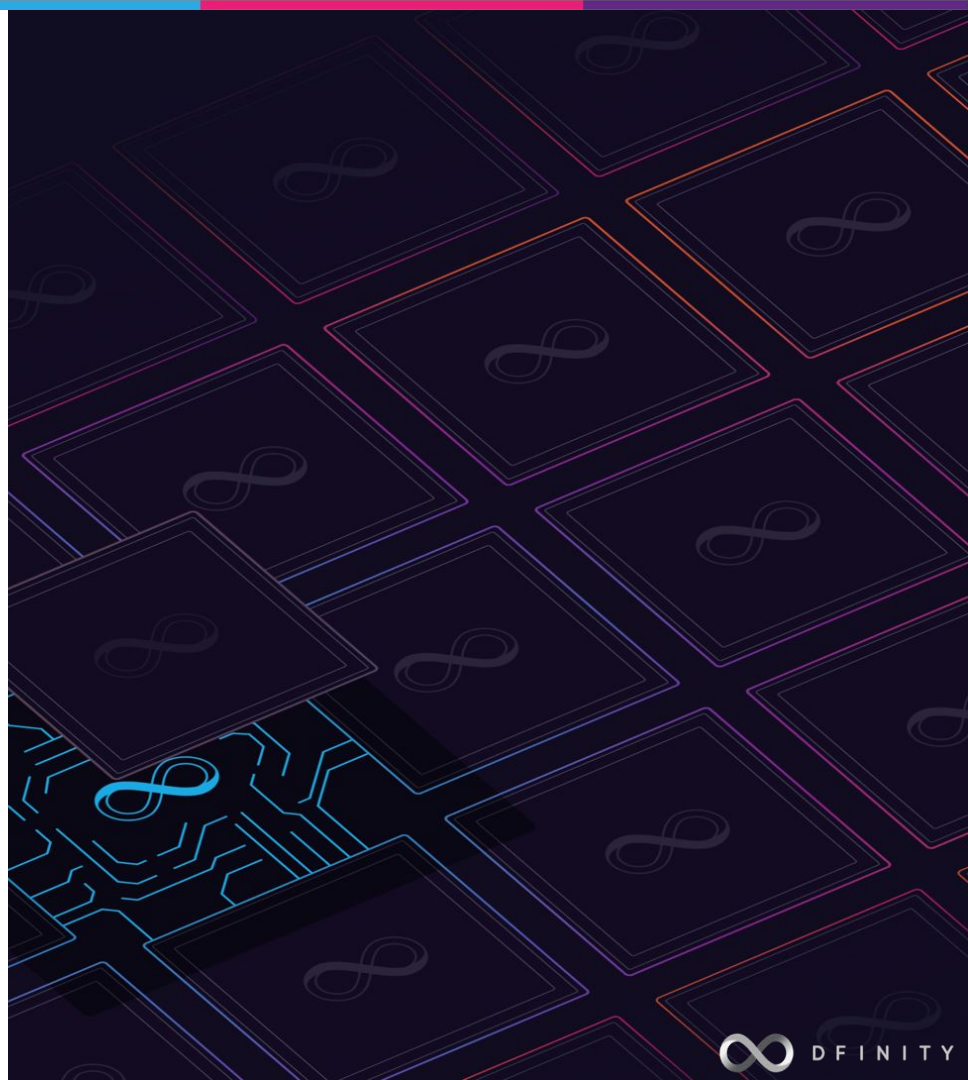
# ICP区块链开发入门课程

3. Canister 智能合约

主讲: Paul Liu - DFINITY 工程师

# 课程大纲

1. 使用 SDK 搭建一个简易网站
2. Motoko 语言简介
3. Canister 智能合约
4. 用 Motoko 做后端
5. 用 Javascript 做前端



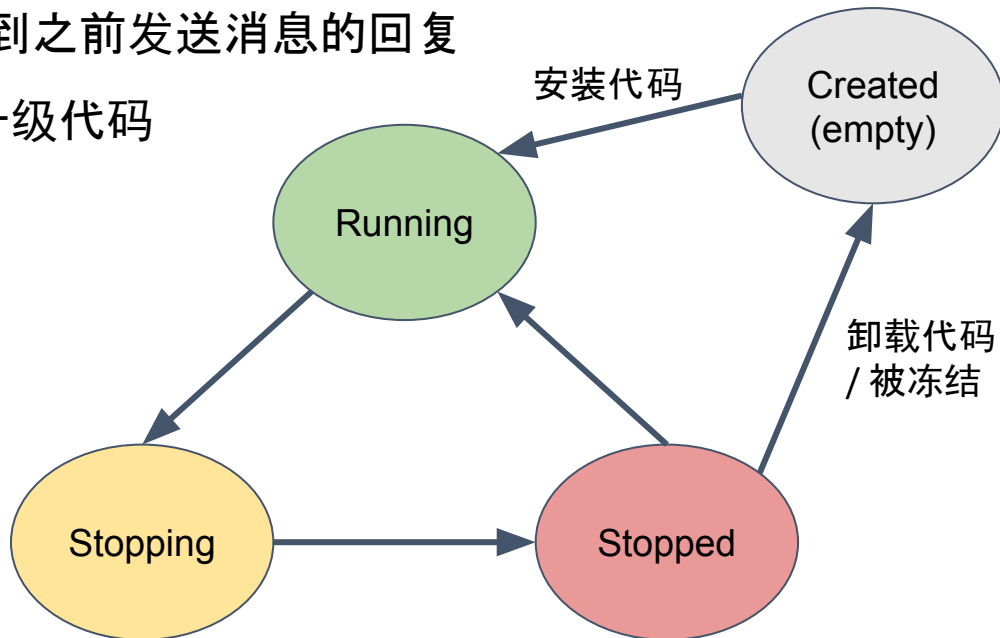
# Canister 智能合约的结构

Canister Id (标识符, 全局唯一)			
元数据 (metadata)			
控制者名单 (Controllers)	Cycle 余额 (128-bit)	当前状态 (status)	资源配置 (resource)
代码 (Wasm bytecode)			
运行时数据			
堆内存 (memory heap)		稳定内存 (stable memory)	
消息队列 (message queue)		调用相关 (call context)	

# Canister 的生命周期

- 已创建 (Created), 无代码, 无内存, 不能接收和发送消息
- 正常运行 (Running), 可以接收和发送消息
- 预备停止 (Stopping), 只允许收到之前发送消息的回复
- 停止运行 (Stopped), 此时可以升级代码

```
dfx canister create  
dfx canister install  
dfx canister stop  
dfx canister start  
dfx canister uninstall-code  
dfx canister delete
```



# Cycles 计费标准

智能合约 Canister 持有 cycles 并为计算与存储付费

价格相对稳定  $1 \text{ SDR} = 1 \text{ Trillion Cycles} = 10^{12} \text{ Cycles}$

Canister 可以相互发送和接收 Cycles (随消息一起)

计费标准由 NNS 控制

- 接收与发送消息(次), 消息的长度(字节数)
- 方法调用(次), 完成运行所执行的指令数量(和类别)
- 创建 Canister(次)
- 存储消耗(字节数、时间), 目前 1GB 约 \$0.46/月
- 预留计算资源(百分比)

**余额不足维系 30 天时 Canister 会被冻结 (frozen), 余额为零则会被删除**

# Canister 的公共接口(方法调用)

Canister 只有接收到消息才会开始运行

每个消息相当于一次异步的方法调用

可以对收到的消息进行答复，对方收到答复也是一次方法调用

只有当一次调用成功完成时，状态才会保存，对外的消息才会发出(原子性)

Canister 的公共接口任何人都可以调用

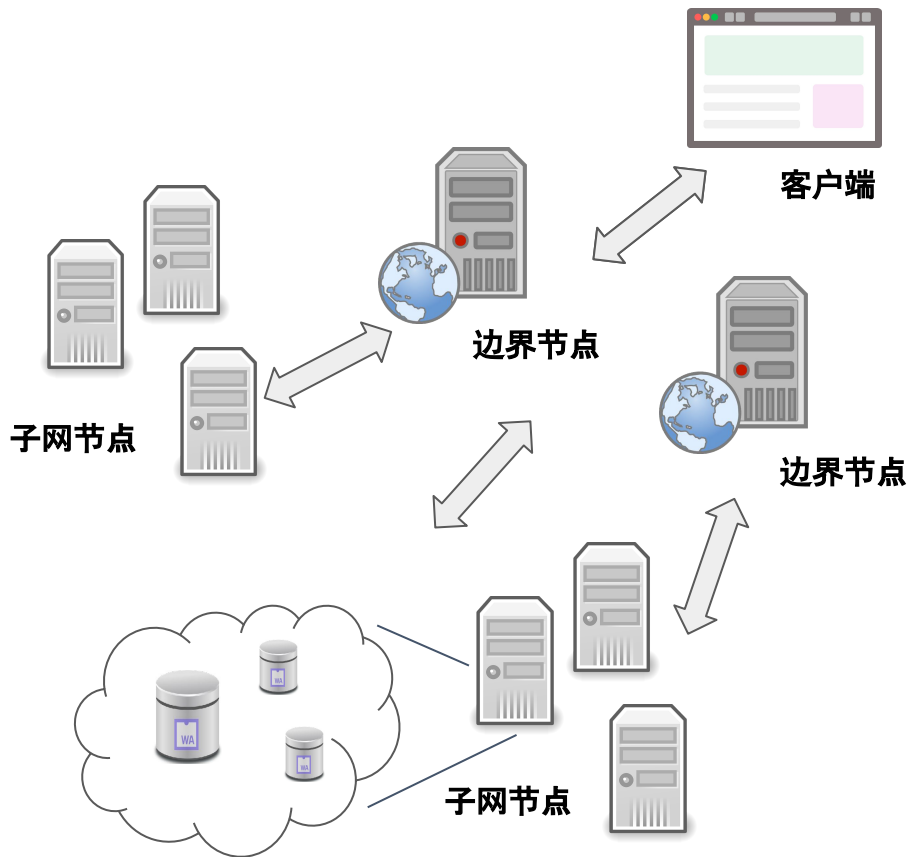




# Cycle 实例演示

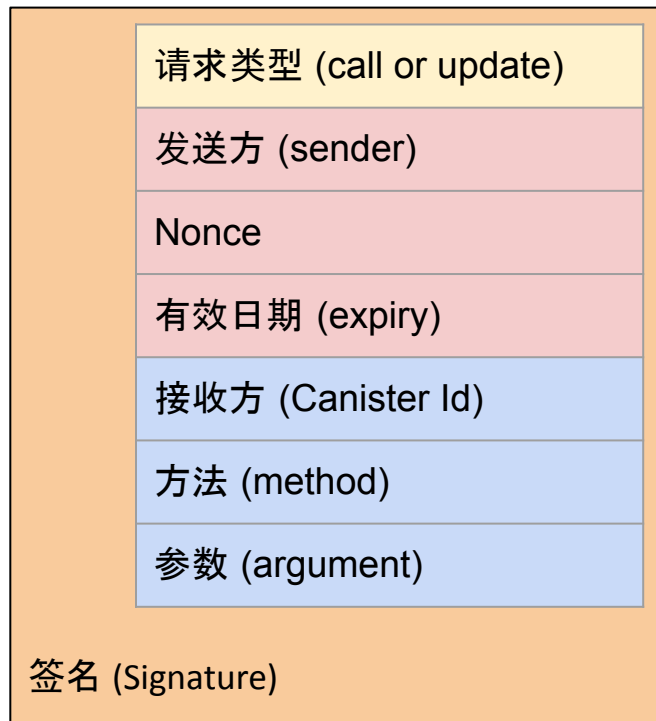
# Canister 调用方式 (Update vs. Query)

	更新调用 Update Call	查询调用 Query Call
共识	需要	不需要
安全性	高	低
响应时间	2~3 秒	~100毫秒
状态改变	持久化	不保存
执行方式	顺序	并行
调用方式	两步	一步





# 给 Canister 发送消息请求



[https://ic0.app/api/v2/canister/<canister\\_id>/call](https://ic0.app/api/v2/canister/<canister_id>/call)

HTTP POST (CBOR 编码)

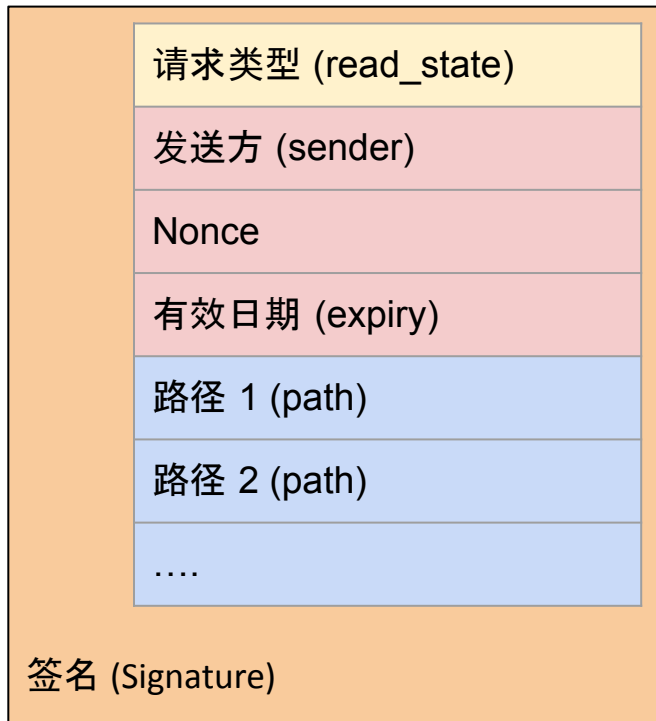
用户签名支持 ed25519 和 secp256k1

有效期需要在五分钟内

**Principal Id:** 通用身份标识

- Sender Id 代表用户身份, 通常是公钥的哈希值
- Canister Id 代表智能合约身份, 由系统分配, 不可改变
- Anonymous Id 匿名身份, 没有签名

# 读取消息的返回结果，或者其它状态



[https://ic0.app/api/v2/canister/<canister\\_id>/read\\_state](https://ic0.app/api/v2/canister/<canister_id>/read_state)

HTTP POST (CBOR 编码)

常用路径

/time

/subnet

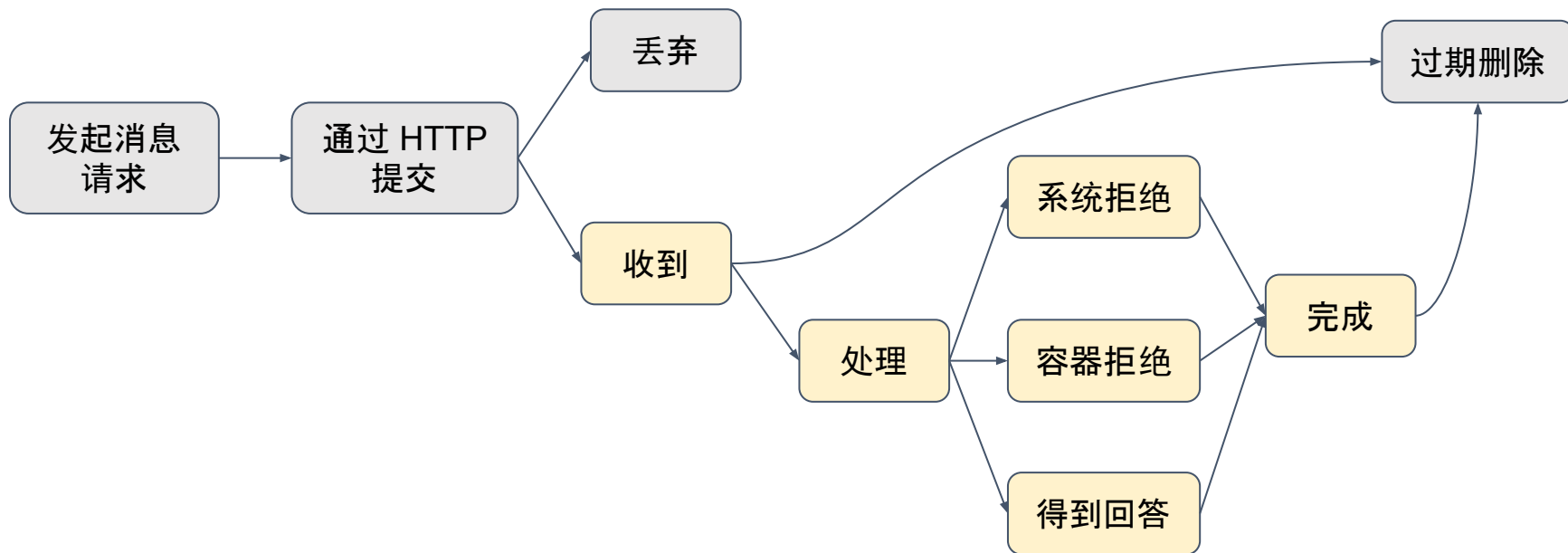
/request\_status/<request\_id>

/canisters/<canister\_id>/module\_hash

/canisters/<canister\_id>/controllers

返回的结果经由子网公证，可以用 merkle hash 验证

# 消息处理的步骤



# 用 http\_request 方法实现网页服务

```
service : {  
  http_request: (HttpRequest) -> (HttpResponse) query;  
}
```

```
type HttpRequest = record {  
  body: blob;  
  headers: vec HeaderField;  
  method: text;  
  url: text;  
};
```

```
type HttpResponse = record {  
  body: blob;  
  headers: vec HeaderField;  
  status_code: nat16;  
  streaming_strategy: opt StreamingStrategy;  
};
```

```
type HeaderField = record { text; text; };
```

# 课程作业

1. 学习 Counter 的例子, 并且部署到主网

<https://internetcomputer.org/docs/current/developer-docs/build/backend/counter-tutorial>

2. 给 Counter 添加一个 `http_request` 方法, 用返回 html 的方式显示当前 count 的值。

## 下一节: 用 Motoko 做后端

- 函数类型
- 结构类型
- Actor 类型
- 身份与权限处理