链滴

Bytom DAPP 开发流程

作者：bytom

原文链接：https://ld246.com/article/1563416224856

来源网站：链滴

从目前已经发布的DAPP来看，DAPP架构大致可以分成3种类型：插件钱包模式、全节点钱包模式和容模式。

- 插件钱包模式是借助封装了钱包的浏览器插件通过 RPC协议与区块链节点通信，插件在运行时会将eb3框架注入到DAPP前端页面中，然后DApp通过Web3来与区块链节点通信。
- 全节点钱包模式需要项目方同步并持有一个区块链节点，并对外提供一个浏览器环境与用户进行交。
- 兼容模式可以在插件钱包和全节点钱包下同时使用，即上述两种方式可以自由切换，安全性能相对高。

接下来介绍的比原链DAPP的架构模式跟账户模型DAPP的插件钱包模式有些相似，都是由DAPP前端插件钱包和合约程序共同组成，其中插件钱包需要连接去中心化的区块链服务器blockcenter，该服器主要是为了管理插件钱包的相关信息。此外，比原链是UTXO模型的区块链系统，合约程序存在于状态的UTXO中，如果要实现这样一个具体的DAPP，就需要在前端和后端多做一些逻辑的处理。

1. 编写、编译并实例化智能合约

编写智能合约

比原链的虚拟机是图灵完备的，理论上可以实现任意图灵计算机能实现的操作。而Equity作为比原链智能合约语言，使用Equity语言可以实现许多典型的金融模型案例，但是为了解决停机问题，比原链设置了手续费的上限，因此用户在设计合约的时候做一下权衡。

合约模板结构如下：

```
contract contract_name(...) locks valueAmount of valueAsset {
  clause clause_name(...) {
    ...
    lock/unlock ...
  }
  ...
}
```

Equity语法结构简单，语句意思明确，有开发经验的童鞋一看基本能明白合约的意思。编写智能合约以参考Equity合约介绍，文档中对Equity语言的语法和编译方法都做了详细的介绍。此外，文档还对些典型的模板合约进行了介绍，开发者可以自己需求进行参考。

编译并实例化合约

编译合约目前支持两种方式，一种是使用Equity编译工具，另一种是调用比原链中编译合约的RPC接口ompile; 而合约实例化是为了将合约脚本按照用户设定的参数进行锁定，编译并实例化合约可以参考译并实例化合约的上半部分说明，该文档不仅介绍了合约的参数构造说明，还对编译合约的步骤进行细说明。而编译器以及相关工具位于Equity编译器中，是使用go语言开发的，用户可以下载源代码并译使用。

工具编译和实例化示例如下：

```
// compile
./equity [contract_name] --bin

// instance
```

./equity [contract_name] --instance [arguments ...]

2. 部署合约

部署合约即发送合约交易，调用比原链的build-transaction接口将指定数量的资产发送到合约program，只需将输出output中接收方control_program设置为指定合约即可。用户可以参考合约交易说明中锁定合约章节，交易的构造按照文档中介绍进行参考即可。如果合约交易发送成功，并且交易已经成上链，便可以通过调用API接口list-unspent-outputs来查找该合约的UTXO。

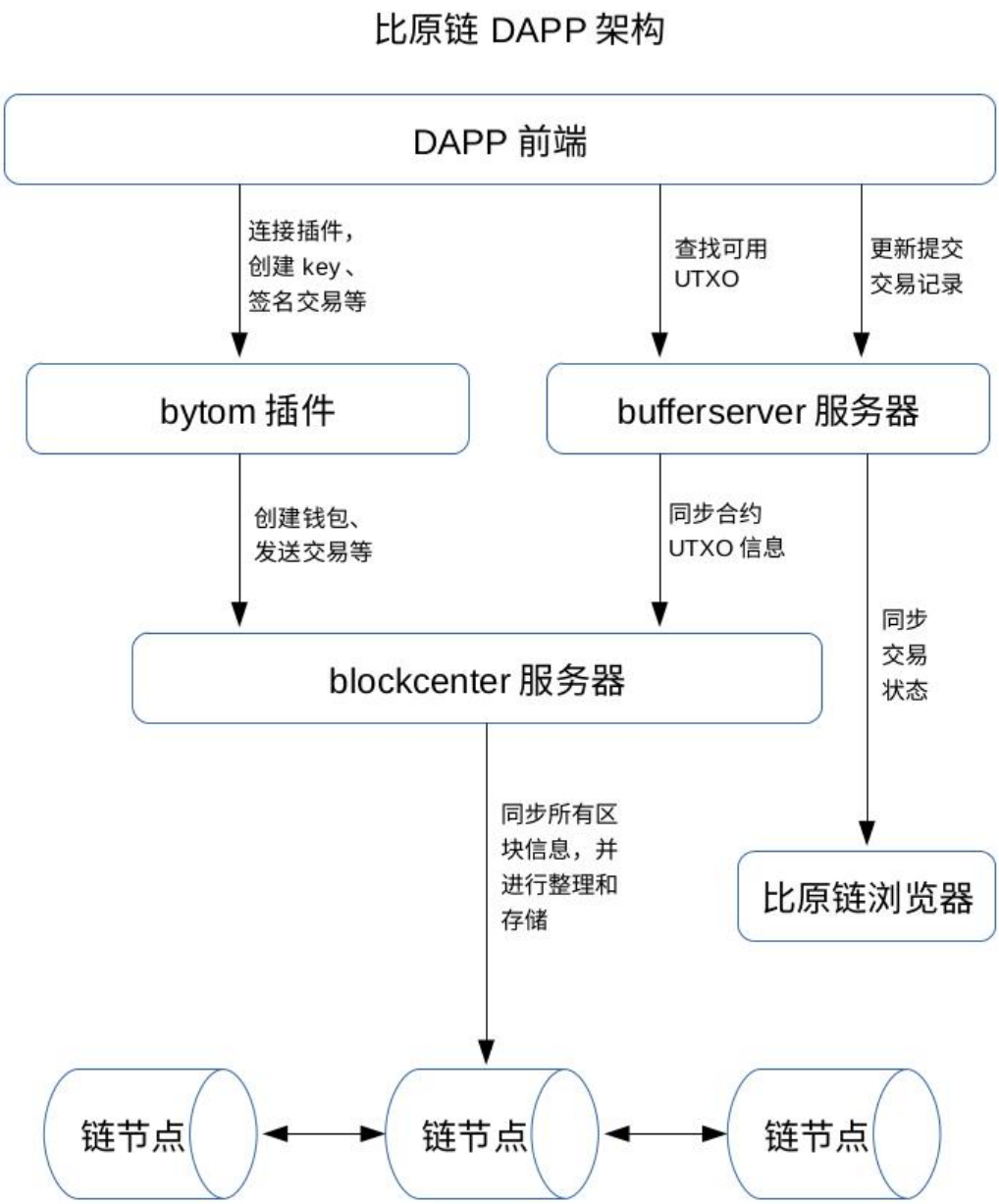
部署合约交易模板大致如下：

```
{
  "actions": [
    // inputs
    {
    // btm fee
    },
    {
    amount, asset, spend_account
    // spend user asset
    },

    // outputs
    {
    amount, asset, contract_program
    // receive contract program with instantiated result
    }
  ],
  ...
}
```

3. 搭建DAPP架构

Bytom的blockcenter服务器是官方开发的去中心化插件钱包服务器，开发者可以按照相关API接口来用即可。比原链的DAPP总体框架模型如下：



DAPP前端

搭建DAPP前端主要包含两个方面：一个是前端与插件钱包的交互，另一个是前端的逻辑处理、以及缓冲服务器的交互。插件钱包是与区块链节点服务器通信的窗口，一个DAPP为了跟区块链节点进行信，需要通过借助插件来与后台服务器节点进行交互。比原的插件钱包除了与后台服务器进行交互之，还包含一些本地业务逻辑处理的接口API，具体内容可以参考一下DAPP开发者向导。由于比原链是于UTXO模型的区块链系统，交易是由多输入和多输出构成的结构，并且交易输入或输出的位置也需按照顺序来排列，因此开发DAPP需要前端处理一些构建交易的逻辑。除此之外，合约中的lock-unlock句中涉及到数量的计算需要根据抽象语法树来进行预计算，计算的结果将用于构建交易，而verify、if-lse等其他语句类型也需要进行相关的预校验，从而防止用户在执行合约的时候报错。

从功能层面来说，前端主要包含页面的设计、插件的调用、合约交易逻辑的处理、缓冲服务器的交互

原文链接：Bytom DAPP 开发流程

。接下来对这几个重要的部分展开说明：

- 1）前端页面的设计主要是网页界面的设计，这个部分开发者可以自己选择页面模式
- 2）插件钱包已经进行了结构化的封装，并且提供了外部接口给 DAPP开发者调用，开发者只需要插件的参数按照规则进行填充，具体请参考DAPP开发者向导
- 3）比原链的合约交易是多输入多输出的交易结构，前端需要进行一些预判断逻辑的处理，然后再择合适的合约交易模板结构。
- 4）DAPP的插件连接的是去中心化的 bycoin服务器，该服务器从比原节点服务器上同步的所有区信息和交易信息，该部分主要是在插件钱包层进行了高度的封装，用户只需按照接口调用即可。除此外，需要开发者搭建一个缓冲服务器，不仅可以在管理合约UTXO层面做一些性能方面的处理，而且可以为DAPP做一些数据存储。开发者可以根据实际需求来开发一些RPC请求接口，然后在前端页面置相关条件来触发这些API的调用。

前端逻辑处理流程大致如下：

- 调用插件，比原的 chrome插件源码位于Bytom-JS-SDK，开发比原DAPP时调用插件的说明可以考Dapp Developer Guide，其网络配置如下：

```
window.addEventListener('load', async function() {

  if (typeof window.bytom !== 'undefined') {
    let networks = {
        solonet: ... // solonet bycoin url
        testnet: ... // testnet bycoin url
        mainnet: ... // mainnet bycoin url
    };

    ...

    startApp();
});
```

- 配置合约参数，可以采用文件配置的方式，该步骤是为了让前端得到需要用到的一些已经固定化的约参数，其前端配置文件为configure.json.js，其示例模型如下:

```
var config = {
    "solonet": {
        ...         // contract arguments
        "gas": 0.4  // btm fee
    },
    "testnet":{
        ...
    },
    "mainnet":{
        ...
    }
}
```

- 前端预计算处理，如果合约中包含 lock-unlock语句，并且Amount是一个数值表达式，那么前端提取计算表达式并进行相应的预计算。此外，前端还需要预判下所有可验证的verify语句，从而判定易是否可行，因为一旦前端对这些验证失败，合约将必然验证失败。此外，如果define或assign语句及的变量，前端也需预计算这些变量的值。
- 构建合约交易模板，由于解锁合约是解锁 lock语句条件，构造交易需要根据lock语句或unlock语来进行变换。解锁合约交易是由inputs和outputs构成，交易的第一个input输入一般都是是固定的，

合约UTXO的hash值，除此之外，其他输入输出都需要根据DAPP中的实际合约来进行变更，其模型致如下：

```
const input = []
input.push(spendUTXOAction(utxohash))
... // other input

const output = []
output.push(controlProgramAction(amount, asset, program))
... // other output
```

- 启动前端服务

编译前端命令如下：

```
npm run build
```

启动之前需要先启动bufferserver缓冲服务器，然后再启动前端服务，其前端启动命令如下：

```
npm start
```

DAPP缓冲服务器

缓冲服务器主要是为了在管理合约UTXO层面做一些效率方面的处理，包括了对bycoin服务器是如何步请求的，此外对DAPP的相关交易记录也进行了存储。bycoin服务器是比原链的去中心化钱包服务，缓冲服务器的UTXO跟它是同步更新的，比原官方插件钱包默认连接的就是该服务器。尽管bycoin务器的也对比原链的所有UTXO进行了管理，但是由于UTXO数量比较大，如果直接在该层面处理会致DAPP性能不佳，所以建议用户自己构建自己的缓冲服务器做进一步优化处理。此外，DAPP开发也可以搭建了自己的去中心化钱包服务器，并且自己开发相关的插件。

缓冲服务器架构可以参考一下bufferserver案例的源代码，其编译和启动步骤如下：

- 编译 bufferserver源代码

按照README安装部署服务需要的软件包Mysql和Redis，然后下载源代码并编译：

```
make all
```

编译完成之后，在target目录下会生成可执行文件api和updater。

- 启动服务

使用root用户创建数据库和数据表，其命令如下：

```
mysql -u root -p < database/dump.sql
```

修改配置文件config_local.json，配置说明参考README的config配置参数详解。

启动api和updater服务器，其中api是提供JSON RPC请求的服务进程，updater是提供同步blockcener和区块链浏览器数据请求的服务进程。

```
./target/api config_local.json

./target/updater config_local.json
```

启动缓冲服务器之后，便可以启动前端服务，然后打开DAPP的网页URL即可使用。

附：缓冲服务器的JSON RPC接口可以参考wiki接口说明。

Bytom DAPP实例

Bytom DAPP 实例说明，请参考储蓄分红DAPP