

PDX BaaP 平台

用户手册

北京全息智信科技有限公司

二〇一七年 六月

目 录

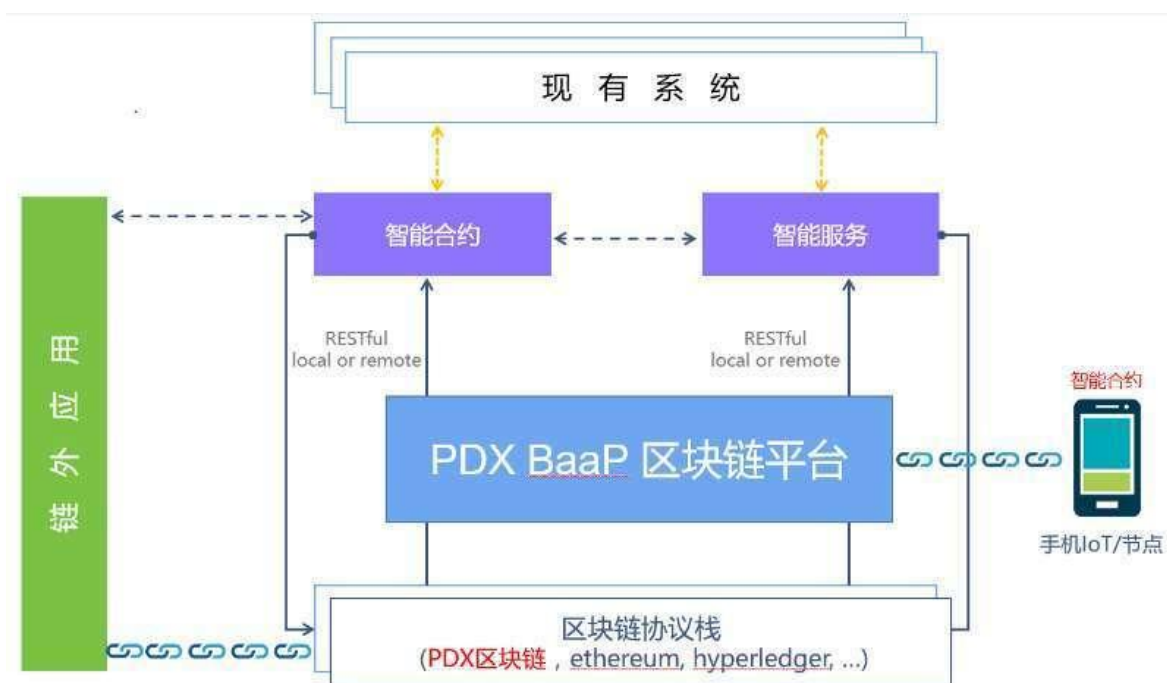
1 简介	4
2 BaaS 区块链平台搭建	5
3 应用开发	5
3.1 智能合约开发	5
3.1.1 智能合约说明	5
3.1.2 智能合约示例代码	11
3.2 智能合约上传	12
3.2.1 设置节点 IP	12
3.2.2 添加 bundle , 上传合约	12
3.2.3 运行合约	13
3.2.4 查看合约发布者地址	14
3.3 客户端应用开发	15
3.3.1 maven 工程创建 pom.xml 配置	15
3.3.2 BcDriver 实例化	16
3.3.3 BcDriver 调用	17
4 智能合约示例	18
4.1 示例说明	18
4.2 示例运行步骤	19
4.2.1 编译打包	19
4.2.2 智能合约部署	19

4.2.3 客户端调用.....	19
5 客户端开发.....	20

1 简介

欢迎使用 PDX BaaP(Blockchain as a Platform)区块链平台 ,PDX BaaP 平台利用创新的区块链技术 ,提供绝对安全和保护商业机密的去中心化应用的开发和运行平台 ,基于主流开发语言的智能合约平台进一步提升了区块链技术应用的广度和安全能力。PDX BaaP 区块链平台提供两种角色给区块链应用技术开发和运行者。

应用开发者 - 可以利用 BaaP 平台提供的 SDK, 利用最主流的 JAVA 语言开发去中心化应用 Dapp, 并部署到区块链上或者部署在远端 ,由 BaaP 平台调度运行 ;BaaP 平台架构如下 :



一些常用概念 :

公钥 / 私钥 —— 无论是应用开发者、节点运行者都需要一对密钥 - 公钥和私钥。这一对密钥 , 可以通过 BaaP 平台提供的客户端工具生成。私钥一定要由自己个

人保管；公钥是可以发布给需要验证的人，用户的公钥同时代表区块云上对应用户的一个公开地址。因此公钥需要通过客户端工具上传到 BaaP 平台；

Bundle – 去中心化的应用程序，是由应用开发者开发；分为两种类型，部署和运行在区块链上的应用及部署和运行在区块链外面的远端应用；每个 Bundle 有一个名字，即 Bundle Name；

Dapp – 一个智能合约，在 PDX BaaP 区块云上对应一个调用入口。一个 Bundle(去中心化的应用程序)可以对应多个 Dapp

Endpoint – 如果应用开发者开发和运行远端应用，可以将远端应用部署在多个环境下以运行多个实例，每一个实例需要提供一个入口 - 即 EndPoint

2 BaaP 区块链平台搭建

此部分内容详见 PDX-BaaP 节点部署手册.pdf

3 应用开发

3.1 智能合约开发

3.1.1 智能合约说明

在 BaaP 平台搭建完成之后就可以在平台上进行智能合约的开发。使用 PDX BaaP 平台开发 Dapp 应用极其简单，一个智能合约对应一个 jax-rs 服务。

实现一个智能合约，需要实现 IDapp 的两个接口 query(查询区块链上交易)，apply (执行交易) 即可。接口如下所示：

```
public interface IDapp {  
  
    /**  
     * Query about a Transaction bypassing the block chain.  
     *  
     * @param headers  
     * HTTP/s headers in the request  
     * @param tx  
     * Criteria in body using JUEL  
     * @return  
     * Transaction List  
     */  
    @POST  
    @Path("/query")  
    @Produces(MediaType.APPLICATION_JSON)  
    @Consumes("application/pdx-DaaP")  
    public List<Transaction> query(@Context HttpHeaders headers,  
    Transaction tx);
```

```
/**
 * Execute a TX via the block chain
 *
 *
 * @param headers
 * HTTP/s headers in the request
 * @param tx
 * Transaction to check
 *
 * @return
 * TransactionResponse
 */
@POST
@Path("/apply")
@Produces(MediaType.APPLICATION_JSON)
@Consumes("application/pdx-DaaP")
public TransactionResp apply(@Context HttpHeaders headers,
Transaction tx);
}
```

Transaction 类数据结构如下 ,

```
public class Transaction {
```

```
    /**
```

```
     * version of this Transaction class 版本
```

```
    */
```

```
    private String ver;
```

```
    /**
```

```
     * DaaP://{tenantId}/{dappId}/{method}), or chain://{hex-of-address
```

合约地址

```
    */
```

```
    private URI dst;
```

```
    /**
```

```
     * OPTIONAL callback d-app to notify TransactionResult.
```

```
    */
```

```
    private URL cbUrl;
```

```
    /**
```

```
     * OPTIONAL transaction IDx that must happen before this one is  
executed.
```

```
    */
```

```
    private List<String> depTXs;
```



```
/**
 * OPTIONAL TX signing algo. Default is blockchain's signing algo.
 */
private String sigAlgo;

/**
 * OPTIONAL Pairs of <signer public key, signature of TX before this
field
 * is set. Sender signature must be present..
 *
 * Key: base64-encoded sender public key
 *
 * Value: signature of TX before this field is set. Sender signature must
be
 * present
 *
 */
private Map<String, byte[]> sig;

/**
 * OPTIONAL salt for sign. The first signer should set this item.
 */
```

```
private byte[] salt;

/**
 * OPTIONAL timestamp for sign. The first signer should set this item.
 */
private long timestamp;

/**
 * OPTIONAL, <b>body</b> encryption algorithm
 */
private String encAlgo;

/**
 * OPTIONAL Multi-recipient encryption, the dst is usually a contract
but
 * not necessarily.
 * Key: base64-encoded SHA3 of recipient public key
 * Value: <b>body</b> encryption key, encrypted by recipient public
key
 *
 */
private Map<String, byte[]> enc;
```

```

/**
 * OPTIONAL Encrypted sender authn/z token, using recipient's
pubk.// 交易授权值
 */
private byte[] token;

/**
 * OPTIONAL extra meta data if needed//交易扩展属性
 */
Map<String, byte[]> meta;

/**
 * TX body //交易包含数据
 */
private byte[] body;
}

```

其中最重要的两个属性需要开发者关注：

byte[] dst --- 合约的地址

byte[] body --- 应用开发者自己定义的数据

3.1.2 智能合约示例代码

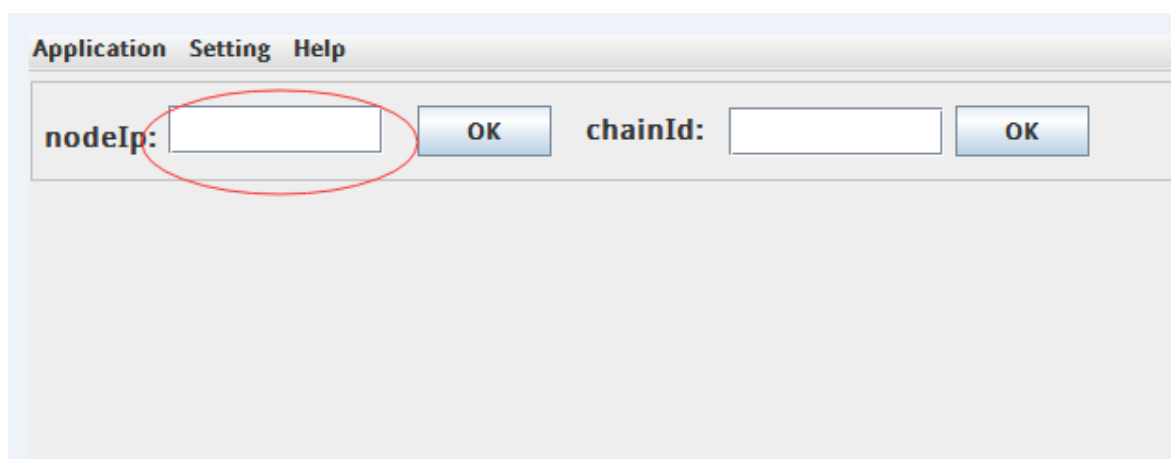
合约示例代码：<https://github.com/PDXTechnologies/baap-contract>

3.2 智能合约上传

合约开发调试完成打成 war 包后可以通过 DaaPUtilsUI 工具上传到节点上。具体步骤如下：解压 DaaPUtilsUI.zip 在命令行使用 java 命令执行 `java -jar DaaPUtilsUI.jar`

3.2.1 设置节点 IP

输入 nodeIp 点击 ok , chainid 可不设置默认 (default_pdx_chain)



3.2.2 添加 bundle , 上传合约

点击左上角 Application->bundle 后在弹出页面点击 Add , 输入 bundle 信息后 , 点击 upload

Application Setting Help

nodeIp: 10.1.1.217 RESET chainId: OK

Bundle

Add List

type: pdx-war

bundleName: dbcontract

jaxrpath: rest

description: dbcontract

file: /home/allenliu/work/h3c/dbcontract/target/db.war scan

Upload

3.2.3 运行合约

添加 bundle 后，点击 list 可展示合约列表，点击 run 发布合约

Application Setting Help

nodeIp: 10.1.1.217 RESET chainId: OK Login

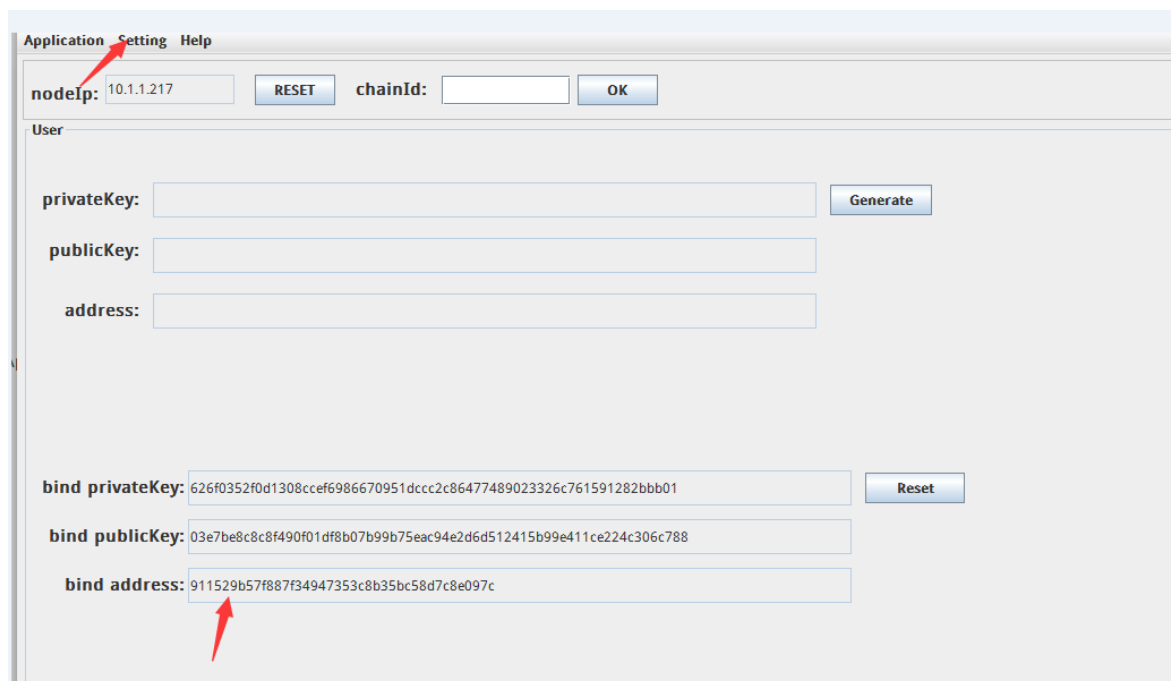
Bundle

Add List

bundleId	bundleType	status	bundleName	operate	operate
bundled34400be0a93406b8a6b17007303c...	pdx-war	stopped	testdb	run	delete

3.2.4 查看合约发布者地址

点击左上角第二个 tab Setting 下 user 查看用户信息，其中 bind address 为合约发布者地址，未合约 dst 中一部分



The screenshot shows a web application interface with a top navigation bar containing 'Application', 'Setting', and 'Help'. The 'Setting' tab is selected. Below the navigation bar, there is a 'User' section. In this section, there are several input fields and buttons. The 'bind address' field is highlighted with a red arrow. The 'bind privateKey' and 'bind publicKey' fields are also visible. The 'bind address' field contains the value '911529b57f887f34947353c8b35bc58d7c8e097c'.

以上红色箭头标注为上传用户地址，合约地址（即 Transaction 对象中 dest）由 DaaP://chainId/user address/restUrl 构成，用户在执行客户端调用时需要设置此地址，见下面示例。

```
try {  
    tx.setDst(new URI("daap://default_pdx_chain/911529b57f887f34947353c8b35bc58d7c8e097c/pdx.dapp/db"));  
    List<Transaction> res = driver.query(tx);  
}
```

3.3 客户端应用开发

智能合约及去中心化应用通常作为系统服务存在，由于在 PDX BaaP 平台上开发的去中心化应用提供 rest 服务接口，所以应用开发者编写面向消费者的客户端应用将会异常简单。

应用开发者可以使用 PDX BlockChain Driver 提供的 SDK 进行客户端应用开发。

3.3.1 maven 工程创建 pom.xml 配置

配置 maven repo

```
<repository>
    <id>pdx-release</id>
    <name>biz.pdxtech</name>

    <url>http://DaaP.pdx.life:8081/nexus/content/repositories/releases</url>
</repository>
```

配置 PDX BlockChain Driver API 依赖

```
<!-- bcdriver api -->
<dependency>
    <groupId>biz.pdxtech.DaaP</groupId>
    <artifactId>daap-bcdriver</artifactId>
```

```

        <version>1.2.*</version>

    </dependency>

```

配置 Default Ethereum Blockchain Driver 实现 依赖

```

<!-- Default Ethereum Blockchain Driver -->

<dependency>

    <groupId>biz.pdxtech.daap</groupId>

    <artifactId>daap-ethbcdriver</artifactId>

    <version>1.2.*</version>

</dependency>

```

3.3.2 BcDriver 实例化

实例化 BcDriver

调用 BcDriver 首先需要实例化 Driver.主要是两个参数。一个是协议栈类型；另一个是用户私钥。缺省情况下，协议栈类型为 ethereum，私钥会由 BcDriver 自动生成。调用者也可以通过如下两个方法之一自己进行参数设置。

1) 通过环境变量实例化

```
BlockchainDriver driver = BlockchainDriverFactory.get();
```

```
PDX_DAAP_CHAIN_TYPE    //协议栈类型
```

```
PDX_DAAP_PRIVATEKEY    //私钥
```


2) 通过 property 配置实例化

```
BlockchainDriver driver = BlockchainDriverFactory.get(property);
```

```
#blockchain Type
```

```
type=ethereum
```

```
#privateKey
```

```
privateKey=*****
```

3.3.3 BcDriver 调用

通过 BcDriver 以下方法调用链上或者远端合约：

```
query
```

```
apply
```

参见 DaaPCaller 例子

通过设置 Transaction 中属性指定 contract 地址和自定义逻辑等

dst-->合约地址

body-->自定义数据结构

客户端需要实现：

对于 query 类型来说，就是调用 Rest call 即可；

为了触发智能合约，客户端要向区块链写入交易。

4 智能合约示例

4.1 示例说明

智能合约的所有示例在 github 的如下地址：<https://github.com/PDXTechnologies/baap-contract>。主要包括 5 个示例：

- baap-contract-simple：简单合约，被触发后只打印相应日志信息，没有其它业务逻辑。
- baap-contract-db：复杂合约，被触发后将交易对象及其各种属性记录入数据库，开发者可以在此处实现自己的业务逻辑。
- baap-contract-state：state 使用示例，此示例展示如何将智能合约的 state 返回并且查询指定的 key 对应的 state 值。
- baap-contract-ledger：账本使用示例。区块链本质上是一个账本的集合，账本下是所有的交易记录。开发者可以在将交易写入时对此交易设置任意的属性和值（key value），然后在后期使用过程中根据此属性和值来查询出交易。
- baap-contract-oobm：oobm 使用示例。如果开发者有大数据的文件处理要求，我们不建议直接将大数据直接写入链中，开发者可以将大数据直接发送给智能合约的 oobm 接口，oobm 接口会自动存储这个大数据文件。同时客户端发送完大数据后会对整个大数据计算摘要，并将摘要写入区块链中。此交易最终会触发智能合约的执行，此时开发者可以在智能合约里处理自己的业务逻辑，比如

读取事先存储的大数据,再次对大数据做摘要计算,并与交易中得到的摘要比较。

如果比较一致则代表数据没有被篡改,然后可以进行进一步的业务处理。

4.2 示例运行步骤

4.2.1 编译打包

将 baap-contract 源代码从 github 上下载下来后可以直接在根目录执行 mvn package, 此时会将 5 个示例工程各个打包。也可以进入单个子工程下执行 mvn package, 此时会打包具体的工程。打包后的文件在各个工程下的 target 目录下, 比如 baap-contract-simple 对应的打包文件在 target 目录下的 baap-contract-simple.war 文件。

4.2.2 智能合约部署

按照“3.2 智能合约上传”章节步骤将打包好的合约 war 文件部署并运行。

4.2.3 客户端调用

在 eclipse 里直接执行 src/main/test/life.pdx.dapp.sample.xxx.calle.DaapCaller.java (右键弹出菜单后直接选择“Run As->Java Application”), 上述地址中 xxx 代表项目名称, 比如如果是 baap-contract-ledger 工程, xxx 代表 ledger。

注意在执行之前需要把以下三项属性更改成相应的值：

```
private static final String DST = "daap://default_pdx_chain/45874a3c0
afc2a4d6cc9dea20245350f2981d3ea/pdx.dapp/sample/ledger";

private static final String HOST = "10.0.0.5";

private static final String PRIVATE_KEY = "137f9a8fa4fac8ad5b3752cc
056eb0f733e5090271d61941a22f790833af4be9";
```

DST 代表合约地址，HOST 代表节点的 IP，PRIVATE_KEY 代表发送交易的用户私钥。

5 客户端开发

如果需要在 BaaP 区块链平台上开发手机客户端应用，可以使用针对手机的 SDK 直接操作区块链，目前只支持 Android 平台。

此部分内容详见 PDX-插件 SDK 用户手册.pdf。