北京全息智信科技有限公司

PDX BaaP 平台

用户手册

2017 年 11 月

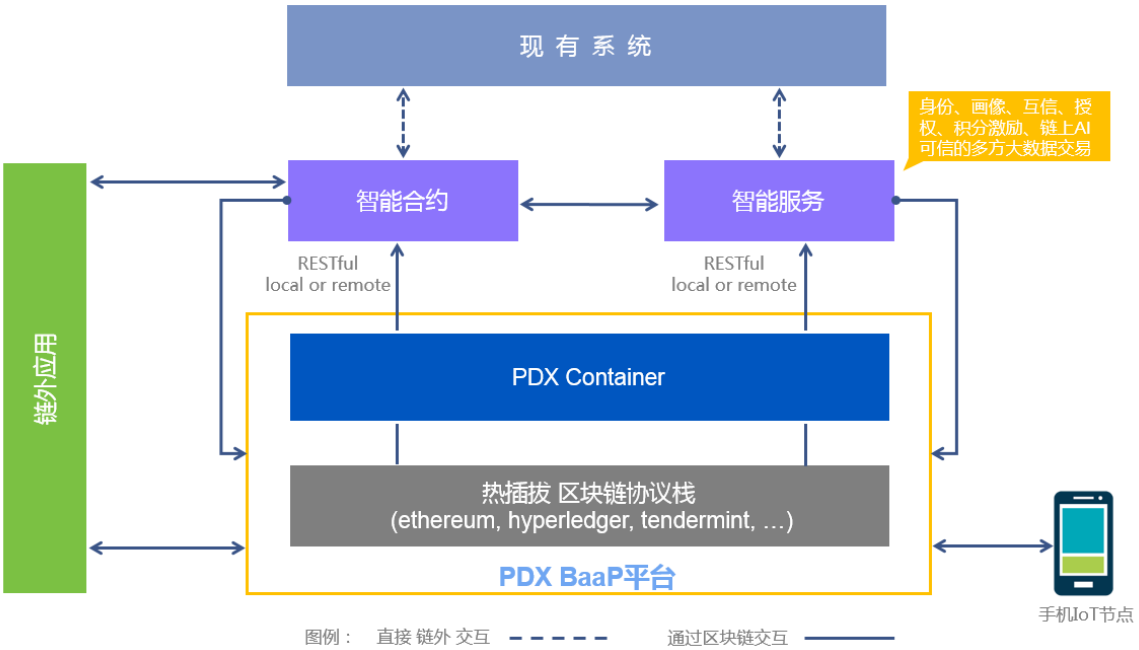
目 录

1 修订记录

编号	修改日期	修改内容	版本号
1	2017 年 10 月 30 日	初版	0.1
2	2017 年 11 月 1 日	修订格式	0.2
3	2017 年 11 月 2 日	修订格式	0.3

2 概述

欢迎使用 PDX BaaP(Blockchain as a Platform)区块链平台，PDXBaaP 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。使用 PDX BaaP 区块链平台，应用开发者只需关注业务逻辑，无需关注区块链底层技术实现，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 Bapp，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：



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，即公钥和私钥。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。

- Bapp

一个智能合约，在 PDXBaaP 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 Bapp。

- 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

3 PDX BaaP 区块链平台搭建

此部分内容详见 PDX BaaP 平台节点部署手册。

4 应用开发

4.1 智能合约开发

4.1.1 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 Bapp 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 IContract 接口的至少一个方法：exec（执行交易）。IContract 接口如下所示：

```
public interface IContract {

    @POST
```

```
@Path("/init")
@Consumes({"application/pdx-baap"})
default void init(@Context BaapContext ctx) {
}

@POST
@Path("/query")
@Produces({MediaType.APPLICATION_OCTET_STREAM})
@Consumes({"application/pdx-baap"})
default InputStream query(@Context BaapContext ctx, byte[] qstr) {
    return null;
}

@POST
@Path("/exec")
@Produces({MediaType.APPLICATION_JSON})
@Consumes({"application/pdx-baap"})
TransactionResp exec(@Context BaapContext ctx, Transaction tx);

@POST
```

```
    @Path("/undo")
    @Produces(MediaType.APPLICATION_JSON)
    @Consumes({"application/pdx-baap"})
    default TransactionResp undo(@Context BaapContext ctx, String txid) {
        return null;
    }

    @POST
    @Path("/fini")
    @Consumes({"application/pdx-baap"})
    default void fini(@Context BaapContext ctx) {
    }
}
```

Transaction 类数据结构如下，

```
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Transaction {

    /**
     * version of this Transaction class
```

```
     */
    private String ver;

    /**
     * contract://{tenantId}/{bappId}/{method}), or chain://hex-of-address
     */
    private URI dst;

    /**
     * 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.
 */
private byte[] token;

/**
 * OPTIONAL extra meta data if needed
 */
private Map<String, byte[]> meta;

/**
 * TX body
 */
private byte[] body;

public Transaction() {
    super();
    this.ver = "1.0.0";
    this.timestamp = System.currentTimeMillis();
}
```

```
public String getVer() {
    return ver;
}

public void setVer(String ver) {
    this.ver = ver;
}

public URI getDst() {
    return dst;
}

public void setDst(URI dst) {
    this.dst = dst;
}

public List<String> getDepTXs() {
    return depTXs;
}
```

```
public void setDepTXs(List<String> depTXs) {
    this.depTXs = depTXs;
}

public String getSigAlgo() {
    return sigAlgo;
}

public void setSigAlgo(String sigAlgo) {
    this.sigAlgo = sigAlgo;
}

public Map<String, byte[]> getSig() {
    return sig;
}

public void setSig(Map<String, byte[]> sig) {
    this.sig = sig;
}
```

```
public byte[] getSalt() {
    return salt;
}

public void setSalt(byte[] salt) {
    this.salt = salt;
}

public long getTimestamp() {
    return timestamp;
}

public void setTimestamp(long timestamp) {
    this.timestamp = timestamp;
}

public String getEncAlgo() {
    return encAlgo;
}
```

```
public void setEncAlgo(String encAlgo) {
    this.encAlgo = encAlgo;
}

public Map<String, byte[]> getEnc() {
    return enc;
}

public void setEnc(Map<String, byte[]> enc) {
    this.enc = enc;
}

public byte[] getToken() {
    return token;
}

public void setToken(byte[] token) {
    this.token = token;
}
```

```
public Map<String, byte[]> getMeta() {
    return meta;
}

public Transaction putMeta(String key, byte[] value) {
    if (this.meta == null) {
        synchronized (this) {
            if (this.meta == null)
                this.meta = new HashMap<>();
        }
    }
    this.meta.put(key, value);
    return this;
}

public Transaction putMeta(Map<String, byte[]> state) {
    if (this.meta == null) {
        synchronized (this) {
            if (this.meta == null)
```

```
                this.meta = new HashMap<>();
            }
        }
        this.meta.putAll(state);
        return this;
    }

    public void setMeta(Map<String, byte[]> meta) {
        this.meta = meta;
    }

    public byte[] getBody() {
        return body;
    }

    public void setBody(byte[] body) {
        this.body = body;
    }
}
```

其中最重要的两个属性需要开发者关注：

URI dst --- 合约的地址

byte[] body --- 应用开发者自己定义的交易数据

4.1.2 智能合约示例代码

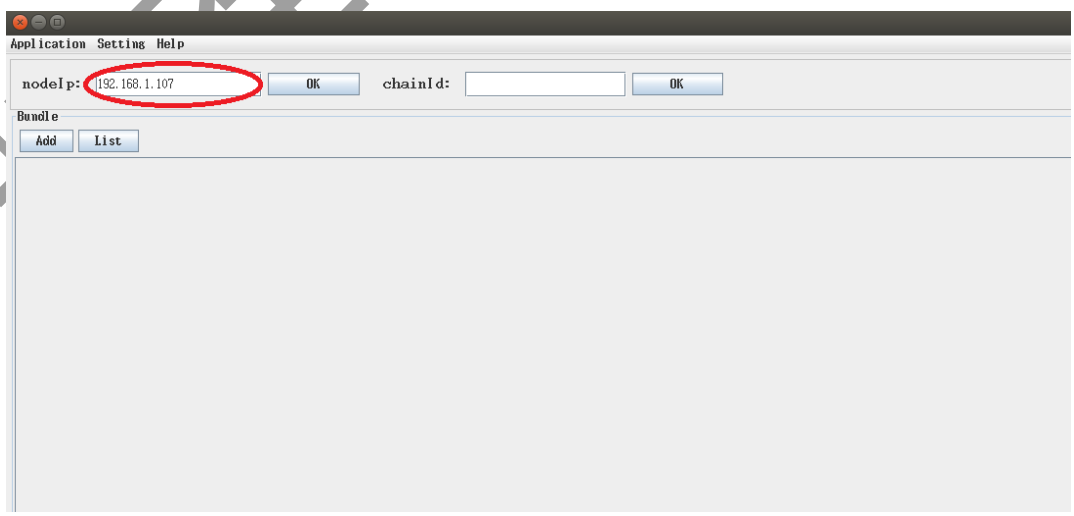
合约示例代码：https://github.com/PDXTechnologies/baap-contract

4.2 智能合约上传

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

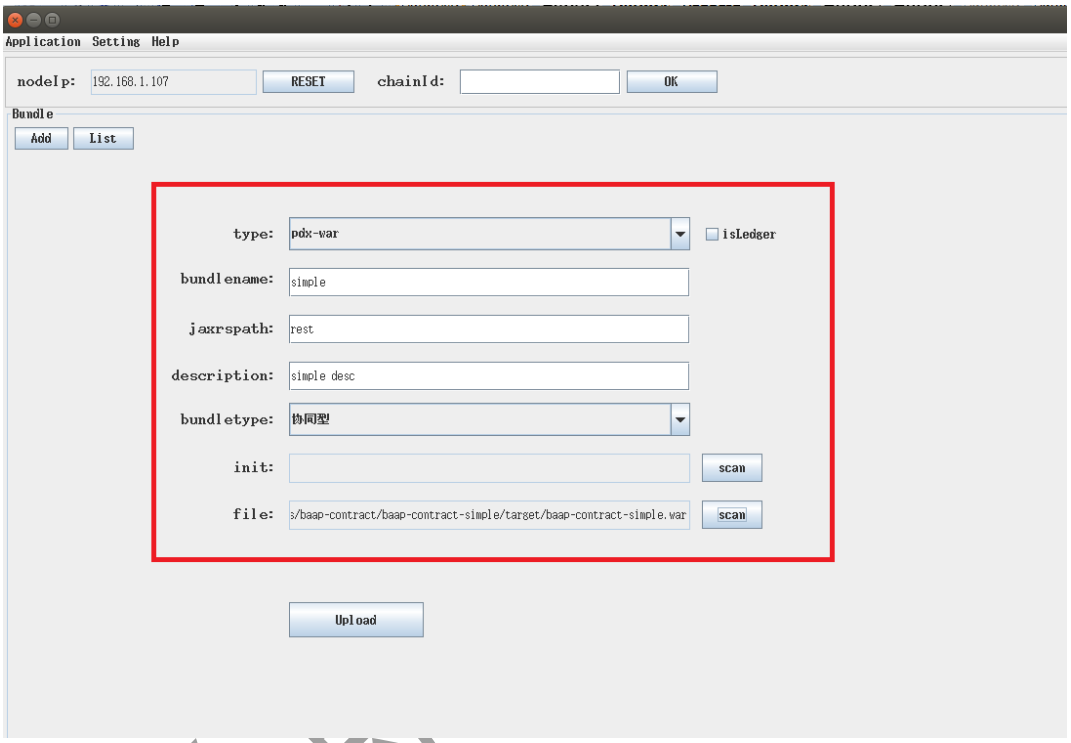
4.2.1 设置节点 IP

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



4.2.2 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



上图各字段说明如下：

Type：分别有如下值 1. pdx-war，合约在 BaaP 平台运行环境下运行；2. App-war，合约在独立的环境运行； 3. Remote，合约在远端运行，此时需要提供合约的调用入口 endpoint。

bundlename：程序包的名称，开发者通过这个名字区别多个不同程序包

jaxrspath：对应 javax.wx.rs.ApplicationPath 设置。

description：程序包的说明

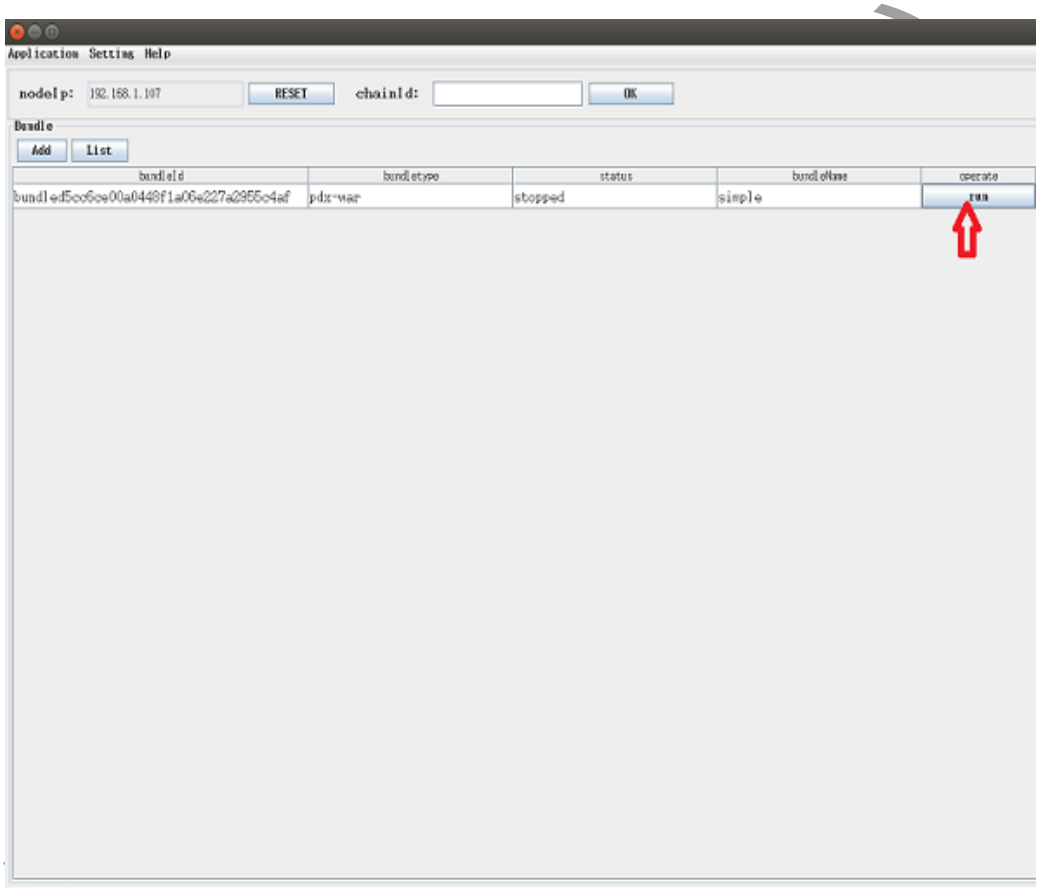
bundletype：对程序包所提供功能的简单分类

init：智能合约的初始化脚本，可空

file：程序包所在文件位置。

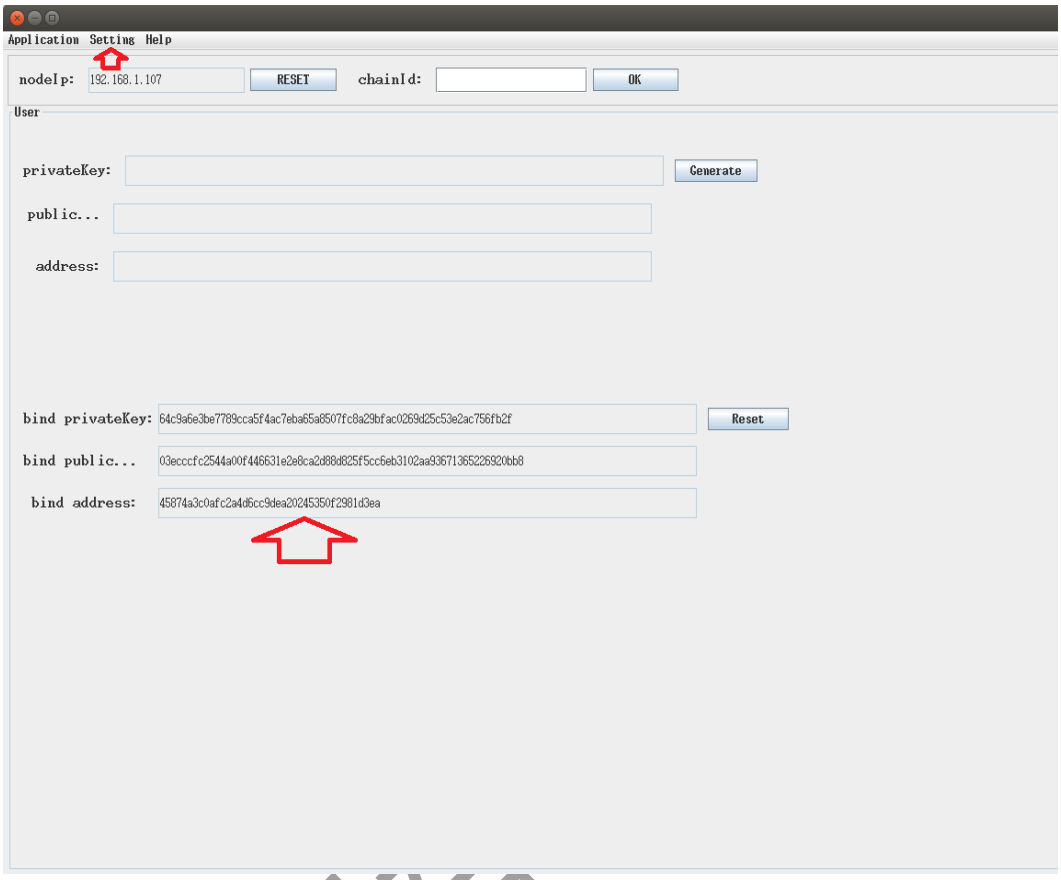
4.2.3 运行合约

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



4.2.4 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 为合约发布者的地址，此地址是 dst 中一部分



以上红色箭头标注为上传用户地址，合约地址（即 Transaction 对象中 dst）由 contract://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("contract://default_pdx_chain/45874a3c0afc2a4d6cc9dea20245350f2981d3ea/pdx.bapp/sample/simple"));
} catch (URISyntaxException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

4.3 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，由在 PDX BaaP 平台上开发的去中心化应用提供 RESTful 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。

4.3.1 配置 maven 仓库

创建 java maven 工程，在 pom.xml 配置 maven repo

```
<repositories>
    <repository>
        <id>pdx-release</id>
        <name>biz.pdxtech</name>
<url>http://daap.pdx.life:8081/nexus/content/repositories/releases</url>
    </repository>
</repositories>
```

配置 PDX BlockChain Driver API 依赖

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

```
        <version>2.1.0</version>
    </dependency>
    <dependency>
        <groupId>biz.pdxtech.baap</groupId>
        <artifactId>baap-setting</artifactId>
        <version>2.1.0</version>
    </dependency>
```

配置 Default Ethereum BlockChain Driver 实现 依赖

```
<!-- Default Ethereum BlockChain Driver -->
<dependency>
   <groupId>biz.pdxtech.baap</groupId>
   <artifactId>baap-driver-ethereum</artifactId>
   <version>2.1.0</version>
</dependency>
```

4.3.2 Driver 实例化

调用 Driver 首先需要实例化 Driver，主要是四个参数：

1、 协议栈类型；

2、 区块链 id；

3、 区块链节点的 baap-url；

4、 用户私钥。

缺省情况下，协议栈类型为 ethereum。调用者可以通 过如下方法进行参数设置。

通过 property 配置实例化

```
Properties properties = new Properties();
properties.setProperty("baap-chain-type",
Constants.BAAP_CHAIN_TYPE_ETHEREUM);
properties.setProperty("baap-chain-id",
Constants.BAAP_CHAIN_ID_DEFAULT);
properties.setProperty("baap-url", "http://192.168.1.107:8080/");
properties.setProperty("baap-private-key",
"137f9a8fa4fac8ad5b3752cc056eb0f733e5090271d61941a22f790833af4be9");
BlockChainDriver driver = BlockChainDriverFactory.get(properties);
```

4.3.3 Driver 调用

通过 BcDriver 的以下方法调用链上或者远端合约：

query

apply

调用 apply 方法时 ,需要通过设置 Transaction 中以下属性指定合约地址和自定义逻辑。BaaP 平台最终会调用到合约的 exec 方法，并向区块链写入交易。

dst-->合约地址

body-->自定义数据结构

调用 query 时，需要带上参数 dst 和查询参数 qstr，BaaP 平台最终会调用到合约的 query 方法。

具体示例参见 BaaPCaller.java 文件

4.4 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 2 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。

4.5 示例运行步骤

4.5.1 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 2 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。

4.5.2 智能合约部署

按照“4.2 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。

4.5.3 客户端调用

在 eclipse 里直接执行 src/main/test/life.pdx.bapp.sample.xxx.caller.BaapCaller.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-simple 工程，xxx 代表 simple。

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 = "contract://default_pdx_chain/45874a3c0afc2a4d6cc9dea20245350f2981d3ea/pdx.bapp/sample/simple";

private static final String HOST = "10.0.0.5";

private static final String PRIVATE_KEY = "137f9a8fa4fac8ad5b3752cc056eb0f733e5090271d61941a22f790833af4be9";

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的私钥。

5 APP 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。