## 引言

本报告制定了城市数据共享与交换平台系统的研究方案。基于区块链的城市数据共享与交换平台系统主要结合去中心化的属性密码的细粒度权限管理访问控制方案与分布式公钥方案，与多维数据确权系统相互支撑从而实现智慧城市中多个组织或系统之间的数据共享。在本系统中，来自于多个城市或者组织的子系统以联盟组织的形式搭建形成联盟链网络，依照各通道的规则制定各自的智能合约进行数据共享与交换。

## 1 研究背景概述

近年来，智慧城市建设已成为城市管理者、建设者、运营者等各方关心的焦点话题，越来越多的城市开启了“智慧城市”的建设热潮。在国家新型城镇化建设战略背景下，发展智慧城市已成为提高城镇化质量、推进内涵型城镇化建设的重要举措，对于国家培育和发展战略性新兴产业，推动经济发展方式转型和生活方式变革以及抢占未来科技制高点，提升城市核心竞争力，都具有非常重要的现实意义。对智慧城市运行产生的各类数据进行存储和分析，实现城市各类业务系统数据之间的共享交换，可以促进城市管理和服务智慧化的新理念和新模式，为城市管理提供强大的决策支持。城市数据共享与交换平台，可以实现多部门、多环节管理无缝化和高效化，有利于政务系统之间数据共享，提高信息共享度，在城市数据的分析处理上，有了数据共享与交换平台，借助于区块链、数据可视化等信息技术，使城市管理部门能够在海量的城市异构数据中挖掘出适用信息，进行数据关联与协同计算，快速分析处理，精准了解和把握城市运行过程中存在的潜在风险和隐患，使决策更具有针对性和实效性，更快更有效地解决突发性灾害；在城市数据的收集上，有了数据交换与共享平台，通过移动互联、区块链、云计算、大数据等技术，可全面、实时的收集城市交通等的基础数据，实现实时检测以及城市管理大数据智能深度分析；在城市数据的加工上，有了数据交换与共享平台，可以通过大数据技术对涉及城市管理的多个领域进行高效率的管理工作。利用各类前端感知城市体征信息变化，有助于城市管理职能部门快速全面了解城市管理部件设施动态，同时与城管业务流程实现可视化系统集成，对业务管理中重要事件主动发现、实时响应，为下达指挥处置预案提供数据支撑，促进城市管理的高效协作。 由于该系统需要针对智慧城市运行数据跨部门/机构共享场景复杂、安全策略不统一、数据跨域共享难控制等特点，支持多源多模信息跨场景的可信治理，实现城市数据的安全交换与受控共享，因此对系统稳定性、系统策略及运算速度要求较高。城市数据共享与交换平台建设的基本策略是首先对现有的城市系统的大量多层次汇集、多源异构、跨域共享数据进行整合，并在此基础上设计出多源异构数据柔性分层汇集机制；突破多源数据汇集关联控制、数据跨域可控安全传输与海量连接状态管理、高并发差异化数据汇集的作业异步处理与状态同步、大流量数据实时清洗与过滤、异构数据智能消冗压缩等关键技术，从而实现海量数据高效分层汇集。另外针对智慧城市海量异构数据结构复杂、数据权限管控复杂、海量数据管理需求不一、信息系统权限体系复杂、运行数据跨部门/机构共享场景复杂、安全策略不统一、数据跨域共享难控制等特点，需要结合区块链、属性密码等新型技术设计解决方案，这点本文会在后续章节介绍。

## 2 系统架构设计

基于区块链的城市数据共享与交换平台系统主要结合去中心化的属性密码的细粒度权限管理访问控制方案与分布式公钥方案，与多维数据确权系统相互支撑从而实现智慧城市中多个组织或系统之间的数据共享。在本系统中，来自于多个城市或者组织的子系统以联盟组织的形式搭建形成联盟链网络，依照各通道的规则制定各自的智能合约进行数据共享与交换。 根据分层设计的理念，我们将城市数据共享与交换平台的系统整体架构分为底层数据库、智能合约层、服务层与应用层，如下图所示：

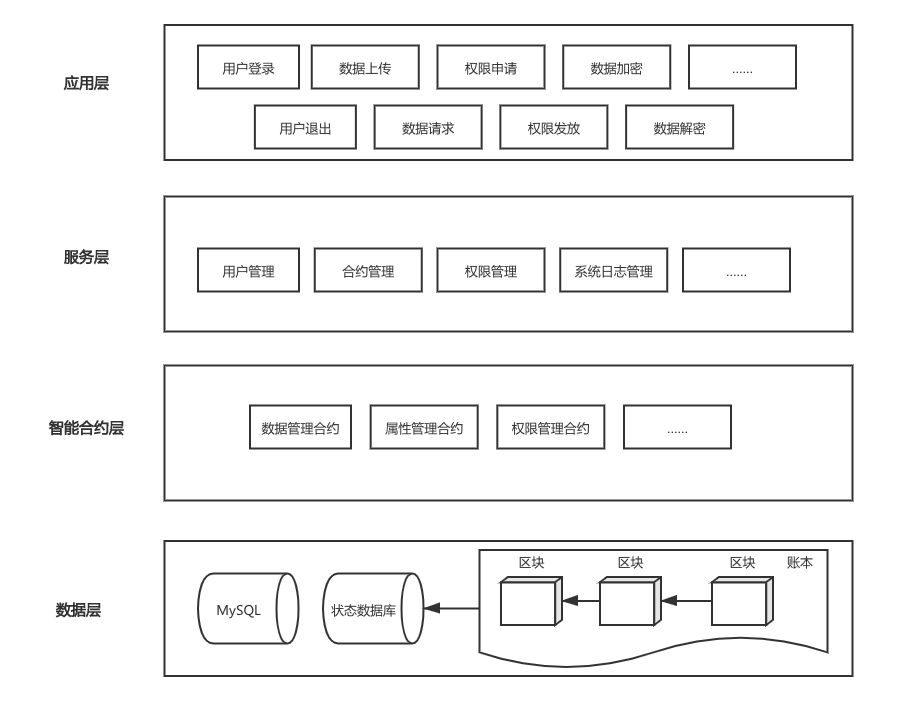


图 2-1 城市数据共享与交换平台系统架构

区块链底层主要在联盟链账本中以交易记录与世界状态的形式存储来自于城市间各组织的用户数据、组织的信息、数据共享、确权与流转过程中的操作日志。本系统使用CouchDB状态数据库作为存储世界状态的数据库，其相较于LevelDB数据库能提供更复杂的数据查询方式。 由于城市数据共享与交换设计与多维数据确权系统、细粒度权限管理访问控制系统等多个子系统多有数据联动，因此智能合约层需要对数据管理、确权验证、权限管理等多个层面的业务逻辑进行规范与限制。服务层主要是针对本系统的功能需要所进行划分的业务逻辑模块，在服务层我们主要通过Java进行后端业务逻辑的实现。其中，我们使用了Fabric-java-SDK来实现java对Fabric通道和用户链码生命周期的管理，通过java调用Fabric-java-SDK来实现在通道中执行链码、查询交易信息等交互操作。本方案的应用层基于Java Web与Node js进行实现，并采取了前后端分离的设计模式。前端界面获取来自于用户基于登录、数据上传、权限申请等交互操作的请求参数，并通过API调用的形式将输入传递给服务层进行处理，并将服务层返回的结果反馈给用户。

## 3 内部模块组成

### 3.1 应用层

应用层主要负责实现用户如何参与到城市数据共享与交换平台中，主要采用Node.js进行前端网页的编写以及采用Java进行后端逻辑的编写。普通用户使用该系统进行注册、登录、上传数据或查询数据等操作。

#### 3.1.1 用户身份模块

用户身份模块主要负责实现用户的注册、登录和身份识别等功能。该模块主要完成以下几项工作：

（1）用户注册时该模块为每个用户生成一对公私钥，该密钥对目的在于加密用户在区块链网络之间的通信。公私钥的生成采用椭圆曲线加密算法（ECC）。

（2）当用户想要在区块链网络中通信时，用户可向其所在组织申请对某一通道的访问，若组织同意该申请，则为该用户生成该通道下的唯一的全局标识（GID），该全局标识用于在不同的通道中标识用户的身份。

#### 3.1.2 用户权限模块

用户权限模块主要负责实现用户的权限申请和查看等功能。该模块主要完成以下几项工作：

（1）向用户显示该用户是否拥有某个属性的属性私钥。

（2）向用户显示某个属性私钥的申请状态。

（3） 如果用户拥有的属性私钥不符合加密策略，为用户开放申请授权的服务，并返回给用户申请成功与否的结果。该申请过程会触发智能合约并记录在数据库中。

#### 3.1.3 用户数据模块

用户数据模块主要负责实现用户数据的存储和查询等功能。该模块主要完成以下几项工作：

（1）用户上传数据并填写对该数据的访问策略属性，其中属性信息是组织发布的属性列表；同时模块对用户上传的数据进行加密，并将通过智能合约将加密后的数据加入到区块链中。

（2）向用户提供数据信息查询功能。如果用户拥有的属性私钥集合满足查看某个数据，则该模块将触发智能合约将对应的数据密文取出返回给用户，然后利用用户的属性私钥集合进行解密从而使得用户可以获取明文数据。

### 3.2 服务层

用户管理模块主要负责实现组织或系统中的管理员对组织内的用户进行管理的功能。该模块主要完成以下几项工作：

（1）用户内容查看。管理员可查看组织或系统中的宏观用户统计数据，例如有多少人在使用系统等。

（2）对于违规的用户，管理员可将该用户置入黑名单，该用户则无法继续使用系统。

（3）其他一些常规系统中管理员应该具有的功能，可根据不同组织的自身需要动态调整。

#### 3.2.1 用户管理模块

用户管理模块主要负责实现组织或系统中的管理员对组织内的用户进行管理的功能。该模块主要完成以下几项工作：

（1）用户内容查看。管理员可查看组织或系统中的宏观用户统计数据，例如有多少人在使用系统等。

（2）对于违规的用户，管理员可将该用户置入黑名单，该用户则无法继续使用系统。

（3）其他一些常规系统中管理员应该具有的功能，可根据不同组织的自身需要动态调整。

#### 3.2.2 合约管理模块

合约管理模块主要负责实现管理员对合约相关的管理控制等功能。该模块主要完成以下几项工作：

（1）管理员参与搭建联盟链，创建组织内的各种节点。

（2）管理员可以选定节点创建通道，或者在其他管理员创建的通道中安装智能合约。其中，管理员之间本身组成了一个通道，该通道子链保存了所有的用户资料和用户上传的数据等信息，未加入通道的用户无法查看该子链上的数据。

（3）管理员可以查看整个组织正在使用中的智能合约。智能合约模块主要负责实现智能合约的安装、升级等功能。该模块主要完成以下几项工作：

（1）智能合约的安装。当管理员在前端提出了智能合约部署等操作的请求时，该模块将创建一个合约并做好相关映射。

（2）当智能合约需要升级时，直接将新的智能合约导入覆盖掉旧的智能合约即可。智能合约部分数据结构如表3-1所示。

#### 3.2.3 组织管理模块

组织管理模块主要负责实现建立组织间的网络和组织的加入退出等功能。该模块主要完成以下几项工作：

（1）初始阶段时，经由几个组织以非技术的形式协商达成共识后，建立起一个开始的Fabric网络；在此之后，如果有组织想要加入该数据共享交换平台网络，需要已存在的组织达成某种共识来同意或不同意新的组织是否可以加入网络。类似地，组织的退出也需要现有的其他组织同意或不同意。

（2）配置组织的内部节点。组织的内部节点主要分为主节点、背书节点和记账节点三类。其中，主节点与用户和管理员直接交互，且管理员和用户交互的节点应当加以区分。背书节点用于初始化MSP，并为组织中的其他节点提供身份认证机制，组织中的实体可以向MSP申请证书然后参与到数据交换中。记账节点则负责真正的数据存储共享。

#### 3.2.4 权限管理模块

属性权限管理模块主要负责实现属性的创建和审批等功能。该模块主要完成以下几项工作：

（1）管理员可以借助该模块创建属性。

（2）当用户利用用户权限模块申请对某属性的授权时，该模块将通知管理员来审批该申请，如果该申请被通过，则该模块将调用智能合约为该用户生成属性私钥，并将对应的公钥加密后存储在区块链上。

#### 3.2.5 系统日志模块

系统日志模块主要负责实现系统内各种操作的记录，由各组织单独记录在MySQL中。如果组织有特殊需要，也可以将记录的数据加密通过智能合约加入到区块链中。

### 3.3 智能合约层

智能合约层主要负责实现各种具体的智能合约，在开发时相对重要的一些接口如表3-1所示。为了方便起见，智能合约执行后存储区块链底层的数据结构在此一并叙述。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 接口名称 | 接口类型 | 参数 | 返回 | 接口说明 |
| create\_attribute | POST | 属性信息 | 无 | 创建属性 |
| create\_gid | POST | 用户信息 | GID | 在通道内为用户创建GID |
| add\_data | POST | 加密数据 | 无 | 用户上传密文 |
| apply\_attribute | POST | 用户信息、属性信息 | 无 | 用户申请属性 |
| approve\_application | POST | 用户信息、属性信息 | 同意或不同意 | 管理员审批申请 |
| read\_data | GET | 加密数据的标签 | 密文列表 | 查询加密数据 |

表 3-1 智能合约中重要接口

#### 3.3.1 数据管理合约

当用户的数据上传完成后，智能合约层将调用用户数据合约将密文加入到区块链中，具体业务逻辑如下：

（1）普通用户如果想要上传数据，需要向管理员申请一个该通道上的GID，之后基于区块链中属性列表公布出来的属性标识和版本号选择特定的访问控制结构发起对明文的加密，加密后的密文会存储在底层区块链的数据账本中。

（2）普通用户如果获取数据，则需要先调用下述的权限合约进行权限的确认，若确认拥有权限，则数据合约将从区块链底层读取数据。数据合约存储在区块链底层CouchDB的主要数据结构如下所示。

|  |
| --- |
| type User struct {  GID string // 用户GID  OrgID string // 归属组织ID  Status string // 用户状态  UPK string // 用户公钥  }  type DataInfo struct {  ID string // 密文ID  GID string // 用户GID  OrgID string // 归属组织ID  Data string // 密文  AttributePolicy string // 属性加密策略  Date string // 数据创建日期  Title string // 数据标题  } |

#### 3.3.2 属性管理合约

属性管理合约主要负责在组织区块链中添加属性，当管理员进行属性的创建或是用户申请某个属性的权限时，将会触发属性管理合约的执行。具体业务逻辑如下：

（1）新建属性时，合约将新的属性记录到区块链中。

（2）如果有属性申请提出，则合约将借助CouchDB查询到对应的属性，并将其返回给管理员进行审批，再将审批结果记录到区块链中。其基本数据结构如下所示。

|  |
| --- |
| type Org struct {  OrgID string // 组织ID  Name string // 组织名  Remark string // 组织备注  OPK string // 组织公钥  }  type AttributePk struct {  ID string // 属性ID  OrgID string // 组织ID  Remark string // 属性描述  Name string // 属性名  Date string // 初始化时间  AttributeU string // 属性全局唯一标识  version string // 属性版本号  } |

#### 3.3.3 权限管理合约

当组织的用户或其他组织的用户想要查看某个密文，智能合约首先需要确定其拥有的属性列表是否满足密文的访问策略。具体业务逻辑如下：如果用户当前拥有的属性列表满足密文的访问策略，则将继续按照用户数据合约的流程执行；如果缺少相关属性，则该用户必须向特定组织申请该属性，对应的管理员需要进行同意或不同意的操作。在此过程中，相关的数据结构如下所示。

|  |
| --- |
| type UserApplication struct {  GID string // 用户GID  OrgID string // 发布属性的组织ID  AID string // 属性ID  ApplicationStatus string // 申请状态  Date string // 申请时间  }  type UserRole struct {  GID string // 用户GID  USK string // 归属于用户加密的属性私钥  OrgID string // 组织ID  Remark string // 属性描述  Name string // 属性名  RoleStatus boolean // 用户属性权限状态  } |

由上述相关合约的保证，数据的确权与追溯得到了保证。

### 3.4 区块链底层

区块链底层采用联盟式管理方式搭建Fabric网络，各个组织的管理员采用这种形式来搭建，组织之间采用Kafka实现共识协议。相关的数据结构已在智能合约层叙述过，在此不做叙述。此外，区块链底层还负责实现通道的创建、加入等功能。对于通道将主要完成以下几项工作：

（1）创建通道。某个组织管理员申请提出需要创建某个通道，其他组织对此请求作出相应回应，最终对是否创立通道达成一致意见。

（2）如果组织管理员同意创立该通道，则该模块为涉及到的组织添加该通道，同时将这一修改记录在区块链中。

## 4 接口说明

接口详细信息及相关数据示例由附件给出。

### 4.1 基本数据接口

对于在组织内的并且不涉及到在平台上共享的数据，可以用MySQL数据库存储。用户数据表主要存储用户名、密码和邮箱等基本信息，其简要数据结构结构如表4-1所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 主键 | 描述 |
| ID | VARCHAR | 36 | YES | 作为主键的标识符 |
| USERNAME | VARCHAR | 32 | NO | 用户名 |
| PASSWORD | VARCHAR | 32 | NO | 用户密码 |
| EMAIL | VARCHAR | 50 | NO | 用户邮箱，用于找回密码 |

表 4-1 用户数据

组织用户表和用户数据大同小异，不同的组织可以根据其自身需求定制数据表，这里不再赘述。 系统日志表主要保存系统内的相关操作记录，如表名、操作类型、操作时间和操作用户等，其简要数据结构如表4-2所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 主键 | 描述 |
| ID | VARCHAR | 36 | YES | 作为主键的标识符 |
| TABLE | VARCHAR | 50 | NO | 受影响的表名 |
| TYPE | VARCHAR | 20 | NO | 操作类型：增/删/改/查 |
| DATE | DATETIME | — | NO | 操作的时间 |
| USER\_ID | VARCHAR | 36 | NO | 操作的用户ID |
| DESCRIPTION | VARCHAR | 1000 | NO | 具体的操作 |

表 4-2 系统日志表

加密数据表主要保存由用户上传的加密数据，如加密后的数据和策略规则等，其简要数据结构如表4-3所示。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 字段名 | 数据类型 | 长度 | 主键 | 描述 |
| ID | VARCHAR | 36 | YES | 作为主键的标识符 |
| USER\_ID | VARCHAR | 36 | NO | 上传的用户ID |
| DATA | LONGBLOB | — | NO | 加密后的数据文件（二进制），原则上不大于4GB |
| DATE | DATETIME | — | NO | 上传时间 |
| RULES | VARCHAR | 1000 | NO | 属性策略规则 |

表 4-3 加密数据表

### 4.2 共享与交换数据接口

数据接口主要有以下几类：

#### 4.2.1 数据上传

数据上传时主要涉及到如下方法：

upload(filename, filetype, location, \*policy)

遇到大型文件时，可能需要划分成小文件单独存储：

decompose(data, \*to[])

#### 4.2.2 数据加密

数据加密接口对用户不可见，用户上传数据时需要确定相应的属性策略组规则，此接口由系统自动调用：

encrypt(data, \*attributes)

#### 4.2.3 数据解密

数据解密接口对用户不可见，用户申请查看某数据时需要提交相应的属性才能保证查看原数据，此接口由系统自动调用：

decrypt(data, \*attributes)

同样地，之前拆分过的大文件可能需要调用如下接口来还原：

compose(data, \*from[])

#### 4.2.4 数据转发

因为每次都从源服务器请求效率较低，因此组织可以授权将加密后的数据在另一组织中保持一份副本：

transfer(data, \*policy, destination)

## 5 系统接入及数据要求

面向智慧城市数据服务的数据交换平台主要分为两大部分，第一部分是底层区块链系统，主要分为两个部分，一个网络的搭建，包括多组织多节点多通道的搭建以及智能合约的编写与部署；第二个部分是web系统的搭建，每个组织都有自己的web系统，其连接不同的peer节点而获得与区块链网络通信的能力，在上面创建属性或者获得其他属性的权限。 依照架构图，在阿里云服务器上通过Docker容器创建三个组织，org1、org2、org3，在第一台服务上创建服务排序节点，每个组织内创建2个peer节点，分别映射到不同的端口。为了部署方便，所有的组织和组织内的节点都在同一个通道下交换与分享数据。合约部分使用Golang语言编写，通过调用Hyperledger Fabric的底层API来实现业务逻辑，至于web后端应用，利用java语言在spring-boot框架上开发开发应用。每个组织的应用程序与数据库mysql部署在同一台服务器上，应用端服务通过调用Fabric-java-sdk与组织内的peer节点通信，调用智能合约接口，前端页面使用node.js编写。

## 6 与其他模块的关系

智慧城市应用由于覆盖系统和用户基数庞大、信息系统种类繁多、数据模式繁复多样、信息迭代迅速、数据共享需求众多，存在城市治理监管需求复杂、数据采集确权困难、数据去向不可控等问题。针对这些难点，依据现有的技术支撑，我们提出了构建一个实现集城市数据收集、处理、访问、分发和交换功能一体的城市数据共享与交换平台。该平台主要包括四部分功能：多链数据汇聚管理、多维数据的确权与认证、细颗粒度权限管理及分布式公钥基础设施信息检测与信任管理。 城市数据共享与交换平台作为整个智慧城市系统的数据中间层，需要将多维数据确权原型系统、细粒度权限管理系统、分布式公钥基础设施等多个模块联结起来，为智慧城市信息系统的运行提供稳定可靠的运作支持。本平台通过接口调用形式，通过分布式公钥基础设置的配置，实现CA间信任管理、相互监督、PKI证书的透明与发放；借助细颗粒度权限管理中基于去中心化属性密码的属性发放与基于属性的数据加解密，从而满足智慧城市中跨组织、跨联盟及跨城市之间数据传递与共享中细颗粒度权限管理的需求；通过多维数据确权原型系统将来自于多方的共享数据的调用记录与日志记录在区块链上，实现数据使用与权限管理的可追溯、不可篡改。 总结而言，城市数据共享与交换平台作为来自城市间各子系统的数据交换的枢纽，打破原有城市组织与系统之间信息孤岛的现状，结合区块链与分布式公钥基础设置设施提供点对点的数据共享机制，实现无缝的数据共享，保证数据安全可靠高效传递。 此外，数据交换平台面向智慧城市汇总给你各业务部门应用，结合细颗粒度权限管理系统实现在多源信息资源整合的同时实现多层次多维度的细颗粒度权限划分与管理，为多城市间的数据流转提供安全性与便捷性的技术保障。