基于区块链的科技资源标识溯源的研究和实现

目录

[1 绪论 2](#_Toc103957111)

[1.1 研究背景和意义 2](#_Toc103957112)

[1.2 研究目标和内容 4](#_Toc103957113)

[1.3 论文的组织结构 5](#_Toc103957114)

[2 研究现状 5](#_Toc103957115)

[2.1 区块链技术 5](#_Toc103957116)

[2.2 区块链溯源技术 7](#_Toc103957117)

[2.3 Redis技术 8](#_Toc103957118)

[3 实现方案 9](#_Toc103957119)

[3.1 需求分析 9](#_Toc103957120)

[3.1.1 数据需求分析 9](#_Toc103957121)

[3.1.2 链上数据存储结构需求分析 10](#_Toc103957122)

[3.1.3 溯源方式需求分析 11](#_Toc103957123)

[3.2 技术分析 12](#_Toc103957124)

[3.2.1 FISCO BCOS 区块链 12](#_Toc103957125)

[3.2.2 Redis协议 14](#_Toc103957126)

[3.3 系统结构设计 15](#_Toc103957127)

[3.4 智能合约结构设计 16](#_Toc103957128)

[3.4.1 FISCO BCOS预编译合约接口 16](#_Toc103957129)

[3.4.2 表存储结构设计 17](#_Toc103957130)

[3.4.3 上链和溯源合约方法设计 19](#_Toc103957131)

[3.5 前后端设计 21](#_Toc103957132)

[3.4.1 数据上链模块 21](#_Toc103957133)

[3.4.2 溯源模块 24](#_Toc103957134)

[3.6 Redis实现 26](#_Toc103957135)

[3.7.1 链上数据存储结构设计和分析 26](#_Toc103957136)

[3.7.2 Redis协议解析的实现 26](#_Toc103957137)

[3.7.3 Redis 指令实现 26](#_Toc103957138)

[4 成果展示 27](#_Toc103957139)

[4.1 溯源系统展示 27](#_Toc103957140)

[4.1.1 上链数据展示 27](#_Toc103957141)

[4.1.2 溯源模块页面展示 27](#_Toc103957142)

[4.2 Redis中间件展示 27](#_Toc103957143)

[5 总结 27](#_Toc103957144)

[6 致谢 27](#_Toc103957145)

[参考文献 27](#_Toc103957146)

# 1 绪论

## 研究背景和意义

科技资源是从事科技活动的人力、物力、财力以及组织、管理、信息等软、硬件的总称，是科技活动的物质基础和基础保障，同时也是经济和社会发展所不可缺少的重要学术资源和战略资源。随着国务院办公厅印发的《科学数据管理办法》（国办发 [2018] 17号）和科技部、财政部印发的《国家科技资源共享服务平台管理办法》（国科发基 [2018] 48号），明确了对科技资源共享的和管理办法，科技资源的共享已经成了大势所趋。目前已经建立了中国科技资源共享网（以下简称共享网），整合了20个国家科学数据中心、31个国家资源库，以及重大科研基础设施和大型科研仪器的资源，收录了超过283万条资源数据，包括由中国热带农业科学院品资所牵头建立的国家热带作物种质资源库已经成为国家种质资源管理的重要基础平台——本课题的研究来源。除此之外，共享网还建立了科技资源标识系统，为每一个资源建立了唯一标识——科技资源标识，每一个标识代表了一条科技资源数据（本课题中科技资源同科技资源标识是一一对应关系，科技资源标识是科技资源的溯源时的唯一标识），并提供科技资源信息检索与查询服务，逐渐成为科技资源开放共享的基础服务平台，为科技资源开放共享做出了突出贡献。

但与此同时，在大数据时代下，看似规律、整齐的科技资源分散在各行各业，不同行业的科技资源在内容上有着不同的结构、特点；在管理上有着不同的规范，标准以及所属机构主体；此外科技资源数量大，科技资源体系存在着主体各异、存储结构各异、存在形式各异，其服务方式、安全保障等都存在着不一致性**，**给科技资源的管理带来了很大的困难，造成了科技资源数据的流通和使用困难，并存在数据的盗用、滥用、误用、侵权、恶意篡改等安全问题，急需构建一个安全、稳定、高效的工具来对科技资源进行管理。

目前传统科技资源数据管理技术仍然采用中心数据库的方式，数据存储在中心数据库中，数据库由管理员进行维护，这种情况下，如果数据库中的数据出现人为的盗取、恶意删除、恶意篡改等安全问题，可能会造成难以估量的损失，却又由于中心数据库的局限性，无法在短时间内对问题的源头进行溯源，并对相关人员进行追责和处罚。而区块链具有去中心化、公开、不可篡改、匿名、独立、集体维护、安全、天然的可溯源等特性，并能够完整、详实、可信地记录科技资源的演变过程，提供溯源支持，为解决上述问题，实现科技资源的管理和追溯提供了一个新思路**[1]。**

与传统中心数据库相比，数据在写入区块链时，会被区块链中的每一个节点验证，在节点达到共识后写入新的区块中，一旦有恶意节点想要篡改区块中的数据，必须得到至少百分之五十一节点的认同，也即至少有百分之五十一的节点都是恶意节点才能够篡改成功，在现实中是几乎不可能实现的。区块链的数据结构和共识机制保证了区块链上的数据的不可篡改。此外区块链上的数据按照时间上链，这与科技资源生命周期相对应，能够为科技资源的溯源提供很好的支持。因此区块链对于解决目前科技资源存在的恶意篡改、滥用、盗用等问题，实现科技资源的管理和追溯有很大的帮助。

目前，国家热带作物种质管理服务系统（以下简称为服务管理系统）在热带作物种质资源的创建、录入、鉴定、共享等环节均需要管理员在后台进行人工信息录入、审核、修改、删除等操作。在上述生命周期过程中产生的数据仍然存储在中心数据库中，由数据库管理员进行管理，在拥有数据库管理员的账号后能够轻松的对数据进行篡改、删除、拷贝等恶意操作，从而导致数据的泄露、窃权等安全问题。此外，在服务管理系统中在由于后台的每一个环节均需人工进行操作，在操作人员受到外界诱惑或操作人员账号泄露时，也会导致上述生命环节过程中出现数据的篡改、泄露、窃权等安全问题。

## 研究目标和内容

鉴于此上述问题，本课题将在国家热带作物种质资源管理服务系统的基础上，基于区块链技术和智能合约，构建一个完整的系统来支持将种质资源“采集-加工-汇交-存储-鉴定-评价-流通-应用”全生命周期过程中在后台人为操作所产生数据的上链和溯源。同时考虑到目前基于区块链的资源管理需要根据不同的场景进行设计不同的智能合约，对区块链开发者和使用者的要求较高，且在高级语言编程中和链上节点进行交互是一件较为复杂和麻烦的事情，本文拟根据共享网规定的科技资源标识数据集的结构，基于Rust语言实现目前较为通用和简单的Redis接口，提供一个能够极大简化和区块链节点进行交互的中间件，以达到降低包括热带作物种质资源在内的科技资源标识数据的上链和溯源难度。

具体工作包括：

1. 研究目前热带作物种质资源管理服务系统中潜在数据安全问题。设计前、后端系统以及区块链之间的交互逻辑，编写智能合约，将相关操作数据存储在区块链上。
2. 设计前端溯源结果展示页面，将溯源信息通过结构化和图形化的方式进行形象的展示，并提供基于时间、用户、科技资源标识等多种方式来支持对（1）中上链数据的溯源。
3. 针对目前科技资源数据的一般结构，设计基本智能合约，使用Rust语言实现一共基于Redis通信协议的中间件，实现SET、GET、Del、Del\_All等基础指令，并基于溯源需求，设计实现自定义History等指令。

## 1.3 论文的组织结构

本文主要内容是在科技资源管理的背景下，基于区块链技术，研究并实现一个完整系统来支持对科技资源全生命周期过程中的数据进行记录，并在需要时能够快速、方便、准确地对数据进行溯源。论文的组织结构如下：

1. 绪论。本章主要介绍研究背景和意义，介绍目前科技资源开放、共享和管理目前取得的成果，分析目前科技资源的管理所面临的问题，进而引出采用区块链的方式来解决目前遇到的恶意篡改、窃权、盗用、滥用等安全问题，并在问题出现时，利用区块链的特性进行溯源来帮助快速定责。最后简要总结本文的研究目标和内容，以及主要工作。
2. 研究现状。本章研究了当前区块链技术发展现状，区块链、以太坊等重要历史事件和天才的想法为当前的区块链技术的发展和应用做出了巨大贡献，接着对区块链溯源技术的发展和应用现状进行研究和总结，提出目前区块链溯源面临的一部分问题，最后对本系统拟实现的Redis中间件的核心——Redis技术进行介绍，并提出基于Redis协议封装区块链底层接口，简化区块链开发和应用的技术门槛的中间件设计。
3. 实现方案。首先对系统的需求进行分析，并对基于需求分析所选择的核心技术进行分析，然后分别介绍了国家热带种质资源服务与管理系统的溯源系统的设计和实现和基于Redis协议，封装FISCO BCOS区块链底层操作的中间件的设计。
4. 成果展示。

# 2 研究现状

## 2.1 区块链技术

2008年11月1日，中本聪发表《A Peer-to-Peer Electronic Cash System》**[2]**首次阐述了基于端到端（P2P）网络技术、复杂加密算法、时间戳技术、分布式共识技术的区块链技术，标志着区块链的诞生。在2009年1月比特币网络诞生，中本聪使用第一版开源客户端挖出第一个创世纪块，比特币系统正式上线。在早期比特币的发展过程中，充分彰显了其去中心化、不可篡改的特性和优势，随着软件和金融生态的完善，比特币逐渐走入大众视野，并出现了矿池，矿机等独特市场。但随着时间的发展，比特币网络本身的劣势和设计的缺陷也在慢慢地突出：栈式脚本过于简单无法执行复杂交易；出块时间过久，交易确认效率低。一部分人们想要扩张区块链技术的应用范围，而不仅仅作为一种虚拟货币交易的底层系统，仅仅局限在金融领域。

2013年到2014年间，俄罗斯程序员Vitalik Buterin在受到比特币的启发下，提出了天才想法——以太坊——下一代加密货币与去中心化应用平台[3]。以太坊相对于比特币系统，提出了智能合约的理念，并提供一个图灵完备的编程语言——Solidity，来支持智能合约的编写和创造。智能合约是一段运行在区块链上的一段静态代码，准确来说是运行在以太坊链的节点上的代码，类似一种协议或者合同（contract），账户通过指定所想要执行的智能合约地址以及传入的参数来执行智能合约，产生交易。智能合约的出现，让以太坊不仅仅是一个简单的加密货币系统，而是成了一种更为通用的基础平台。通过编写智能合约，开发者可以在遵循和实现以太坊制定的代币标准（实质上是一个智能合约接口）的基础上，发行属于自己的代币，还可以通过智能合约进行资金募捐，发布股权，债券等金融衍生品。除智能合约外，以太坊极大的降低了出块时间，把出块时间从比特币网络的10分钟降低到了15秒，极大的提高了交易执行和确认的效率。除上述特性外，以太坊还提出了GHOST协议、基于账户的账本等很多不同于比特币网络的新理念。

以太坊的新理念和设计为加密货币的金融体系扩展提供了很多的可能，同时也为区块链应用到实体经济、政府监管等场景提供了可能。如果说比特币是区块链1.0时代，以太坊代表了区块链2.0时代，为区块链技术在各个领域百花齐放提供了广阔平台。

目前区块链技术，在各个领域蓬勃发展。非同质化代币（Non-Fungible Token，NFT）是一种发行在链上的虚拟货币，以其不可进行互换和可追踪的特性被用来作为数字资产（画作、艺术品、声音、影视、游戏）的代；在游戏领域，基于以太坊的DApp和GameFi的游戏开发也在蒸蒸日上，比如Dark forest，CryptoKitties等，区块链游戏在逐渐定义新的游戏模式：玩家在游戏中既是一个玩家，也是资源的拥有者；世界上很多国家和组织对区块链的发展和应用也持有积极态度，如英国政府的区块链报告[4]，习近平总书记也在2019年10月对区块链技术的发展做出了重要指示[5]。除上述之外领域外，还有物流、社会公共服务，政府监督和管理，知识产权保护等个领域，区块链技术以其独特的优势，为各领域的发展。

## 2.2 区块链溯源技术

随着以太坊的提出，区块链技术发展进入了“大航海时代”，诸如FISCO BCOS，Hyprtledger Fabric、Truffle、Libra等开源区块链框架和平台相继推出。开源区块链底层框架上手简单，能够快速开发、部署和测试，并配有详细文档，在很大程度上降低了区块链应用开发的难度，为区块链落地在各行各业提供了较为强大的支持。

得益于此，区块链溯源技术在很多行业都有一定的应用。在食品领域，陈飞，叶春明等人[6]通过对食品的供应链、物流、销售等环节进行分析，采用Truffle框架设计了溯源数据的存储结构和唯一标识，并将唯一标识作为溯源码，来达到溯源的目的。谯业石[7]针对猪肉质量问题，设计了一套溯源系统来对猪肉的生产、批发、物流、零售等各环节进行记录和追踪，并将数据分为链上数据和业务数据，充分提高系统效率。在农产品领域，Qijun Lin[8]等人基于EPCIS架构，针对在农产品溯源体系中的参与者：企业、消费者和政府监管者三类角色的需求，设计并分析了系统的功能，并在以太坊上部署智能合约，来实现溯源系统。范茂顺[9]针对FISCO框架的底层共识和签名算法进行修改和优化，并据此构建具有隐私保护功能的农产品溯源系统。在物流运输方面，郑君臣[10]分析了跨境物流中的信任问题，并结合区块链技术阐述了区块链技术在解决问题的过程中的作用，并据此基于Fabric框架设计了区块链系统，并实现了较好的性能。在科技资源的管理和追溯领域，王瑞丹[1]等人采用Hyperledger创建的Fabric联盟链，设计一个完整的交易流程，定义了联盟链中各个节点的通信过程和功能，并使用科技资源标识作为科技资源元数据的唯一访问标识，通过记录全生命周期的数据来支持溯源。

总的来说，目前区块链技术在解决目前各行各业所面临的数据安全、数据记录、追溯、监管等传统技术所难以解决的行业痛点问题上有着得天独厚的优势。在不同的溯源领域，根据具体场景使用不同的框架，并将区块链技术同传统技术相结合，在充分发挥区块链的不可篡改性的同时，注重整个系统的效率以及吞吐量，最终达到1+1大于2的成果。不过在使用区块链技术进行溯源实现时，溯源数据上链之后虽然可以利用区块链技术的共识算法来实现数据的不可篡改，但是如果数据从源头上出错，即便链上数据真实可靠，也会面临着溯源难，追责难的问题。

## 2.3 Redis技术

目前区块链开发在现有框架的支持下，尽管框架能够降低开发上手难度，并集成了一部分功能，使得开发者快速开发、测试、部署，极大的提高了开发效率。但是在特定领域数据的溯源中，开发者仍然需要使用较为复杂的调用机制来开发区块链应用，需要编写较为复杂的智能合约来实现，也需要开发者对区块链底层有着较为深刻的理解才可使用框架进行开发，这极大的阻碍了区块链的应用和落地。本文采用Rust语言实现Redis协议，实现一个在调用者和区块链之间的中间件，调用者不需要知道区块链的底层逻辑，只需要通过执行相应的Redis命令，即可实现数据的上链以及数据的溯源。

Redis全称是Remote Dictionary Server，是一个使用ANSI C编写的开源、支持网络、基于内存、分布式、可选持久性的键值对存储数据库，提供多种语言的API。Redis是一个高性能的key-value数据库，支持多种数据结构，例如：字符串，散列，链表，集合，带范围查询的排序集合、位图、超日志、地理空间索引和流式存储等。因为Redis将数据存储在内存中，读写速度非常快，常常用来作为数据库、缓存、消息中间件。

Redis的存储介质一般是内存，即将数据存储在内存中，但只要实现了Redis的协议，存储介质可以不仅仅局限于简单的内存，可扩展性很强。采用Rsut来实现支持Redis协议的存储后端，Redis的存储介质可以是内存，磁盘，数据库等传统介质，也可以是区块链——本课题将采用的方式，通过把区块链作为Redis协议的存储介质，用户只需要通过Redis客户端连接到Redis服务端，发送想要执行的命令，服务端通过解析命令，向区块链发送相应的请求，来进行数据的创建、修改、查询和删除等操作。此外，在采用Rsut实现Redis协议时，Redis协议具有高度可扩展性，可以通过自定义指令的形式，来扩展Redis命令，从而实现区块链作为存储介质的特殊需求，也据此来支持溯源查询。通过自行实现Redis协议的方式，将Redis和区块链进行整合，利用Redis命令本身的简洁和Redis兼顾计算机分析、人类理解和实现效率的通信协议设计，屏蔽掉开发者和链上节点通信复杂的过程和流程，降低区块链开发的技术成本和使用难度，让区块链应用开发更为方便。

# 3 实现方案

## 3.1 需求分析

### 3.1.1 数据需求分析

在种质资源的收集、录入、鉴定、共享等生命周期过程中，会产生很多数据。在种质资源收集环节，管理员需要输入种质资源的作物类别、作物名称、科名、属名、学名、种质资源类型、收集方式、收集单位、收集地点、收集人等多种信息，包含着种质资源的收集时五花八门的属性信息，在后续每一个环节中都需要参考录入时的信息。收集时越详细，越全面，对种质资源的管理和应用的作用越大；在种质资源录入时需要对种质资源的主要特征、主要用途、保存设施、种质类型、保存数量、计量单位、入库年份、资源描述、备注、照片等详细信息进行记录，为后续对资源的鉴定提供更好的参考；在种质资源鉴定时，需要录入鉴定人、鉴定机构、鉴定地点、鉴定年份、种质资源的可操作范围以及鉴定备注等信息，此环节中涉及到鉴定人员和鉴定机构等个人敏感信息，同时鉴定信息是后续进行种质资源共享的重要参考信息，鉴定人和鉴定机构是否权威会影响到种质是否能够进行共享；在进行种质资源共享时，需要录入共享对象、联系电话（便于进行协商与沟通）、共享方式、共享用途、共享数量、共享的起始时间和结束时间、共享的效果以及其他关于共享的备注等信息；当然，在种质资源不愿进行共享时，可以录入不共享状态，此时需要录入不共享的原因。

此外除上述录入信息，在每一个环节均可对上一个环节录入的信息进行检查是否合格，并由此选择对信息进行录入（即同意上一环节的信息，并填写下一环节信息）或驳回（拒绝进入下一环节，需要继续进行修改），同时根据情况分析，可以选择对每一个环节的录入信息进行删除，此时必须重新由上一个环节重新录入。

在对上述过程中的信息进行分析可知，无论是哪一环节的数据，其实均类似于Key-Value键值的方式，每一个环节有多个信息属性，每一个信息属性对应一个值，同时为支持溯源查询，每一个环节的记录信息除了管理员在的录入信息之外，需要额外增加信息录入的时间戳信息，每一个记录上链时所产生的交易的Hash值，以及交易所在块号。在目前前后端交互常用的数据格式比如XML、SOAP、Atom、CSV、字符串等，本文拟采用JSON格式来作为数据传输和存储格式。

JSON格式全称为JavaScript Object Notation，是由道格拉斯·克罗克福特（Douglas Crockford）设计出的一种轻量级数据交换格式，完全独立于编程语言的信息交换和传输格式。JSON的语法简单、便于阅读和编写，解析速度快，支持数组、整数、字符串等多种数据类型，很多编程语言对JSON均有较好的支持，无论是JSON对象转换为字符串（序列化）和JSON格式字符串转换为JSON对象（反序列化）都简单、快捷，此外，JSON也是各类NoSQL的常用数据存储格式，应用极为广泛。在本课题中，管理员对各个环节的录入信息以键值对的形式将信息进行存储在JavaScript的JSON对象中（todo JSON数据图），前端对JSON数据序列化后，发送到后端，字符串形式的数据无论是Java、Rust、Solidity支持度都是相当高的，非常便于处理和存储。在查询时，前端也能够非常简单快捷的将JSON字符串进行反序列化得到JSON对象，进而对数据进行处理和展示。

### 3.1.2 链上数据存储结构需求分析

由于前后端交互数据交互是基于JSON数据的序列化和反序列化，那么链上的数据的存储格式便有了两种选择：SQL形式和NoSQL形式。

采用SQL形式存储，也即在3.1.1中所述的JSON数据在后端需要进行反序列化后，其中每一个键，都是数据库表中的一个字段，会产生一个字段很多的数据库表，在增加、删除、修改、查询时会传入很多的参数，而Solidity的一个函数中只能拥有16个变量，面对如此多的参数如果使用SQL方式，智能合约的编写会举步维艰，很难实现数据上链和溯源的复杂功能。

采用NoSQL（Not Only SQL）的方式，也即可以采用一部分内容使用类SQL的方式存储，一部分使用字符串的形式存储，在3.1.1中所述的JSON数据在到达后端后，无需进行JSON数据的反序列化，只需将JSON字符串作为一个字段存储到链上即可，此时只需要一个表结构的一个字段，而其他的数据比如时间戳、交易Hash、块号等可以分别作为一个单独的字段存储在表结构中，这样既不会出现字段过多的情况，又可以极大的简化后端和链上结点对数据的处理过程。

### 3.1.3 溯源方式需求分析

在本课题中所溯源的数据是管理员在后台的所有行为记录，包括但不限于哪一个管理员，在什么时候，操作了哪一个资源（科技资源标识），属于什么操作，操作的数据的是什么，操作的结果是什么等，但是作为一个管理系统，由于不同的用户的权限不同，可进行的操作不同，在进行溯源时，如果只使用时间作为一个溯源范围，在茫茫区块中寻找特定时间范围内的交易，无异于大海捞针，十分困难。为此本课题采用了三种方式作为溯源查询方式：

（1）根据时间范围进行溯源。在目前的溯源需求下，在出现安全问题或者监管人员想要对某段时间的行为进行阶段性的记录、总结和查看，此时需要通过确定一段时间（比如：十分钟，一个小时，一天，一周）来进行溯源。通过对一段时间内的行为记录的溯源类似一种模糊查询的方式，在时间片较长的情况下，可能会溯源出现大量的数据，但能够为接下来进行更为精确的查询提供支持，提高溯源效率。

（2）根据管理员ID和时间范围进行溯源。在经过上述“模糊查询”后，若发现某个管理员的行为异常，比如：在短时间内进行大量的删除操作或进行大量的审核且审核数据不合常规，可以通过管理员的ID并设定时间范围来对该管理员进行更为精准的溯源，以便查找出更多违法操作和行为。

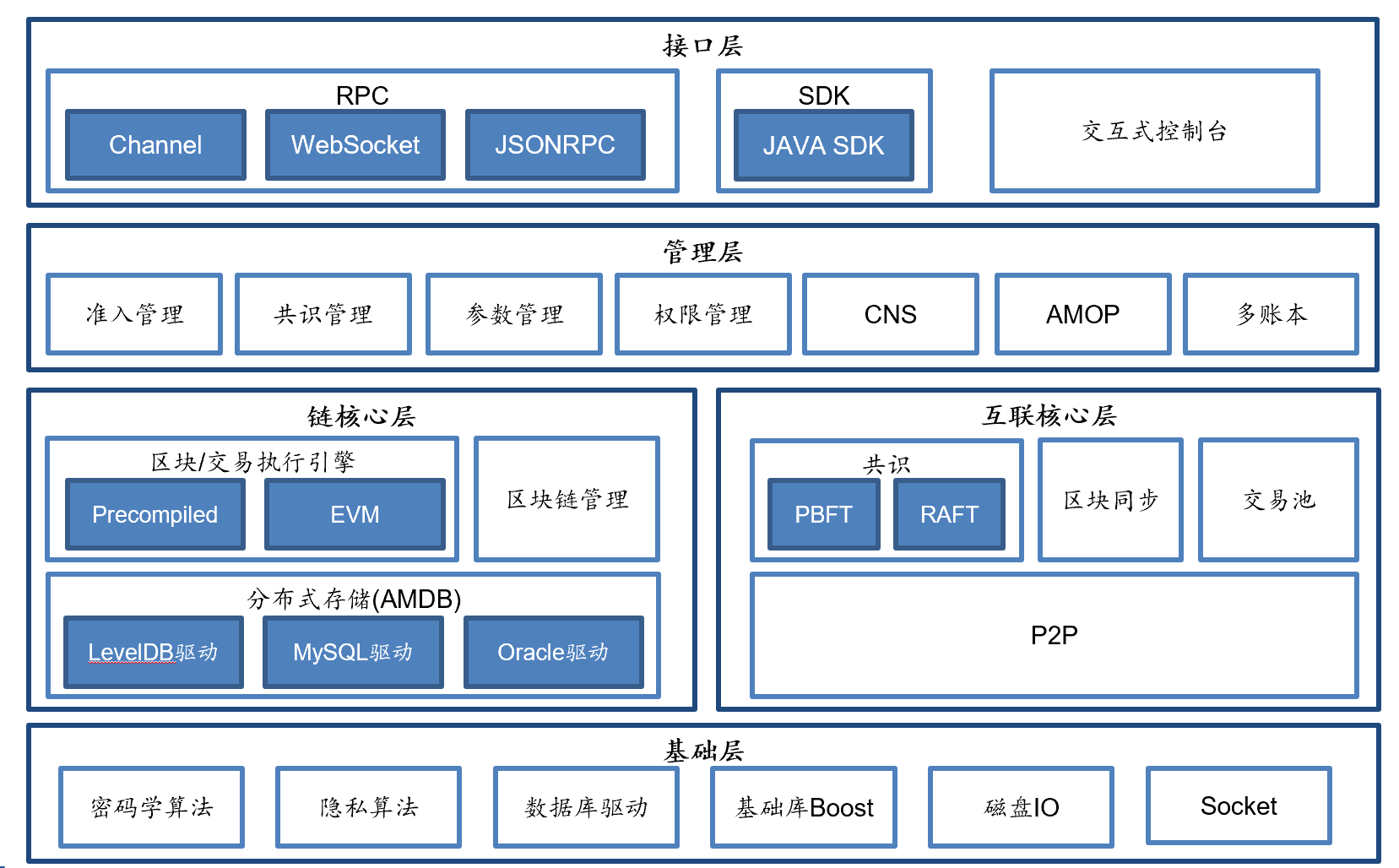
（3）根据种质资源标识（科技资源标识）和时间范围进行溯源。假如在某个种质资源流通过程中，出现了诸如数据被篡改、盗用、误用、丢失等安全问题，并造成了严重的损失，可以根据种植资源标识并精准定位时间来查找资源在生命周期的哪一个环节出现了问题，导致问题的出现的管理员是谁，并据此追责。与此同时，种质资源标识和种质资源是一一对应的，一个种质资源标识对应一个种质资源详细信息，据此在溯源时，每一条溯源记录都会对应一个资源，极大的方便了溯源管理。

## 3.2 技术分析

### 3.2.1 FISCO BCOS 区块链

本课题拟采用FISCO BCOS作为区块链底层平台。FISCO BCOS（Be Credible，Open & Secure） 是安全可控、稳定易用、高性能的金融级区块链底层平台，由金链盟开源工作组于2017年正式开源。社区以开源链接多方，目前汇聚了超过1000家企业及机构、超过万名社区成员参与共建共治，发展成为最大最活跃的国产开源联盟链生态圈。底层平台可用性经广泛应用实践检验，数百个应用项目基于FISCO BCOS底层平台研发，超80个已在生产环境中稳定运行，覆盖文化版权、司法服务、政务服务、物联网、金融、智慧社区等领域。

FISCO BCOS平台架构如下图所示，整体架构可以分为基础层、区块链层、管理层、接口层。基础层包括底层加密算法、哈希算法、数据库的驱动、存储和网络链接等功能，是整个平台最接近硬件的一层；区块链层可以分为链核心层和互联网核心层，链核心层规定了节点内部交易产生、执行、存储的方式，主要包括预编译合约引擎、以太坊虚拟机、分布式存储驱动以及整个区块链的管理方式，互联网核心层规定了节点之间的通信方式，主要包括的共识算法、节点的区块同步方式，节点间采用P2P（点到点）的方式进行信息交流；管理层的主要功能是对链上节点的状态进行管理，包括节点状态、节点权限、节点之间的消息传播等；接口层定义了链上节点和外界信息交互的方式，主要调用方式是RPC（Remote Procedure call，远程过程调用 ），并提供了多种语言的的SDK来支持对链上的调用。



在3.1.2中所述，本系统拟采用NoSQL的方式来存储存储，但是在Soldity语言中，数据只能存储在合约内部诸如数组，映射，字符串等数据结构中，要实现对数据的上链和存储，需要编写很复杂的智能合约逻辑，才能勉强实现溯源需求。FISCO BCOS平台提供链上节点管理、分布式存储、预编译合约引擎、CRUD接口以及KV和SQL两种链上数据存储方式等功能，极大的简化了区块链应用开发，为解决上述问题提供了思路。

FISCO BCOS平台底层存储结构并没有使用传统的MPT（Merkle Patricia Tree）存储结构，而是采用了基于表结构的方式，一方面避免了世界状态急剧碰撞而导致的性能下降问题；另一方面，表结构是一个抽象接口层，底层存储能够兼容各种存储引擎，实际平台所采用的底层存储方式为RocksDB，抽象底层存储为表结构使得业务开发更加的方便。平台使用类SQL的存储结构，不会改变区块链具有去中心化、不可篡改、不可逆、匿名等特性，在使用基于表的存储时，数据更方便管理，且在执行合约时，对表结构数据的读写请求可以不再经过MPT，依赖缓慢的EVM引擎，而是直接访问存储，读写性能得到了很大的提升。得益于上述存储设计，FISCO BCOS平台顺势提供的预编译合约引擎和SQL数据存储方式，是解决本系统数据在链上存储的不二选择。

### 3.2.2 Redis协议

Redisk客户端和服务器通信使用名为RESP（REdis Serials Protocol）协议，此协议是专门为Redis设计。Redis协议的设计是三个原则的折中：实现简单、解析速度快、便于人类阅读，在实现中兼顾了计算机分析、人类理解和实现效率，是非常优秀的通信协议。Redis客户端和服务器端采用TCP连接，默认连接端口号为6379，客户端通过向Redis服务器发送指令来实现特定功能的请求，且所有请求均为二进制安全，（二进制安全指可以正确处理任何字符串，无论字符串包含什么字节）服务器端在接收到客户端请求之后，便会立刻对其进行处理，并将应答返还给客户端，所以Redis协议是一个简单的请求-响应请求。

Redis协议是一个支持字符串、错误、整型数、超长字符串和数组等多种数据结构类型的序列化协议，协议主要内容分为请求和响应两部分。

请求协议。在Redis的协议请求中所有请求的结尾和每一部分的结尾必须是CRLF（\r\n）；请求必须指定命令参数的个数，每一个参数的长度以及每一个参数的内容，用以满足二进制安全，参数包含Redis命令本身，其中每一个参数以 \* 为开头，每一个参数的长度以 $ 为开开头，参数的内容无需特殊符号开头。例如如下Redis指令：set key value，在请求发送时应该被转换为：

\*3\r\n$3\r\nSET\r\n$5\r\nmykey\r\n$7\r\nmyvalue\r\n。

响应协议。Redis服务器不同的响应以不同的特殊号开头，一共包括以下五种，

1. 简单字符串回复（Simple Strings reply），第一个字节是“+” 。
2. 错误回复（Errors reply），第一个字节是“-”。
3. 整数回复（Interger reply），第一个字节是“.”。
4. 批量回复（Bulk Strings reply），第一个字节是“$”。
5. 多条批量回复（Multi Bulk Reply），第一个字节是“\*”。

简单字符串回复，是一共状态回复，或者说是单行回复，是一段由“+”开始，“\r\n”结尾的单行字符串，常常用来作为状态回复（status reply），例如常用的状态回复：+OK。

错误回复和简单字符串回复很相似，在内容上没有什么区别，唯一去区别为开始字符为“-”，具体内容字符串一般是原生报错内容。

整数回复是一个以“:”开头，并以CRLF结尾表示的整数，一般整数回复不会代表很多意义。例如：“:20\r\n”，“:0\r\n”等都是一共整数回复。

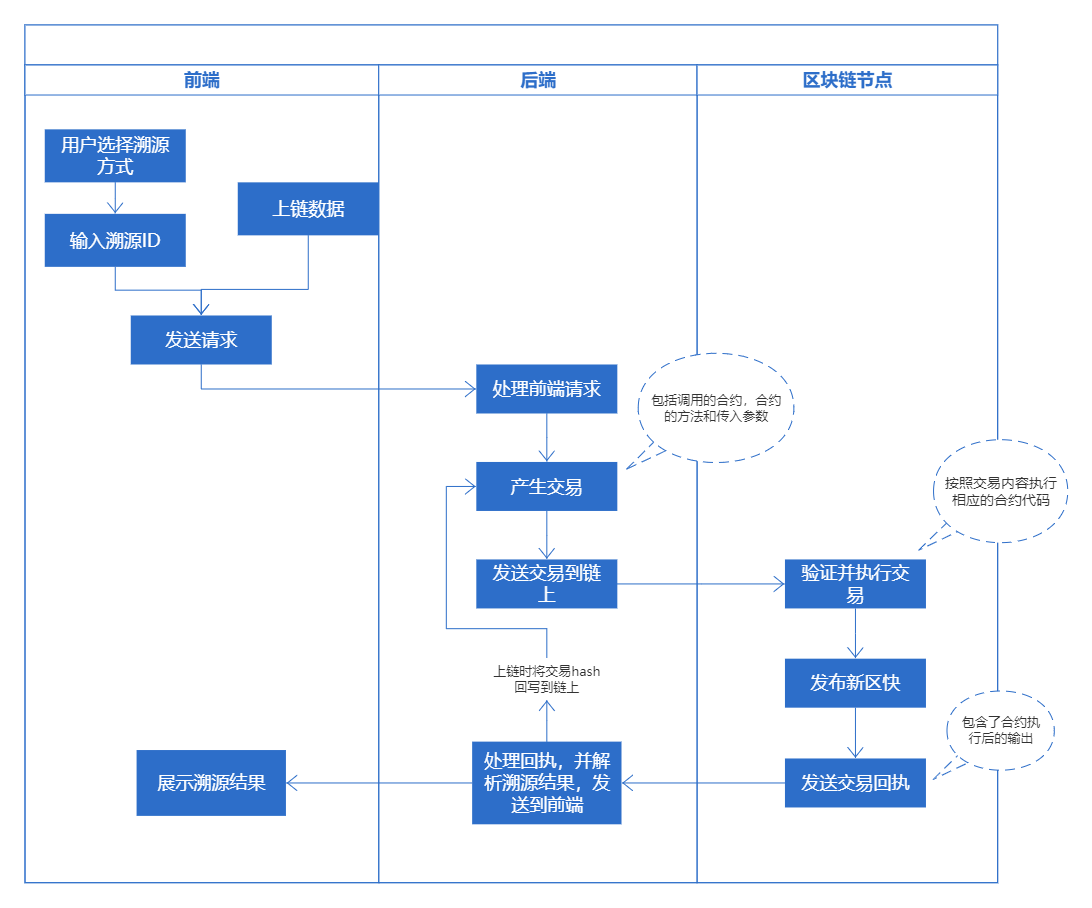
批量回复以“$”开头，被用来返回二进制安全的字符串，字符串的最大长度为512MB，需要按照类似请求的格式，必须同时包含参数长度和参数内容，每一部分结尾都使用“\r\n”。例如：假设服务端返回的内容为“computerScience”，则实际回复应该转换为“$15\r\ncomputerScience\r\n”。此外，当执行批量回复的指令的请求时，如果在请求对象不存在时，则返回空批量回复（NULL Bulk Reply）。

多条批量回复第一个字节是“\*”本质是一个数组，其中包含很多回复，每一个回复的类型都是任意类型的，包括多条批量回复本身，即允许进行嵌套回复。同样的类似请求格式，必须同时包含回复数组的长度和每一个回复的内容。同样的，在执行多条批量回复指令的指令的请求时，如果请求对象不存在，则返回空批量回复（NULL Bulk Reply），不 能返回一共空数组。

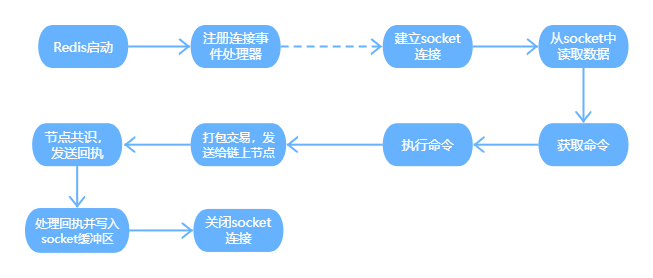
由于Redis协议的独特设计，使其协议实现性能和二进制协议的实现性能相媲美，并且如此简单、轻巧、可扩展性极高的设计，使得大部分高级程序设计语言都能够轻易地实现这个协议，尽管此协议为Redis所专门设计，但可以使用在其他的客户端-服务器的通信模式的软件上。本课题将根据科技资源场景，实现基于Redis通信协议的中间件，由于存储介质的特殊和使用场景需求的特殊（数据上链、数据溯源）会将Redis原生指令的意义进行稍作改变。此外由于Redis官方指令数量非常多，其中大部分指令在本课题的应用场景下，有着难以甚至无法实现或者无实际使用意义的情况，所以本课题只实现了部分Redis指令，同时增加了部分Redis不具备的指令。

## 3.3 系统结构设计

图是本课题在种质资源服务管理系统的基础上所开发溯源系统的结构流程图，主要分为前端、后端和区块链节点三部分。前端主要功能是根据管理员行为获取上链或溯源数据发送请求到后端，并对溯源结果进行展示。后端主要功能是处理前端请求，根据不同的请求调用不同的合约接口并打包交易发送到七区块节点，同时对区块节点的交易回执信息解析根据不同的接口选择是再次发送交易或将结果返还给前端。区块链节点主要功能是执行交易内容，打包区块，实现数据的上链和溯源查询。



图为Redis中间件结构流程图，在结构上和上述溯源系统相似，Redis客户端在和中间件建立连接后发送命令，本中间件解析命令，根据命令调用智能合约接口，实现数据的上链和溯源，并将结果返回给Redis客户端。

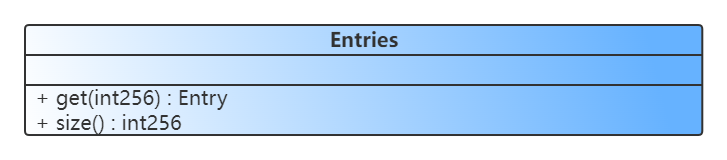
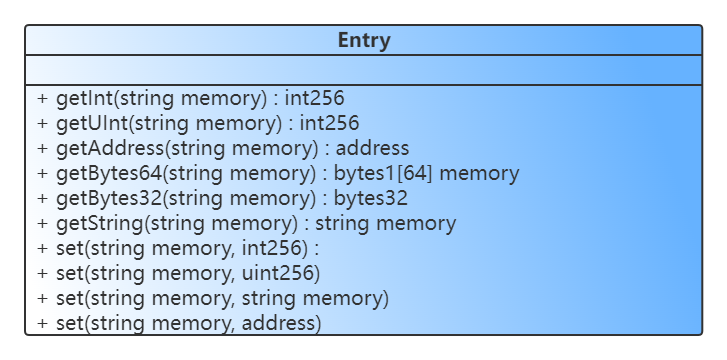
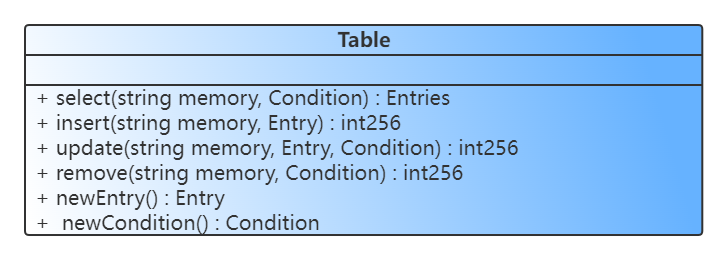
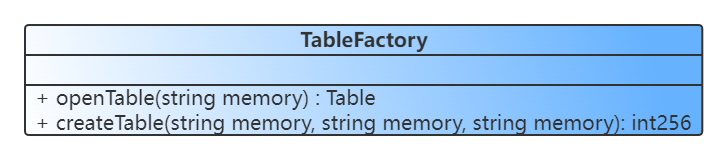


## 3.4 智能合约结构设计

### 3.4.1 FISCO BCOS预编译合约接口

智能合约是运行在区块链上的静态代码，更为具体的说是一段运行在EVM（以太坊虚拟机，Ethereum Virtual Machine）上的二进制代码，在本课题主要依靠智能合约来实现数据的上链和溯源。根据3.2.1的分析，本课题使用FISCO BCOS作为区块链平台，并基于其提供的预编译合约Table.sol接口来实现本课题数据的上链和溯源。

类似SQL中创建数据库的表的过程，FISCO BCOS平台在链上提供了一个固定地址为0x1001的TableFactory合约，合约方法如下图所示，通过调用createTable并传入表的名称，表的主键，表中各属性的名称，即可在底层创建一个对应的表，在使用时通过openTable方法并指定一个表名即可得到该名称所对应表的引用，并根据该引用来对表进行操作。如图是Table合约中所提供的对表的操作，可以对表中数据进行查询、插入、修改、删除，其中Condition类型为每次对表格数据进行操作时，被操作对象应该满足：条件，同样的类似SQL，支持对某一个属性的值进行根据条件进行筛选比如相等、大于、小于、大于等于、小于等于、数量限制的等；Entry类型为表格中的一个表项，包含了一个表项的全部属性信息，如图，Entry类型提供了多种获取属性值的方式，根据属性的名称和属性对应数据的类型选择不同的get方法来获取一个表项种某一个属性对应的值，同时在为表种插入数据时也是通过插入一个Entry的方式进行插入，其中每一个属性值是通过Entry中的set方法进行设置的；Entries类型是Entry的复数，也即代表了多个表项，一般作为select选择的返回值，如图，它仅有两个方法：get和size，它的使用类似数组，我们通过下标和get方法来得到表项数组中某个Entry的引用。



### 3.4.2 表存储结构设计

本课题在上述FISCO BCOS所提供的面向SQL编程的区块链应用开发规范的基础上设计了满足种质资源数据上链和溯源场景的存储结构，主要分为三个表：资源表、管理员表、操作记录表，表结构如下表，其中操作记录表为核心表，用来存储管理员操作的数据，资源表和管理员表是为支持溯源所额外建立的表。

在操作记录表中，Index为主键，并包含了每一个管理员操作对应的数据：管理员ID（AdminID），资源唯一标识（ResourceID），操作的数据（Data），进行操作时的时间戳（Time），操作类型（Operation），该操作数据上链时所产生交易对应的Hash值（TxHash）以及交易所在的区块号（BlockNum），在后端将数据发送到链上，会调用上链对应的智能合约方法，将数据存储在操作记录表中。而在资源表中资源唯一标识为主键，在管理员表中管理员ID为主键，这两个表中的index为属性字段。

在三个表中的Index是为了支持溯源以及解决FISCO BCOS所提供的预编译合约的限制所单独设立的字段。预编译合约主要有两个限制，一个是在使用上述预编译合约中的TableFactory来创建新的表时，只允许创建单主键表，不允许创建双主键表；另一个是在调用Table合约中的select方法时，必须提供一个主键值，否则不能进行查询。而在3.1.3中分析中，本课题既需要根据管理员ID进行溯源，又需要根据资源唯一标识进行溯源，在支持双主键的表结构中，可以直接指定管理员ID和资源唯一标识作为主键，在查询时只需指定主键名和主键值即可，在不支持双主键的表结构中，如果可以不指定主键值进行查询，也可以直接指定管理员ID和资源唯一标识作为查询的属性和值，对表中数据进行查询。但是由于FISCO BCOS预编译合约的限制，本课题提出了一种自定义主键和额外的存储表来解决这样的限制。

Index是上链和溯源合约的全局变量，初始化为0，在后端每一次调用合约中的溯源数据上链方法时将Index加一，利用Table合约中的newEntry方法创建一个空表项，并将Index同上链数据一同set到该表项中之后插入到操作记录表中，与此同时分别创建一个资源表Entry和管理员表Entry将上链数据中的资源唯一标识和管理员ID分别与Index组合作为两个表中的表项，插入到资源表和管理员表中。在根据资源唯一标识或管理员ID进行溯源时，首先根据资源唯一标识或管理员ID查找资源表或管理员表，找到某一个资源唯一标识或者管理员ID所对应的全部Index值。由于在操作资源管理表中，Index是主键值，在拿到一个资源唯一标识或管理员ID对应的全部Index后，即可根据Index在操作记录表进行增删改查等操作。

|  |  |
| --- | --- |
| 字段 | 类型 |
| ResourceID | String |
| Index | Int256 |

|  |  |
| --- | --- |
| 字段 | 类型 |
| AdminID | String |
| Index | Int256 |

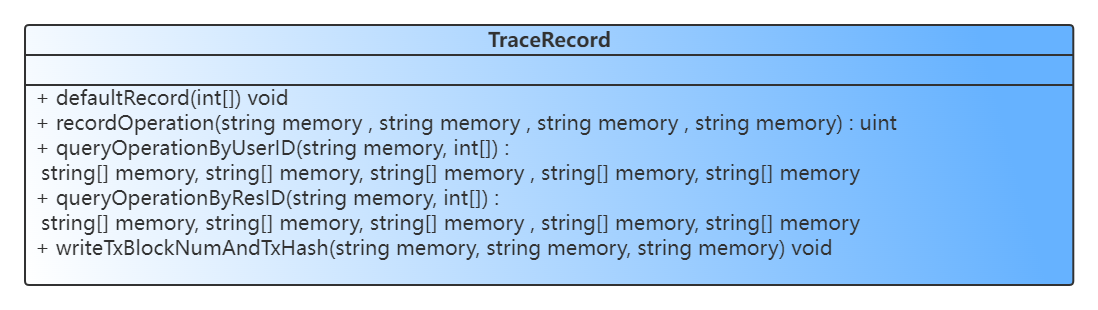
|  |  |
| --- | --- |
| 字段 | 类型 |
| Index | Int256 |
| AdminID | String |
| ResourceID | String |
| Data | String |
| Time | String |
| Operation | String |
| TxHash | String |
| BlockNum | String |

### 3.4.3 上链和溯源合约方法设计

上链和溯源是通过执行合约中的方法，调用预编译合约接口来对表中数据进行操作。如图，本课题智能合约对外暴露了五个接口供后端进行调用，默认溯源方法（deafultRecord），操作数据上链（recordOperation），根据用户ID进行溯源（queryOperationByUserID），根据资源唯一标识溯源（queryOperationByResID），写回操作记录对应的交易哈希值和交易所在区块（writeTxBlockNumAndTxHash）。

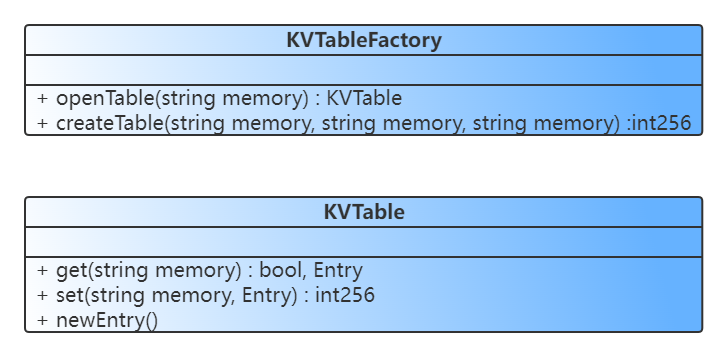
在本课题中由于上链数据不仅仅包括前端操作产生的数据，还包括数据上链时所产生交易的信息，而此信息只能在前端操作产生的数据完成上链操作后才能得到，所以数据上链需要分为两步：第一步是将上链数据包括管理员ID、资源唯一标识、操作时的数据、操作类型等数据存储到3.4.2的三个表中，第二部是在交易完成后，将上链所产生交易的Hash值和交易所在的区块号重新存储到第一步上链的表项中。第一步通过调用recordOperation接口，并传入上链数据，此接口打开3.4.2中的三个表，得到表的引用，将上链数据插入到表中。其中属性Time在本课题中采用区块内置的时间戳来赋值，即获取block.timestamp的值来作为Time属性的值。第一步结束后，链上节点会给后端调用者返回交易执行结果，其中包含了交易执行的各种信息，此时后端通过回调的方式调用writeTxBlockNumAndTxHash接口，并传入产生交易主键值、交易Hash值和交易所在区块号，更新该主键值对应的表项。

溯源接口的实现本质是根据条件在资源表、管理员表和操作记录表中按照条件进行查询，并将查询结果以数组形式返回到后端，主要实现为三个方法：deafultRecord、queryOperationByUserID、queryOperationByResID，其中后两个方法在实现逻辑上是一致的，根据查询的时间范围、需要返回的表项数量（类似SQL的分页查询）以及查询的资源唯一标识或用户ID来设置Table合约中select方法的Condition类型。defaultRecord方法是提供一种最近、最新的一定数量的操作记录，它会根据当前合约状态（Index值）返回最近上链的部分操作数据，以达到前端能够实时刷新得到最新的操作记录。



除上述对外暴漏接口外，由于在Solidity编程中对方法中变量的限制，本课题智能合约设计时，尽可能的抽离出了一部分公共合约代码以简化暴漏接口中参数数量，否则，合约会因在执行栈过深而无法编译。

上述的Index设计在开发初期并未出现什么问题，但在后续对合约的修改和完善过程中发现，当合约重新部署在区块链上时，由于区块链的不可更改特性，无法删除原有合约，重新部署只能部署新的合约。由于利用FISCO BCOS底层预编译合约来存储数据，实现了代码和数据存储分离的形式，重新部署新的合约原有数据仍然存在，但合约中的Index值却被重新初始化0，此时导致原有表中的测试数据无法使用。本课题为支持后续系统对智能合约的应用与开发，建立了一个新的表结构，利用图 中的KVTableFactory合约，建立一个KVTable，名为IndexTable，通过KVTable中的set和get方法，在开发时，每当重新部署合约时便从IndexTable中获得上次合约中Index的值，并在每次数据上链时对Index值更新时，向IndexTable中set更新后的Index值。这样能够保证在开发时每次部署合约能够继续利用以前的测试数据，降低测试时的成本。



## 3.5 前后端设计

本课题在种质服务管理系统的基础上进行设计和实现溯源功能，在原有系统的基础上增量开发了两个功能模块：数据上链模块和溯源模块。

### 3.4.1 数据上链模块

数据上链模块主要功能是收集前端管理员操作产生的数据，处理后发送到后端，后端对数据进行处理，指定调用合约的接口，对数据进行加密、签名、打包成交易发送到链上节点。

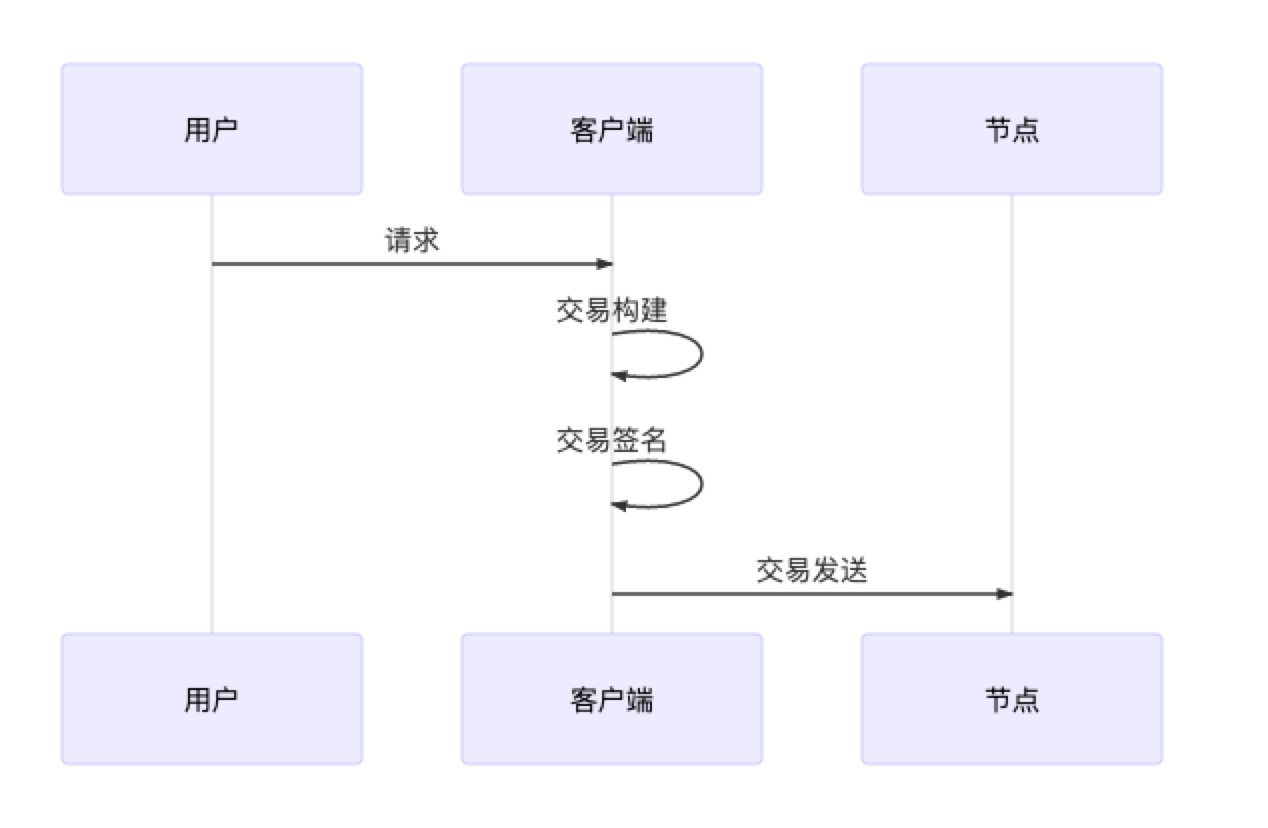
在原有系统中对资源的管理主要将种质资源的收集、录入、鉴定、共享等生命周期过程的业务数据存储到后端中心数据库中，核心逻辑为对种质数据进行创建、修改、查询和删除。在本系统中为支持溯源，不仅仅需要将上述的种质资源业务数据进行上链，同时还需要将进行操作的管理员ID、操作的时间戳、被操作的资源唯一标识以及对资源进行了何种操作的数据传输到后端，由后端进行上链。

管理员ID可以直接通过原有后端接口查询当前登录的用户信息得到；被操作的种质资源唯一标识和资源一一对应，贯穿整个系统，可以轻易得到；操作的时间戳被封装到管理员操作产生的JSON格式数据中，序列化后发送到后端。

**图**为在管理员后台种质资源的生命周期过程流程图，从图中可以看出在整个种质资源流程过程中，每一个阶段都可以对当前阶段数据进行录入、驳回、修改、删除等操作。为详细记录不同操作的类型，本系统定义了13种管理员操作，具体操作类型和意义如表xxx所示。本系统在原有服务管理系统的基础上，根据表中的操作类型，在每一个原有操作结束后增加一步，将通过上述方法所获得的上链数据通过http协议的POST方式调用后端接口，把数据发送到后端。

|  |  |
| --- | --- |
| 操作类型 | 操作意义 |
| CREATE | 创建新的资源 |
| CHANGE | 修改创建资源创建时的信息 |
| COLLECT\_REFUSE | 驳回资源创建申请 |
| COLLECT\_APPROVAL | 同意资源创建申请，并录入 |
| COLLECT\_DELETE | 删除资源创建申请 |
| SAVE\_CHANGE | 修改资源录入信息 |
| SAVE\_REFUSE | 反驳资源的录入 |
| SAVE\_APPROVAL | 对资源进行鉴定，资源进入待共享状态 |
| SAVE\_DELETE | 删除资源录入信息 |
| APPRAISAL\_CHANGE | 修改对资源的鉴定信息 |
| SHARE | 分享资源 |
| NOT\_SHARE | 不分享资源 |
| ALL\_DELETE | 删除资源对应的收集、保存、鉴定、共享信息 |

后端在得到前端请求和数据后，需要构造出一笔交易，并对交易进行签名后发送到链上节点，大体流程如下图所示。



FISCO BCOS提供了Java SDK为后端进行交易的打包、签名、交易发送提供了较为完善的支持，具体过程如下：

1. 构造交易。交易的内容主要采用ABI编码，主要编码内容为所调用智能合约的ABI文件（合约编译得到）和二进制文件、需要调用智能合约的方法名（recordOperation）、被调用方法传入的参数，最终得到编码后的交易内容，接下来将交易发送的目的地址，区块限制主要包括GAS\_PRICE、GAS\_LIMIT等数据进行打包得到未签名、未编码的交易。
2. 对交易进行编码。使用交易编码器对交易进行RLP编码，并通过Hash算法对编码后的内容。
3. 对编码后的Hash值进行签名。通过本地硬件加密或远程签名服务对第2步中的Hash值进行签名。签名是指使用私钥对交易进行加密，解密时只能通过该私钥或该私钥对应的公钥才能进行解密。
4. 拼接未签名交易和签名交易，并发送交易。将第2步得到的交易同第三步签名后的交易进行拼接，拼接后通过RPC的方式发送到链上节点中。节点收到交易后，能够使用公钥对签名后的交易解密，与未签名的交易进行对比，进而对交易的合法性进行验证，节点会将一段时间内接收到的合法交易进行打包一个共区块中，然后发布新区块。

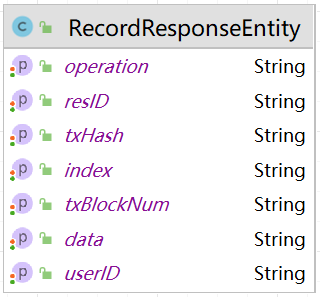
### 3.4.2 溯源模块

溯源模块的过程类似数据上链模块，主要有两部分不同：向后端发送的数据不同，后端调用的合约方法不同。此外前端溯源模块需要展示溯源结果，前端需要增加页面来实现溯源信息的展示。

溯源模块向后端发送的数据根据溯源的方式不同分为三种：

1. 默认溯源。此方式是一种默认展示最近、最新的管理员操作（defaultRecord接口），但也允许仅指定时间范围来查找一定时间范围的操作记录
2. 根据管理员ID和时间范围溯源。在此方式下，用户在前端通过输入管理员ID和时间范围（实际为两个时间戳）作为溯源的信息，前端将数据发送到后端，后端调用合约中的queryOperationByUserID接口来对链上符合条件的数据进行查询。
3. 根据资源唯一标识和时间范围溯源。同（2），只是前端输入的是资源唯一标识和时间范围，后端调用合约中的queryOperationByResID接口。

在上述过程后，节点在执行完合约，达成共识后，会向后端返回交易执行回执其中包含了合约执行过程中的各种信息：包含该交易所在区块Hash，区块高度、执行的合约地址、发送者地址、交易消耗的Gas、执行合约方法传入的参数、合约执行过程中产生的Log信息、合约方法执行后的返回值、交易的Hash值、交易在区块中的序号、交易的状态等。由于这些信息在交易回执的格式是16进制编码数据，很难被人类阅读，特别是其中的合约执行后的返回值，其内容是溯源的结果，所以后端会根据合约的ABI编码和二进制文件对交易回执中的部分字段进行解析，将其转换为UTF-8字符串或者10进制数据等便于人类阅读的格式。后端会将溯源结果进行处理后转换为RecordResponseEntity实体类（图），每一个该实体类对象对应着一条溯源结果，并转换为JSON格式数据发送到前端。



前端在得到后端溯源查询的结果后通过页面进行展示，本系统根据种质资源的收集、录入、鉴定、共享四个流程状态，将3.4.1中定义的操作类型划分为四大类，每一类对应一个流程状态，每一个流程状态需要一个页面来展示进入此流程状态的信息。CREATE、CHANGE、COLLECT\_REFUSE是在收集状态下的操作，SAVE\_CHANGE、SAVE\_REFUSE、COLLECT\_APPROVAL、是已经录入状态下的操作，SAVE\_APPROVAL、APPRAISAL\_CHANGE是鉴定状态下的操作，SHARE、NO\_SHARE是待分享状态下的操作，剩下的操作类型为删除操作，此操作并无信息记录。每一流程状态下的信息是一致的，本系统在此基础上构建了四个不同的信息展示页面，展示操作时的数据，由于操作的数据为JSON数据序列化后的字符串，所以在展示数据前需要将RecordResponseEntity中的data对应的字符串在前端进行反序列转换为前端JSON数据对象。

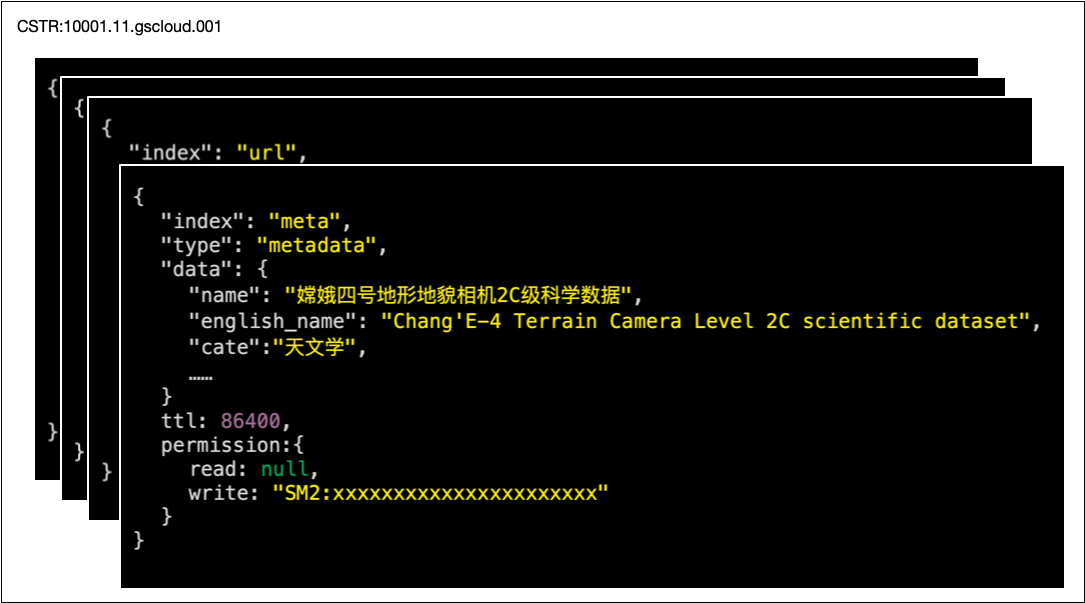
除上述流程页面外，为保证溯源信息的准确和真实，本系统提供该溯源记录上链时所产生交易的信息，为此本系统设计了两个页面来展示交易和交易回执信息。针对交易和交易回执中的输入和输出，在前端提供简便方式能够在16进制的数据信息和解码后的字符串进行转换。

## 3.6 Redis实现

本系统分析科技资源标识数据集的数据特征和结构，采用FISCO BCOS区块框架作为底层支持，使用Rust实现在客户端和区块链间的Redis中间件，并根据科技资源标识应用场景设计智能合约和链上存储结构，屏蔽底层区块链结构，降低开发难度，简化开发流程，为科技资源标识数据上链和溯源提供支持。整个中间件的实现分为三部分：链上数据存储结构的设计和分析、Redis协议解析的实现和Redis指令的实现。

### 3.7.1 智能合约和存储结构设计

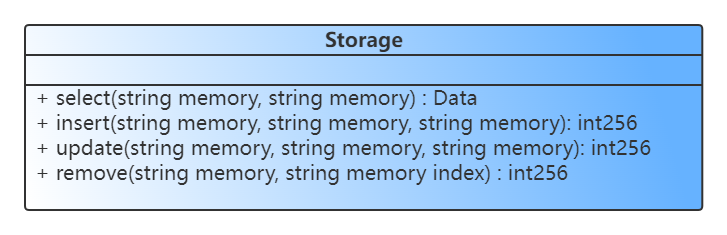
在共享网的标识系统中，每一个科技资源标识对应一组标识数据集，数据集中的数据使用不同的索引进行访问，每一个数据项都采用JSON格式进行组织。如下图，标识数据由若干个数据项组成，不同的数据项根据不同的索引（index）进行区分，index索引目前主要包括“node”，“meta”，“url”，每一个index对应一个不同的数据项，一个科技资源标识即图中左上角的字符串，可以对应多个资源标识。



在共享网标识系统中，一个标识的生命周期是从创建、查询、修改、删除的过程，只是会进行简单的数据增删改查，所以在链上的数据存储结构不需要像3.4.2中所设计的复杂表结构。由于仍然使用FISCO BCOS作为底层区块链平台，类似3.4.2的设计，根据上述数据结构特征设计了如下表结构，Id字段是表的主键，由于一个科技资源标识对应一组数据集，本系统采用此标识作为主键Id字段的值，Index字段对应上述数据集中的Index字段，在标识系统中一个Index和科技资源标识能够唯一确定一个数据项，所以Data字段代表该数据项的内容，Deleted代表了一个数据项是否已经被删除，用来支持对数据的删除操作。

|  |  |
| --- | --- |
| 字段 | 类型 |
| Id | String |
| Index | String |
| Data | String |
| Deleted | Int256 |

由于对科技资源标识只有创建、修改、查询和删除的操作，在设计智能合约时对外暴露了四个接口，分别对应为创建（insert），修改（update），查询（select）和删除（remove），其中Data类型为一个结构体，包含了上述表中的一个表项的数据。



### 3.7.2 Redis协议解析的实现

在实现Redis协议解析前，Redis中间件需要同Redis客户端建立TCP连接，本系统使用Rust的TcpListener打开服务器端口，并监听外部连接，在建立连接后，Tcp连接数据转换为TcpStream的实例化对象，客户端每发送一条命令，都会通过TcpStream构建为Framed结构，可以对其中的数据流进行接收和发送。

Redis协议解析的实现需要完成两个任务，一个任务是能够对Redis客户端所发送的包含Redis命令的数据进行解码，从中获取指令以及指令所对应的参数，另一个任务是将指令执行后的数据编码为符合Redis协议，能够被Redis客户端解析和接收。

对Redis客户端发送的数据解析，由于Redis协议的设计简单并符合二进制安全，在进行解析时只需要从开头到结尾进行顺序遍历即可。请求指令的开头均为“\*”，并在后面跟上了参数个数，每一个参数以“$”开头，并给出了参数的长度，在解析时只需进行逐个解析即可。

对于Redis回复数据的封装，需要根据指令的类型进行不同封装。数据封装的过程是将原有数据开头根据返回数据类型不同增加不同的开头字符，并在返回数据和参数后面拼接“\r\n”。由于Redis协议的简单设计，在实现过程中所有的数据只需要遍历一遍，在相应位置拼接对应的特殊字符即可。

### 3.7.3 Redis 指令实现

本文根据科技资源标识上链和溯源的需求设计并实现了一部分Redis原生指令，并根据溯源需求扩展了部分指令。

set指令和mset是Redis的原生指令，set指令原生一般使用形式为set key value，为适应一个科技资源标识对应多个index，每一个index对应一个数据项的场景，本系统对原有set和mset的意义进行了修改和补充。第一部分修改是在将一个数据项上链时，需要采用“id#index”的形式作为key，其中“#”作为一个分隔符，分隔科技资源标识和index，此时比如要将id为“CSTR:17977.08.GBWE091117”，index为“meta”的数据项上链，指令可以为：

set CSTR:17977.08.GBWE091117#meta data

其中data为数据项JSON格式数据的序列化结果。由于一个科技资源标识可以对应多个index，每次使用一次set来对一个index进行数据上链是比较繁琐的，本系统第二个修改部分为mset指令的意义，采用和set指令同样的方式，每一个key必须包含上链的科技资源标识ID和数据项对应的index值，命令示例如下：

mset id#index1 data1 id#index2 data2 id#index3 data3.......

这样的方式能够简化一个科技资源标识数据上链的时间和操作，多个数据项只需要一个mset指令即可。set指令和mset的主要作用是对上链数据进行创建和修改，在中间件拿到set指令后，会执行和3.4.2中类似的过长，调用合约中的方法将向3.7.1的表中插入或更新数据。

get指令和mget和是Redis原生指令，get指令原生一般使用形式为get key，主要功能是通过指定一个key值来获取Redis数据库中存储的对应value值。同样类似set和mset指令，在本系统中get指令和mget的key必须包含科技资源标识和数据项对应的index.。get和mget指令的主要功能是实现对链上数据的查询，根据科技资源标识和数据项对应的index查询一个数据项的内容。

del 指令是Redis原生指令，del指令使用形式和mget指令类似一般为

del key [key]

也即允许一次性删除多个key值的数据。在本系统中，同样的key的意义同mget指令中key的意义。del指令的主要功能是删除某一个科技资源标识指定index的数据项，由于考虑到需要对溯源的要求，在数据被删除时，会将表项中的deleted的值由0改为被删除时的时间戳，其他非主键字段都被清除为空。

在上述del指令中，删除时只能指定某个或几个index值，但是由于无法知道一个科技资源标识所对应的所有index，所以在对一个科技资源标识的彻底删除del指令做不到，为此本系统设计并实现了扩展指令del\_all。指令形式类似get指令：

del\_all id

指令实现的主要思路是既然在del时无法拿到一个标识对应的全部index，那么就在每次set时对index进行记录。为此单独设计了一个index字段值为“all\_index”，此时在每次执行set操作时，根据该标识ID，以“all\_index”作为index的值得到该标识对应的全部index字段取值，并将当前set操作所传入的index值和全部index取值进行比较，若不存在，则额外执行一次set操作，更新all\_index所对应的数据项。如此以来，在执行del\_all命令时首先查询该标识的全部index字段取值，然后执行del命令，将全部index删除。

为支持溯源，记录对科技资源标识的操作，本系统设计了history指令。history指令的格式是：

history id index

指令意义是查询某一个科技资源标识数据集中某个数据项的历史操作记录。为记录一个科技资源标识对应的历史操作记录，类似del\_all指令，在这里增加了一个index字段取值“history”，其数据项内容为一个JSON数组，记录了这个科技资源标识数据集中的全部操作记录。一个数据项的操作记录是在这个数据项被删除或被修改（包括创建）时，将操作数据记录到链上，执行过程类似3.4.2中后端回调的方式，在数据项被删除或修改（set 或 del指令）时，链上节点在执行交易结束后会给交易发送者一份交易回执，其中包括了交易的hash值、交易执行的时间戳等交易具体信息，通过将交易hash值、被操作的index字段值、交易时间戳以及version（格式化后的交易时间戳）组成四元组作为一条操作的历史记录，然后查询当前history数据项中的数据，将上述四元组转换为JSON格式的字符串插入到当前history数据项的末尾，并更新history数据项的内容（执行insert方法）。在记录历史操作记录后，当使用history命令进行查询某个科技资源标识的特定数据项历史记录时，首先得到当前科技资源标识在链上存储的history数据项，然后筛选数据项中全部历史记录中和当前所查询的index值相等的记录，然后将其编码为Redis协议数据，返回给Redis客户端。

在上述通过history指令获得一个操作的历史记录后，每一个历史记录有四个字段：index，timestamp，tx\_hash，version。index即为该数据项index字段值，timestamp是上链时间戳，tx\_hash是交易的哈希值，version作为溯源时的一个版本信息，可采用get“id#index@version”的指令来对某一个历史操作后数据的信息，以此来支持对科技资源标识的溯源查询。

# 4 成果展示

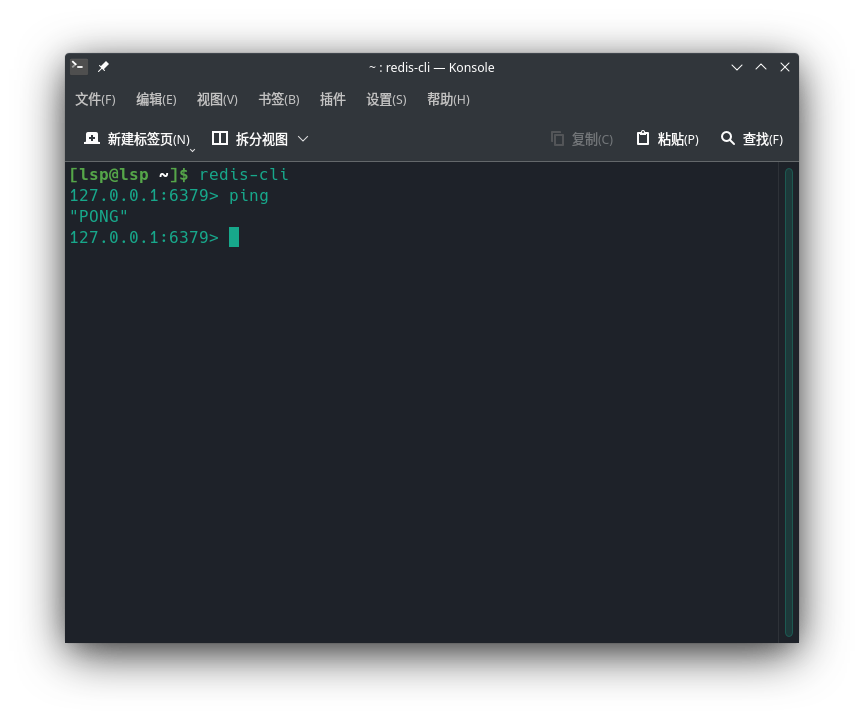
## 4.1 溯源系统展示

溯源系统最终实现和表现为

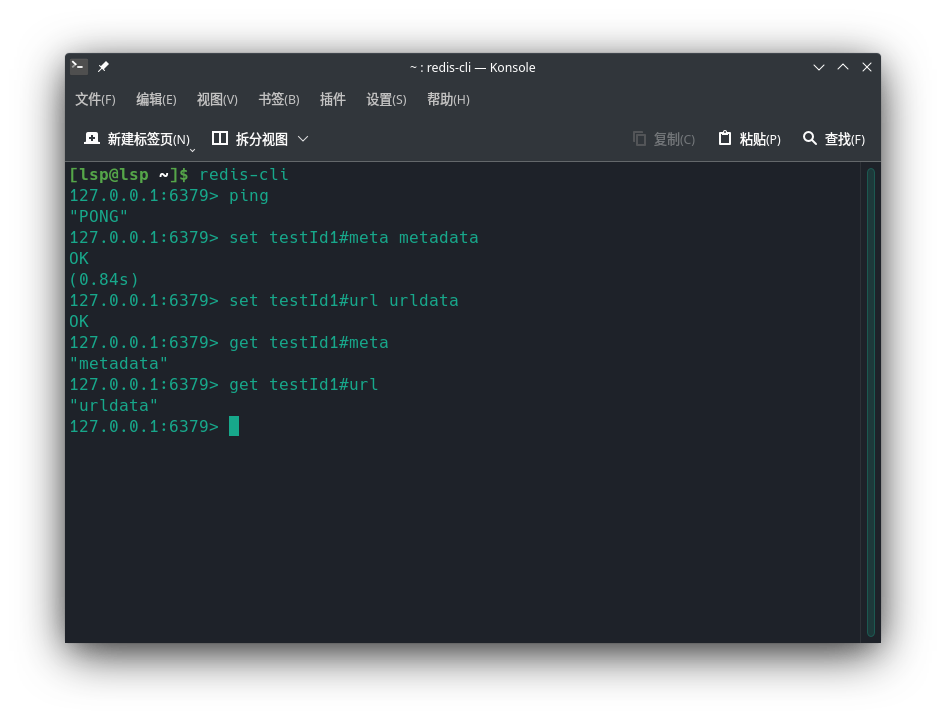
## 4.2 Redis中间件展示

Redis中间件的实现展示采用Redis官方工具包提供的redis-cli作为Redis客户端和Redis中间件进行连接，中间件运行在本机6379端口。在使用redis-cli作为客户端能够正常运行，说明本系统能够在其他实现Redis协议的客户端情况下正常使用。

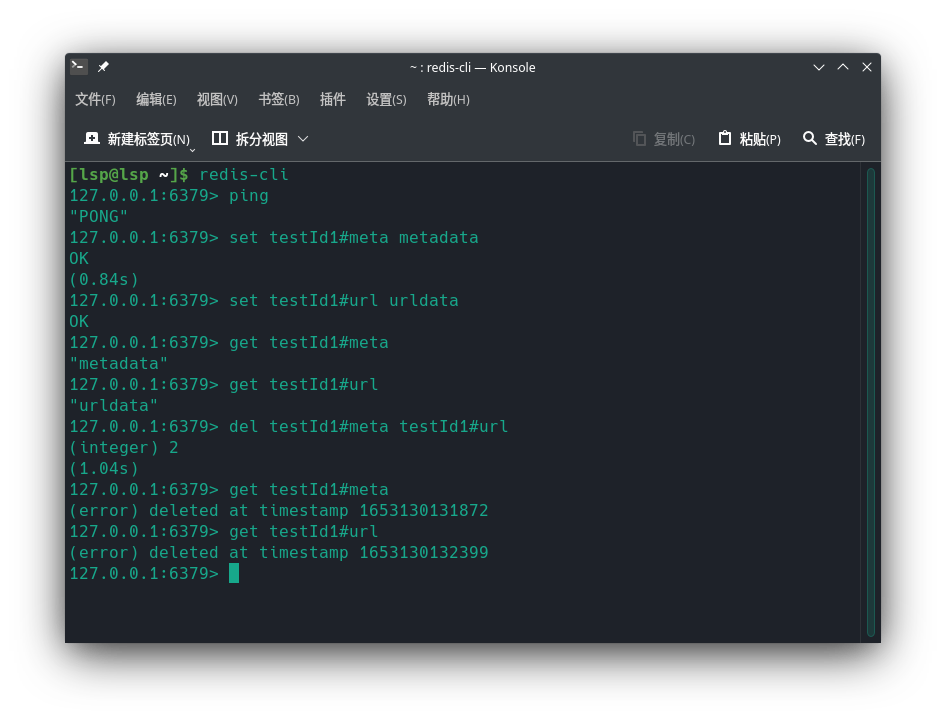
在使用redis-cli连接成功后，使用ping指令来验证是否连接成功，连接成功会返回PONG，如图，连接成功。



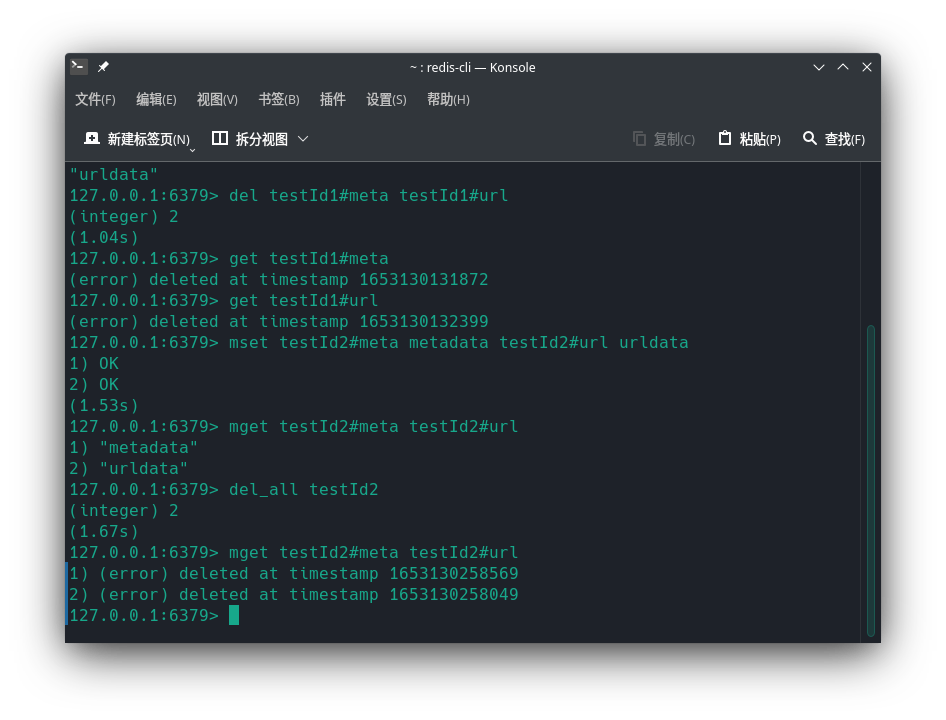
使用set指令向链上存储一个资源的数据集，在这里采用测试数据的形式，资源标识为testId1，数据也为测试数据，正式数据中也为字符串，在测试数据运行成功表明在真实数据中也能够成功。如下图



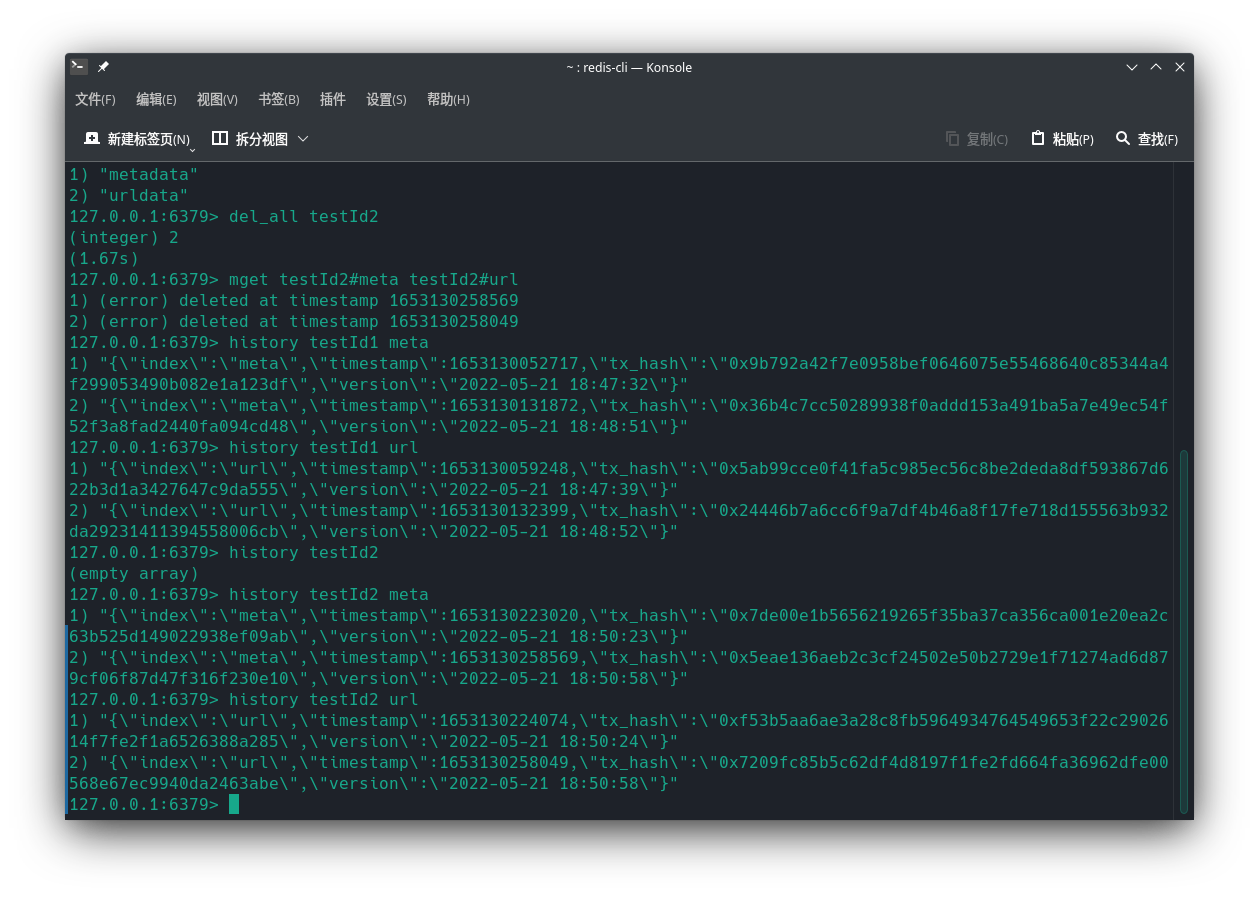
del指令，在上述过程中设置了两个index值分别为meta和url，在删除时可以选择删除其中一个或两个，如下图，删除了两个index数据项，再使用get命令会无法得到数据项的值。



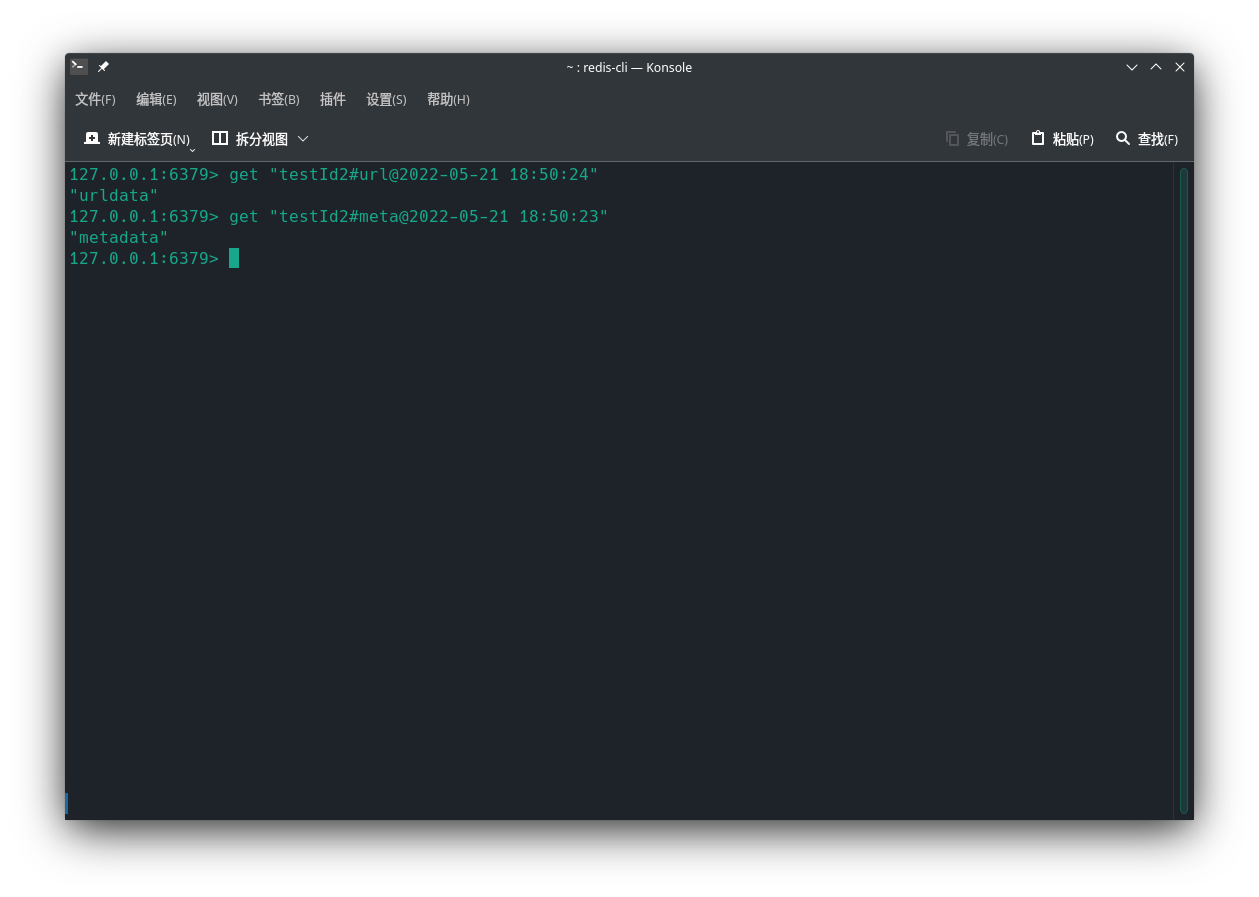
del\_all 指令是删除一个id全部数据项，如下图，在这里使用mset指令和mget指令，可以看出在del\_all之后全部index都会被删除，无法再通过get获取。



history指令来展示testId1和testId2的操作记录如下图，从图中可以看出每一个meta和url数据项都有两个历史操作记录，包含创建和删除两项记录。



在查询到history信息后，我们可以通过get命令来查询指定版本的内容，来达到溯源的目的，如下图。



# 5 致谢

# 参考文献