装
订
线

大连海事大学

毕 业 论 文

二〇一七年六月

基于秘密共享的
多方协作撮合服务系统设计

专业班级：　　软件工程 3 班

姓　　名：　　廖　　添

指导教师：　　李　志　淮

信息科学技术学院

摘　　要

目前来看大部分合作平台倾向于尽可能地收集用户信息，将其加入到搜索引擎的资料库中，以便其他用户进行匹配。这种简单直接的服务模式虽然能够在一定程度上解决不同用户之间寻求合作的需求，但还不够理想。本论文所设计的系统更专注于寻找和设计一个有效的解决方案，提供精细化、智能化的合作匹配服务。旨在严格保护用户个人信息的同时，更能够非常准确的匹配出一个合适的合作结果。而且相较于目前流行的中介服务模式，更加安全和智能。

系统为用户提供的所有服务都围绕着四个基本原则进行设计：1.严格地保护用户信息安全；2.使用分布式计算结构将服务分散化；3.允许具体服务具有多版本，能够精细化、快速演进迭代；4.支持模糊计算、同时可将服务智能化。通过保证这四个基本设计原则，能够有效地保证系统在设计和开发的过程当中始终不偏离最初的设想和要求。

本论文先从阐述设计一个安全的合作撮合系统的必要性和相关背景开始。介绍系统使用和依赖到的若干关键基础技术，并举例其典型的应用方式和案例。然后再描述系统方案的需求分析、概要设计和详细设计，逐渐地绘制一个清晰完整的系统轮廓。最后展示系统的原型和相应的测试用例。具体以若干完整的假设用户和场景，通过数据和结果来论述系统的功能和效果。

关键词：合作；匹配；信息安全；P2P；秘密分割；区块链

ABSTRACT

At present, most of the cooperation platform tends to collect user information as much as possible, add it to the search engine database, so that other users can use them to match their requirements. This simple and straightforward service model, although able to a certain extent, to solve the needs of different users to seek cooperation, but not ideal. The system designed in this paper is more focused on finding and designing an effective solution to provide a meticulous and intelligent partners matching service. Designed to strictly protect the user's personal information at the same time, more able to match the exact match with a suitable result. And compared to the current popular intermediary service model, more secure and intelligent.

All services provided by the system are designed around four basic principles: 1. Strict protection of user information security; 2. Use of distributed computing architecture to decentralize services; 3. To allow specific services with multiple versions, to refine, fast evolution iterations; 4. Support fuzzy computing for providing intelligent services. By ensuring these four basic design principles, can effectively ensure that the system in the design and development process which does not always deviate from the original ideas and requirements.

This paper begins with the necessity and background of partners matching services. Introduce the key technologies used by the system and rely on, and give examples of its typical application and case. And then describe the requirements of the system program analysis, summary design and detailed design, and gradually draw a clear and complete system profile. Finally show the prototype of the system and the corresponding test cases. Specifically with a number of complete assumptions on the user and the scene, through the data and results to discuss the system functions and effects.

Keywords: Cooperation, Matching, Information Security, P2P, Secret Sharing, Blockchain

目录

第 1 章 绪论

1.1 系统开发背景

信息技术不仅提升了工业效率，而且改变了团队合作的方式和合作理念。例如，人们可以在几分钟内通过发送电子邮件或更新到云服务器来提交作品，而不是乘坐出租车，花几个小时到达目的地，并将文件交给对方。另外，如果人们想建立一个网站作为他们公司的门户网站，他们可能不需要聘请一个正式员工来建立这个网站，而是可以在网上咨询自由职业者或者相关服务供应商。因此，限制员工时间和空间的传统工作形式将会逐渐消失，一种全新的合作方式将取代它。这样一来，每个参与者都可以专注于自己的专业知识，而不用将重点放在克服物理限制和就业人事关系等事务上。人们通过自己的专业技能和知识为需求而工作和努力，让平台服务管理人事关系，从而淡化雇主和雇员之间的界限概念。

目前来看，互联网上的“Elance”，“Odesk”和“Freelancer”等现有平台提供的众包服务，不利于保护用户的创意、隐私等重要信息。尽管这类平台造就了蓬勃发展的市场和互联网社会关系，但仍然存在一个重要问题需要解决：如何在保障用户信息安全的前提下，提升创造者和创新者的团队建设工作的效率和成功率？本文试图为解决该问题提出一个可行的解决方案并实现一套服务系统。

由于中心化设计本身存在难以解决的信任风险问题，本文会基于目前已经拥有成功实践经验的去中心化技术，设计新的平台结构，提供一个在具有去信任前提的同时也能够保障用户信息安全，且能够为需求供需双方提供可靠匹配服务的解决方案。

1.2 国内外现状

关于提供服务供需双方匹配的服务，国内业务和技术都比较成熟是“猪八戒网”以及“商理事”两大平台。其中“商理事”是基于企业资源共享和 SaaS 模式的企业合作撮合服务平台，运用企业智能、大数据技术以及云计算技术以尝试重构商业合作营销方式，以“企业网” 、“资源网”、“BD 网” 三网为中心，不同于传统的人工获取销售合作线索和粗颗粒度营销合作方式，融合商机搜索引擎、商业数据库、商业资讯以及活动等功能，通过主动查询和智能推送为商业从业者

提供企业资源服务。“猪八戒网”是服务众包平台，创办于 2006 年。涉及的服务交易品类涵盖创意设计、网站建设、网络营销、文案策划、生活服务等多种行业。“猪八戒网”有大量服务商为企业、公共机构和个人提供定制化的解决方案，将创意、智慧、技能转化为商业价值和社会价值。

与以上平台专注业务类似的国外服务提供商有以“Elance”为代表的大量外包网站，也有像“MatchPool”这样的创新类用户匹配服务网站。其中“Elance”是国外成熟的一套业务外包平台，外包项目类型以软件和网站为主，这个平台上包含平面和动画设计，网站设计，软件编码设计，商业计划寻找技术合作商等各类需求。其主要业务和模式都与国内的“猪八戒网”相似。而“MatchPool”则基于虚拟货币以及区块链等技术，加之新的匹配机制和算法，提供一个去中心化的用户社交匹配服务方案。

1.3 解决的问题

首先，创新创业者和普通社会公司员工之间的对于合作需的求的差异在于创新者通常需要保护他们重要的创意和资料，在寻找合作伙伴时不被能泄漏和被盗。因此，收集大量用户信息和私有数据的通用服务模式（集中式）具有严重的数据安全问题。一方面，创新创业者会考虑避免上传数据安全性重要的文件，因此很难获得一个找到合作伙伴的好机会。另一方面，即使有很多用户在网站上公开他们的想法，以吸引好的合作伙伴，很有可能使网站成为一个免费创意的搜索引擎，无法响应用户的期望。用户上传他们的信息和资料到网站，是因为用户相信它。但是，如果网站的运营商私下背叛用户，使用这些用户数据获得更高的黑色利润呢？没有人可以给出一个肯定的承诺，这样的问题不会在集中式技术中发生。因此，去中心化的解决方案能有助于我们找到一种相对正确的方法来保护用户的数据安全并保持服务的可信度。本论文提出的主要框架是设计为去中心化的分布式解决方案。它使用一些 Peer-to-Peer（以下称 P2P）技术和秘密分割加密来确保网络中没有包含所有或大部分用户数据的节点，用户可以自由选择多个节点来存储其信息片段。设计的算法和结构保护用户的信息片段不被恢复，除非相反是真正的潜在合作伙伴。

下面列举分析现有网络平台所存在的问题：

1) 集中化存储个人用户信息，信息泄露风险高，并且存在平台私自利

用用户信息进行保密协议外其他用途的可能风险。仅中国范围内，据中国互联网协会的一项调查报告，国内网购用户的规模已高达约 4.8 亿人，其中过半在网购过程中经历过个人信息泄露的不幸遭遇，而由此造成的经济损失每年高达 915 亿元。

表 1.1：近年发生的用户数据泄露事件举例

- 雅虎，时间 2016 年 9 月，受影响用户数量至少 5 亿；
- 英国宽带服务提供商 TalkTalk，时间 2015 年 10 月，受影响用户数两至少 400 万；
- 智联招聘，时间 2014 年 12 月，86 万条求职者简历数据泄露。

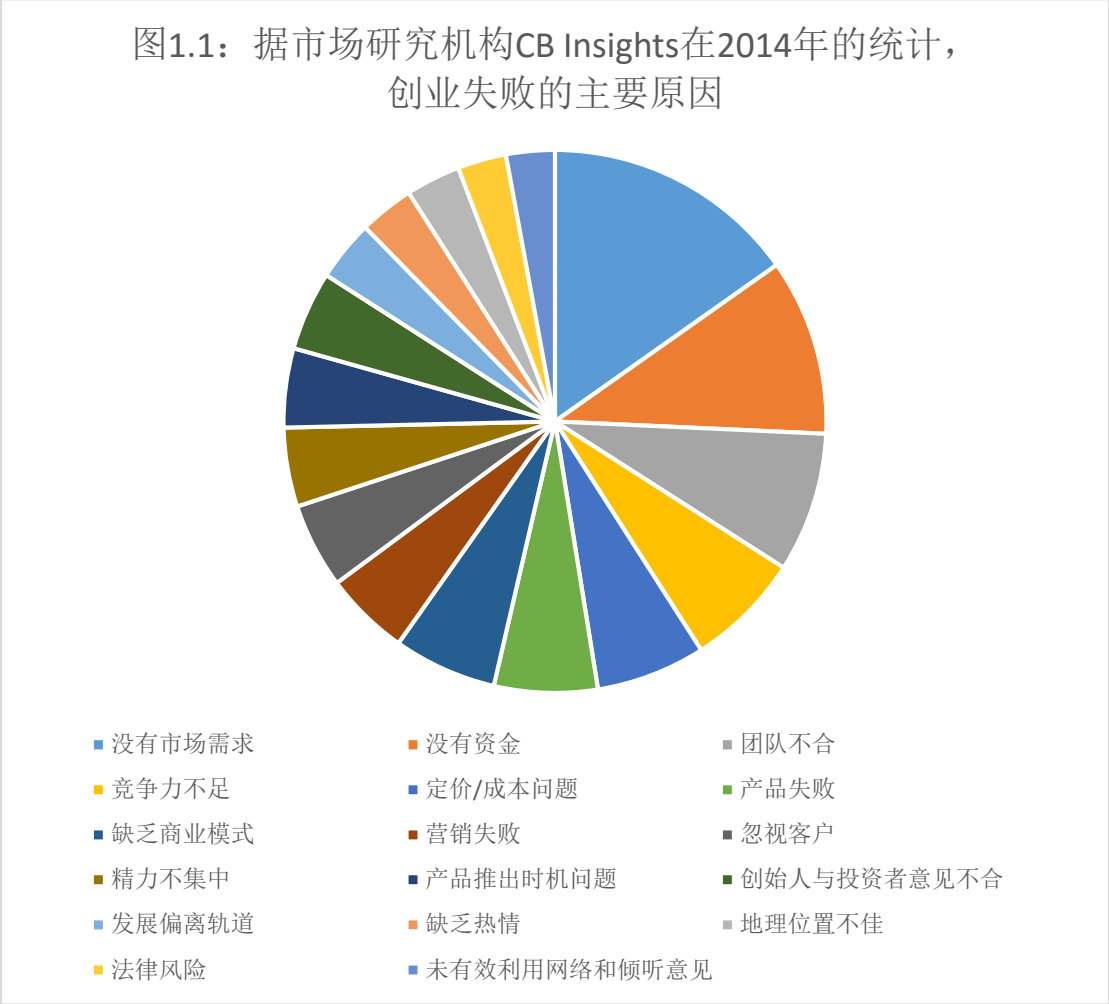
对于信息安全敏感的创新创业者和自由工作者们来说，采用数据集中化管理办法的平台不能给予他们足够的安全需求（至少在现今的安全环境之下）。假如为了保证用户信息安全而又过度封闭用户信息又将导致另外一个重要矛盾的产生：寻找到一个合适的合作伙伴需要逐渐地互相公开一定的敏感信息。这也是我们要描述的第二个问题。

2) 信息公开与封闭的矛盾致使现有平台表面上来看能够解决合作伙伴的寻找问题，不过实际运用当中会发现：很多有价值的项目或创意急需有能力和相同志趣的人与之合作，但要想寻找到这样的合作伙伴若不公开一部分项目核心内容是不可能轻易完成的任务，而公开项目核心内容于互联网中是显然绝对不可行的办法，剽窃和抄袭等问题会将无法解决。

本论文的目标正是力图为致力于创新创业事业的用户们提供一个能够保障信息安全的、合作撮合高效的网络平台。一直以来创新者都处在一个比较尴尬的合作困局当中，即拥有独特创新的想法和理念，甚至是一个成熟的设计方案，但在现代社会当中抛弃合作仅凭一己之力是几乎不可能成功完成一项合格的创新创业事业；然而如果选择合作，又面临着不方便过度透露自己的创意以防止知识剽窃行为，但若不透露关键信息又难以寻找到真正的合作伙伴。如此矛盾的合作撮合困境可能在无形当中抹杀了许多非常优秀的创意创造走向实现道路的机会。

该解决方案不同于以往的传统网站平台，而是采用去中心化设计的方式尽最大可能保证用户信息的不受制于任何一个特定的运营商；同时，通过我们设计的解决方案，创客双方根据自身的技术能力与合作需求能够更准确的将自己的部分信息公开与需要与之公开的可能合作对象，避免了无关人士对于重要信息的访问，

从而在一定程度上保证了创客们的创意与技术秘密的安全；而且鉴于用户账户注册数据集中化之后产生的安全问题，该解决方案将采用区块链即服务(BaaS, Blockchain-as-a-Service)的技术制作认证服务平台以担任整个方案中的认证任务。



其次，大部分众包网站都像中介机构一样工作，其重点是把工作伙伴介绍到一起，但对于后续工作漠不关心。用户来到网站，使用其服务寻找好的合作伙伴。但他们的最终目标不是合作伙伴。他们想找到合作伙伴，是为作出一些作品或工作。最终的目标是让合作者们一起成功地完成一个工作。本论文希望通过使用智能合约技术，让解决方案能够支持后续跟进工作。智能合约是一基于块链的概念和技术，它们像标记化程序一样运行，它们像网络上的任何其他东西一样具有公钥，但是它们具有代码，可以像存储过程那样“处理”业务。运用这样的技术可以通过规则的手段让供需双方签署生效的协议不受人为干扰地自动执行，最大化地保证了协议的公平性和严格性。

第 2 章 系统使用的概念和技术

2.1 中心化概念

在网络当各节点之间中有着明显从属关系或服务于客户关系的结构都可以考虑成一个中心化设计。其特点是所有的客户节点主要负者提出服务要求并接受和处理由服务端返回的数据，而服务端主要负责处理来自客户端的请求。网络当中存在一个中心节点或中心节点群，中心内的服务节点与中心外客户节点是对称但不对等的关系。所有的客户节点必须按照与服务节点的通信协议才能正常地工作。而且往往重要的数据都存储在服务端，从而导致大量的信息安全问题。

中心化架构并不意味着只有一个数据中心，它也可以是多数据中心的，如下图：

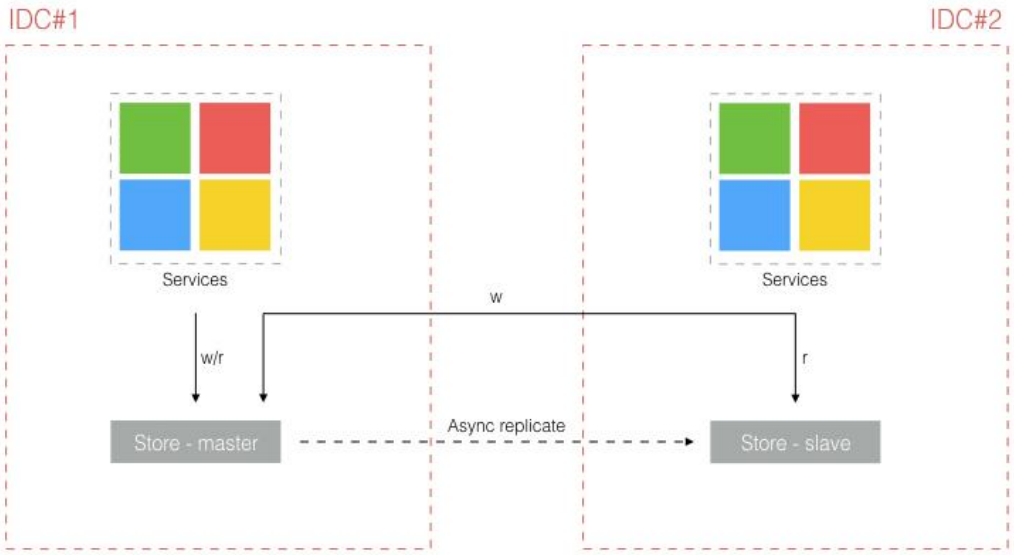


图 2.1：一个中心网络可以不仅仅只存在一个物理节点，服务器之间可以有从属关系，在保持对外抽象的一致性的同时提高服务处理能力和运算规模。

之所以说它是中心化架构，关键特征是其存在共享的数据存储。部署在两个数据中心的应用需要共享访问统一的数据存储，而这种共享访问实际是依赖数据中心之间的专线连通，这样的架构也限制了能选取的数据中心地理位置的距离。而实现去中心架构的关键点就在于规避跨数据中心的共享存储访问，使得应用在其自身数据中心实现访问闭环。

2.2 去中心化概念

在具有许多节点的系统中，每个节点具有高度的自主性。节点可以互相连接，形成连接单元。任何节点都可能成为系统的中心，但不具有强制性的中央控制功能。节点和节点之间的关系将通过网络形成非线性因果关系。这种开放、平等和扁平化的系统现象或结构，我们称之为去中心化，它必须存在于具有大量节点或一组个体的系统中。

去中心化的系统中，通常每一个节点都平等地存储数据，而且相互之间存在一定的共识机制。同时，去中心化网络结构中的身份认证往往是匿名的、去信任的，从而保证每一个用户在保护自身信息安全的同时也能够与其他用户进行可信任的数据来往。

2.3 秘密分割方案

秘密共享（也称为秘密分割）是指在一组参与者之间分配秘密的方法，每个参与者分配一部分的秘密。只有当足够数量的可能不同类型的秘密碎片结合在一起时，才能还原秘密。个别碎片自身是没有意义的。

在一种类型的秘密共享方案当中，有一个分配者和 n 名共享人。分配者给予共享人一个秘密的碎片（也称为影子）。但是只有当具体协定的条件得到满足时，共享人才能从碎片中还原秘密。如果，分配者通过给予每个共享人一个秘密碎片，使得任何一组 m（阈值）或更多的共享人可以一起还原秘密，但是没有达到 m 名共享人则不能还原这个秘密。这样的方案被称为（m，n）阈值方案（有时它也被记为（n，m）阈值方案）。

秘密共享方案是存储高度敏感和非常重要的信息的理想选择。典型的示例有：加密密钥，导弹发射代码和银行账户编号等。这类信息中都必须保持高度的机密性，因为它们被曝光之后产生的影响是巨大的。但保证信息保密的同时也保证信息不被丢失是一件非常重要的问题。传统的加密方法不适合同时满足高水平的机密性和可靠性。这是因为当存储加密密钥时，必须选择在单个位置保存单个密钥副本以获得最大的保密性，再者是在不同的位置保留密钥的多个副本以获得更高的可靠性。通过存储多个副本来提高密钥的可靠性的同时降低了机密性，而提高机密性则会降低可靠性。秘密共享方案成功地解决了这个问题，并且能够满足任

意级别的机密性和可靠性。

2.4 非对称加密技术

对称加密算法加密和解密相同的密钥，而非对称加密需要两个密钥来单独加密和解密。非对称加密能为数字签名提供良好的安全保证。例如，若要在区块链的地址中操作比特币，则必须通过数字签名的验证。在比特币中，算法采用了椭圆曲线密码学（ECC）。用户可以通过 ECC 生成自己的私钥，再通过私钥可以生成相应的公钥。数字签名需要私钥进行签名处理，此证书和公钥将发送给收件人进行验证。在比特币的 PoW 协议区块链中，接收者是参与到区块链维护的挖掘节点，每个节点也维护着整个区块链数据的数据。对于交易的验证，它需要使用接收的公钥进行检查，验证其是否由私钥持有者发送，并且公钥可以通过两次特殊的哈希生成唯一的地址。验证完成后，地址中的比特币就可以运行。每个用户在比特币钱包应用程序中都有自己的私钥，而且私钥不会在网络上传播，它可以生成独特的相应公钥，公钥可以生成唯一对应的地址。整个区块链数据是公开的，任何人都可以查看块中的数据。想要操作比特币就必须知道相应的私钥。而使用不同明文数据进行安全哈希运算得到相同哈希值的概率非常低，所以几乎不可能获得与其地址对应的私钥。

2.5 区块链技术

区块链本质上是一个简单的链式数据结构。具有点数量具有随时间增加、数据不可修改、开放且支持匿名等诸多特点。每一个区块与特定信息相互捆绑，整个区块链是分布式、P2P 和去中心化的。当前已经成功应用了区块链技术的案例有“比特币”（Bitcoin），“以太坊”（Ethereum）等虚拟货币，以及由微软的身份认证服务为代表的区块链 2.0 技术支持的产品。

区块链可以被看作是一个数字账本，并且其区块和支链的维护需要通过多个节点合作进行。在“比特币”的应用中，每个矿工计算机都是一个有效节点，每个节点都在本地存储整个区块链的数据并一直更新。对于一个新的事务（我们把所有的数据操作称为区块链的交易，对应于比特币则是一个输入和输出的数据流），许多节点都需要进行检查其是否有效的确认工作，这需要节点之间建立安

全合理的共识机制。

区块链采用匿名的方式存储和访问数据。以“比特币”为例，每个比特币都有其唯一的地址，也就是区块当中的一个标记。这种地址是经过安全哈希变换后的哈希值字符串。虽然地址是开放的，但为能保证匿名功能的同时具有极强的安全性，不可或缺的就是非对称加密技术及其签字技术来支持比特币地址相关操作。

第 3 章 系统需求分析

3.1 系统概述

3.1.1 总体目标

本论文中的提出的系统主要面向既有合作需求也需要保证自身信息安全的网络用户，作为一个完整的解决方案，同时也是一个应用平台，解决社会当中安全合作过程的关键问题。从而试图推进一个更好的合作模式，让用户可以无需顾虑信息安全问题，同时也能大大提高匹配服务的成功率。

目前市场上已有的此类系统基本存在着以下问题，也正是本论文需要解决的问题：

1）中心化服务导致的集中化管理数据的方式不可避免地存在一定的信息安全问题，易被盗取、篡改和滥用。

2）合作撮合的准确度不高。采用搜索引擎方式进行合作信息查找，有效的匹配结果出现的概率并不高。

3）合作进行过程当中的信用度不能保证，存在单方面诈骗行为的可能。

本论文设计的系统适用于任何具有合作、撮合需求的人群，无论是公司的大型项目、还是个人的创意想法，都可以在我们的平台上寻求合适的合作；而且该平台更擅长支持对合作内容对保密需求、撮合准确度需求有一定高度要求的一类用户提供更好的服务。

用户无论在公司还是家中，通过浏览器都可以访问打开系统的网页，因为系统平台具有良好的移动性和交互性，利用 Web 服务无需提前下载应用的优势，尽最大努力为有需求的客户提供便利的服务。其前期操作方式与传统的平台相近，因此有利于用户从其他传统平台迁移到该系统提供的平台上。

系统的基本目标以及要求有：

1）保证用户在寻找合作伙伴过程中使用的数据和信息的安全。通过去中心化的设计，防止数据的存储、管理、收集的权利过于集中，防止任何一方滥用侵犯个人隐私和威胁数据安全大数据技术。

2）设计对使用者、开发者、客户端用户、服务用户均友好的交互方式。既

要保证健全的开发接口、调试接口和配置空间，也要为客户端用户提供能够简单上手、低学习成本的交互方式。因此，所有核心操作都由命令行风格的程序完成，而客户端单独设计具有图形化界面的工具，辅助用户进行操作。

3）用户账户使用匿名的方式，采用基于区块链的身份认证方案，必要情况下考虑集中化的用户开户审核方式。最大化地保证用户在进行数据交换时的身份确认建立在由密码学保证的去信任环境上。

3.1.2 业务描述

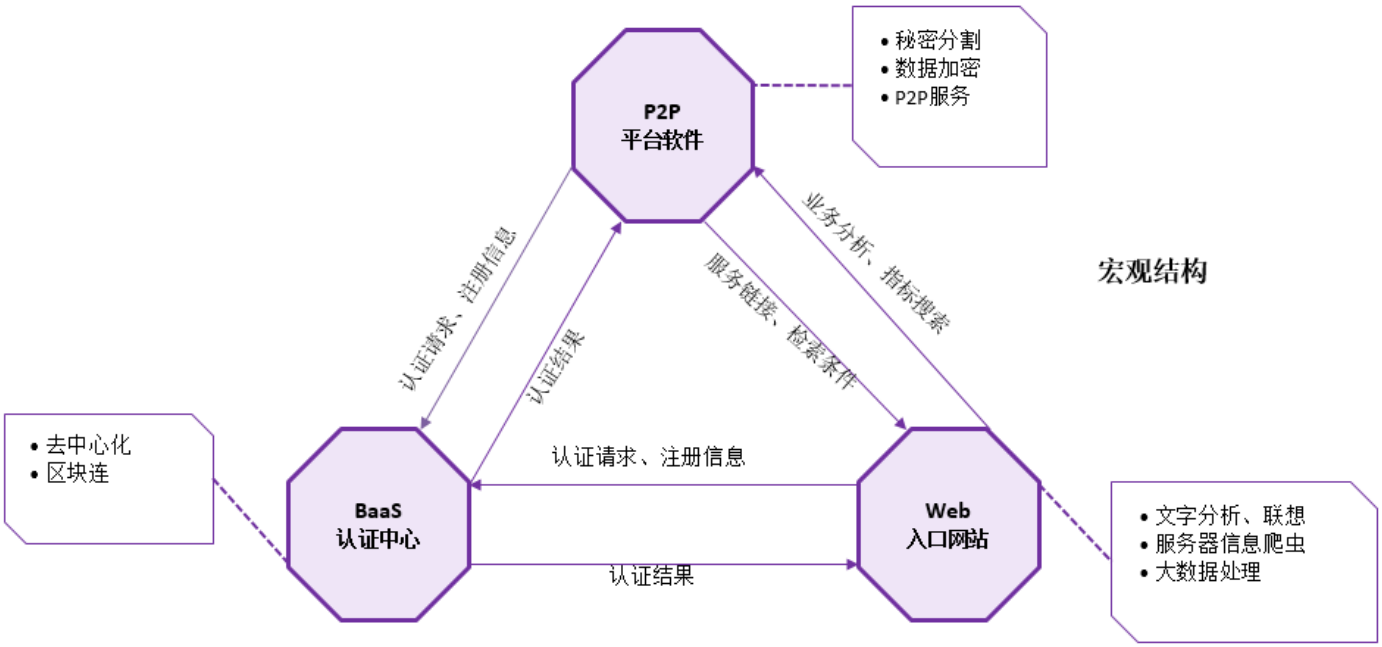


图 3.1：整个系统的工作需要三个相对独立的模块（或称角色）互相提供支持才能为用户提供合作伙伴的匹配服务。

系统的主体业务是帮助合作供需双方能够安全地寻找到合适的合作者。首先一方合作寻求者将自己的信息资料通过不同的需求和要求标准发布至平台，以供系统进行用户匹配。系统会根据合作寻求者的要求，如指标数据，来按照不同服务器自定义的算法进行双方或者多方匹配。整个匹配过程只有进行匹配运算的服务器和提交指标数据的用户知道详细的指标数据和算法，其他无关匹配和信息服务器无法通过正常方式窃听到有关信息，从而最大化地保证了用户信息的安全。

然后，当有另一个合适的合作寻求者向平台提交其合作需求和指标时，系统就会将其匹配出来。当每个匹配服务成功匹配出一对指标时，匹配服务器将把当时双方提交的信息资料的碎片发送到对方。当获得到的信息碎片数量满足还原条件时，原始的信息将会被用户在本地计算还原。此时服务双方的撮合任务就完成了。如果双方选择在特定的业务按特定的规则开始合作，则平台可以开始后续的合作跟进服务。

为在本论文当中更好地描述业务流程以及系统参与者的动作、属性和任务等各项特性，现使用若干角色代表不同的用户、程序和服务器，明确地分配业务内容和阐述业务关系。这些角色有：

1）终端用户角色。即用户，是系统提供的主要服务的最终使用者。

2）服务供应角色。为终端用户提供指标服务，主要负责用户数据存储，指标匹配以及服务映射功能。

3）搜索引擎角色。针对用户提供的简单需求描述，尽可能地提供最符合用户需求的服务供应条目以供用户选择，从而提高系统的使用效率、降低使用门槛和学习成本。

4）身份认证平台。对各个角色的身份进行管理和认证，以保证系统的安全运行。

5）保险柜角色。针对个人用户的私钥存储问题，提供一个保险柜平台帮助用户管理私钥。

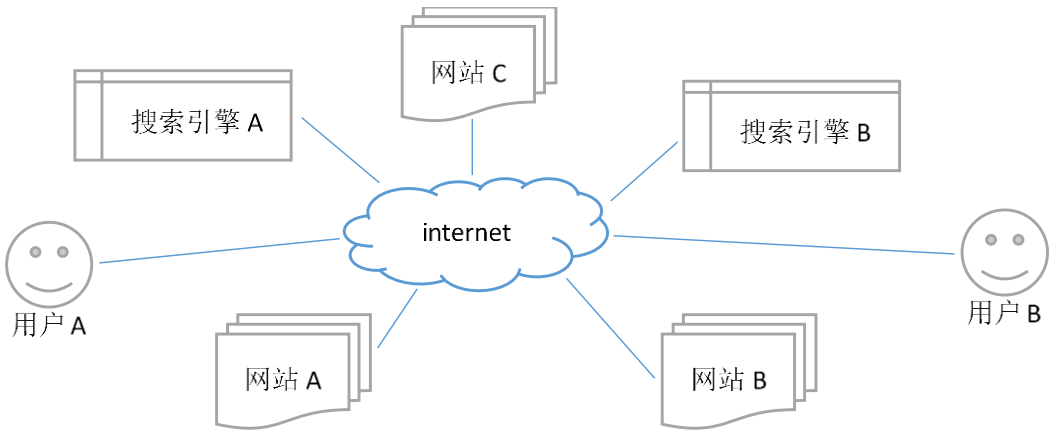


图 3.2：目前互联网中的主要角色及其结构的简单图示。

互联网本身是去中心化的结构，每一个网站都是相对独立的，每一个用户也

不例外。为了让用户更容易、更简单地访问到合适的网站，一个被称为“搜索引擎”的角色的网站出现，为用户分析其访问需求，并提供经过筛选的合适的网站链接。这样一个结构非常稳定和安全。因为去中心化的结构，保证了没有任何一个搜索引擎、网站、用户对于整个网络来说是必不可少的。例如，没有“Google”用户仍然可以使用微软的“Bing”或是百度来代替它。

类比这样的结构，本论文所设计的系统在网络当中的存在形式及其相似，如图 4。

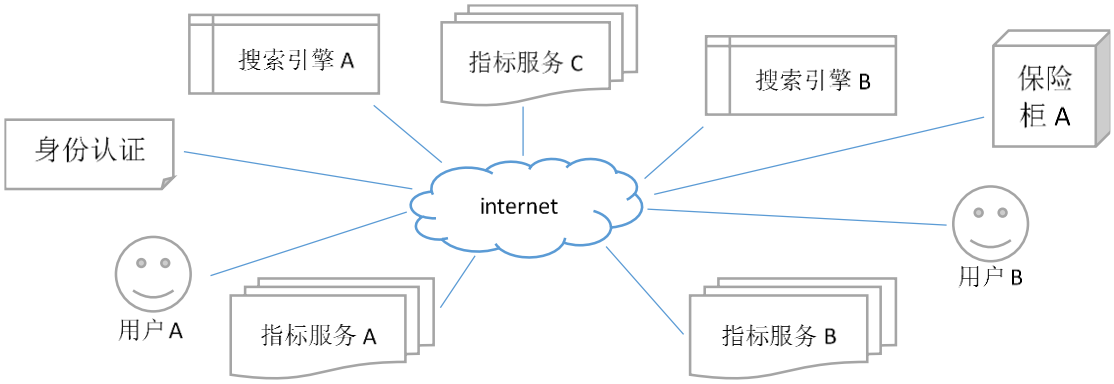


图 3.3

其特点是：

1）去中心化的服务结构。整个解决方案当中没有任何一个独立节点能够收集用户的完整信息。

2）与现有网络运行模式相兼容，不存在重新建设基础网络设施的成本。

3）面对用户拥有一致的用户接口，降低普通用户使用该解决方案的学习成本，具有快速推广的潜力。

核心业务流程：

核心角色协作流程 - 合作申请
终端用户角色
服务供应角色
身份认证平台
开始
选择服务供应者
填写服务表格
提交申请单
申请单
接收结果
保存令牌
结束
开始
监听用户申请
接收申请
验证终端用户身份
接收结果
身份有效
写入服务数据
生成令牌
令牌
尾部处理
验证申请单
验证结果
开始
监听身份验证申请
接收申请
验证身份
尾部处理

图 3.4

3.2 功能性需求

3.2.1 数据要求

计算机程序在本质上其实可看作一个数据过滤器，其拥有一个输出要求和一个可预期的输出能力。因此，为了保证次系统在后期能够更加顺利、合理地进行实际开发过程，在此作一个对必要输入输出数据的规范和要求。

经过对需求的详细分析，以及调试为此建立的多个原型程序，总结出以下若干主要的程序输入数据。由于目前需求暂时处于非常模糊且不确定的状态，因此只给出必要数据的要求和其描述。

• 指标匹配服务器信息：由文本格式编写的服务器相关信息。编码格式优先采用 UTF-8，其次是 Unicode。后期为更好地扩展这一信息文件的功能，可通过制定不同的协议，来支持以 HTML 或 XML 等方式渲染服务器信息。

• 配置文件：由文本格式编写的配置信息。编码格式优先采用 UTF-8，其次是 Unicode。程序的启动和初始化操作应尽可能地不依赖于系统环境变量以提高其移动性。因此，需要一个配置文件详细地描述程序所需要的必要和扩展信息。内容主要分为三部分：全局配置、客户端方配置、服务端方配置。

• 指标解算插件：动态链接库。实现预先设计的统一解算接口。服务端程序解算不同指标时会通过使用不同的算法来满足不同的需求，因此提供可变解算模块的能力是必须的。

• 指标数据格式：由文本格式编写的格式信息。编码格式优先采用 UTF-8，其次是 Unicode。使用 XML 描述指标数据的格式和相关信息，类似数据库的表定义。

• 解算代码：由文本格式编写的格式信息。编码格式优先采用 UTF-8，其次是 Unicode。使用 lua 语言实现解算插件的核心匹配算法。该文件会由解算插件加载和执行。

• 指标数据：经过加密的文本，原文格式编码优先采用 UTF-8，其次是 Unicode。用户根据指标数据格式按自身需求填写指标数据。数据会由用户在客户端完成填写，然后提交至对应的服务端，其他无关服务器无法通过正常手段获取该指标数据。

• 影子数据：每一条指标数据会跟随一个对应的影子数据。影子数据是由秘密分割算法对用户的秘密进行分割之后得到的碎片数据。

• 日志文件：由文本格式编写的格式信息。编码格式优先采用 UTF-8，其次是 Unicode。按重要程度分别记录程序运行过程中各个环节的关键状态和产生的重要数据，以供调试。客户端、服务端都应具有记录日志的功能，并且可由用户选择是否开启该功能。

3.2.2 系统功能需求

总体要求

在开发和测试前期的很长时间的阶段当中，应保证系统的 P2P 性质和能力。因此，服务端和客户端的功能需要集中地编译到一个可执行文件当中。保证每个系统的拷贝都同时拥有服务端和客户端的所有功能。用户只需要通过输入不同的命令即可使用相应的所有功能，无需再次下载基础功能包。

对于客户端用户，提供友好的人机交互方案——大量的图形界面工具、操作辅助工具以及相应的教程和帮助文档。

对于服务端用户需要保证其运行性能和部署能力，因此应给予其宽松的调试和配置的空间。提供配置模板和插件的示例代码。

功能要求

用户在客户端可以进行的基本操作有：

• 配置基本功能选项；

• 新建、撤销和删除事务；

• 通过添加和移除指标项来调整事务内的指标；

• 启动和停止事务；

• 填写指标数据；

• 分割秘密，为每条指标数据生成对应的影子数据；

• 还原秘密，对由多个服务器返回的影子数据进行最大努力的还原；

- 生成账户密钥。

服务端进行的基本操作有：

- 配置所有功能选项；
- 接受客户端请求；
- 保存合法的指标数据；
- 保存合法的影子数据；
- 按用户提供的令牌进行高效地查找；
- 按指标数据通过配置和插件进行匹配运算。

3.3 非功能性需求

3.3.1 可用性要求

既要保证一般用户能通过阅读简单的入门演示说明后快速上手使用该系统，也要保证专业用户能够通过专业概念和工具使用该系统的所有功能和拓展功能。这要求系统的所有操作都可以通过命令行支持，并且还有提供一个操作简洁的客户端图形界面。

关于可用性问题需要注意的要点：

(1) 客户端图形界面的目标平台是 Windows 7 及以上。而命令行风格的核心程序的目标平台应是 Windows 7（及以上）和 Linux kernel 2.6（及以上）。跨平台代码应做到不需要修改代码内容而是通过修改编译参数即可成功编译至目标平台。

(2) 插件的开发规则和交互原则应遵守系统的规定和风格。

(3) 程序执行过程中出现错误时，为保证用户的业务数据安全，应该尽可能提供详细的错误报告和解决建议，辅助用户进行安全的错误退出。

(4) 系统在后期迭代更新的过程当中应始终保持向前的兼容性。

3.3.2 稳定性要求

客户端的期望持续运行时间较短，相对于服务端并不是一个数量级，因此其重点不是持续运行的稳定。相反，客户端对命令的执行速度、响应还有图形界面的启动时间等对用户的使用体验影响非常大。所以，客户端的时间稳定性要求集中在提高响应速度和启动速度。

服务端的程序需要长期执行，每周七天、每天二十四小时，重启时间应控制在五分钟内。而且必须保证每个服务线程出现致命错误都不会导致整个系统进程崩溃。

3.3.3 性能要求

(1) 服务端应在任何时候都能够支持最少 512 个用户并发的服务请求；

(2) 服务端对匹配解算的时间复杂度必须优于或等价于$O(n)$。平均每个指标项的匹配速度应该控制在 1ms 之内；

(3) 服务端对至少 80%的业务请求应在 1s 内完成计算相应，具体的数据返回速率由网络带宽决定，但其数学期望应该在 10s 内完成。

(4) 客户端图形界面的平均启动时间应该控制在 2s 内；

(5) 秘密分割速率应该控制在$2\ \mathrm{s}/(\mathrm{MBytes}\cdot\mathrm{Hz})$内。

3.3.4 安全要求

(1) 只有通信双方才能交换正确的信息，必须避免第三方窃听。实现上可以通过标准的协议来保证系统的可维护性，同时也可以提升开发效率。建议使用 SSL。

(2) 保证被分割的秘密碎片，即影子数据只能通过其算法按合法条件还原，避免攻击行为导致影子被意外恢复。

(3) 通过版本控制、用户授权、信用管理和特征值校验等方式保证服务端和客户端的可信度。

(4) 单个节点的被破坏或停止工作时不能对整个网络平台造成破坏和影响。

3.3.5 扩展性要求

(1) 所有版本的服务端都必须保持完整的向前兼容性，保证任何旧版本的客户端请求都能够得到正确响应。

(2) 命令行风格的内核程序必须与图形化界面分离。保证人机交互范式设计的自由度，满足不同用户的不同需求。同时也有利于二次开发。

(3) 秘密分割产生的影子数据应标有算法编号，以保证能够通过正确的算法进行还原，提高秘密分割模块的扩展、更新和迭代的能力。

第 4 章 系统概要设计

4.1 目标和约束

系统的设计目标要严格按照需求分析当中的总结的若干需求进行对应和满足。同时也要保证系统在开发过程当中不会偏离这些目标、脱离所指定的约束，即开发过程当中出现镀金和蔓延现象。整个系统的开发过程应该在以下若干目标和约束的限制下，通过迭代和更新不断地收敛至一个成熟的系统。

(1) 安全目标。系统的设计和实现必须始终保证用户的个人信息和相关资料（包括存档资料和网络会话资料）的安全。这里的安全指用户信息不会未经用户同意授权就被第三者截取或收集。用户在整个网络当中拥有选择节点的权利和被遗忘的权利。尽可能保证用户的匿名身份。

(2) 设计目标。系统在不断的迭代过程中，应保证每个模块的低扇入和尽可能高的扇出。提供抽象设计的内聚、降低其各个部分的不必要耦合。必须杜绝设计和代码中出现内容耦合现象。而且，公共耦合只允许在无法找到其他适用解决方案或是在简单测试中少量使用。尽可能对相对独立的功能模块设计高功能内聚的结构。

(3) 可靠性目标。系统对应非法输入应该有强制的抵抗性，不能因为恶意输入而致使系统宕机。当遭受到非法输入或恶意攻击时，最坏的情况应是系统停止工作，而不是产生和返回无法预料的其他无效结果。网络模块应考虑如何防止和避免 DDoS 攻击造成恶劣影响。

(4) 扩展性目标。系统应支持运行时更新插件的能力，即支持插件热插拔。这样做的目的是最小化系统更新升级对正常业务造成的影响。而且，系统必须保留百分之百的向前兼容能力。

(5) 简洁性目标。系统提供的接口必须是最小化参数要求的。所有接口和抽象必须保持一定的简洁度。同时，所有的功能模块应该简单地完成其自身部分的所有功能。系统代码也应按照编码规范保持其简洁以获得更好的可读性。

为了更好更快地实现系统功能，我选择借鉴 RUP（Rational Unified Process）的优秀之处，如其中对需求的分析方法、系统的设计原则与工具以及部署原则与方法。加之快速原型模型考虑到的开发过程中的特点，最终应用到螺旋模型当中。

一方面详细合理地分析需求与设计系统，另一方面可以很好的控制系统开发过程中可能遇到的风险。约束和规范有：

(1) 设计和分析需要采用：面向对象的分析与设计；建模语言使用 UML。

(2) 开发策略：使用合适的开发平台和工具，内核模块由 C++开发，使用 C++11 标准，编译器使用 MSVC 或 GCC。整个开发过程当中必须伴有测试。

(3) 技术规范：根据不同语言、框架和环境，使用开源社区或官方推荐的技术规范要求。同时，根据自身以往的项目开发经验，可以对模糊的标准作出自己擅长的技术要求和规范。

(4) 平台规范：内核模块采用 C++及其相关技术。图形界面采用支持 C#的 WPF 技术，对于 Windows 系统，整个图形界面方案应建立在.Net Framework 的基础之上。对于 Linux 系统的支持，目前考虑使用 GCC 作为内核的编译器，而图形界面使用 Qt 去支持。

4.2 总体架构

根据前面对需求所作的分析，现将系统设计为由三个主要部分组成的集成平台，而且每一个部分都是相对独立的。为方便实现，将开发代号命名为 Enco、Cert 和 Agent。它们的结构如图所示。

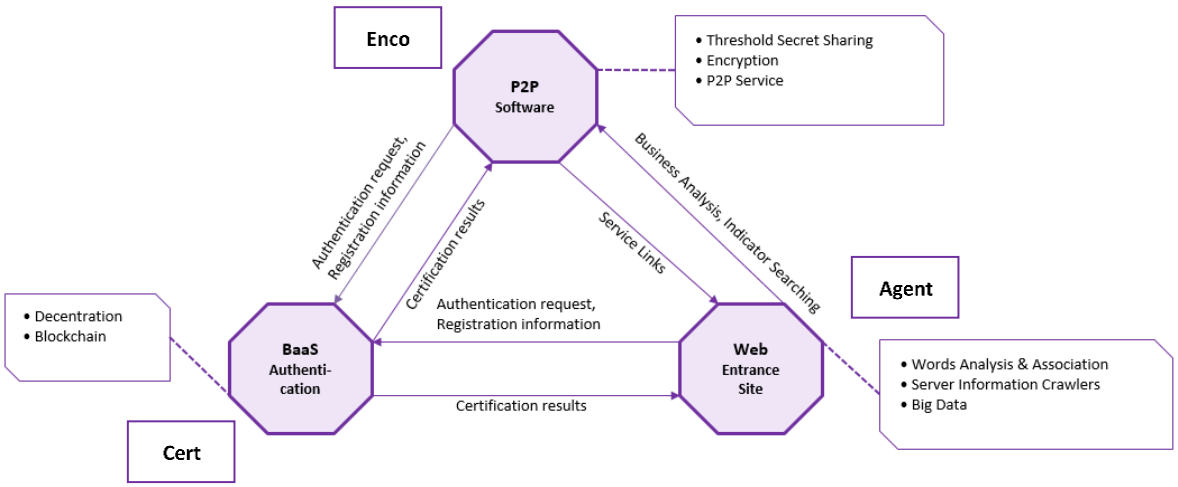


图 4.1

Enco 是作为客户端在用户个人计算机上运行的软件，也可以配置为在服务器上运行的服务器程序。它可以被认为是一个点对点（P2P）软件，但并不支持 DHT（分布式散列表）方法或任何其他类似的技术，它不负责与其他服务器共享整个数据，这有助于我们保护用户的数据和信息。

Cert 是一个认证服务中心，但它的内核完全基于块链技术，这有助于我们分散身份验证功能，也没有人可以控制整个信息。 其次，用户在服务上是匿名的，直到特殊方法干预该过程，用户的真实身份被安全地保护。

Agent 是很像是一个传统网站，扮演着搜索引擎的角色，帮助用户通过分析他们的需求来检索正确的服务。它还具有搜索 Enco 端点的爬虫工具，但不是传统的网站。

在实际实现之后，其服务结构如下所示：

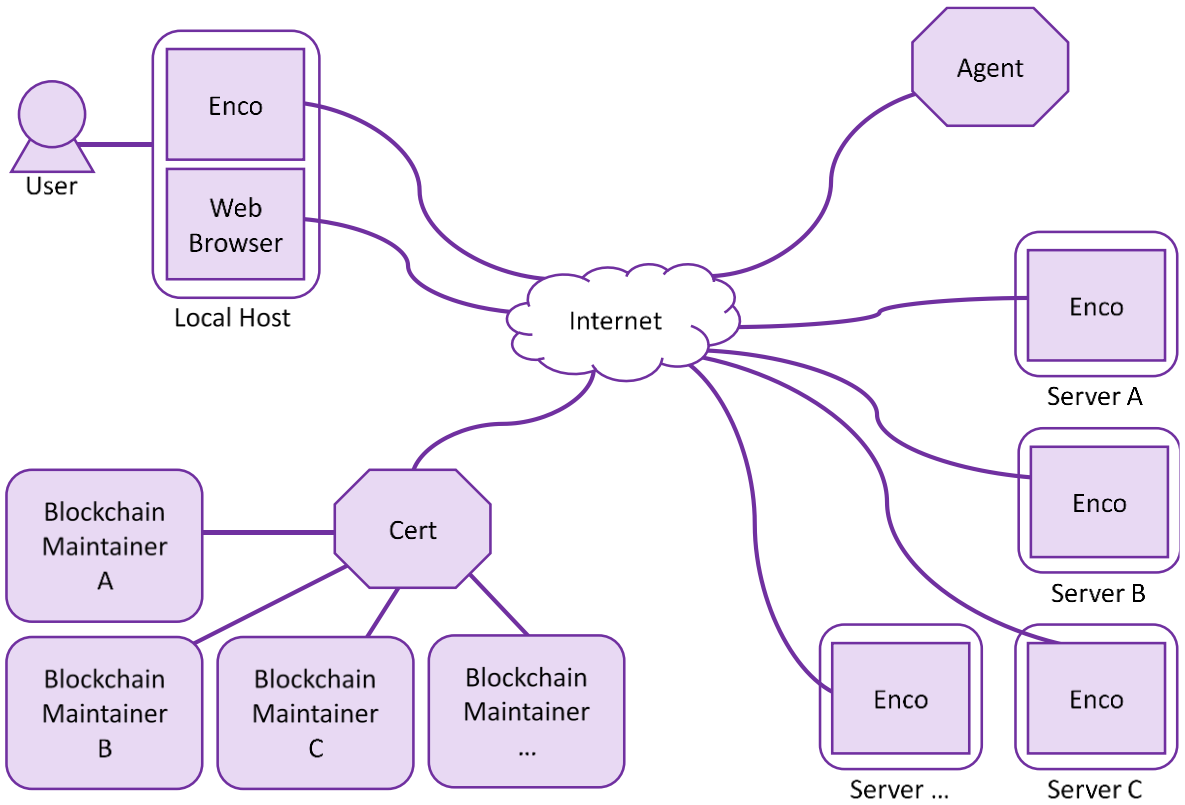


图 4.2

如果用户尝试找到潜在的合作伙伴，他/她可以按照以下步骤操作。

1. 使用 Web 浏览器（如 Microsoft Edge、Google Chrome）访问 Agent 网站。

通过告知 Agent 自己的需求和要求以寻找合适的指标服务。

2. 选择指标服务，并将相应的服务器信息文件下载到本地主机。

3. 使用 Enco（或 Enco-UI，即图形化界面）在本地主机上创建新的事务。事务的执行取决于具体的指标项目，用户必须首先将步骤 2 中下载的指标服务器信息文件放入到事务当中。

4. 根据每个指标服务填写指标数据。

5. 选择附加到事务的秘密文件，之后启动业务。

6. 使用 Enco 主动查询业务结果。

7. 如果任何其他用户提交了能够成功匹配的指标，其秘密部分将在查询过程中通过指示器服务器发送到用户本地计算机的 Enco。Enco 尝试恢复在本地主机收集的所有秘密。

8. 如果秘密恢复，Enco 将帮助主要用户将其打开。

4.3 功能架构

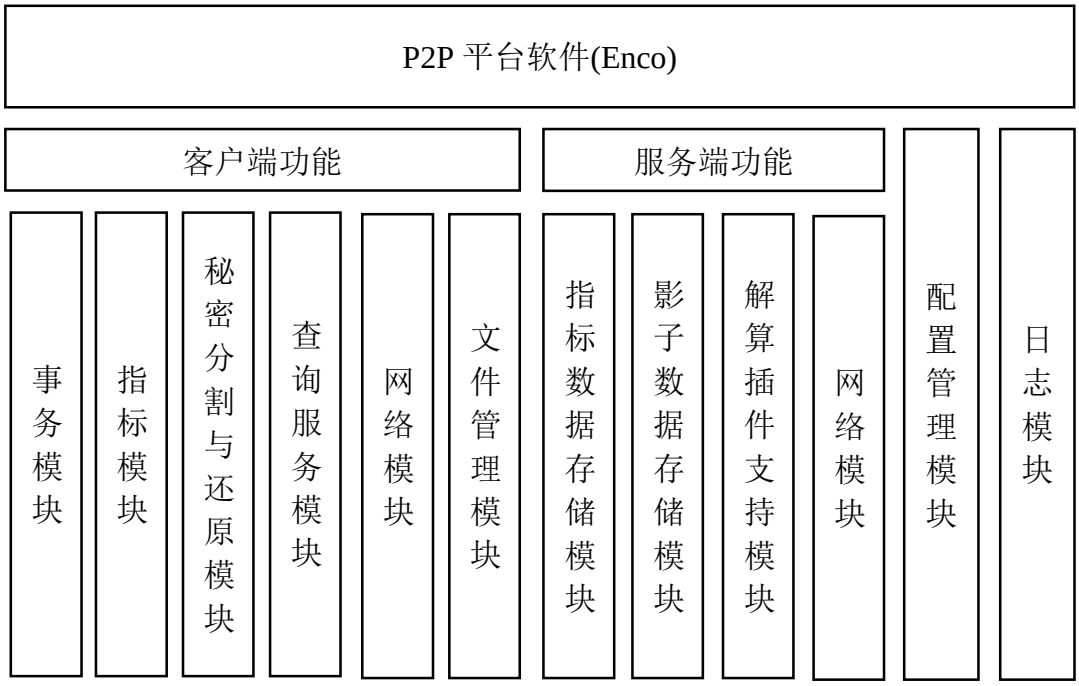


图 4.3

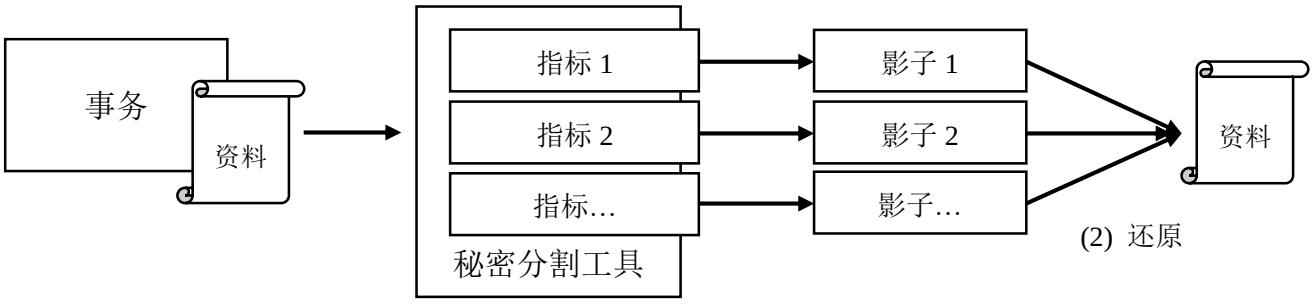
系统的三个主要组成部分之一——P2P 平台软件。它的基础功能可以满足目前在需求分析中所描述的大部分合作者匹配需求的基本服务要求。其中的功能主要分为三类：全局类功能、客户端功能、服务端功能。全局类功能为整个 P2P 平台软件提供了基础而且频繁重用的功能，比如基础的密码学操作（安全哈希校验、ECC 加密、AES 加密等），软件命令处理以及程序配置管理等功能。客户端功能者面向客户端用户设计，从而提高整个软件的可用性，降低使用的学习门槛。其更加专注于细化一般客户的业务需求并提供相应的功能模块去满足这些需求，比如将用户的每个细致的合作匹配需求抽象为用户的事务，从而可以进行量化、单元化的管理等等。服务端的功能主要专注于数据的存储能力以及访问效率，还有就是网络的稳定性和多用户高度并发访问时服务器负载的问题。

配置管理模块。全局功能模块，使用自定义风格的配置语言对软件的启动参数进行配置。早期版本可以使用简单的实现，比如按行扫描，以井号（“#”）开头的字符串作为备注被忽略等功能。后期待功能实现具有一定良好轮廓时，在迭代过程中可以使用优秀的脚本语言进行配置，如 python、javascript 或者 lua 等等。

日志模块。全局功能模块，提供带有时间戳功能和级别控制功能的日志输出能力以供其他模块调用。该模块可以贯穿整个软件，作为一个最基础的输出模块使用，从而记录软件在运行工程中的状态和产生的数据。该模块是软件调试的重要工具，也是测试报告需要依赖的重要数据源的提供模块。

客户端功能中：

事务模块。每个事务只负责处理一项合作匹配需求，需要由若干指标对合作需求进行细分拆解。每个事务只能携带一份秘密资料，多个资料可以整理至一个压缩包中作为一份资料去分割。如果事务中包含的指标数量越多，指标所描述的内容准确度越高，所使用的指标服务器热度越大，则匹配成功率越高而且安全系数也越大。



(1) 分割　　(2) 还原

图 4.4

指标模块。每个指标由维护其指标类型的服务器提供服务支持。服务器会提供该指标的含义、目的、适用场合以及配对算法的详细定义和说明。客户端可以随时从服务端拉取指标类信息。值得注意的是：每个事务需要启动之后才能够生效，而事务的启动依赖于每个指标在其指标类服务器上的成功写入。

服务端功能中：

影子数据存储模块。影子数据是一一对应于每一个指标数据的秘密碎片数据。服务端应具有稳定地进行存储大量影子数据的能力，同时也需要具有快速随机访问的能力。在早期的原型阶段，该模块可以考虑利用操作系统本身的文件系统辅助完成该功能。后期迭代阶段可以使用 Memcached 以提高随机访问效率。

指标数据存储模块。每一个指标服务器维护着一个指标类。用户通过访问指标类服务器获取相关信息，然后决定是否使用该指标类来匹配合作伙伴。如果用户选择了一个指标类，那么它将在其事务启动的时候将填写好的指标数据发送到一个指标服务器。指标服务器获取到指标数据后，先将插入到指标数据队列当中，然后服务器会给客户端返回一个令牌。之后，用户可以根据该令牌要求服务按匹配算法进行配对解算，如果成功匹配出若干指标数据，那么服务器会将其对应的影子数据返回给客户端。

4.4 部署架构

由于使用系统的用户类型很多，知识水平不等，对计算机的了解程度也不同，涉及的行业和领域之多，导致系统不能局限于固定的配置和部署方案。在这个部分，先不考虑第三方的系统和环境所造成的因素，例如依赖的物理平台、操作系统以及网络防火墙等等。整个 P2P 平台软件的部署可结构可以简化为一个很简单的关系。

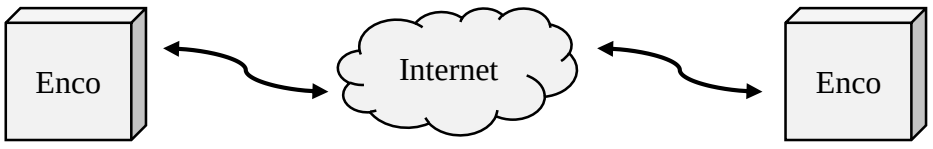


图 4.5

在后期的迭代过程中，考虑数据量、访问量的快速增长、客户端对服务稳定性要求的提高以及对业务处理速度的容忍度的降低，系统需要考虑使用一定的分

布式计算技术来缓解单个网络节点的压力。

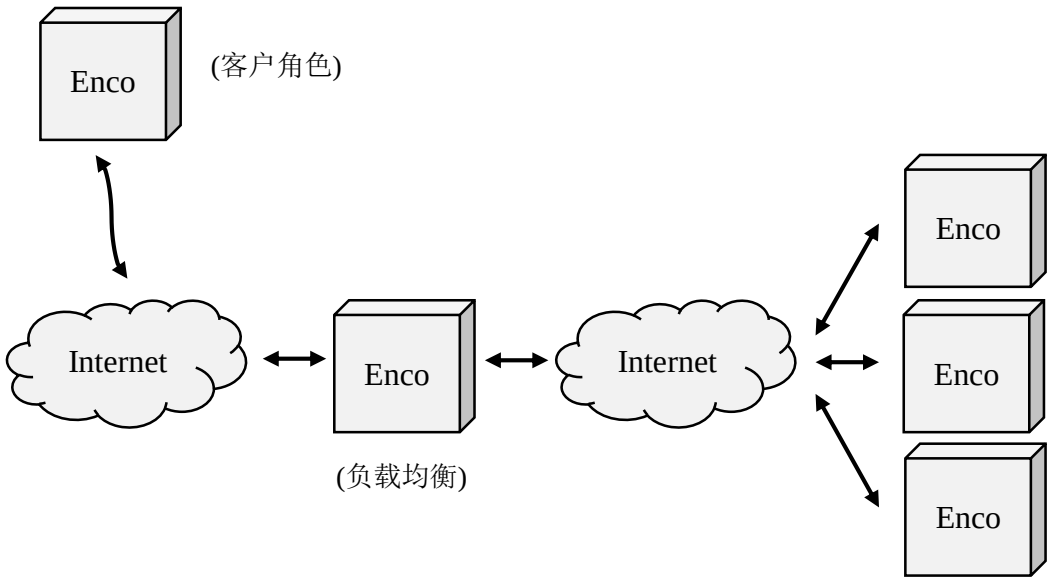


图 4.6

通过赋予每个节点潜在的负责均衡功能。当服务能力对于网络需求的处理达到瓶颈时，用户可以通过修改配置文件、重启服务系统来开启分布式式计算功能。从而提高系统服务的稳定性和执行效率。

第 5 章 系统详细设计

5.1 数据详细设计

本系统区别于传统的信息管理系统，根据对于需求详细分析和对于实现的考量，放弃使用成熟的数据库对数据进行管理。关于在放弃使用数据库这一点，我在早期已经放弃的原型当中提供了一个使用 MySQL 实现的指标数据管理功能，经过实践发现，本系统的数据要求想比使用传统的数据库来说，自己创建工具来管理数据能够更好地发挥其特性、提高开发数据、拜托传统数据管理方式的拘束。因此，对于指标数据的存储，该系统因自己掌握完整控制数据的能力。

指标数据管理基础的设计：

基本概念。每一个指标服务器维护着一个指标族，也可以是多个指标服务器同时维护同一个指标簇。每一个指标簇有且仅有一个解算方法，用于对其定义的指标项进行求解，得到一个参考值。每一个指标簇可拥有多个字段（可理解为参数形式），每个字段都属于且仅属于一个元（或维度）。元是表明该指标所能联系的维度的；维度越低，参考值可信度越低命中率越高，反之则可信度越高命中率越低。每一个指标项对应于与其唯一的一份影子数据。

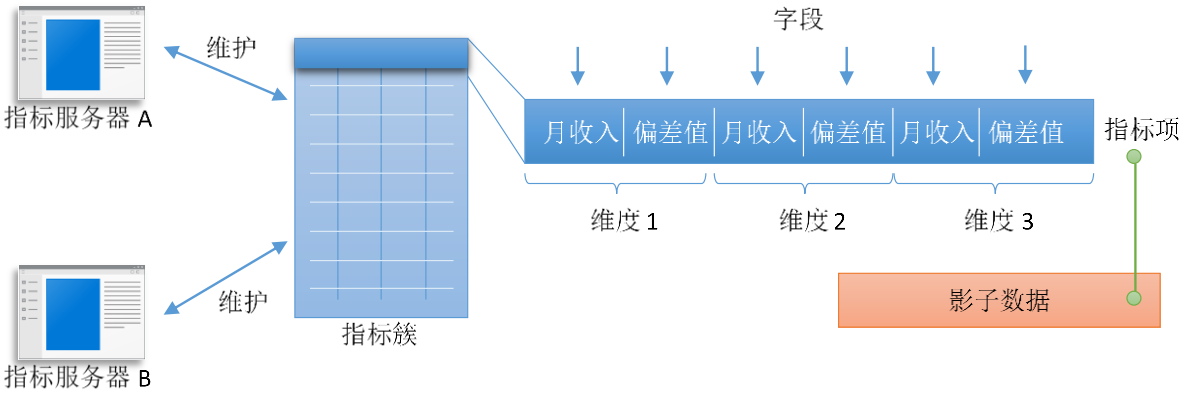


图 5.1

详细定义。

(1) 影子数据

影子数据块中储存着由秘密分割算法生成的部分数据，用于还原指标项拥有者留下的重要信息。必须采用具有门限功能的秘密分割算法对信息进行分解和还

原。

(2) 字段

字段定义了其数据位置的数据类型和含义，可以看作是函数的参数声明，也可以类比数据库对字段的定义。主要作用是描述指标项的参数数据块中不同区域的数据意义。目前支持的数据类型有：

类型	字节长度	描述
Int32	4	32 位整数类型（有符号）
Int64	8	64 位整数类型（有符号）
Byte	1	字节类型（无符号）
Boolean	1	布尔值
Single	4	单精度浮点数（有符号）
Double	8	双精度浮点数（有符号）
Char	1	窄字符类型（有符号）
WChar	2	宽字符类型（有符号）

表 5.1

(3) 元与维度

元和维度是一个概念的不同称呼，描述了指标解算过程中一个指标项与其他指标项的关系。是指标服务在实际应用当中发挥理想效果的重要因素。有且仅有一个维度的指标簇称作一维指标簇或一元簇；有且仅有两个维度的指标簇称作二位指标簇或二元簇；以此类推。

维度，准确的说是解算过程中必须参与的指标项的数量。这个数量必须为大于或等于 2 的整数才能进行解算。

图 5.2　一元簇解算的对应关系

一元簇是最简单的指标簇，也是一个特殊的指标簇。按规律推算一元簇的参与指标项数目应为 1，但只有一个指标项的解算是没有意义的，因此可以考虑一个特殊情况：传统按行匹配技术的完全对应方法，虽有其局限性，但在很多细节方面却是非常实用和高效的。所以，将完全对应的解算方法赋予一元簇，以保持规律的连贯性。

二元簇是推荐使用的指标簇。对应关系非常清晰，使用场景广泛。适用于对等合作对象的搜索参数。二元簇的解算值由最小的维度解算值决定，意味着如果任何一个维度难以成立则整体结果也难以成立。映射至实际情况可以理解为：如果用户 A 满足用户 B 的指标要求，而用户 B 不满足用户 A 的指标要求，则不推荐相互揭示；同样，如果用户 B 满足用户 A 的指标要求，而用户 A 不满足用户 B 的指标要求，也不推荐相互揭示；当且仅当用户 A 满足用户 B 的指标要求，用户 B 也满足用户 A 的指标要求时，A、B 之间可能相互揭示。

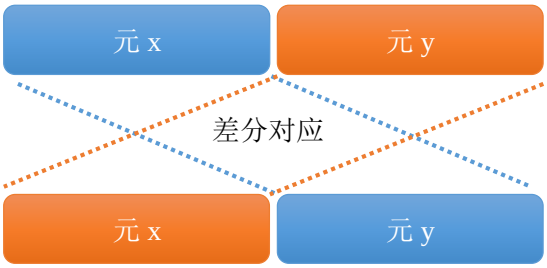


图 5.3 二元簇解算的对应关系

三元簇适用于与三方协作等场景，相比于二元簇多要求一枚指标项进行解算，意味着多一位参与者加入到合作寻求的过程当中，适当改进后可用于更多的团体合作的搜索工作。

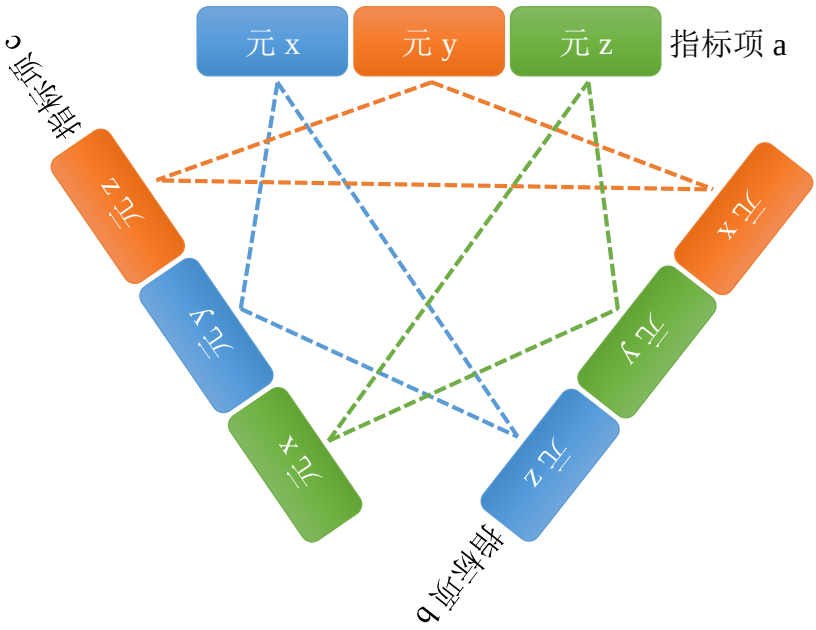


图 5.4 三元簇解算的对应关系

多元簇的解算对应关系以此类推。

假设 a 为一个二维指标项，定义 x(a) 为取 a 中 x 元的运算，y(a) 为取 a 中 y 元的运算，S(x,y)为解算方法，则对于指标簇 A 中的任意一个指标 a_i 解算指标项 t 的方法为：

$$R_2 = \min\{S(x(a),y(t)),S(x(t),y(a))\}$$

同样，对于三元簇：

$$R_3 = \min\begin{Bmatrix} S(x(a),y(b),z(t)),S(x(b),y(t),z(a)),S(x(t),y(a),z(b)), \\ S(x(a),y(t),z(b)),S(x(b),y(a),z(t)),S(x(t),y(b),z(a)) \end{Bmatrix}$$

(4) 指标项

指标项是指标簇实际储存的数据。指标项集合与解算方法是一个指标簇的核心部分，任何一个指标簇都必须包含这两项内容。其他的部分，可以根据具体情况进行改进。例如为每个指标项添加时间戳，实现差异化服务，提供一种商业模式。

(5) 指标簇

由指标项集合与解算方法组成的数据表单。记录着一个指标类型的实际指标数据。是对用户开放的概念。

(6) 指标服务器

指标服务器是维护指标簇的进程。一个指标服务器只能维护一个指标簇，而一个指标簇可被多个指标服务器共同维护。

5.2 软件结构详细设计

在第一个实现当中（版本：0.1）总体结构主要分为五个部分：全局元素、服务端部分、客户端部分、项目通用部分以及秘密共享部分。

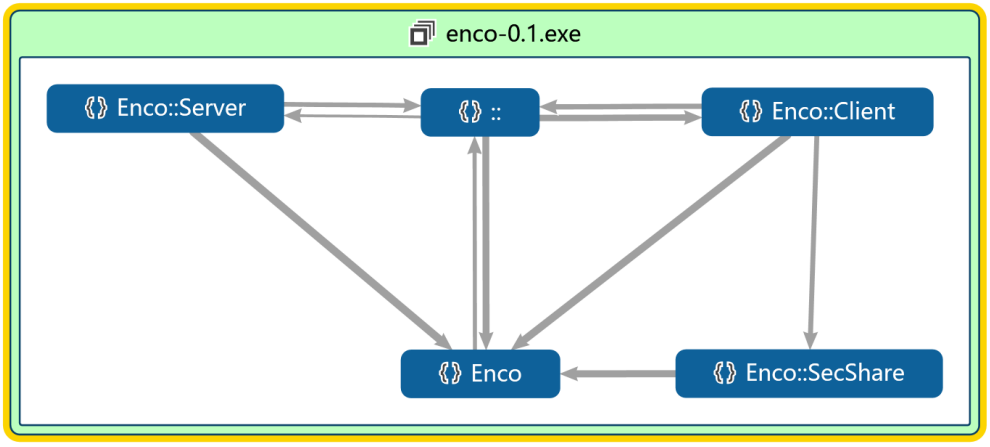


图 5.5

其中：

全局元素（“::”）主要包括系统调用的操作、工具函数的实现以及未设计的临时实现。这一部分里实现的函数和数据定义应该在后期迭代中通过使用合适的设计模式和数据结构，分配到不同的模块当中或者算法当中。

具体使用到的和实现的全局元素如下图：

LoadSolverLib
FreeSolverLib
SOLVERLIB
tag_solver_lib
_report_rangecheckfailure
LoadConfig
main
KeyValSet
CheckArgs
WrongArgs
KEYVALLIST_ITEM
PrintVersionInfo
_FPTR_UNINITSOLVER
SOLVERINITINPUT
_FPTR_ISOLVE
_FPTR_ISOLVERDIMENSION
_FPTR_UINITSOLVER
tag_solver_init_input
LISTEN_THREAD_PARAM
SERVICE_THREAD_PARAM
_tag_exeunit_list_item
_tag_kevalset_list_item
EU_IgniteBusiness
ClearDir
WorkThread_Listen
_tag_listen_thread_param
_tag_service_thread_param
PFN_EXECUTE_UNIT
PFN_KEYVAL_SETTER
ReadAllChar
RemoveDir
WorkThread_Service
defcolor
greencolor
bluecolor
redcolor
lredcolor
EXEUNIT_LIST_ITEM
SetConsoleColor
`dynamic initializer for 'keyval_setter_list''
`dynamic initializer for 'global_logs_path''
`dynamic atexit destructor for 'local_main_socks''
`dynamic initializer for 'exeunit_list''
`dynamic initializer for 'listen_threads''
`dynamic initializer for 'global_main_jps''
`dynamic initializer for 'local_main_socks''
`dynamic initializer for 'global_main_ports''
`dynamic initializer for 'Enco::Logger::m_mtx''
`dynamic atexit destructor for 'global_main_ports''
`dynamic atexit destructor for 'listen_threads''
`dynamic initializer for 'global_shadow_data_path''
`dynamic initializer for 'global_cli_idcts_depot''
`dynamic initializer for 'global_cli_busis_depot''
`dynamic atexit destructor for 'global_main_jps''
`dynamic initializer for 'global_cluster_info_path''
`dynamic atexit destructor for 'global_logs_path''
`dynamic initializer for 'global_indicator_data_path''
`dynamic initializer for 'global_indicator_desc_path''
`dynamic initializer for 'global_cli_busis_secret_path''
`dynamic atexit destructor for 'Enco::Logger::m_mtx''
`dynamic atexit destructor for 'global_cluster_solver''
`dynamic atexit destructor for 'global_cli_idcts_depot''
Service
StrtoByte
BytetoStr
DatetoStr
StrtoDate
GuidtoStr
StrtoGuid
TimetoStr
StrtoTime
Int32toStr
Int64toStr
StrtoInt32
StrtoInt64
SingletoStr
StrtoSingle
stackDump
ReadAllText
DoubletoStr
StrtoDouble
ANSItoUTF8
UTF8toANSI
StrtoBoolean
BooleantoStr
PickCmdHead
DatetimetoStr
StrtoDatetime
UnexpectedExit
EU_ListBusiness
EU_TellIndicator
KVS_svr_main_ip
KVS_cli_sec_path
EU_TestIndicator
EU_NewBusiness
EU_StopBusiness
EU_StartBusiness
EU_ViewBusiness
EU_ViewIndicator
KVS_cli_logs_path
EU_WriteIndicator
EU_BusinessResult
EU_StatusBusiness
KVS_svr_logs_path
KVS_cluster_solver
KVS_cli_idct_depot
KVS_svr_main_port
KVS_cli_trans_depot
EU_RemoveBusiness
EU_RemoveIndicator
ConnectToIdctServer
KVS_cluster_info_path
KVS_svr_main_backlog
KVS_cluster_dimension
KVS_shadow_data_path
KVS_indicator_data_path
KVS_indicator_desc_path
EU_AddIndicatorToBusiness
KVS_cluster_solver_script_file
EU_DelIndicatorFromBusiness
`dynamic initializer for 'version''
`dynamic atexit destructor for 'global_cli_busis_depot''
`dynamic initializer for 'global_cluster_solver_script_file''
`dynamic atexit destructor for 'global_cluster_info_path''
`dynamic atexit destructor for 'global_shadow_data_path''
`dynamic initializer for 'global_indicator_data_path''
`dynamic atexit destructor for 'global_indicator_desc_path''
`dynamic atexit destructor for 'global_cli_busis_secret_path''
`dynamic atexit destructor for 'global_cluster_solver_script_file''
Listen

图 5.6

服务端部分主要包括网络功能的支持、配置的应用、对客户端服务请求的相应。对全局元素中插件调用的部分依赖较大，后期可以考虑重新进行抽象和设计，提出更合理的方案。

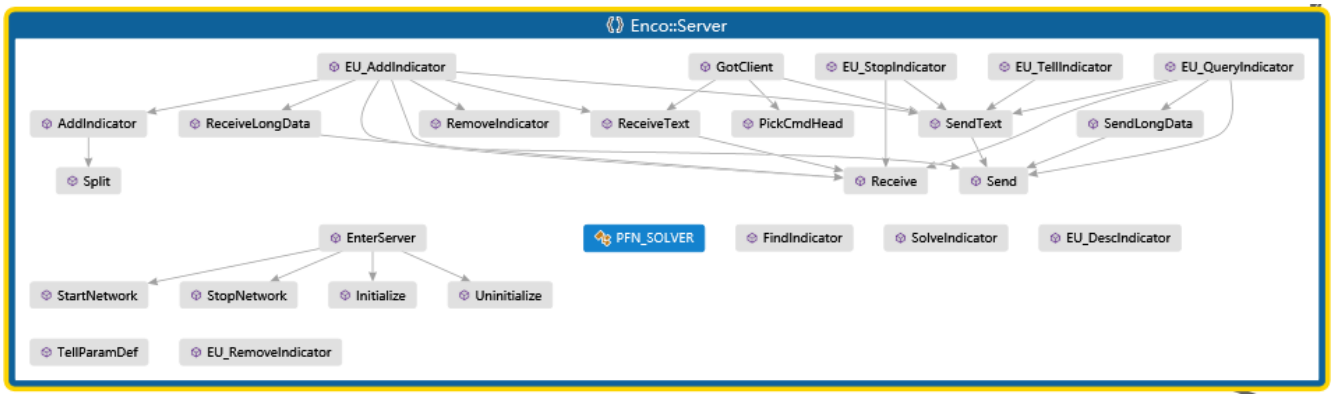


图 5.7

客户端部分主要包括事务管理、指标管理、查询功能、网络支持、配置应用等功能。其对文件读写与管理的要求较高。为了更好地管里用户的业务，提供了一些数据信息文件的抽象以便在内存中建立与硬盘相对应的镜像。

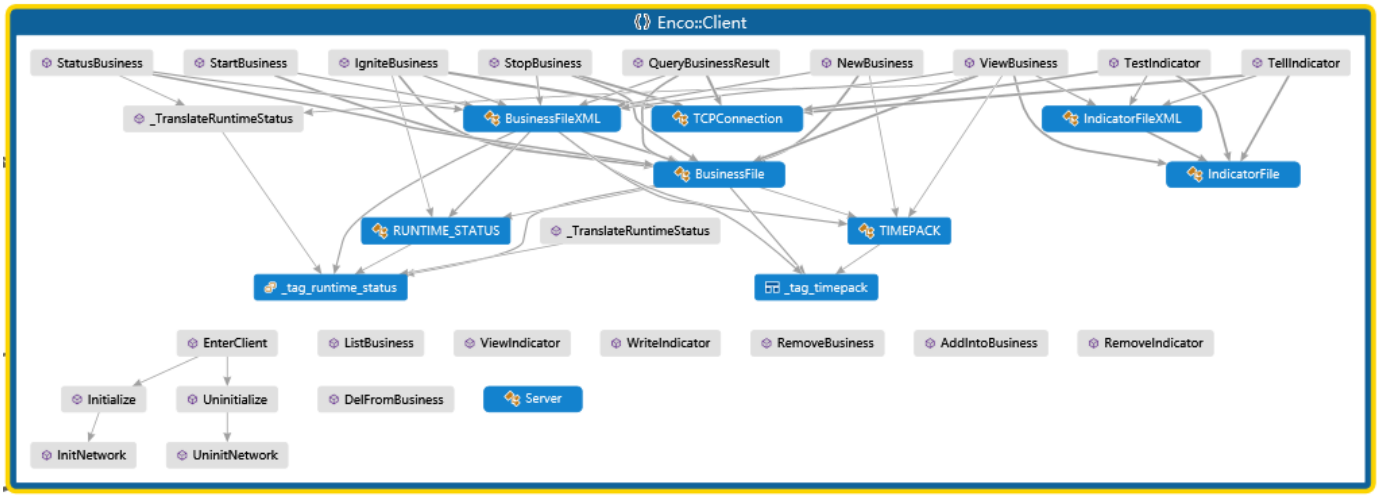
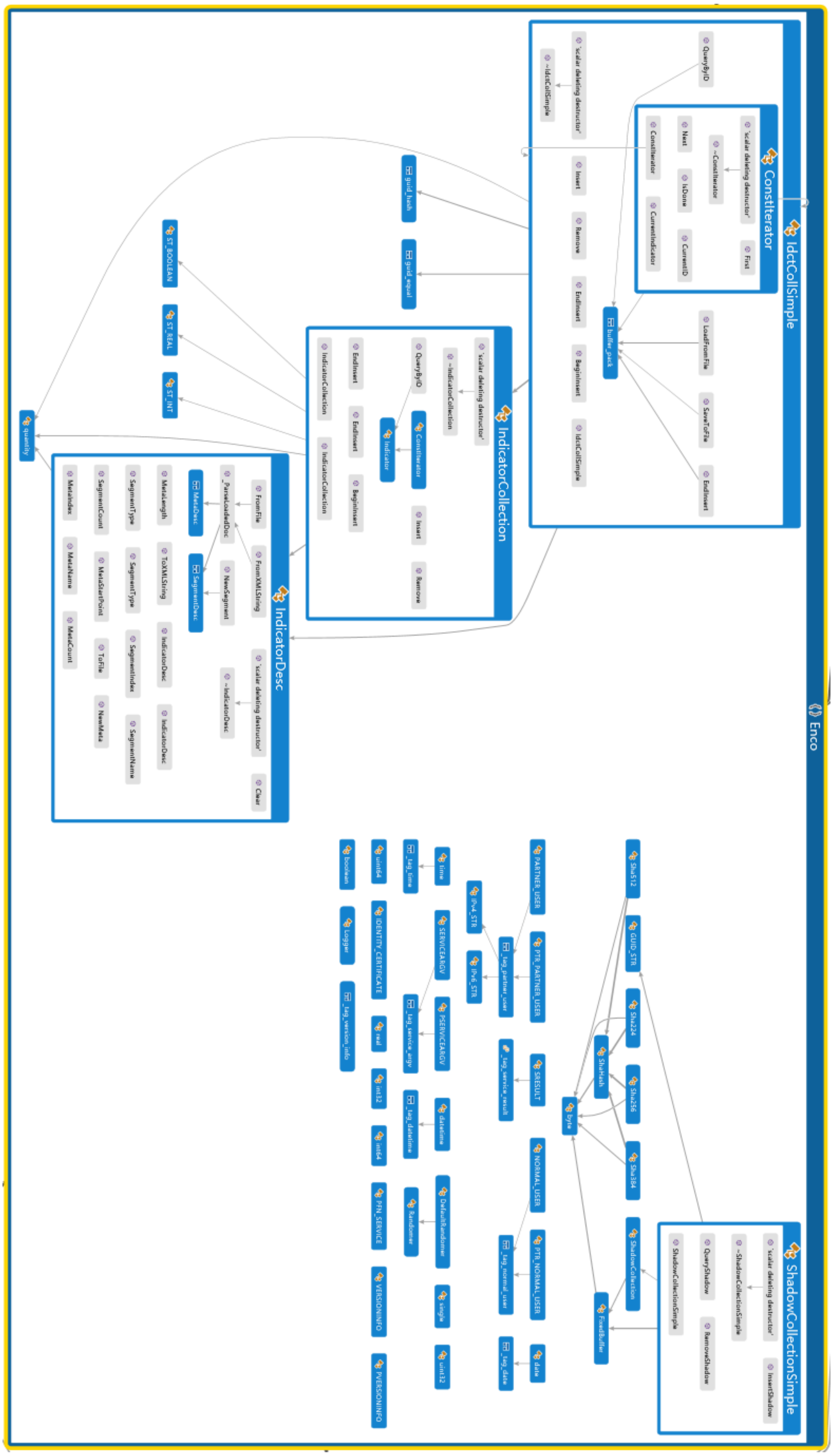


图 5.8

项目通用部分主要包括数据存储管理的实现、秘密分割的实现、安全哈希的实现以及加密算法的实现等。目前第一个版本（0.1）的实现当中，为了提高随机访问的效率，使用了 STL（MSVC 的实现）中的 Unordered_Map，其基础数据结构是哈希表，有效地将随机访问的时间复杂度将至$O(1)$。



Enco
IdctCollSimple
ConstIterator
IndicatorCollection
IndicatorDesc
ShadowCollectionSimple

图 5.9

秘密共享部分主要负责秘密分割算法的实现，并且对实际使用的文件分割与还原场景提供基础支持，减少上层不必要的重复代码，降低偶然耦合程度。

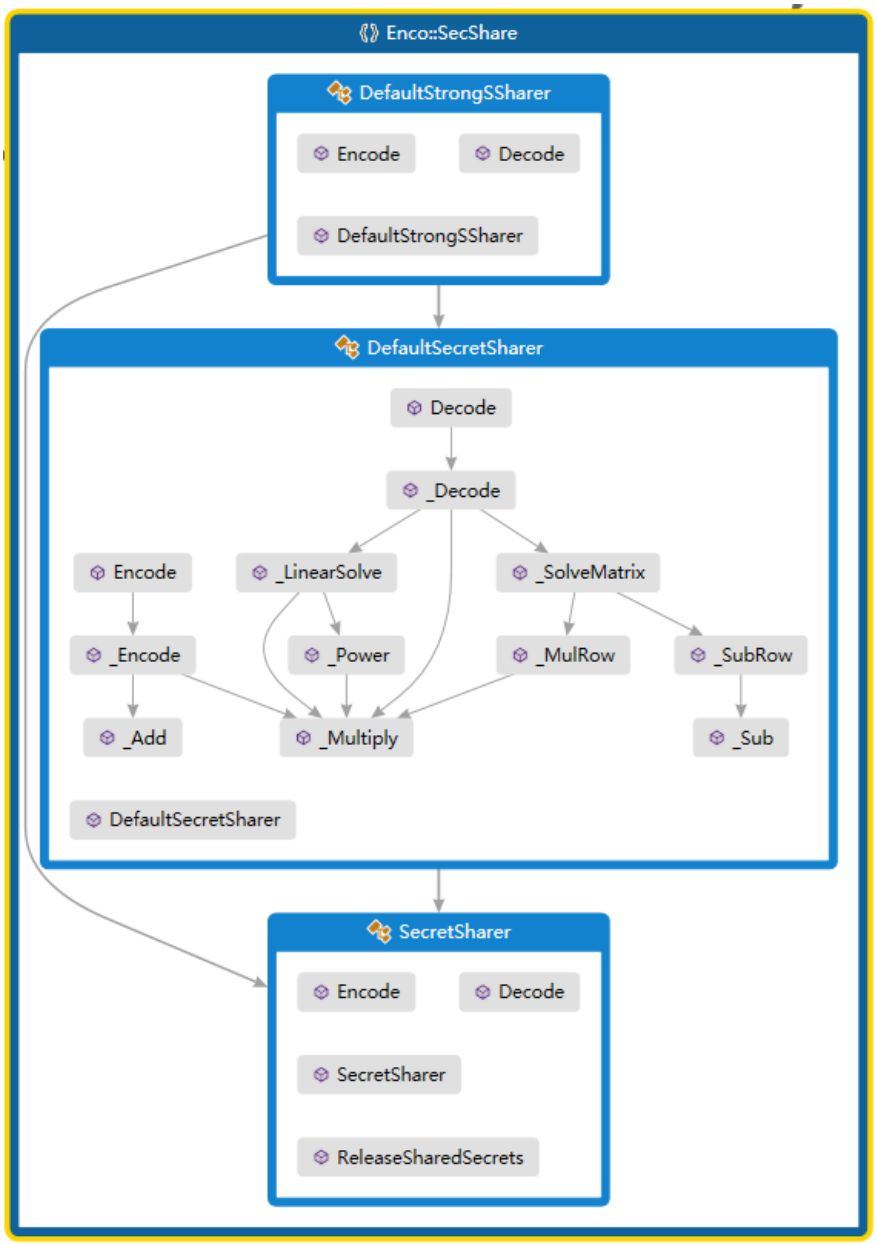


图 5.10

5.3 类详细设计

通过 UML 的类图，本论文可以更直观地展现系统当中的类结构和关系。下面将按一定顺序依次说明每个重要功能模块的类设计。

5.3.1 安全哈希模块

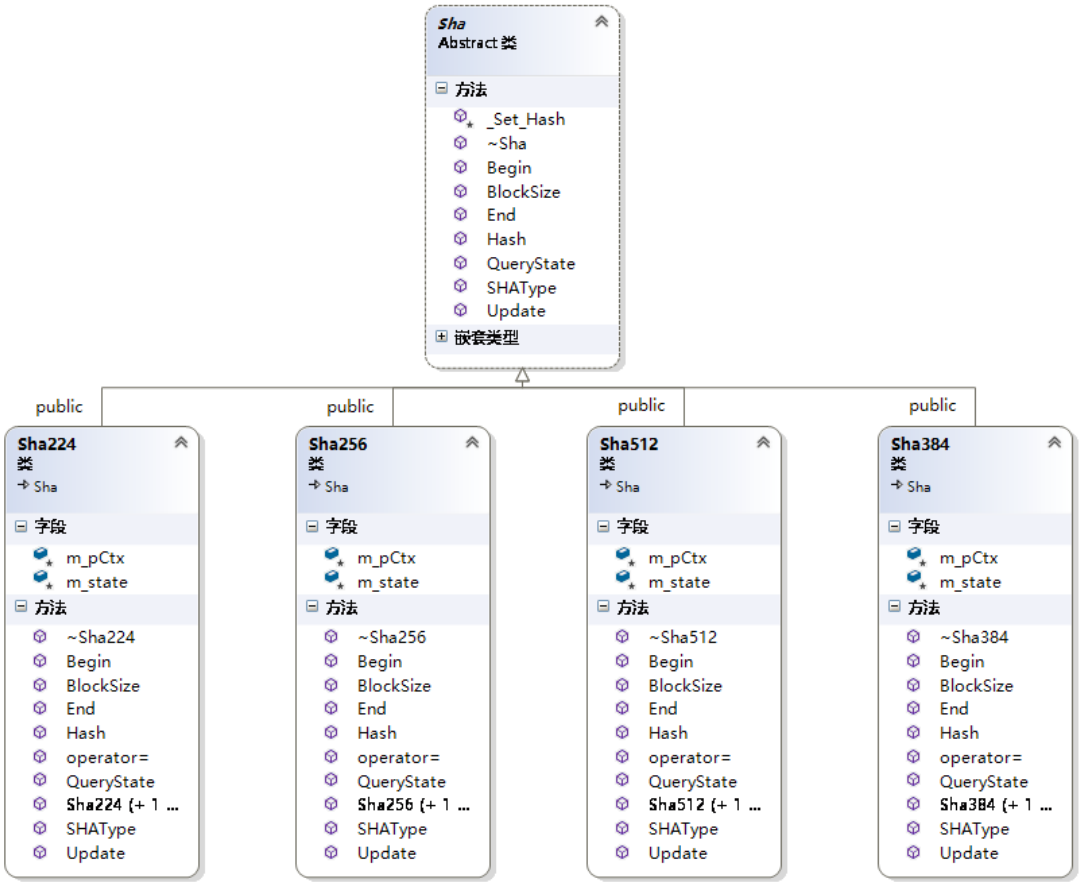


图 5.11

类 Sha 是一个抽象的父类，用于定义安全哈希操作的抽象接口，统一所有与安全哈希相关的操作，根据依赖倒置原则将抽象和实现分离并倒置。在后期的软件迭代过程中，可以通过继承和实现该类来优化和修改安全哈希的具体实现。

现准备了四个具体的实现为软件的其他模块所用：Sha224、Sha256、Sha512 和 Sha384。它们分别实现了 224 位、256 位、512 位和 384 位长度的安全哈希算法，符合 Sha 2.0 的标准。

5.3.2 秘密分割模块

类 SecretSharer 定义了秘密分割的接口。类 DefaultSecretSharer 通过使用 Shamir 的门限秘密分割算法实现了秘密分割，并作为系统的默认方案提供给其他模块调用。类 DefaultStrongSSharer 从使用了设计模式中的适配器模式（类继

承适配）重新封装了 DefaultSecretSharer 的主要功能，目的是使其更好地适应于文件资料的分割和还原。

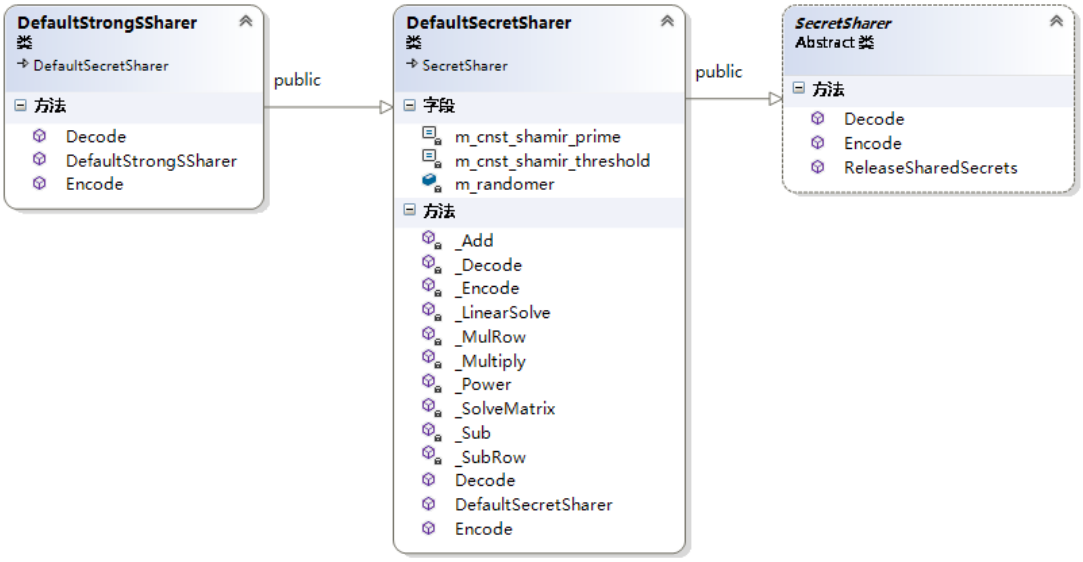


图 5.12

5.3.3 指标数据与影子数据存储模块

指标数据的存储与访问的关键功能由抽象类 IndicatorCollection 定义接口。并且，本论文中描述了一个通过 STL（MSVC 的实现）中的 Unordered_Map 实现快速随机访问能力的具体实现，该类直接继承于 IndicatorColletion，并命名为 IdctCollSimple。后期的迭代过程当中应考虑如何使用新的实现来替代这个不成熟的实现，不过目前的测试来看该模块工作情况还可以接受。

值得注意的是，对于 IndicatorCollection 的遍历访问，本论文要求使用迭代器模式进行。而且，在 IdctCollSimple 中已经实现了迭代功能。具体接口请参照 IndicatorCollection 的子类 IndicatorCollection::ConstIterator 的定义。

影子数据的存储于访问的关键功能由抽象类 ShadowCollection 定义接口。但请注意，由于处于用户数据安全的考虑，目前没有设计遍历影子数据的方法，也不允许在后期的迭代中加入遍历影子数据的设计和实现。

具象类 ShadowCollectionSimple 通过使用系统的文件系统管理能力简单有效的实现了 ShadowCollection 的基本功能。

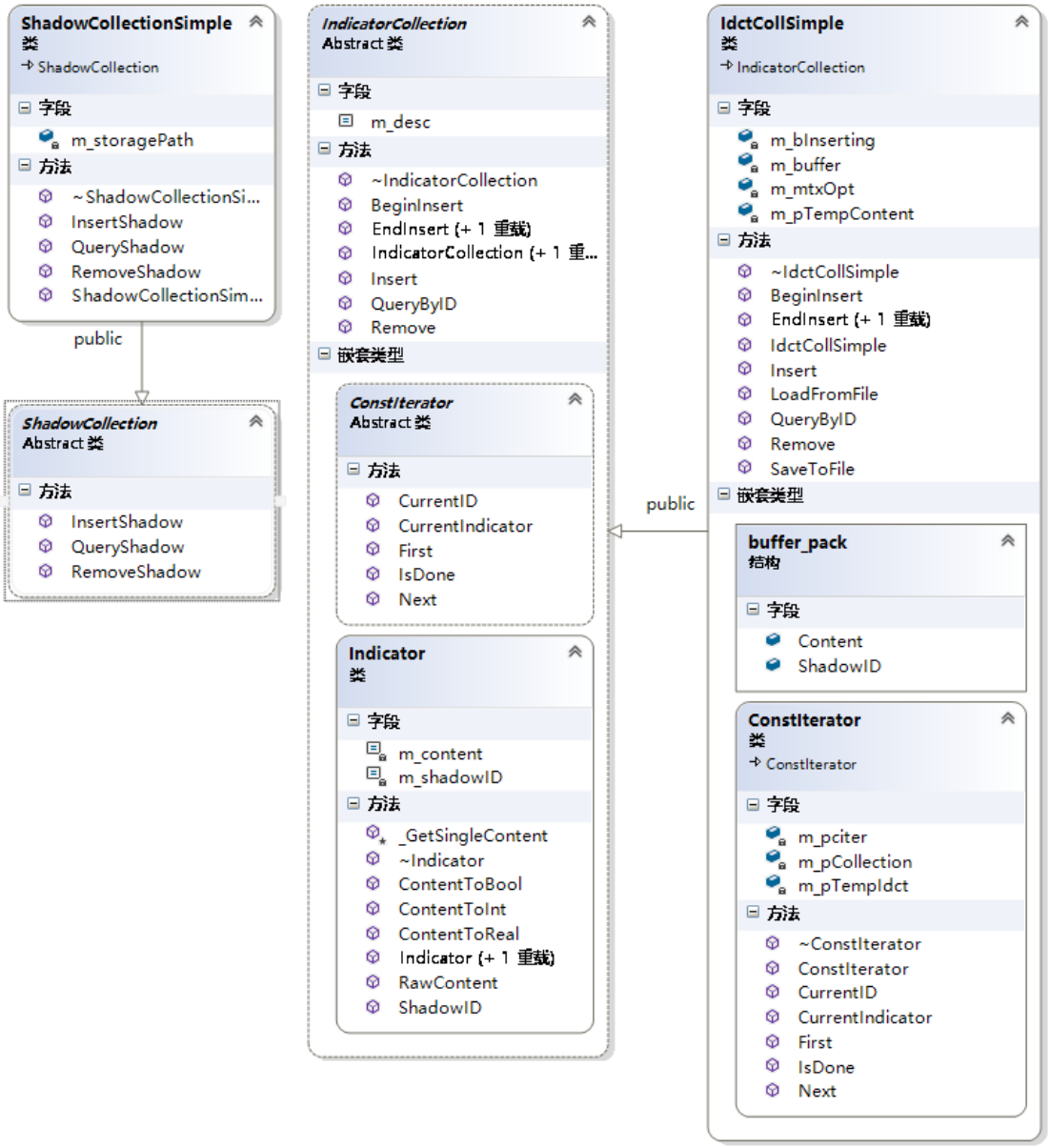


图 5.13

5.3.4 磁盘数据内存镜像

在客户端的功能当中，经常频繁访问和写入硬盘上的信息文件，为了更方便、准确地完成这些操作，需要一个合理的内存管理来辅助这个过程。现针对客户端的两个频率最高的业务——事务业务和指标业务，作一个简单有效的设计。

抽象类 IndicatorFile 和 BusinessFile 分别定义了指标信息文件和事务信息文件当中重要数据项读写接口。具象类 IndicatorFileXML 和 BusinessFileXML 都通

过 XML 对实际的磁盘文件上的信息进行读取和写入。

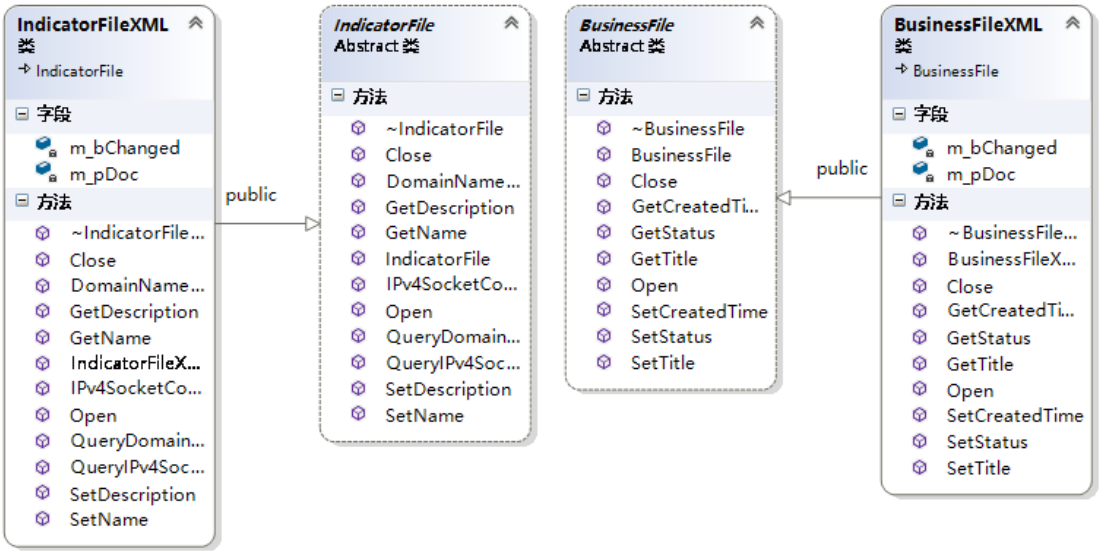


图 5.14

第 6 章 系统实现

6.1 程序入口部分

由于 P2P 平台软件（Enco）需要同时处理客户端命令和服务端命令，因此将整个程序的命令在入口部分做一个分流，分配给客户端功能模块与服务端功能模块。而且，在客户端功能模块与服务端功能模块当中，也具有子入口，用于针对带有参数的不同子命令作出对应的响应。

6.1.1 概要流程

在主入口函数当中，执行以下流程：

(1) 获取用户输入的命令行；

(2) 检查命令的合法性，失败则返回并退出程序；

(3) 进行必要的初始化操作；

(4) 判断命令类型；

(4.1) 如果是客户端命令，则进入客户端功能子入口；

(4.2) 如果是服务端命令，则进入服务端功能子入口；

(4.3) 其他情况，作为未知命令处理。（该分支不应被执行，所有的非法输入都应该在这之前被检测到）

(5) 结束程序。

在子功能模块入口函数当中，通过键值对命令派送功能实现按命令名称进行响应的功能。其大致流程如下：

(1) 事先注册命令名称，并实现响应实现；

(2) 检查输入的子命令行的合法性，失败则返回并退出模块；

(3) 进行必要的初始化操作；

(4) 根据注册列表中的命令名称查找相应的处理函数，并调用它。失败则返

回并退出模块；

(5) 结束。

6.1.2 关键代码

程序入口的 main 函数实现

```
/**
* @function: main
* @description: The entrance of entire program
*/
int main(int argc, char* argv[])
{
    if (false == CheckArgs(argc, argv))
    {
        WrongArgs(argc, argv);
        return ERR_WRONGARGUMENT;
    }
    if (strcmp("--server", argv[1]) == 0 ||
        strcmp("server", argv[1]) == 0)
    {
        return Enco::Server::EnterServer(argc, argv);
    }
    else if (strcmp("--client", argv[1]) == 0 ||
        strcmp("client", argv[1]) == 0)
    {
        return Enco::Client::EnterClient(argc, argv);
    }
    return 0;
}
```

客户端子入口函数的实现

```
/**
@function: EnterServer
@description: the entrance of server services
*/
int Client::EnterClient(int argc, char* argv[])
{
    int exeres;
```

```
    exeres = CheckArgs(argc, argv);
    if (exeres != SUCCESS)
    {
        UnexpectedExit(exeres);
        return exeres;
    }
    exeres = Client::Initialize();
    if (exeres != SUCCESS)
    {
        UnexpectedExit(exeres);
        return exeres;
    }
    // initialize logger
    global_pLogs = Logger::Create(global_logs_path);
    if (nullptr == global_pLogs)
    {
        exeres = ERR_CREATELOGS;
        UnexpectedExit(exeres);
        return exeres;
    }
    global_pLogs->LogInfos("\nStart logging.\n>\n>\n>");
    global_pLogs->LogExceptions("\nStart logging.\n>\n>\n>");
    // execute commands ------------------------------------------------
    boolbExed = false;
    for (EXEUNIT_LIST_ITEM& item : exeunit_list)
    {
        if (strcmp(argv[2], item.pObj) == 0 &&
            strcmp(argv[3], item.pOpt) == 0)
        {
            string logText = "Execute command: [";
            logText.append(item.pObj);
            logText.append("] with option: [");
            logText.append(item.pOpt);
            logText.append("] and parameters: [");

            std::vector<std::string>params;
            for (int i = 4; i < argc; ++i)
            {
                logText.append("<");
                logText.append(argv[i]);
                logText.append(">");
```

```
                params.push_back(argv[i]);
            }
            logText.append("].");
            global_pLogs->LogInfos(logText);
            item.pfnExeunit(params);
            bExed = true;
            break;
        }
    }
    if (bExed == false)
    {
        global_pLogs->LogInfos("Cannot find the command.");
        PRINT_LINE("Cannot find the command.");
    }
    // disposing ----------------------------------------------------
    Uninitialize();
    global_pLogs->LogInfos("\nStop logging.\n<\n<\n<");
    global_pLogs->LogExceptions("\nStop logging.\n<\n<\n<");
    SAFE_RELEASE(global_pLogs);
    return SUCCESS;
}
```

服务端子入口函数的实现

```
SRESULT ENCO_NETWORK __stdcall  Server::GotClient(const PSERVICEARGV& argv)
{
    char    cmdhead[CMDHEAD_LEN + 1] = "";
    std::string cmdLine;
    while (true)
    {
        // read command
        if (SRESULT::Succeeded != ReceiveText(argv->Socket, cmdLine)) break;
        // pick cmd head from the command line.
        if (PickCmdHead(cmdhead, cmdLine.c_str()) < 0)
            continue;
        bool bExecuted = false;
        // try to execute command
        for (EXEUNIT_LIST_ITEM& item : exeunit_list)
        {
            if (strcmp(item.pCmdHead, cmdhead) == 0)
            {
```

```
                item.pfnExeunit(argv, cmdLine.c_str());
                bExecuted = true;
                break;
            }
        }
        if (false == bExecuted)
        {
            // send "Not found Cmd" result back to client.
            if (SRESULT::Succeeded != SendText(argv->Socket,
                    signal_not_found_cmd))
                break;
        }
    }
    return SRESULT::Succeeded;
}
```

6.2 秘密共享模块

6.2.1 概要流程

分割秘密：

(1) 计算原文散列值，用于后期还原校验；

(2) 准备计算空间和缓存；

(3) 通过 Shamir 的门限算法逐单位（字节、字或双字等）进行秘密分割运算，并将结果写入预先准备的缓存当中；

(4) 为每个秘密碎片索引顺序；

(5) 清理计算空间和内存资源；

(6) 完成分割。

还原秘密：

(1) 获取每个碎片索引标记和长度；

(2) 准备计算空间和缓存；

(3) 尝试将碎片还原，得到还原后数据；

(4) 通过校验散列值检查是否还原成功；

(5) 清理计算空间和内存资源；

(6) 完成还原。

6.2.2 关键代码

分割秘密的实现

```
/*
* @implementation: Encode
* @description:
* @protocal:
*   ----------------------------------------
*   |  * shared index   [  4 bytes ]        |
*   |  * sha256 value   [ 32 bytes ]        |
*   |  * secret data    [  x bytes ]        |
*   ----------------------------------------
*/
bool SecShare::DefaultSecretSharer::Encode(std::vector<FixedBuffer*>&
sharedSecrets, const unsigned int& n, const unsigned int& k, const
FixedBuffer& secretToShare)
{
    // release the possible trash data.
    ReleaseSharedSecrets(sharedSecrets);
    // fetch the size of origin secret to share.
    size_t originSize = secretToShare.Size();
    // calculate hash value
    unsigned char hash_value[32] = { 0 };
    sha256((const unsigned char*)secretToShare.Buffer(), (unsigned
int)secretToShare.Size(), hash_value);
    // prepare the buffers to catch secrets
    for (Enco::uint32 i = 0; i < n; ++i)
    {
        FixedBuffer*pBuf = new FixedBuffer(originSize * 4 +
sizeof(Enco::uint32) + 32);
        Enco::uint32sidx = i + 1;
        sharedSecrets.push_back(pBuf);
        pBuf->Write(0, &sidx, sizeof(Enco::uint32));    // record sequence
        pBuf->Write(sizeof(Enco::uint32), hash_value, 32);  // record hash
```

```
    }
    // loop each bytes
    Enco::byte const*   pChar = static_cast<Enco::byte
const*>(secretToShare.Buffer());
    Enco::uint32*       pShrs = new Enco::uint32[n];
    const size_t    soff = sizeof(Enco::uint32) + 32;
    for (size_t idx = 0; idx < originSize; ++idx)
    {
        Enco::uint32 sec = *pChar;
        ++pChar;
        _Encode(sec, n, k, m_randomer, pShrs);
        for (unsigned int i = 0; i < n; ++i)
        {
            sharedSecrets[i]->Write(sizeof(Enco::uint32)*idx + soff,
&(pShrs[i]), sizeof(Enco::uint32));
        }
    }
    delete[] pShrs;
    if (sharedSecrets.size() > 0)
        return true;
    return false;
}
```

还原秘密的实现

```
/*
* @implementation: Decode
* @description:
* @protocal:
*   ----------------------------------------
*   |  * shared index   [  4 bytes ]        |
*   |  * sha256 value   [ 32 bytes ]        |
*   |  * secret data    [  x bytes ]        |
*   ----------------------------------------
*/
bool SecShare::DefaultSecretSharer::Decode(std::vector<FixedBuffer*>&
recoverdSecrets, const std::vector<FixedBuffer*>& sharedSecrets)
{
    // release the possible trash data.
    ReleaseSharedSecrets(recoverdSecrets);
```

```
    if (sharedSecrets.size() <= 0)
        return false;
    // fetch indices and check the length of data.
    size_t          secLen;
    Enco::uint32*   pIndice = new Enco::uint32[sharedSecrets.size()];
    unsigned char   hash_value[32] = { 0 };
    unsigned char   tmp_hash_value[32] = { 0 };
    secLen = sharedSecrets[0]->Size();
    for (unsigned int i = 0; i < sharedSecrets.size(); ++i)
    {
        sharedSecrets[i]->Read(&(pIndice[i]), 0, sizeof(Enco::uint32));
        if (secLen != sharedSecrets[i]->Size())
        {
            if (pIndice != nullptr)
                delete[] pIndice;
            return false;
        }
        if (i == 0)
        {
            sharedSecrets[i]->Read(hash_value, sizeof(Enco::uint32), 32);
        }
        else
        {
            sharedSecrets[i]->Read(tmp_hash_value, sizeof(Enco::uint32), 32);
            if (memcmp(tmp_hash_value, hash_value, 32) != 0)
            {
                if (pIndice != nullptr)
                    delete[] pIndice;
                return false;
            }
        }
    }
    Enco::uint32*   pShr = new Enco::uint32[sharedSecrets.size()];
    char        data;
    size_t          soff = sizeof(Enco::uint32) + 32;
    const size_t origin_secLen = (secLen - sizeof(Enco::uint32) - 32) / 4;
    FixedBuffer* pRecoverdSecret = new FixedBuffer(origin_secLen);
    for (size_t idx = 0; idx < origin_secLen; ++idx)
    {
        for (unsigned int j = 0; j < sharedSecrets.size(); ++j)
        {
```

```
            sharedSecrets[j]->Read(&pShr[j], sizeof(Enco::uint32)*idx + soff,
sizeof(Enco::uint32));
        }

        data = (char)_Decode(pIndice, pShr, (int)sharedSecrets.size());
        pRecoverdSecret->Write(idx, &data, sizeof(char));
    }
    sha256((const unsigned char*)pRecoverdSecret->Buffer(), (unsigned
int)pRecoverdSecret->Size(), tmp_hash_value);
    if (memcmp(tmp_hash_value, hash_value, 32) != 0)
    {
        if (pRecoverdSecret != nullptr)
            delete pRecoverdSecret;

        if (pShr != nullptr)
            delete[] pShr;

        if (pIndice != nullptr)
            delete[] pIndice;

        return false;
    }
    recoverdSecrets.push_back(pRecoverdSecret);
    if (pShr != nullptr)
        delete[] pShr;
    if (pIndice != nullptr)
        delete[] pIndice;
    return true;
}
```

第 7 章 安装与运用

本论文论述了一个基于秘密共享的多方协作撮合服务的系统，并且给出了其中三个重要组成部分之一的 P2P 平台软件的实现。因此，本章提供一个 P2P 平台软件的 0.1Beta 发行版（以下称其代号，Encoagent 2017）的安装指引和使用案例。根据该章内容用户完全可以快速入门，并且通过或多或少的案例修改就能够适用于其自身的需求。

7.1 安装

Encoagent 2017 的软件包目录内容如右图。

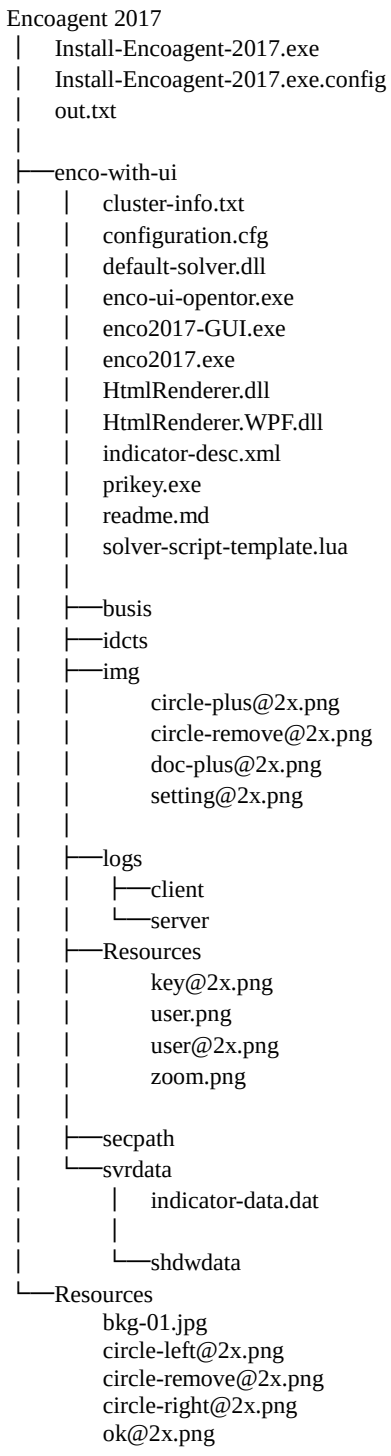


图 7.1

7.2 运用

第 8 章 结论

参考文献

致谢