信贸链协议

——全球贸易互操作协议

编写单位

慧贸天下（北京）科技有限公司

2025 年 2 月预发布版

v1.1.1-beta-20250224-zh

编制及发布说明

贸易数字化也称无纸贸易，是当前数字经济国际合作的主要共识领域。2024 年 6-7 月，70 多个 WTO 成员国达成一致的 WTO《电子商务协定》稳定文本，所凝聚的主要共识就是贸易数字化。联合国国际贸易法委员会（UNCITRAL）自 2017 年以来，也不断推动《电子可转让记录示范法》、《仓单示范法》、《可转让货物单证公约》等国际规则文件，推动国际贸易数字化合作。

2022 年成立的北京贸易科技联盟，在北京市商务局、北京海关、朝阳区政府等政府部门的指导下，共同推动建立可信贸易协作网络（“信贸链”），打造开放、中立的贸易数字化网络。联盟秘书长单位——慧贸天下（北京）科技公司牵头开发了“信贸链”技术协议和相关的“慧贸 OS”操作系统，已经应用于机场“双枢纽”、中关村综保区等北京市贸易数字化重点建设项目。同时，在北京市“两区”办和新加坡资讯通信媒体发展局（IMDA）的大力支持与推动下，“信贸链”兼容了 IMDA 开发的 TradeTrust 框架，以电子提单为突破口，实现了中国-新加坡首个全环节、分布式（互操作）贸易数字化实单试点（中粮--一丰试点）的落地。

下一步，为加快贸易数字化国际合作，慧贸天下在北京贸易科技联盟成员的大力支持下，决定向全球发布已遵循或兼容了 UN/EDIFACT、TradeTrust、URDTT 和 KTDDE、ANSI、RosettaNet、IATA 等国内和国际上诸多贸易数字化技术标准的“信贸链协议”（即全球贸易互操作协议），并将适时发布“慧贸 OS”产业操作系统的开源版、打造开源社区，推动各国贸易数字化系统的互联互通，不断提升跨国网络数据安全互信，共建全球可信贸易协作网络。

2025 年 2 月，“信贸链协议-全球贸易互操作协议”首次实现预发布。欢迎贸易数字化领域的各国企业、科研院所、政府机构、国际组织、专业人士提宝贵意见。慧

贸天下与北京贸易科技联盟成员，将不断完善该协议的后续版本和相关开源软件，为国际社会提供优质、开放、透明的国际数字公共产品。

法律声明概要

本协议为慧贸天下（北京）科技公司（以下简称“本公司”）撰写的预发布版本，本公司依法享有本协议当前版本及其后续版本的完整知识产权，未经本公司事先书面授权，任何单位或个人不得以任何形式复制、修改、传播、展示、演绎本协议或用于商业用途等。本公司保留依法追究侵权者法律责任的权利，且不对任何因使用本协议产生的后果承担责任。完整法律声明请见本协议文末。

目录

1、协议介绍

1.1 定义

“信贸链协议”，即全球贸易互操作性协议，是一套实现贸易参与方之间数字化系统高效互联互通和全环节贸易可信协作的通用技术规范。该协议分为五层，包括数字身份握手层、跨链流转与同步层、数据交换与存证层、数据标准转换层、交易过程管理层，其设计严格遵循并兼容包括UN/EDIFACT、TradeTrust、URDTT、KTDDE、ANSI X12、RosettaNet、IATA Cargo-XML等在内的多项国际及区域性贸易数字化技术标准。该协议基于区块链技术、人工智能技术底座所开发，旨在为构建数字化交易生态提供一套通用技术语言。

利用“信贸链协议”，贸易参与各方可以共建一个开放、中立、包容、安全且全球互联的分布式全球可信贸易协作网络（简称“信贸链”），推动传统贸易向数字贸易全面升级。在“信贸链”中，卖方与买方能够以高度灵活性发布和检索商品及服务信息，同时实现精准的供需匹配。与此同时，政府机构、行业组织以及第三方服务机构亦可深度参与其中，完成包括身份认证、业务撮合、交易执行以及行业监管在内的多样化功能。通过“智能合约+人工智能”的协同机制，“信贸链”能够实现交易的自动化撮合与执行，从而显著提升操作效率并降低人为干预带来的风险。另外，“信贸链”确保各参与方的原始贸易单据与敏感数据仍保留在其自有系统中，仅将不可逆的数字指纹上传至区块链网络进行存证，这样既保证了商业隐私，又利用区块链的不可篡改特性，确保了交易记录的真实性、完整性和可追溯性，从而大幅度降低信任成本与交易摩擦，为全球贸易生态提供了更高的透明度和安全性！

1.2 背景

当前，全球正处于百年未有之大变局之中，地缘政治博弈日趋激烈，人工智能技术革命迅猛演进，这些因素共同加剧了全球经济贸易环境的不确定性。在此背景下，全球贸易的发展进程正面临一系列亟待解决的深层次挑战与复杂难题。比如：

开放方面的问题：现阶段数据标准缺乏统一性，技术底座呈现出各式各样的形态。各个系统或者平台之间通常难以实现便捷的互联互通，相互之间的操作性较差，数据无法实现共享，进而催生出了更多的数据孤岛现象。

信任层面的问题：信任难以自我证明，也无法有效地进行传递。在国际贸易领域，非法违规的主体屡见不鲜，假冒伪劣的商品或服务充斥其中，虚假交易的情况也普遍存在。由于在线贸易主体以及商品的可信度欠佳，致使众多潜在的贸易机会最终未能成功转化为实实在在的交易。

安全相关的问题：就现有的各类系统而言，即便部分是基于区块链技术构建的，但其中绝大部分从本质上来说依旧属于中心化的架构模式。并且，很多平台还企图利用用户沉淀下来的数据去开展增值服务，这无疑给贸易安全带来了一定隐患。

贸易政策与法规差异问题：不同国家和地区有着各自独特的贸易政策和法规体系，这些差异可能导致企业需要花费大量的时间和资源去研究并遵守不同目标市场的贸易政策与法规，比如进出口关税规定、产品认证标准、环保要求等，这无疑增加了企业的运营成本和管理难度。

物流与供应链挑战问题：国际油价、汇率变化、运输能力供需不平衡、天气、自然灾害、港口拥堵、海关查验效率等因素会导致海运、空运等物流运输成本大幅波动。全球供应链的复杂性使得任何一个环节出现问题，严重影响了国际贸易的正常开展。

人工智能的应用问题：人工智能在贸易场景中的应用面临数据分散、质量不均的挑战，跨境数据流动受限于隐私法规，算法透明性不足易引发信任危机，加之各国伦理与安全标准差异，导致技术应用复杂化，难以充分发挥其潜力。

总结起来，当前地缘政治竞争导致的供应链重构、区域化趋势以及贸易壁垒的抬升，正在深刻重塑全球价值链的布局。与此同时，人工智能技术的快速迭代及其在各行业的广泛应用，虽然为效率提升和模式创新提供了前所未有的机遇，但也引发了诸如技术伦理、数据安全、算法偏见等新型风险。此外，技术鸿沟的扩大可能进一步加剧全球经济发展的不平衡性，从而对多边贸易体系的稳定性和包容性构成威胁。因此，在这一充满不确定性的时代，如何有效应对上述挑战，平衡技术创新与治理需求，构建更具韧性和可持续性的全球贸易体系，已成为国际社会亟需解决的核心议题。我们的方案建议是，利用区块链等分布式技术为贸易各方构建一个开放、中立、包容和安全的可信协作网络，充分释放人工智能的潜力，推进贸易环节各机构之间业务协作的自动化、智能化，促进供应链价值创造!

1.3 目标

基于上述背景，我们推动形成一套国际共识的全球贸易互操作协议，即“信贸链协议”，并以此为基石，推动各方在统一框架下按照“共创、共治、共享、共赢”的核心原则，共同建设多层次、多维度的分布式可信贸易协作网络。这一网络将涵盖区域级、行业级、链主级等不同层次，并最终构建起一个开放、中立、包容且全球互联的贸易新生态体系。具体目标如下：

首先，“信贸链协议”旨在通过技术与制度的双重保障，实现跨国家、跨行业、跨平台之间的无缝互操作性。当前，全球贸易数字化进程虽已取得显著进展，但因缺乏统一的标准和规范，各地区、各行业的数字贸易基础设施往往呈现碎片化状态，导致信息孤岛现象频发，阻碍了资源优化配置与效率提升。因此，建立一套具有高度灵活性与适应性的全球互操作协议至关重要。“信贸链协议”不仅能够提供底层技术支

持，还将充分考虑法律合规性、数据隐私保护以及文化差异等因素，确保其在全球范围内的普适性和可执行性。

其次，“信贸链协议”需要从以下几个方面着手构建针对人工智能应用的规范体系。首先，在数据治理层面，应建立统一的数据交换共享与保护机制，确保不同的人工智能系统在处理贸易相关数据时既能实现高效运算，又能严格遵守国际隐私保护标准。例如，可以通过区块链技术实现数据的分布式存储与加密传输，从而有效防止数据泄露与篡改。其次，在算法透明性方面，协议应要求所有参与贸易的人工智能系统具备可解释性，即其决策过程必须能够被人类理解与监督。这不仅能增强各方对人工智能系统的信任，也能避免因算法偏见或错误导致的不公平贸易行为。此外，在技术伦理层面，“信贸链协议”还需明确规定人工智能在贸易场景中的使用边界，例如禁止利用人工智能进行价格操纵、市场垄断或其他损害公平竞争的行为。

最重要的是，需基于“信贸链协议”逐步搭建起多层次的分布式可信贸易协作网络。在区域层面，该网络应能促进区域内经济体之间的高效协同，减少跨境交易中的摩擦成本；在行业层面，则要针对特定领域（如制造业、农业或服务业）定制专属解决方案，以满足行业特有需求；而在链主层面，大型企业或平台作为关键节点，需发挥引领作用，带动上下游中小企业融入这一生态系统，从而形成完整的价值链闭环。这种分层设计既体现了对复杂现实环境的深刻洞察，也兼顾了实际操作中的可行性。

此外，为了实现上述愿景，还需注重以下几个方面的工作：一是加强国际合作，争取更多国家和地区加入到“信贸链协议”的实践中来，不断扩大影响力；二是持续完善技术架构，包括区块链、人工智能、物联网等前沿技术的深度融合，以支撑更广泛的应用场景；三是秉持开源开放原则，基于“信贸链协议”不断升级开源“慧贸 OS”产业操作系统，吸引更多主体参与改进、优化及应用开发，促进技术传播共享，推动信贸链生态发展。四是建立健全法律法规体系，为新型贸易模式提供坚实的制度保障；五是培养专业人才，通过教育与培训提高全社会对数字经济的认知水平与实践能力。

总而言之，“信贸链协议”及其衍生的分布式可信贸易协作网络，不仅是应对全球化挑战的重要工具，更是重塑国际贸易秩序的关键一步。它将以技术创新驱动经济变革，以开放包容凝聚全球共识，最终助力人类社会迈向更加繁荣、和谐的未来。

1.4 治理机制

“信贸链协议”是由慧贸天下在合作伙伴的大力支持下所编制发布的。为促进“信贸链”生态的良好发展，慧贸天下联合相关权威机构于2022年中国服贸会上发起成立了贸易科技联盟。该联盟由多家知名机构共同参与，包括中国电子口岸数据中心北京分中心、北京微芯区块链与边缘计算研究院、电子商务交易技术国家工程实验室、富士康、中国外运等。经过两年的持续发展，联盟已成功吸引了包括科大讯飞、中国信息通信研究院（信通院）以及阿联酋中华工商总会在内的50余家国内外知名机构达成战略合作，构建起一个覆盖广泛、资源丰富的合作网络。在此基础上，联盟正式启动了全球贸易科技联盟的筹备工作，旨在打造一个引领全球数字贸易发展的核心平台，推动国际间贸易科技的深度协作与生态共建，为全球数字贸易治理体系的完善提供重要支撑。

慧贸天下将联合联盟成员持续升级“信贸链”协议的相关技术规范，推动成为行业、国家、国际层面的共识标准，共同建设“慧贸OS”产业操作系统开源社区，研究并试点示范基于“信贸链协议”的新场景、新模式和新服务。此外，联盟还将建立跨专业、跨界的专家会商机制，为政府和行业提供贸易创新发展的专业建议，并推动由“信贸链协议"建立的各种业务网络与更多贸易、物流与金融平台互联互通，支持实现全链条跨境贸易可信协作。

在推动“信贸链协议”形成共识并建设可信贸易协作网络的工作中，“共创、共治、共享、共赢”是贯穿整个体系的核心理念。所谓“共创”，是指所有参与方都应积极贡献智慧与资源，在技术研发、规则制定及应用推广等方面携手合作；“共治”强调治理结构的公平透明，避免权力集中于少数利益相关者手中，同时引入多方监督机制，确保决策过程科学合理；“共享”意味着打破传统壁垒，让每个参与者都能从

数据流动、知识积累和技术进步中获益；而“共赢”则是最终追求的结果——通过协同效应最大化整体价值创造能力，使所有成员在竞争中获得可持续发展优势。

1.5 法律声明

本法律声明由慧贸天下（北京）科技有限公司（以下简称“本公司”）作出，适用于本公司预发布的全球贸易互操作协议（以下简称“本协议”）。为保护本公司对本协议当前版本的合法权益，维护本协议的知识产权，特声明如下：

一、本协议性质与知识产权声明

本协议为本公司研发的底层技术协议的预发布版本，其主要目的是对相关理念和技术进行初步展示。本协议内容并非最终版本，后续可能根据反馈进行调整、优化，并发布更全面的正式版本。

本协议当前版本的内容，包括但不限于技术文本、架构设计、技术方案、算法逻辑、文档说明等，均为本公司独立开发或依法取得合法授权的成果。

本公司依法享有本协议当前版本及其后续版本的完整知识产权，包括但不限于著作权、专利权、商标权及其他相关权益。上述权利受《中华人民共和国著作权法》《中华人民共和国专利法》《中华人民共和国商标法》等相关法律法规保护。

未经本公司事先书面授权，任何单位或个人不得以任何形式复制、修改、传播、展示、演绎、反向工程、反编译本协议；将本协议内容整合至其他产品或服务中；将本协议用于商业化用途；传播或发布超出本公司许可范围的内容。任何超出授权范围的使用行为将被视为侵犯本公司的知识产权。

二、侵权责任与维权措施

若任何单位或个人未经授权复制、传播、使用或挪用本协议当前版本的内容，均属于侵权行为。本公司保留依法追究侵权者法律责任的权利，包括但不限于采取民事、行政或刑事法律手段。

对于侵权行为，本公司将要求：立即停止侵权；删除或销毁所有侵权内容；赔偿因侵权行为造成的经济损失；公开赔礼道歉。

三、免责声明

本协议当前版本仅为预发布版本，内容可能存在一定的不完善之处。本公司不对任何因使用本协议产生的后果承担责任。

本协议的后续正式版本将以最终发布的内容为准，预发布版本中的任何内容均不得作为正式协议的解释依据。

2.1 架构需求

“信贸链协议”旨在为贸易生态中的各参与方提供一套通用的技术规范，以实现数字化系统的高效互联互通，构建一个高效、透明且安全的全球化交易网络。以下从买家、卖家、物流服务商、金融机构、行业组织及政府单位等核心参与方的角度，对其业务需求与技术需求进行深入剖析。

2.1.1 买家

业务需求

买家的核心诉求在于提升交易透明度、支付安全性和物流可追溯性。具体而言，买家期望通过交易网络自动检索卖家公开的商品和服务信息，并通过智能合约+人工智能与卖家实现供需自动匹配，及时达成交易。同时能够实时获取商品的全生命周期供应链信息，包括生产溯源、运输轨迹和交付状态，以验证商品的真实性和质量。此

外，买家希望借助协议优化跨境支付流程，减少中间环节带来的额外成本（如手续费）和时间延迟（如清算周期），从而实现无缝化的国际交易体验。

技术需求

从技术层面来看，协议需具备强大的数据加密和隐私保护机制，确保买家的个人敏感信息（如身份数据、支付记录）在传输和存储过程中免遭泄露或篡改。同时，协议提供标准化的 API 接口，支持与主流电商平台、移动应用及第三方服务的无缝集成，使买家能够随时随地查询订单状态、物流进度及相关交易详情。

2.1.2 卖家

业务需求

相较于传统的中心化交易模式，在基于“信贸链协议”构建的交易网络中，卖家享有更高的自主权和灵活性。他们可以通过接入信贸链网络发布商品或服务信息，并根据商业策略选择性地将这些信息公开给平台上的潜在买家或需求方，亦可将其设定为私密状态。这种高度灵活的信息管理机制使卖家能够精准匹配市场需求，同时保护其核心商业利益。卖家需要一个基于区块链的信任机制，用于记录买家的交易行为，从而有效防范恶意差评、虚假投诉及欺诈性退款等风险。

技术需求

技术上，协议需支持智能合约功能，以自动执行合同条款（如付款条件、退款规则及违约处罚），减少人为干预导致的操作失误或纠纷。同时，协议应具备高并发处理能力，能够应对促销活动期间的瞬时流量峰值，确保系统稳定运行。此外，协议还需提供灵活的扩展性，以适应不同规模企业的业务需求。

2.1.3 物流服务商

业务需求

物流服务商的核心诉求在于提高运输过程的透明度与可追溯性，从而为客户交付更精准的服务体验。具体而言，他们希望通过协议实现货物全生命周期的数字化跟踪，涵盖仓储管理、运输监控、清关操作等关键环节，并基于数据分析提供时效预测与异常预警。此外，物流服务商需要一种可信的交接验证机制，以明确责任归属，避免因信息不对称引发的纠纷。

技术需求

技术层面，协议兼容物联网（IoT）设备的数据接入，例如GPS定位器、温湿度传感器及RFID标签，以实时采集运输环境的关键参数。同时，协议采用分布式账本技术（DLT），确保物流数据的不可篡改性，并支持快速生成合规报告供监管机构审查。此外，协议具备跨平台兼容性，以整合不同物流服务商的异构系统。

2.1.4 金融机构

业务需求

金融机构在贸易生态中扮演着资金流转与风险管理的重要角色，其核心需求是简化跨境支付与融资流程，并提升信用评估的准确性。具体而言，金融机构希望通过协议实现基于区块链技术的信用证开立与结算服务，以降低操作复杂度与人工成本。此外，他们需要利用协议中的大数据分析功能，对企业信用状况进行动态评估，为供应链金融提供科学决策依据。

技术需求

技术上，协议支持多币种支付与实时汇率转换功能，并符合国际反洗钱（AML）与了解你的客户（KYC）合规标准。同时，协议应提供细粒度的权限管理机制，允许金融机构根据不同的业务场景设置访问控制策略。此外，协议具备高可用性与容错能力，以保障金融交易的安全性与可靠性。

2.1.5 行业组织

业务需求

行业组织致力于推动行业的标准化发展与整体利益最大化，其核心诉求是通过协议建立统一的技术规范，促进成员企业之间的互操作性，消除因系统差异导致的合作障碍。此外，行业组织希望通过协议收集行业数据，用于分析市场趋势、制定政策建议，并推动行业的可持续发展。

技术需求

协议采用模块化设计，支持行业组织依据自身特点定制功能模块，并兼容去中心化治理模式，赋予其参与规则制定与技术升级的权利，以确保协议与行业发展需求动态契合。协议具备强大的数据聚合与分析能力，整合多源异构数据，通过先进算法生成行业洞察，同时支持开放接口标准（如 RESTful API 或 GraphQL），实现高效的数据交换与系统协同。此外，协议内置隐私计算技术（如联邦学习、同态加密），在数据共享中保护企业敏感信息，满足数据主权与合规性要求。

2.1.6 政府单位

业务需求

政府单位在贸易生态中承担着监管、服务与政策引导的多重职能，其核心诉求是通过协议提升贸易活动的透明度与合规性，从而维护市场秩序并促进经济发展。具体而言，政府希望通过协议实现对跨境贸易全流程的实时监控，包括商品进出口申报、

税收征缴、外汇管理等关键环节，以防范偷税漏税、非法走私及金融欺诈等违法行为。此外，政府需要借助协议推动数字化政务转型，为企业提供更加便捷高效的公共服务。

技术需求

从技术角度来看，协议需支持与现有政务系统的无缝对接，例如电子口岸平台、税务管理系统及海关清关系统，以实现数据的自动化流转与共享。同时，协议应符合国际通用的技术标准（如 ISO/IEC 27001 信息安全管理体系认证），并具备高度的可审计性，确保所有交易记录均可追溯且不可篡改。此外，协议集成人工智能（AI）与大数据分析工具，用于识别潜在的风险行为（如异常交易模式或可疑资金流动），为监管部门提供决策支持。

2.1.7 总结

“信贸链协议”作为贸易生态数字化转型的重要基础设施，其成功落地离不开对各参与方需求的精准把握与技术规范的科学设计。通过深入剖析买家、卖家、物流服务商、金融机构、行业组织及政府单位的核心诉求，我们可以清晰地看到，这一协议不仅需要在技术层面解决互联互通、数据安全与性能优化等问题，还需在业务层面兼顾效率提升、成本节约与信任构建等目标。

未来，“信贸链协议”的发展重点关注以下几个方向：一是加强与新兴技术（如人工智能、物联网、隐私计算）的深度融合，进一步拓展应用场景；二是推动国际化合作，吸引全球范围内的企业与机构加入生态，形成规模效应；三是持续完善治理机制，确保协议的公平性、透明性与可持续性。只有这样，“信贸链协议”才能真正成为驱动全球化贸易数字化变革的核心引擎，为构建高效、透明且安全的交易网络奠定坚实基础。

四、声明的生效与修改

本声明自本协议预发布之日起生效。

本公司保留对本声明内容进行修订的权利，修订后的内容将通过官方网站或其他适当方式通知相关方。

2、协议架构

2.2 设计原则

为确保“信贸链协议”在技术实现与业务应用中具备科学性、规范性和前瞻性，其设计需遵循一系列核心原则。这些原则不仅是协议开发的指导方针，也是保障其在全球贸易数字化生态中高效运行的基础。以下从开源开放、法律合规、标准兼容等多个维度展开具体阐述。

- 开源开放原则：推动“信贸链协议”的发展与广泛应用。一方面，相关代码应尽可能开源，吸引更多开发者、研究人员和企业参与协议改进、优化及应用开发。开源代码可接受广泛审查，利于发现潜在问题与漏洞，促进技术传播与共享，共建贸易数字化网络与生态。另一方面，提供开放接口，便于不同系统、平台和应用程序接入，且接口具有明确规范和文档说明，方便第三方开发与对接，拓展信贸链应用场景和服务范围。

- 法律合规原则：要求“信贸链协议”在设计、开发和应用全过程严格遵守国际国内相关法律法规及行业规范。遵循 DEPA、WTO 等国际高标准经贸规则，在法律法规方面，需遵循数据保护法、电子商务法、金融监管法规等，确保贸易数据处理、交易执行等符合法律要求，保护贸易主体合法权益。在行业规范方面，要遵循国际贸易、区块链等相关行业规范和标准，确保在技术实现和业务运作上符合行业要求，提升互操作的专业性和可靠性。

- 国内外标准原则。促使“信贸链协议”更好地服务国际贸易业务。积极采用国际通用标准和规范，如 UN/EDIFACT、URDTT、IATA、ANSI、RosettaNet，TradeTrust 协议等，在数据格式、通信协议、质量控制等方面与国际标准接轨，便于与国际市场上其他相关系统对接和互操作。同时，兼顾国内相关标准，满足国内市场特殊要求，实现内外兼顾，确保协议既能在国际上通用又能适应国内市场环境。

- 兼容性原则。旨在确保协议能与不同底层区块链技术架构及各类信贸链系统良好对接。无论是公有链、联盟链还是私有链，都要在技术层面适配其特性，如共识机制、智能合约语言等差异。同时，规定统一的数据格式和标准，保障不同系统中的贸易相关数据能准确识别、解析与交互，避免格式不兼容或语义歧义问题，从而实现各信贸链系统间的互操作。

- 安全性原则。对于“信贸链协议”至关重要。通过设计严格的身份认证机制，利用数字签名、公钥基础设施等技术手段验证贸易主体身份真实性，防止非法入侵与恶意操作。在数据传输过程中，借助加密技术保障数据的保密性、完整性和可用性，防止窃取、篡改并确保可恢复性。对涉及的智能合约进行严格安全审计和代码审查，杜绝逻辑漏洞被恶意利用，避免贸易资产错误转移等安全事故发生。

- 可扩展性原则。“信贸链协议”能够灵活适应国际贸易业务发展与规模变化。一方面，要能支持新贸易模式、金融服务等业务扩展需求，当出现新情况时，可快速调整与扩展以保障在信贸链上顺利开展互操作。另一方面，考虑到参与主体增多导致信贸链规模扩大，需优化共识机制、网络拓扑结构等，确保在不影响系统性能前提下容纳更多节点、链路和交易，实现高效稳定运行。

- 透明性原则。体现在两个方面。一是明确规定互操作的具体流程，涵盖数据传输、交易发起与确认等环节，让参与各方清晰知晓整个过程，减少误解与操作失误，便于各方监督。二是对互操作过程中数据使用情况予以明确说明，包括哪些数据会被共享、用途及使用主体等，确保数据使用基于各方知晓并同意的基础上，有效保护贸易主体的数据隐私权益。

- 高效性原则。聚焦于提升贸易互操作的效率。在交易处理方面，设计高效机制，通过优化共识机制、减少不必要验证环节等措施，尽力缩短交易确认时间，使贸易主体能快速完成交易，降低交易成本，提高国际贸易运作效率。在数据交互

方面，采用优化数据传输协议、高效缓存和索引机制等手段，确保数据在跨链交互时能快速、准确传递，让贸易主体及时获取所需信息。

2.3 技术边界

“信贸链协议”是一套基于区块链技术、分布式存储技术以及人工智能技术构建的通用技术标准，其核心目标是为全球贸易相关业务提供一套高效、可信且灵活的技术解决方案。相较于具体的底层技术实现，“信贸链协议”定位为一种中立的技术协议，旨在解决贸易领域中的关键痛点问题，包括但不限于数字身份认证、数据交互存证、数据标准化转换、跨链资产流转与同步，以及交易全流程管理等核心环节。这些功能的设计不仅体现了对当前贸易生态需求的深刻理解，也为未来多链融合和技术演进预留了广阔的空间。

在当今数字化贸易快速发展的背景下，传统的中心化系统已难以满足全球化贸易场景下的高并发、高安全性和高透明度要求。而“信贸链协议”通过引入分布式架构，从根本上改变了传统贸易信息传递和价值交换的方式。这一协议的核心设计理念在于打造一个端到端的分布式贸易生态系统，从而实现数据的去中心化存储、不可篡改记录以及跨平台互操作性。为了支撑这一宏伟愿景，底层存储架构采用了分布式设计作为其不可或缺的技术基础，并结合多种主流区块链框架，确保系统的兼容性与开放性。

秉承“共创、共治、共享、共赢”的建设理念，“信贸链协议”致力于适配多种主流区块链底层技术框架，如长安链（ChainMaker）、星火链网（BitXHub）、Hyperledger Fabric、以太坊（Ethereum）等。这些框架各具特色，例如长安链以其高性能和国密算法支持著称，适合国内企业使用；星火链网则专注于工业互联网领域的应用；Fabric 因其模块化设计成为企业级联盟链的首选；而以太坊凭借其庞大的开发者社区和丰富的智能合约生态，为跨行业协作提供了坚实基础。通过对这些多样化技术框架的支持，“信贸链协议”能够灵活应对不同应用场景的需求，同时促进异构区块链网络之间的互联互通。

为进一步降低接入门槛并提升系统互操作性，“信贸链协议”在技术规范中优先推荐采用开源无币公链作为分布式信息交互平台。这类公链通常具备较高的安全性与稳定性，且无需依赖代币激励机制即可运行，因此特别适用于企业间合作与跨境贸易场景。此外，无币公链还避免了因加密货币波动带来的经济风险，使得更多传统企业和机构愿意加入分布式贸易生态。这种选择不仅符合监管合规的要求，也为未来更大规模的产业协同奠定了基础。

在分布式文件存储层面，“信贸链协议”建议采用星际文件系统（IPFS, InterPlanetary File System）作为核心技术解决方案，用于实现去中心化存储与内容寻址功能。IPFS 通过将文件分割成小块并分散存储在网络节点上，有效提高了数据存储效率和访问速度，同时还解决了传统 HTTP 协议中存在的单点故障问题。然而，在具体业务场景中，各参与方可以根据实际需求选择更加灵活的数据交换方式。例如，对于高频次但低敏感性的数据传输，可以选择点对点直接传输的方式进行交换，仅利用区块链网络完成数据哈希值的存证与验证。这种方式既降低了主链的负载压力，又确保了数据完整性和不可篡改性，为复杂的贸易环境提供了多样化的技术选项。

值得一提的是，“信贸链协议”不仅仅关注现有技术能力的集成，更着眼于未来的扩展性与可持续发展。例如，随着人工智能、物联网、边缘计算的普及，分布式贸易生态将迎来更多创新机会。届时，“信贸链协议”可以通过升级其技术架构，进一步整合智能撮合、实时物流追踪、动态定价模型以及自动化结算等功能，从而全面提升贸易链条的整体效率。与此同时，该协议还在设计之初便充分考虑了跨链互操作的重要性，通过标准化接口和统一的数据格式，实现了不同区块链网络之间的无缝衔接。这不仅有助于打破“数据孤岛”现象，还为多链生态的融合发展铺平了道路。

综上所述，“信贸链协议”凭借其先进的技术架构与全面的标准设计，不仅能够满足分布式贸易的高并发、高安全性和高透明度要求，还为未来多链生态融合与技术演进预留了充分的扩展空间。它不仅是推动贸易数字化转型的重要工具，更是连接全球贸易参与者的桥梁，为构建一个更加开放、公平且高效的国际贸易体系贡献了力量。

2.4 总体架构

“信贸链协议”采用五层架构设计，自下而上分别为：数字身份握手层、跨链流转与同步层、数据交换与存证层、数据标准转换层、交易过程管控层。每一层均具备独立的功能模块和协议规范，同时通过标准化的接口实现与相邻层级的无缝通信与协同运作。

该架构通过多层次的功能解耦与协议分层设计，确保了分布式系统在交易全流程中的高效性、安全性和可信度。其中，底层的数字身份握手层负责建立可信的身份认证与初始连接；跨链流转与同步层实现多链间的资产与信息流转及状态一致性维护；数据交换与存证层提供高效的端对端交换机制和不可篡改的数据记录；数据标准化映射层完成异构数据格式的统一转换与语义对齐；顶层的交易过程管控层则聚焦于业务逻辑的全生命周期管理与不同业务角色的权限控制。

通过上述五层架构的有机协作，“信贸链协议”在技术层面实现了分布式交易环境下的高鲁棒性、强一致性和可验证性，为构建下一代可信贸易生态系统奠定了坚实的技术基础。

2.5 协议分层

2.5.1 数字身份握手层

数字身份握手层 是一个基于区块链技术的身份管理与交互协议层，旨在为组织、个人、产品、系统和硬件设备分配唯一的、不可篡改的数字身份，并通过去中心化的方式实现身份验证、数据交换和交互协作。该层的核心功能是通过区块链的透明性、安全性和可追溯性，确保所有实体的身份信息真实、唯一且可验证，同时支持跨实体之间的安全握手和可信交互。

- 组织身份分配：为各类组织（如企业、政府机构、非营利组织等）在区块链网络上分配唯一的数字身份。这些身份具有不可篡改性和可追溯性。

- 员工身份分配：为个人用户即员工在区块链网络上分配唯一的数字身份，确保用户身份的真实性和唯一性。

- 产品身份分配：为产品在区块链网络上分配唯一的数字身份，确保产品的产地、原料信息、环保认证可以通过区块链技术进行追踪和验证。

- 系统身份分配：为各类系统（如应用、设备、服务等）在区块链网络上分配唯一的数字身份，以便于系统间的交互。

- 硬件身份分配：为硬件设备在区块链网络上分配唯一的数字身份，确保设备的安全性和可管理性，同时也为设备的追踪和管理提供便利。

2.5.2 跨链流转与同步层

跨链流转与同步层 是一个用于实现异构或同构区块链之间资产、数据和业务互通的技术协议层。它通过统一的监控、传输和验证机制，解决不同区块链之间因共识算法、账本结构、加密机制等技术差异而导致的价值互通难题。

- 新兴区块链生态系统的数量和规模都急剧增加，用户未必单一使用一条区块链。而各种链的共识算法、账本结构、加密机制等技术各不相同，链与链之间缺乏统一的互联互通机制，难以做到价值互通。

- 跨链流转与同步层用于解决异构/同构链之间交易的监控、传输、验证等核心问题，实现多链之间资产互换及转移、数据交互共享、业务协同等场景，支持长安链/星火链/Fabric/以太坊等区块链底座，采用 tradetrust 等协议标准，使得不同区块链之间的资产和资讯能够互相传输，提升了区块链的应用空间和价值流通性。

2.5.3 数据交换与存证层

数据交换与存证层是分布式贸易数据交换与共享体系的核心组成部分，旨在通过区块链技术实现贸易数据的安全高效交换、可靠存证以及快速核验。该层包含以下功能模块：

- 数据交换：让参与方数据安全有效交换，并增强数据的安全性。
- 数据存证：为每份单证、事件生成唯一数据指纹进行加密上链保存。
- 链上核验：参与方可以通过存证核验入口查询文件的上链状态，验证文件的真实性。

2.5.4 数据标准转换层

数据标准转换层是为了实现不同系统间数据的无缝对接与高效转换而设计的核心功能模块。在实际应用中，不同用户系统对同类业务的数据标准可能存在差异，数据标准转换层通过预定义的转换规则和标准化协议，实现源系统数据格式与目标系统数据格式的自动转换。具体内容如下：

- 不同用户系统对同类业务定义的数据标准往往存在差异，传统方式下，跨系统对接时需要双方改造
- 通过数据标准转换层，可以通过转换规则按源系统中的数据格式和目标系统的数据格式进行自动转换，而无需对现有系统进行大规模修改或替换，从而降低了集成的复杂性和风险。

2.4.5 交易过程管理层

交易过程管理层是基于AI+区块链技术构建的、用于规范和管理贸易交易全流程的核心功能模块。该层通过AI助理和智能合约技术，实现贸易交易的全生命周期管理，确保交易过程透明、可追溯、高效且可信。其主要功能包括：

- 圈子管理：贸易主体可以创建圈子并邀请协作方加入圈子，可通过建立不同的圈子形成不同的垂直贸易业务圈，也可以通过圈子来控制信息流转的范围。

- 需求发布：需求方发布明确的需求信息到链上，包括产品、服务的详细描述，确保需求方的交易意图得到充分表达，同时提高需求的可见性和匹配效率。

- 供给发布：供给方发布产品、服务等资源的详细信息到链上，包括基础参数，确保供给信息透明公开、易于匹配。

- 供需匹配：系统基于大模型技术构建AI助理按照地理位置、时效性、价格区间等多维度规则，智能推荐最佳匹配交易对象，优化供需对接的精准度。

- 执行交易：在双方确认交易条件后，基于智能合约进行支付、交付、验收等操作，全程上链记录交易过程，确保操作透明、可追溯，并通过实时监控功能提示交易关键节点，支持异常情况的及时响应。

- 交易完成：交易结束后，所有参与方将交易的过程和结果记录在区块链上，并同步提供给交易双方及授权监管机构，为未来的审计、争议解决和法律依据提供坚实的基础。

2.6 关键技术

“信贸链协议”着重聚焦于借助技术创新以及对合作模式的优化改进，全力推动国际贸易朝着数字化转型的方向迈进，并助力建立起稳固可靠的信任体系。涉及的具体关键技术如下：

- 分布式数字身份：充分运用分布式数字身份（DID）这一先进技术，对参与贸易的主体以及各类商品的真实性展开严谨验证。通过此举，能够有效化解贸易过程中存在的信任难题，切实提升在线贸易主体与商品在市场中的可信度，为国际贸易活动营造更为诚信可靠的环境。

- 数据交换与存证：在整个贸易流程的各个环节当中，针对单证以及各类事件所产生的数据，均会进行严格的签名操作，并将这些数据妥善存证。而且，所有的数据交换过程都会被详细记录在区块链之上，凭借区块链不可篡改的特性，从根

本上杜绝任何数据被恶意篡改的可能性，以此确保数据的透明度以及可信度，为贸易各方提供准确可靠的数据依据。

- 跨链流转与同步：提供区块链之间资产与信息顺畅流转。采用 tradetrust 等技术标准，推进不同区块链上的各类资源得以实现相互连通，从而极大地提升了信息在各个链条之间的流通性，并且进一步强化了价值在不同链上的传递能力，有效打破了区块链之间的信息壁垒，促进了贸易资源的优化配置。

- 数据标准转换：采用自动化的数据格式转换方式，以此来全力支持源系统与目标系统之间实现无缝对接。这种方式的优势在于，无需对现有的各个系统进行大规模的改造工程，从而显著降低了系统集成过程中所面临的复杂程度以及可能出现的风险，使得不同系统之间的数据交互能够更加顺畅高效，为国际贸易的数字化运作提供有力支撑。

- 自动化交易：支持企业独立自主地发布自身在贸易活动中的各类需求、供给等相关信息。通过人工智能的先进匹配机制，迅速为企业找到合适的交易对象，从而实现交易的高效达成。有效消除传统交易平台所存在的垄断弊端以及信息隔离问题，大力推动整个市场朝着全面共享的方向发展，让国际贸易市场更加公平、开放、高效。

- 共识机制：优化共识机制，增强对网络攻击的防御能力，保障区块链系统的安全稳定运行。完善跨链互操作性，实现不同区块链之间的数据交换和互操作，降低跨链交易成本，促进区块链生态的融合与发展。

- 智能合约：优化智能合约设计，减少资源消耗，提高合约执行速度。加强对智能合约的审计和测试，减少漏洞和风险。同时，引入形式化验证等手段，提高合约安全性。拓展智能合约功能，使其支持跨链调用和交互，实现不同区块链之间的智能合约互操作。

- 数据隐私保护：采用隐私保护技术，实现数据在区块链上的安全存储和传输。这些技术可以保护用户隐私，防止数据泄露。通过权限管理、数字身份认证等手段，限制用户对区块链数据的访问权限，确保数据安全。针对跨链数据交互，采用隐私保护协议实现数据在跨链过程中的隐私保护。

3、技术规范

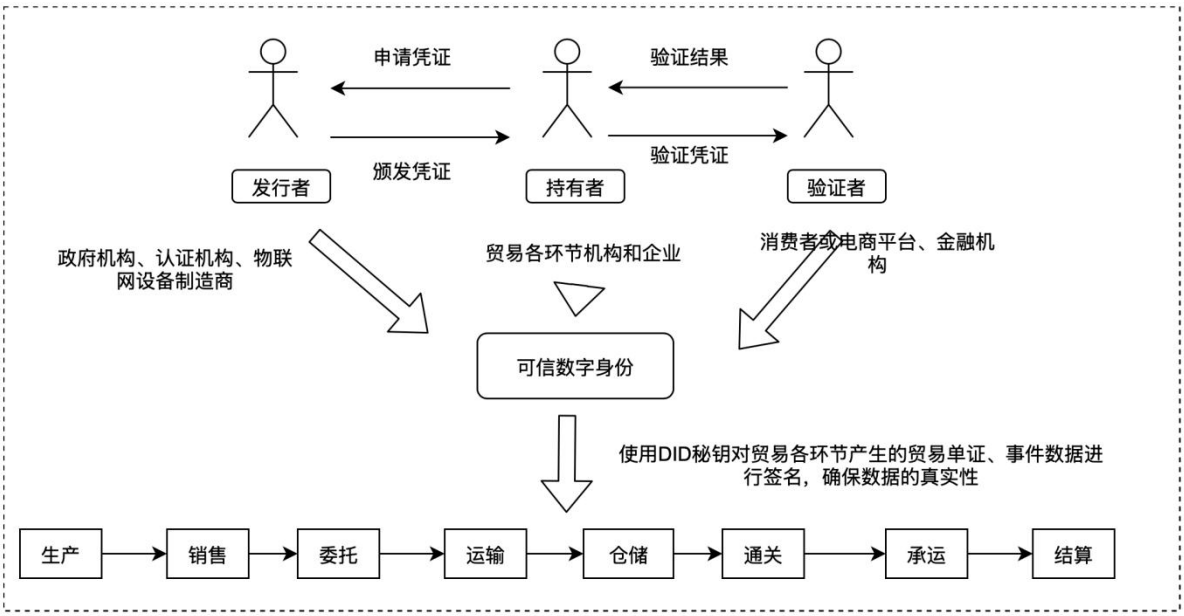
3.1 数字身份握手层

数字身份握手层主要包括三类角色，分别是身份持有者、凭证发行者、凭证验证者。

- 身份持有者：拥有链上数字身份的任何组织/产品/设备等主体，DID 的创建者和私钥拥有者，同时也是可验证凭证的申请者、接收者和使用者。

- 凭证发行者（认证方）：可发行数字凭证的组织，例如政府机构、认证机构，接收持有者的申请自动审核并签发身份证明凭证。

- 凭证验证者（验证需求方）：需要验证持有者身份的组织，例如银行、贸易伙伴，接收并验证由身份持有者出示的数字身份合法性。



3.1.1 流程说明

（1）注册数字身份

- 生成 DID 标识符：用户提供主体信息，根据主体类型不同，提交不同主体信息，例如组织主体类型包括但不限于组织名称、法定代表人、联系方式、注册地址等，系统调用注册数字身份智能合约。

- 智能合约按照预设的编码规则自动创建唯一、不可篡改的 DID 标识 。

- 生成 DID 文档：生成 DID 标识的同时，也会生成一对秘钥，并将公钥、加密算法和 DID 标识的绑定关系等公开信息以 DID 文档的方式保存。私钥则保存在用户自己设备中。

- 保存 DID 文档：完成 DID 文档的创建后，需要将其哈希值发布到区块链上，以确保其公开可查、不可篡改。具体方式为，DID 文档完整内容以 json 文本形式保存在 IPFS 中，将 DID 文档的 IPFS 链接地址与 DID 标识符、DID 文档哈希值保存在区块链上。

（2）认证数字身份

- 申请认证：用户向凭证颁发者（如政府、银行或大型企业）提交认证请求，例如向海关申请 AEO 认证，根据凭证类型提交 DID 及认证所需信息。

- 签发凭证：凭证颁发者通过其系统自动化验证用户提供的身份信息，并签发带有该机构签名的认证凭证。

- 保存凭证：在区块链上存储一个指向 VC 的链接或引用，而 VC 本身存储在链下的分布式存储系统（如 IPFS）中。

（3）同步数字身份

在不同区块链底座中通过跨链通信机制，将源链上的身份信息同步到目标链上，确保不同区块链之间能够互相识别和验证 DID，实现跨链复用。

（4）身份信息核验

- 用户提交凭证：用户向身份验证方提交相应凭证，例如包括组织机构代码、注册资本等属性的工商营业执照凭证。

- 验证凭证：身份验证方根据用户提交的 DID，从区块链上找到该 DID 对应的 DID 文档的 IPFS 地址，从 IPFS 获取 DID 文档内容，读取其中的公钥及加密算法，利用公钥对用户提交的凭证进行验签，验证通过，证明提交的信息可信，无需额外补充提交其他纸质材料。

（5）身份信息销毁

保护个人隐私和数据安全，允许终止数字身份。销毁操作则将彻底删除 DID 及其关联的所有信息。

3.1.2 数据结构

（1）DID 文档结构设计

每个 DID 都有一个相关的 DID 文档。DID 文档中包含有关 DID 主体的元数据，包含 DID 基本属性信息、验证信息、验证方法、服务端点和完整性签名，其中最关键的是 DID 与公钥的对应关系。

根据 W3C 的 DID 规范，定义 DID 文档的基本结构

字段	类型	含义
context	文本	上下文，用于描述 DID 文档的结构信息和 DID 所使用方法的规则。
ID	文本	表示 DID 标识符本身，因此 DID 文档是自描述的。
Public Key	文本	公钥描述信息，用于身份验证或者与 DID 主体进行交互。
Authentication	文本	用于证明 DID 与当前 DID 文档的关联性。
Service	文本	用于描述与 DID 相关的服务，例如与 DID 主体交互的位置和方式。可以是区块链节点地址，以便于其他实体知道如何与该身份持有者进行交互。
Recovery	文本	用于 DID 私钥丢失后重置公私钥。

DID 文档示例

```
    {
  "@context": "https://did.method.com/did/v1",
  "id": "did:method:123456789abcdefghi",
  "created": "2023-03-10T17:00:00Z",
  "updated": "2023-03-10T17:00:00Z",
  "publicKey": [{
    "id": "did:method:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:method:123456789abcdefghi#keys-2",
    "type": "Secp256k1VerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyHex":
"02b97c30de767f084ce3080168ee293053ba33b235d7116a3263d29f1450936b71"
  }],
  "authentication": [{
    // this key can be used to authenticate as DID ...
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:method:123456789abcdefghi#keys-1"
  }
  ],
  "service": [{
    "type": "drivingCardService",
```

```
    "serviceEndpoint": "https://method.com/endpoint/8377464"
  }, {
    "type": "padiCertificateService",
    "serviceEndpoint": "https://method.com/endpoint/8377465"
  }],
  "recovery": ["did:method:2323e3e3peewee2","did:method:2323e3e3dweweewew3"],
}


    {
  "@context": "https://did.method.com/did/v1",
  "id": "did:method:123456789abcdefghi",
  "created": "2023-03-10T17:00:00Z",
  "updated": "2023-03-10T17:00:00Z",
  "publicKey": [{
    "id": "did:method:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:method:123456789abcdefghi#keys-2",
    "type": "Secp256k1VerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyHex":
"02b97c30de767f084ce3080168ee293053ba33b235d7116a3263d29f1450936b71"
  }],
  "authentication": [{
```

```
    // this key can be used to authenticate as DID ...
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:method:123456789abcdefghi#keys-1"
  }
  ],
  "service": [{
    "type": "drivingCardService",
    "serviceEndpoint": "https://method.com/endpoint/8377464"
  }, {
    "type": "padiCertificateService",
    "serviceEndpoint": "https://method.com/endpoint/8377465"
  }],
  "recovery": ["did:method:2323e3e3peewee2","did:method:2323e3e3dweweewew3"],
}


    {
  "@context": "https://did.method.com/did/v1",
  "id": "did:method:123456789abcdefghi",
  "created": "2023-03-10T17:00:00Z",
  "updated": "2023-03-10T17:00:00Z",
  "publicKey": [{
    "id": "did:method:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
```

```
    "id": "did:method:123456789abcdefghi#keys-2",
    "type": "Secp256k1VerificationKey2018",
    "owner": "did:method:123456789abcdefghi",
    "publicKeyHex":
"02b97c30de767f084ce3080168ee293053ba33b235d7116a3263d29f1450936b71"
  }],
  "authentication": [{
    // this key can be used to authenticate as DID ...9938
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:method:123456789abcdefghi#keys-1"
  }
  ],
  "service": [{
    "type": "drivingCardService",
    "serviceEndpoint": "https://method.com/endpoint/8377464"
  }, {
    "type": "padiCertificateService",
    "serviceEndpoint": "https://method.com/endpoint/8377465"
  }],
  "recovery": ["did:method:2323e3e3peewee2","did:method:2323e3e3dweweewew3"],
}
```

（2）DID 可验证凭证

可验证凭证是一个 DID 给另一个 DID 的某些属性做背书而发出的描述型声明，并附加自己的数字签名，用以证明这些属性的真实性，可以认为是一种类型的数字证书。

例如：当公安机关给我们颁发了身份证，在 DID 中，这个身份证就是 VC；当银行给某家企业签发一个凭证，声明该企业资产规模大于 1000 万，而不披露实际的资产金额，保护了用户的隐私信息。

VC 包括的信息

字段	类型	含义	备注
id	文本	本 VC 的唯一标识，也就是证书 ID	
type	文本	vc 内容的格式	
issuer	文本	本 VC 的发行方	
issuanceDate	文本	本 VC 的发行时间	
credentialSubject	文本	VC 声明的具体内容	比如身份证作为公安机关颁发给我的 VC，在声明中会包含：姓名、性别、出生日期、民族、住址等信息。
proof	文本	对本 VC 的证明	通常就是颁发者的数字签名，保证了该 VC 能够被验证，防止 VC 内容被篡改以及验证 VC 的颁发者。

组织工商信息凭证示例

```
  {
"@context": [
  "https://www.w3.org/2018/credentials/v1",
  "https://schema.org/"
],
"id": "urn:uuid:bbba8d33-4e0d-4d40-8b8d-0b8b9a3d47ef",
"type": ["VerifiableCredential", "BusinessRegistrationCredential"],
"issuer": "did:web:example.gov.cn",
"issuanceDate": "2023-04-01T12:00:00Z",
"credentialSubject": {
  "id": "did:web:example.enterprise.com",
  "type": "企业",
  "name": "示例企业有限公司",
  "registrationNumber": "123456789012345",
  "legalRepresentative": "张三",
  "registeredCapital": "1000 万人民币",
  "establishmentDate": "2010-05-20",
  "businessScope": "人工智能技术研发；软件销售和技术服务。"
},
"proof": {
  "type": "EcdsaSecp256k1Signature2019",
  "created": "2023-04-01T12:00:00Z",
  "verificationMethod": "did:web:example.gov.cn#controller",
  "proofPurpose": "assertionMethod",
  "jws": "..."
}
```

```
}
```

产品基本信息凭证示例

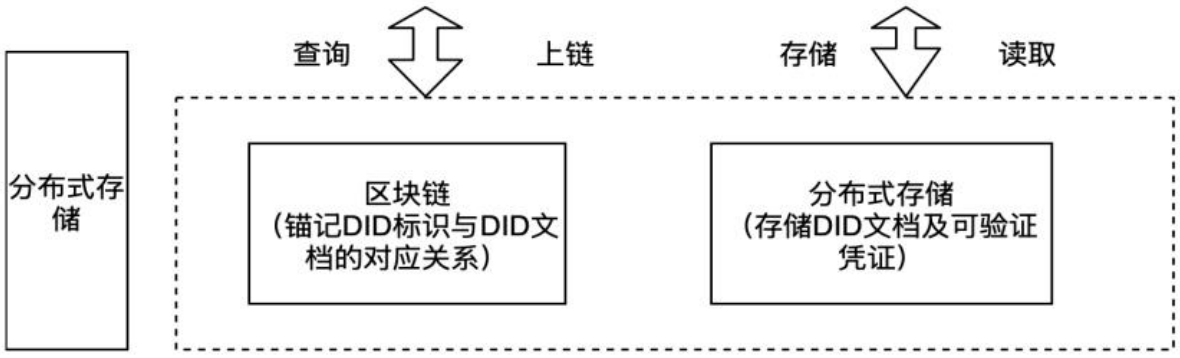
```
    {
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://w3id.org/security/suites/jws-2020/v1"
  ],
  "id": "urn:uuid:3fa85f64-5717-4562-b3fc-2c963f66afa7",
  "type": ["VerifiableCredential", "ProductCredential"],
  "issuer": "did:example:123456789abcdefghi",
  "issuanceDate": "2024-11-21T10:00:00Z",
  "credentialSubject": {
    product:{
      "productid": "9359502000041",
      "productClass": "Beef",
      "prodWeight": {
        "value": 1.5,
        "unit": "kg"
      },
      "productDescription": "This is a high-quality product with excellent
performance."
    }

  },
  "proof": {
```

```
    "type": "JsonWebSignature2020",
    "created": "2024-11-21T10:00:00Z",
    "verificationMethod": "did:example:123456789abcdefghi#keys-1",
    "proofPurpose": "assertionMethod",
    "jws":
"eyJhbGciOiJFUzI1NksiLCJiNjQiOmZhbHNlLCJjcmlOljpblml2NCJdLCJhbGciOiJFUzUxMiJ9..Q
WxOZWRTY3JpcHQiOiJyZWFkliwicmVmljpblmJhciJdLCJkZWxldGVkljpblml2NCJdfQ.SW4gUmVhZ
CBpbiB0aGUgc2VjdXJlIG1lc3NhZ2U."
  }
}
```


3.1.3 技术实现

（1）存储方案



由于区块链数据是对每个人可见的，必须谨慎决定将哪些身份数据存在在区块链上（链上），以及将哪些数据存储在可以链接至区块链的其他地方（链下）。

DID 标识与 DID 文档链接存储在链上：出于隐私保护、数据大小限制或交易效率的考虑，DID 文档仅将一个指向其内容的链接存储在区块链上。

DID 文档存储在链下：将实际的文档内容存储在链下，仅在提出专门请求时可以访问，这就保障了链下数据的隐私性，并且可在点对点基础上共享。

可验证凭证存储在链下：在区块链上存储一个指向 VC 的链接或引用，而 VC 本身存储在链下的分布式存储系统（如 IPFS）中。

（2）程序设计示例

通过程序生成 DID 与 DID 文档

```
    const (
    // IssuerDID is the issuer did
    IssuerDID = "did:aeotrade:3SADbinfdxiQTkZ7RHS558LxAmEo"
    // IssuerDIDPrivKey is the issuer did priv key
    IssuerDIDPrivKey =
"703e05ceb2b8b6b068a700235d8ee5fc09af4d5a0d797556b20c351d24bc6a6e"

    // HolderDID is the holder did
    HolderDID = "did:aeotrade:38Fj8zDy5nPcjjjopA62GcJqsmeY"
)

func main() {
    // issue a claim
    claim := issueClaim()
    claimBytes, _ := json.MarshalIndent(claim, "", "  ")
    fmt.Printf("issue claim success, claim: \n %s\n", string(claimBytes))

    // verify this claim
    if !verifyClaim(claim) {
```

```
        fmt.Printf("verifyClaim failed")
        os.Exit(-1)
    }
    fmt.Printf("verifyClaim success")

}

func issueClaim() *sdk.Claim {
    // gen claim
    claim := &sdk.Claim{
        BasicClaim: sdk.BasicClaim{
            Context:         []string{"https://www.w3.org/2018/credentials/v1"},
            Id:              "123",
            Type:            []string{"ProofClaim"},
            Issuer:          IssuerDID,
            IssuanceDate:    time.Now().Format("2006-01-02T15:04:05Z"),
            ExpirationDate: "2066-01-02T15:04:05.999Z",
            CredentialSubject: sdk.CredentialSubject{
                Id:                HolderDID,
                Type:              sdk.RealNameAuthentication,
                ShortDescription: "实名认证",
                LongDescription:  "该 DID 用户已在本机构完成实名认证",
            },
            Revocation: sdk.RevocationService{
                Id:   "http://example.com/v1/claim/revocations",
                Type: sdk.RevocationServiceTypeV1,
            },
```

```
        },
    }

    // sign claim
    issuerDIDPrivKeyByte, err := hex.DecodeString(IssuerDIDPrivKey)
    if err != nil {
        fmt.Printf("DecodeString err: %v", err)
        os.Exit(-1)
    }
    privKey, _ := btcec.PrivKeyFromBytes(btcec.S256(), issuerDIDPrivKeyByte)
    err = claim.Sign(IssuerDID, privKey)
    if err != nil {
        fmt.Printf("Sign err: %v", err)
        os.Exit(-1)
    }
    return claim
}

func verifyClaim(claim *sdk.Claim) bool {
    // get issuer did document
    resolver, _ := sdk.NewResolver("https://did.aeotrade.com")
    issuerDIDDocument, err := resolver.Resolve(claim.Issuer)
    if err != nil {
        fmt.Printf("Resolve err: %v", err)
        os.Exit(-1)
    }
```

```
    // use pub key from issuerDIDDocument to Verify claim issued by issuer, get
the DID by blockchain
    if claim.Verify(issuerDIDDocument, false) != nil {
        return false
    }
    return true
}
```

- 存储 DID 文档到 IPFS

```
    package main

import (
    "bytes"
    "encoding/json"
    "fmt"
    "log"

    "github.com/ipfs/go-ipfs-api"
)

// DIDDocument represents a decentralized identifier document
type DIDDocument struct {
    Context      string   `json:"@context"`
    ID           string   `json:"id"`
    PublicKey    []string `json:"publicKey"`
    Authentication []string `json:"authentication"`
```

```
}

func main() {
    // 初始化 IPFS 客户端，这里使用的是本地节点的默认地址
    ipfs, err := api.NewLocalApi()
    if err != nil {
        log.Fatal(err)
    }
    defer ipfs.Close()

    // 创建一个 DID 文档示例
    didDoc := &DIDDocument{
        Context: "https://w3id.org/did/v1",
        ID: "did:example:123456789abcdefghi",
        PublicKey: []string{
            "did:example:123456789abcdefghi#keys-1",
        },
        Authentication: []string{
            "did:example:123456789abcdefghi#keys-1",
        },
    }

    // 将 DID 文档编码为 JSON
    docJSON, err := json.Marshal(didDoc)
    if err != nil {
        log.Fatal(err)
    }
```

```
    // 将 DID 文档添加到 IPFS
    resp, err := ipfs.Add(bytes.NewReader(docJSON))
    if err != nil {
        log.Fatal(err)
    }

    // 输出 IPFS 返回的 CID
    fmt.Printf("DID Document stored with CID: %s\n", resp.Hash)
}
```

（3）智能合约设计示例

- 数字身份管理智能合约（注册、撤销、以及可验证凭证的颁发与验证）

```
 pragma solidity ^0.8.0;

contract DigitalIdentityRegistry {
    event DIDRegistered(bytes32 did, address indexed owner, string type);
    event DIDRevoked(bytes32 did);
    event CredentialIssued(bytes32 did, bytes32 credentialId);
    event CredentialVerified(bytes32 did, bytes32 credentialId, bool result);

    struct DIDDocument {
        bytes32 did;
        address owner;
        string type; // 身份类型（组织、员工、产品、系统、硬件等）
```

```
        string ipfsHash; // IPFS 哈希值
        bool isRevoked;  // 是否被撤销
    }

    struct Credential {
        bytes32 credentialId;
        bytes32 did;
        string ipfsHash; // 凭证的 IPFS 哈希值
        bool isValid;    // 凭证是否有效
    }

    mapping(bytes32 => DIDDocument) public didDocuments;
    mapping(bytes32 => Credential) public credentials;

    // 保存 DID 文档哈希值到区块链
    function registerDID(bytes32 _did, string memory _type, string memory
_ipfsHash) public {
        require(bytes(_type).length > 0, "Type cannot be empty");
        require(bytes(_ipfsHash).length > 0, "IPFS hash cannot be empty");
        require(didDocuments[_did].owner == address(0), "DID already registered");

        didDocuments[_did] = DIDDocument({
            did: _did,
            owner: msg.sender,
            type: _type,
            ipfsHash: _ipfsHash,
            isRevoked: false
```

```
        });

        emit DIDRegistered(_did, msg.sender, _type);
    }

    // 撤销 DID
    function revokeDID(bytes32 _did) public {
        DIDDocument storage doc = didDocuments[_did];
        require(doc.owner == msg.sender, "You are not the owner of this DID.");
        require(!doc.isRevoked, "This DID has already been revoked.");

        doc.isRevoked = true;

        emit DIDRevoked(_did);
    }

    // 获取 DID 文档
    function getDIDDocument(bytes32 _did) public view returns (DIDDocument
memory) {
        require(!didDocuments[_did].isRevoked, "This DID has been revoked.");
        return didDocuments[_did];
    }

    // 颁布可验证凭证
    function issueCredential(bytes32 _did, bytes32 _credentialId, string memory
_credentialIpfsHash) public {
        require(didDocuments[_did].owner != address(0), "DID not registered");
```

```
        require(credentials[_credentialId].did == bytes32(0), "Credential ID
already in use");

        credentials[_credentialId] = Credential({
            credentialId: _credentialId,
            did: _did,
            ipfsHash: _credentialIpfsHash,
            isValid: true
        });

        emit CredentialIssued(_did, _credentialId);
    }

    // 验证可验证凭证
    function verifyCredential(bytes32 _did, bytes32 _credentialId) public view
returns (bool) {
        Credential memory credential = credentials[_credentialId];
        require(credential.did == _did, "Credential does not match DID");
        require(credential.isValid, "Credential is not valid");

        DIDDocument memory didDoc = didDocuments[_did];
        require(!didDoc.isRevoked, "DID has been revoked");

        return true;
    }

    // 可选：撤销凭证
```

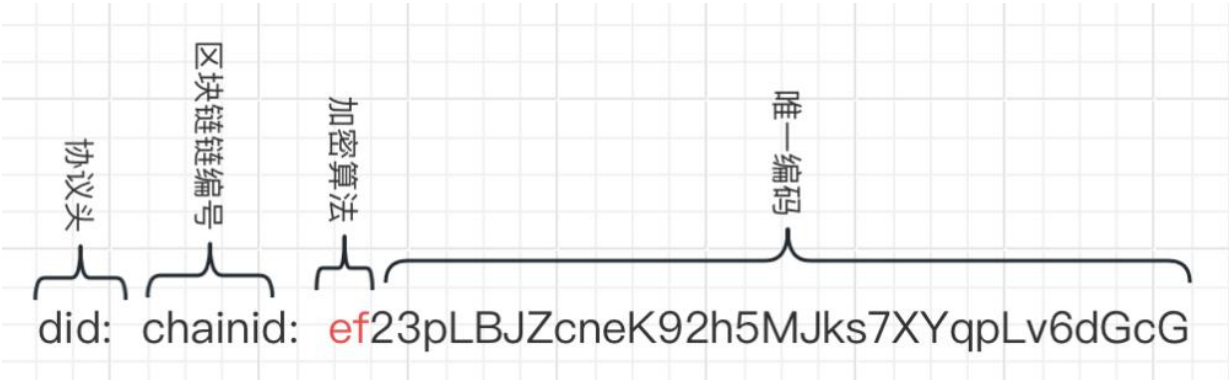
```
    function revokeCredential(bytes32 _credentialId) public {
        Credential storage credential = credentials[_credentialId];
        require(credential.isValid, "Credential is already revoked");
        require(didDocuments[credential.did].owner == msg.sender, "You are not
the owner of this DID");

        credential.isValid = false;

        emit CredentialVerified(credential.did, _credentialId, false);
    }
}
```

3.1.4 数字身份 ID 编码规则

数字身份 DID：由用户创建、拥有、并独立于任何组织的可验证标识符，用来达标一个实体的数字身份，这里的实体可以是个人、组织、实物、数字商品，它是一个不可变的字符串，可通过 DID 关联查找到 DID 文档。



DID 标识组成如下图所示

身份标识有协议头、信贸链专网编号、加密算法和唯一编码组成，由于每个专网采用了相同的标准和不同的编号，因此不会重复，但又能互认证。专网编号由贸易科技联盟进行分配，便于制定统一规则。

- 不需要中心化注册机构实现全球唯一性的分布式数字身份
- 在不同的信贸链专网各自注册，后续需要数据互通时能进行身份同步互通
- 统一签名算法
- 同一个公私钥生成的身份标识可在不同的链上使用

3.1.5 凭证类型

支持默认凭证类型及扩展凭证类型

- 默认凭证类型（应由对应认证机构通过智能合约发布）：中国国际贸易单一窗口用户凭证、全球通用法律实体可验证标识 vLEI、邓白氏凭证等
- 扩展凭证类型：

可以由发证方新增新的凭证类型，不断扩展，新增凭证类型时需提供如下信息，需通过智能合约进行创建，凭证内容：

- did：发证方的 DID
- website：公布发证方相关信息的官网，通常可以在发证方官网上提供凭证核验方式，方便大家来审查和校对身份持有人提供的信息是否属实。
- shortDescription:该凭证类型的短描述
- longDescription：该凭证类型的长描述
- requestData：用户申请该类凭证时需要提供哪些数据

3.2 跨链流转与同步层

通常情况下不同的贸易方通过节点接入到同一个区块链网络，但由于国际贸易的复杂性在信贸链网络中我们也将支持多个区块链进行接入，为了让打破数据壁垒，让业务更好的在跨区块链中进行开展，因此我们需要开放跨链流转与同步层。

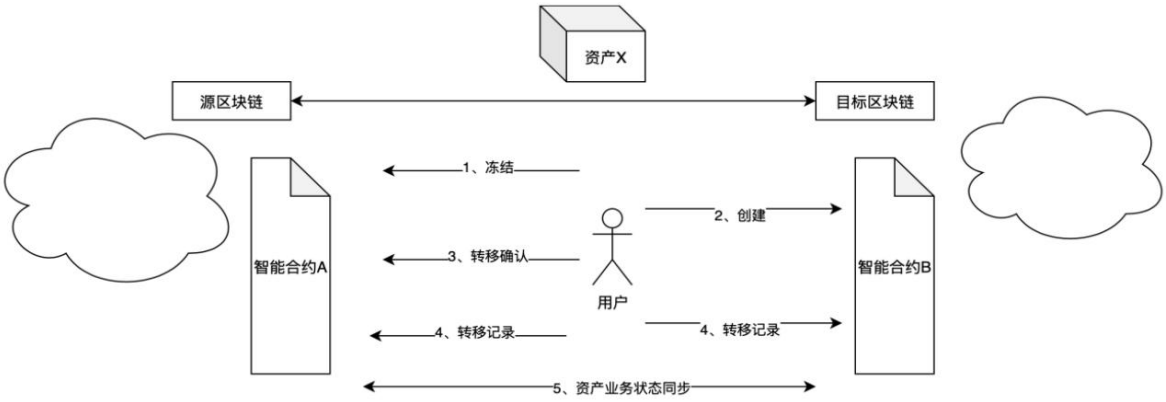
我们认为区块链中的跨链分为两种情况：

- 第一种需要跨链流转的资产：如代币、NFT、积分等，大家希望用等价交换或资产转移，原存在于 a 链中的资产记录应该销毁或扣减。

- 第二种是需要跨链同步的信息：如不同区块链中身份信息，产品服务信息等，大家要做的不是在 a 链中删除 b 链中添加，而是希望扩大信息的传播范围。

3.2.1 流程说明

- 跨链流转流程(资产、token 等)



在跨链技术实现过程中，保持资产的一致性是一项巨大的挑战。这是因为跨链交易涉及到不同链的独立性和异构性，一旦某一个环节出现问题，可能导致整个跨链交易异常。因此我们将分步进行跨链操作，具体资产跨链转移流程说明如下：

1. 冻结操作 用户在源区块链上将要转移的资产 X 进行冻结操作，确保资产在跨链过程中的安全性。

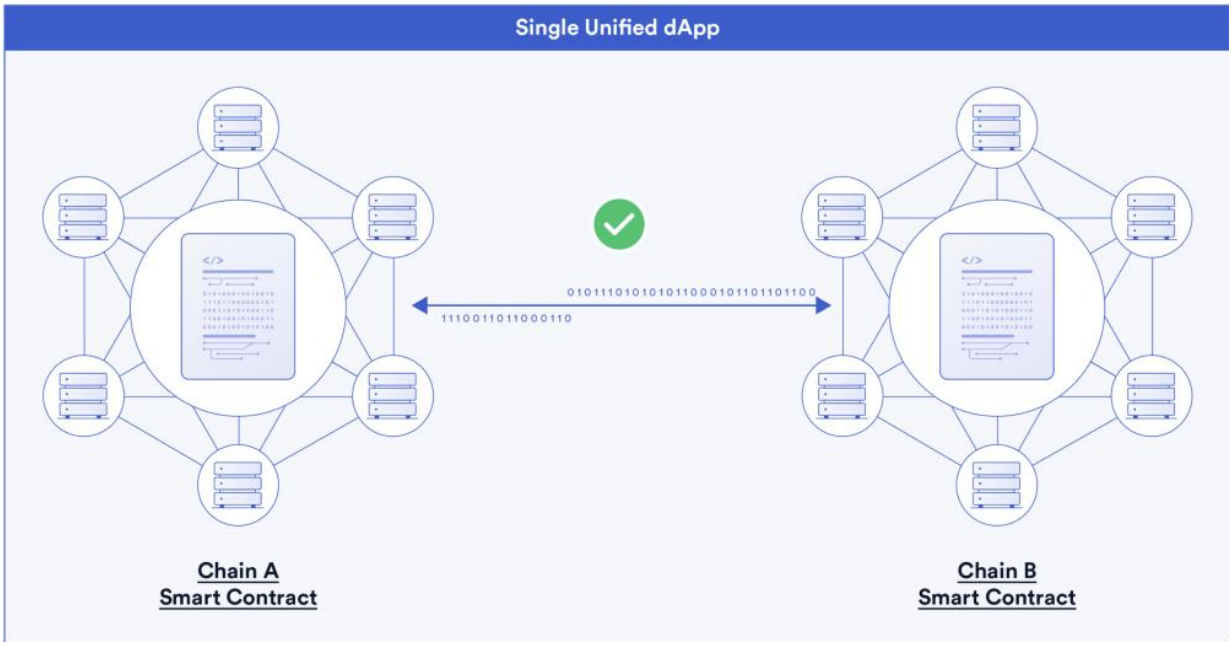
2. 创建对应资产 用户在目标区块链上通过智能合约 B 创建一个与资产 X 对应的资产，以便在目标区块链上使用。

3. 转移确认 用户确认资产 X 已成功冻结，并且在目标区块链上创建了对应的资产，确保转移过程的透明性和可靠性。

4. 转移记录同步 智能合约 A 和 B 分别记录资产 X 的转移过程，包括冻结时间、创建时间和转移确认时间，确保转移过程的可追溯性和不可篡改性。

5. 资产业务状态同步 智能合约 A 和 B 同步资产 X 的状态，确保两个区块链上的资产状态一致，完成资产转移。

- 跨链同步流程(数字身份、产品服务、交易事件信息等)



在跨链同步的实现过程中，有一些数据是希望它在多个链上同时存在，并进行扩散的如数字身份、产品或服务信息，具体资产跨链同步流程说明如下：

1.用户在源区块链上生成信息X，并声明需要同步到B链。

2.系统跨链同步程序将监控到同步需求，并将这个哈希值提交到目标区块链上的智能合约B。

3.智能合约B验证这个哈希值的有效性，确保它对应于源区块链上是需要扩散的属性。

4.如果验证通过，智能合约B在目标区块链上创建信息X的副本，并记录信息的原始来源。

5.同时，智能合约A在源区块链上记录信息X的扩散状态。

3.2.2 实现方案

（1）跨链转移实现方案

在当前主流的跨链方案中，我们选择哈希锁定来作为跨链转移的主要技术实现，该方案不需要一个第三方来作为信任中介，双方可以互为主链，主要通过技术手段来保证交易数据的一致性，无需借助可信的第三方来进行跨链的实现。

哈希时间锁合约协议（Hash Time-Locked Contract, HTLC）是一种在区块链中实现跨链资产交换的机制。它允许在不同区块链网络间的用户进行资产交换，同时确保交易的原子性、公平性和透明性。以下是关于HTLC合约协议的一些关键点：

- 定义与原理：哈希时间锁合约结合了哈希锁和时间锁两种机制。哈希锁确保只有知道特定秘密（称为“原像”）的用户才能解锁资产，而时间锁则设定了一个时间限制，超过这个时间限制未解锁的资产将自动退回给原始发送者。

- 跨链资产交换流程：

用户A生成一个随机数（秘密）并计算其哈希值，将哈希值发送给用户B。

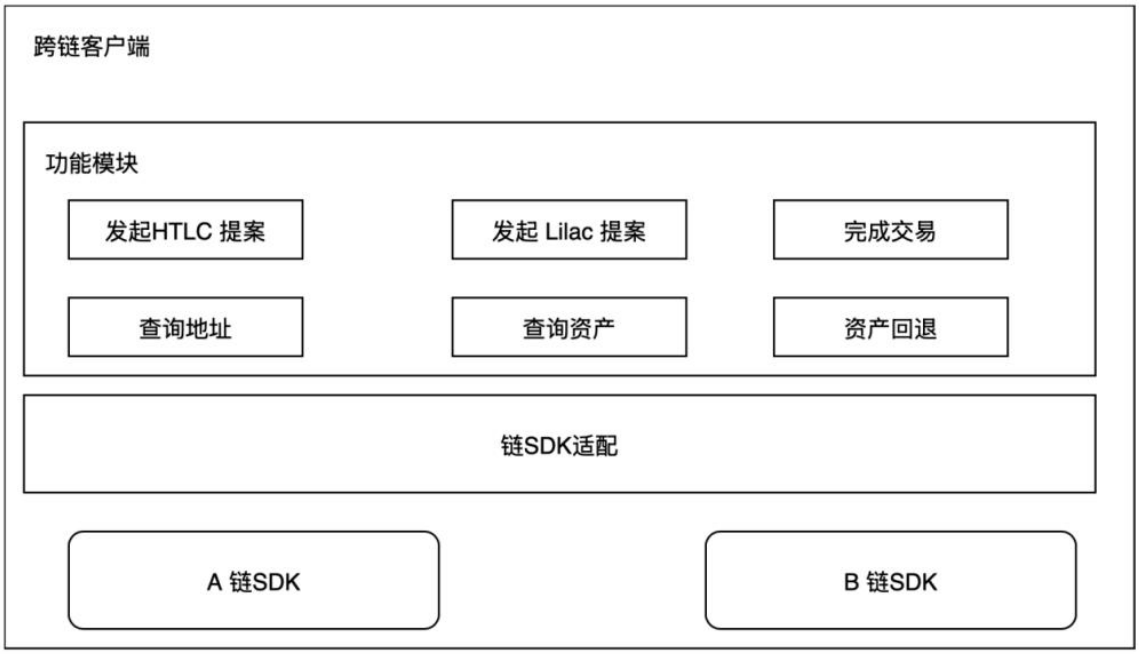
用户 A 在链 A 上创建一个资产锁定合约，使用哈希值和时间锁锁定资产。

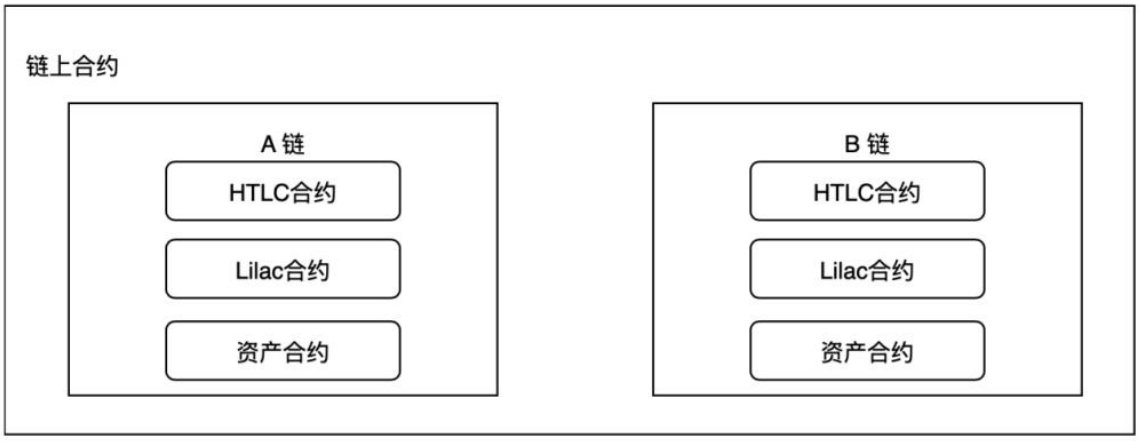
用户 B 在链 B 上创建一个哈希锁定合约，使用相同的哈希值和较短的时间锁锁定资产。

用户 A 使用秘密在链 B 上提取资产，或者用户 B 在时间限制内使用秘密在链 A 上提取资产。

如果在时间限制内未提取资产，资产将退回给原始发送者。

具体实现上我们在面向资产互换的原子性保障应用包含两种事务原子性保障协议，哈希时间锁合约协议(hash time lock contract, HTLC)和哈希时间锁合约优化协议 Lilac。





链上合约主要实现三个合约，分别是跨链资产 Assets 合约，哈希时间锁 HTLC 合约，哈希时间锁优化 Lilac 合约 Assets 合约主要包括以下功能：

approve 授权其他账户能够从当前账户中花费一定金额的 token

allowance 获取当前账户能够从指定账户中花费的 token 数量

transfer 转移一定数量的 token 给指定用户

balanceOf 获取当前账户剩余 token 数量

transferFrom 转移被授权人的 token 给指定用户

getAddress 获取当前用户的地址

HTLC 合约主要包括以下功能：

newProposal 发起一个新的哈希时间锁提案

getProposal 根据 proposal id 获取完整的提案信息

withdraw 根据 proposal id 获取转给自己的 token

refund 对一个超时的提案，取回自己锁定的 token

Lilac 合约主要包括以下功能：

newProposal 发起一个新的 Lilac 提案

getProposal 根据 proposal id 获取完整的提案信息

withdraw 根据 proposal id 获取转给自己的 token

refund 对一个超时的提案，取回自己锁定的 token

实现参考

提案数据结构

```
    // Proposal 提案结构
type Proposal struct {
    Sender     string `json:"sender"`     // sender account Address
    Receiver   string `json:"receiver"`   // receiver account Address
    Amount     int64  `json:"amount"`     // transfer/lock token amounts
    PosalTime int64  `json:"posaltime"` // proposal generation time
    TimeLock   int64  `json:"timelock"`   // timelock
    // hashlock, 在 htlc 中是 string, 在 lilac 中是[]string, 需要先转换成 any/[]any
```

```
    HashLock any `json:"hashlock"`
    // 原像, 在 htlc 中是 string, 在 Lilac 中是[]string, 需要先转换成 any/[]any
    Preimage any `json:"preimage"`
    // BlockDepth int64  `json:"blockdepth"`
    Locked      bool `json:"locked"`      // proposal locked status
    Unlocked    bool `json:"unlocked"`    // proposal unlocked status
    Rolledback bool `json:"rolledback"` // propsoal rolledback status

    // 以下是为了实现 proposal 匹配而额外增加的属性
    ProposalID              string `json:"proposalID"`      // proposal id
    ChainName               string `json:"chainName"`       // proposal chain name
    UserName                string `json:"userName"`        // proposal user name
    ContractName            string `json:"contractName"`    // 这个 proposal 合约名
    ProposalType            string `json:"proposalType"`    //proposal type such as
'htlc' 'lilac'
    CorrespondingChainName string `json:"correChainName"` // 对端的名字
    CorrespondingAmount     int64  `json:"correAmount"`     // 对端应该转给我多少钱
}
```

哈希锁定智能合约示例

```
    pragma solidity ^0.8.0;

contract HashTimeLockContract {
    // 事件声明
    event AssetLocked(bytes32 hashLock, address sender, uint amount, uint
timeLock);
```

```
    event AssetWithdrawn(address receiver, uint amount);
    event AssetRefunded(address sender, uint amount);

    // 资产锁定结构
    struct LockedAsset {
        bytes32 hashLock;
        address sender;
        uint amount;
        uint timeLock;
        bool withdrawn;
    }

    // 映射锁定资产 ID 到 LockedAsset 结构
    mapping(bytes32 => LockedAsset) public lockedAssets;

    // 锁定资产
    function lockAsset(bytes32 hashLock, uint amount, uint timeLock) public
payable {
        require(msg.value == amount, "Amount mismatch");
        require(timeLock > block.timestamp, "Time lock must be in the future");

        emit AssetLocked(hashLock, msg.sender, amount, timeLock);
        lockedAssets[hashLock] = LockedAsset({
            hashLock: hashLock,
            sender: msg.sender,
            amount: amount,
            timeLock: timeLock,
```

```
            withdrawn: false
        });
    }

    // 提取资产
    function withdrawAsset(bytes32 hashLock, bytes32 preImage) public {
        LockedAsset storage lockedAsset = lockedAssets[hashLock];
        require(!lockedAsset.withdrawn, "Asset already withdrawn");
        require(lockedAsset.timeLock <= block.timestamp, "Time lock has not
expired");
        require(keccak256(preImage) == lockedAsset.hashLock, "Invalid preImage");

        lockedAsset.withdrawn = true;
        payable(lockedAsset.sender).transfer(lockedAsset.amount);

        emit AssetWithdrawn(lockedAsset.sender, lockedAsset.amount);
    }

    // 退还资产
    function refundAsset(bytes32 hashLock) public {
        LockedAsset storage lockedAsset = lockedAssets[hashLock];
        require(!lockedAsset.withdrawn, "Asset already withdrawn");
        require(lockedAsset.timeLock <= block.timestamp, "Time lock has not
expired");

        lockedAsset.withdrawn = true;
        payable(msg.sender).transfer(lockedAsset.amount);
```

```
        emit AssetRefunded(lockedAsset.sender, lockedAsset.amount);
    }
}
```

lockAsset：用户 A 调用此函数锁定资产，并提供一个哈希值、资产数量和时间锁。资产被锁定在合约中，直到时间锁过期或资产被提取。

withdrawAsset：用户 B 知道哈希的原像（preImage），可以在时间锁过期前提取资产。

refundAsset：如果时间锁过期且资产未被提取，用户 A 可以调用此函数退还资产。

（2）跨链同步实现方案

在不需要转移的跨链信息，我们将采用信息同步方案，在绝大多数区块链系统中,网络层都采用了点对点组网的方式,区块链中节点可以自由地组网,任意两个节点之间通过消息传播协议实现消息和交易的传播。

P2P 组网方式有如下特点:

• 节点之间采用扁平化拓扑结构进行连接,不存在任何中心化的节点和层级结构。

• 节点的地位相互平等,每个节点均承担了节点发现、网络路由、传播区块、验证区块等功能。

数据通过消息传播协议在区块链网络中传播。不同的区块链系统采用的消息传播协议不同,例如,比特币系统和 Fabric 系统采用了 Gossip 协议,以太坊系统采用了 Kademlia 协议。

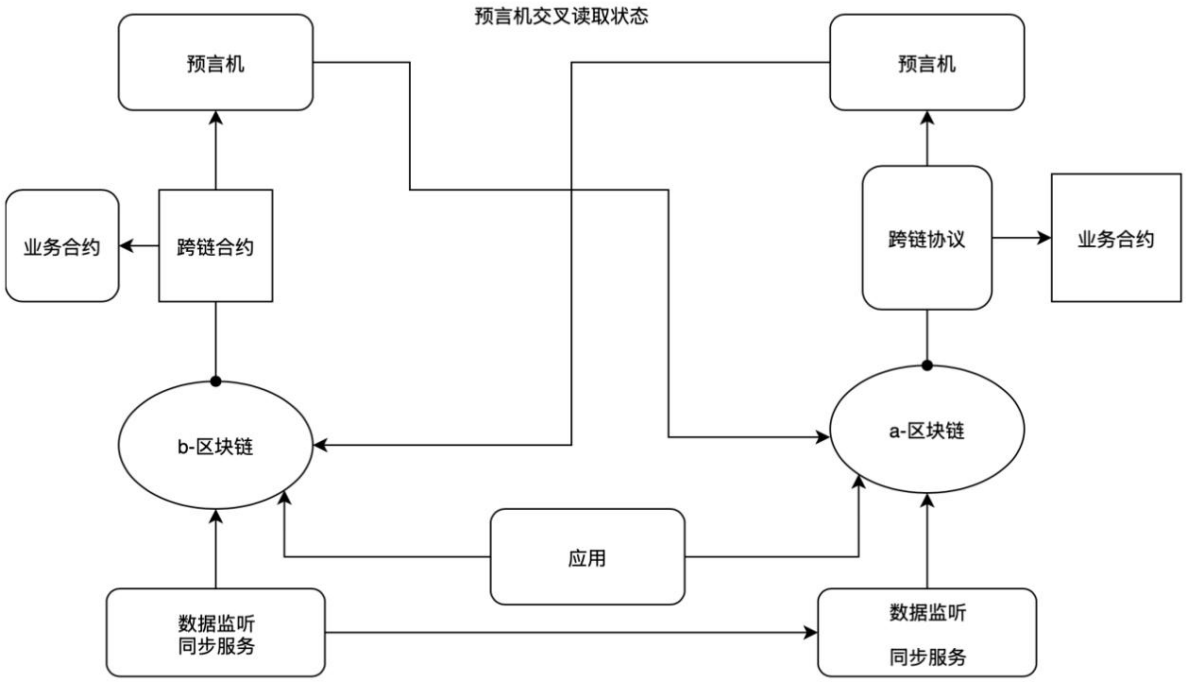
为了在不同的区块链网络中能够进行消息同步，我们需要做如下的技术假定：

技术假设：

- 各方使用统一的DID分布式数字身份，以保证同步到不同区块链中的数据能指向唯一的数据实体

- 用户可从任意底链进行接入，底层记录的数据身份为did，用户在不同的区块链上的操作能被唯一人口

- 如果是可操作的数据，每次操作都做一次对方链的状态检查，状态一致后才能做下一次操作



3.3 数据交换与存证层

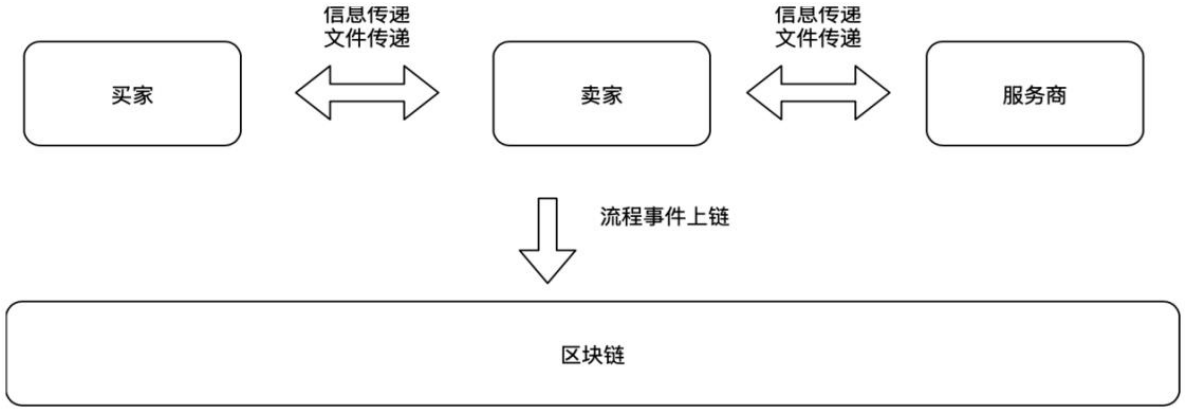
随着电子商务的繁荣发展，运输智能化和物流自动化程度的提高以及人工智能（AI）和大数据的预测性分析能力等方面的进步，行业应用所产生的数据量呈爆炸性增长，与此同时，数据平台架构更加趋于复杂化。因此，对全球供应链数据的质量、时效和准确性的要求已成为贸易物流领域不可或缺的标准要求。

当前，供应链数据流仍然围绕传统纸质文档过程而建立，不能提供准确、及时和真实来源的数据。为了满足不同政府机构和相关业务领域的多方要求，供应链数据流在与各方共享的过程中通常会被重复解释甚至修改，这就导致了贸易流通中的数据缺乏准确、完整的信息。

本协议参考 UN/CEFACT《业务需求规范 数据管道载体 管道数据交换架构》，并根据我国对国外标准转化规定，结合当前实际需要而制定。本协议层主要解决的是商家之间进行信息交换之间的规定。

3.3.1 业务场景

在数据交换业务中有我们主要考虑这三类业务场景，一是用户通过去中心化网络进行商品和服务信息发布，二是支持双方通过点对点的信息交互，三是数据的交换过程可信。



流程示例，假设有一个国际贸易场景，其中进口商（Importer）需要向出口商（Exporter）系统订货，并且双方还有一些凭证文件需要发送。

流程说明:

1.数据交换流程

a. 发布业务场景：各方在发布商品的时候，将商品订阅接口发布到区块链。

b. 业务预定场景：进口商通过出口商提供的 api 进行进行商品预定。

c. 报文交互场景：由于海关等机构采用 EDI 进行报文交互，各方在批量文件传输中将信息包装成报文进行传输。

2.数据存证流程

a. 数据指纹生成：发送方为发送文件或事件信息生成哈希值（数据指纹）。

b. 数据上链：发送方将数据指纹发送到区块链上，进行存证。

c. 存证查询：通过上传原始文件或原始数据信息验证是否被篡改。

3.链上核验流程

a. 文件存证：出口商将文件的哈希值存储在区块链上。

b. 核验请求：进口商或监管机构请求核验文件的上链状态。

c. 智能合约验证：智能合约自动验证文件的哈希值与链上记录的一致性。

d. 核验结果反馈：系统反馈核验结果，确认文件的真实性和完整性。

3.3.2 数据交换流程设计技术方案

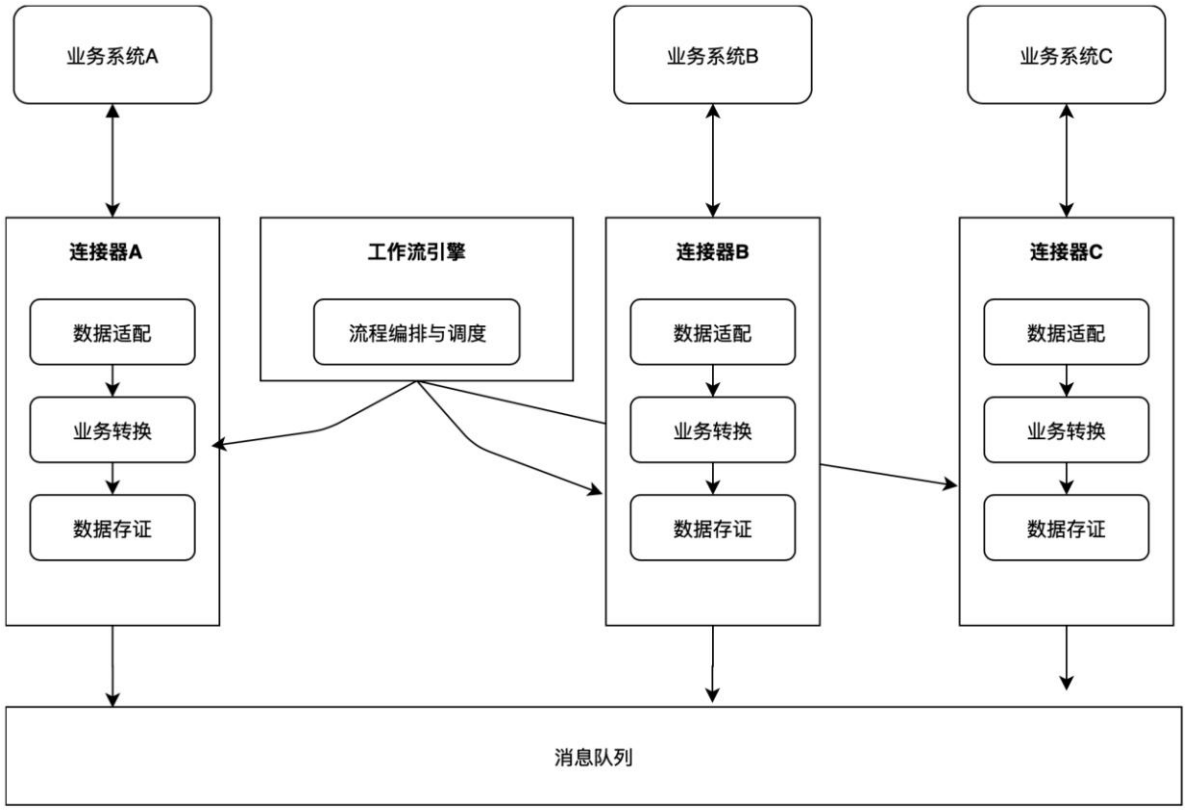
为了解决各参与方业务系统繁多问题，我们提出了通过工作流和连接器的技术设定，通过工作流+连接器的技术方案，参与方可以在不改变现有系统的情况下，像搭积木一样重构业务流程，快速实现业务自动化处理和执行，让业务系统更加平滑的接入到信贸链网络中。

工作流是一种通过计算机技术对业务流程进行抽象建模、自动化执行与监控管理的技术方案其核心是通过预定义规则协调多个参与者，实现文档、信息或任务在业务流程中的自动流转。

连接器是一个应用系统的连接插头，具有区块链、RPA、数据标准转换、通讯方式适配、API 统一管理等能力，帮助各异构系统以零代码方式实现无缝连接，完成自动化数据处理、交换和存证。

数据交换流程设计业务流程图

连接器通过与业务系统适配，帮助不同业务系统完成数据转换与存证，并在工作流引擎的指引下通过与其他连接器的对接完成业务流程的重构。如下图通过工作流的配置，ABC 三个业务系统可以实现业务流程的平滑重构。



连接器架构

解决了企业内外部各个应用系统对接问题，通过连接器你可以像搭积木一样，快速实现业务自动化处理和执行，连接器的架构如下，后续我们也将开源连接器的开发示例。

主要功能包括

序号	功能	描述
1	账号管理	用户在操作账号时，连接器需对对应账号格式和真实性进行校验，以确保数据有效性。连接器在登录对应业务系统时也需要获取账号信息模拟用户登录。本模块包

		括：账号检验、获取账号信息。
2	任务管理	任务是连接器中的输入，连接器外界的指令都是通过任务的形式的发送到连接器。本模块包括：任务管理、执行任务。
3	数据传输	在连接器执行过程中，需要多个连接器互相协作，数据的传输方式为 MQ 队列，连接器需要负责接收和发送数据。
4	数据解析	在整个协作过程中都需要基于传输标准，其业务报文数据需要符合数据模型。
4	心跳连接	主动发送组织连接器任务的心跳，以方便连接器管理系统监控组织连接器任务的执行状态。
5	异常监控	主动发送组织连接器任务的状态和异常信息，以方便连接器管理系统收集组织连接器任务的异常和状态信息
6	数据统计	在连接器启动任务后，主动发送一次组织连接器任务的总协作数量，以方便连接器管理系统统计。
7	可信存证	主动获取组织连接器对应的链上身份信息，以方便后续对数据进行上链时需要 主动发送组织连接器任务对应的数据进行上链，以方便连接器管理系统代理上链后记录为协作记录
8	版本管理	版本管理包括版本控制、版本发布等功能。版本控制可

		以确保版本的可追溯性、可维护性；版本发布是指将产品版本发布给用户使用。

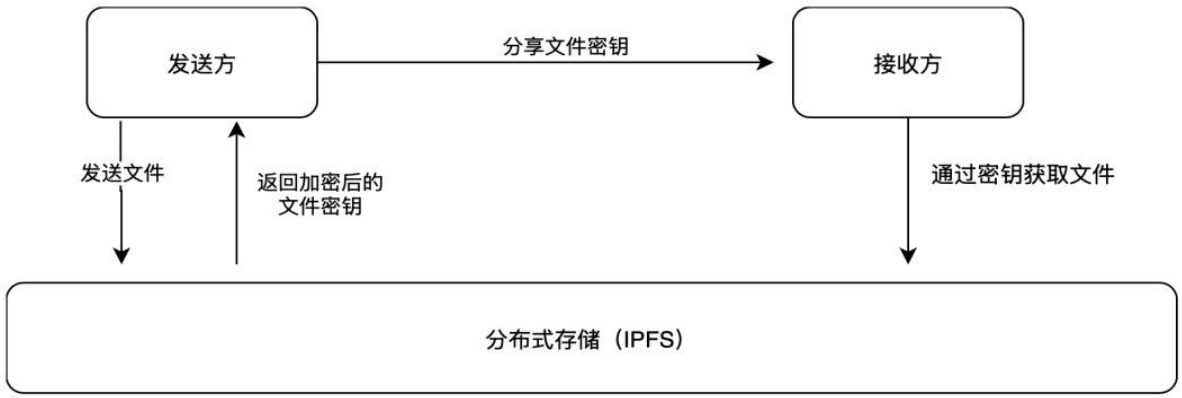
3.3.3 数据交换数据传递技术方案

为了解决各参与方的数据传递问题，我们提出了三种技术方案，分别对应不同的业务场景

1.通过去中心网络进行 1 对多数据交换

这种数据交换方式主要可用于商品信息发布等**“1 对多”**的信息场景，卖家可以通过信贸链网络将自己的商品或服务信息发布到公共网络或特定“圈子”，需求方可以通过搜索获得相关信息，并建立联系，然后开展商业活动。如果各参与可以通过建立 AI 助理来进行商品或服务信息的智能匹配，【详见 **3.5.2 商品和服务的匹配方案**】，如需交互相关的文件信息可通过 IPFS 进行交互。

IPFS 会对每一份数据文件都会进行加密，只有用密钥才能打开进行访问，发送方将文件上传到 IPFS 后会获得一个文件密钥，他可以将文件密钥发送给接收方，接收方拿到密钥后可以去 IPFS 上打开对应的文件。

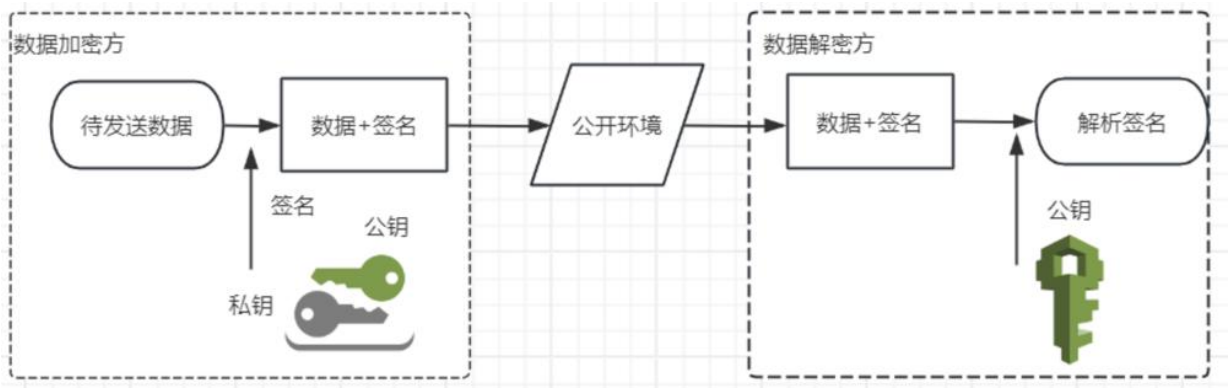


2.通过 API 数据交换点对点进行数据交换

这种数据交换方式主要用于“1 对 1”的特定信息交换场景，在商业活动中为保密需要有些信息仅希望在双方之间传递，因此双方直接提供 api 双方进行对接，对关键步骤进行存证即可。这样业务信息只在双方自己的业务系统中，区块链上仅保存交易事件的哈希，用于证明交易的真实性和不可篡改性。

通过 api 进行对接我们建议各发采用统一的 RESTful API 作为编码规范，使用 did 中的身份密钥进行签名来进行身份认证和验证。

数据交换过程中的我们的身份认证采用非对称加密的方式来进行，技术流程图如下，签名的公私采用用户的数字身份中的公私钥来进行签名。



在公开环境下的数据传输安全我们建议采用 https，HTTPS（Hypertext Transfer Protocol Secure）是一种用于确保数据传输安全的通信协议，它通过在 HTTP 协议上添加 SSL（Secure Sockets Layer）或 TLS（Transport Layer Security）协议来实现加密和认证。

具体的设计思路如下

- **身份验证机制**

Header 中的身份验证信息

在 HTTP 请求的 Header 中加入以下身份验证信息：

Authorization: 用于传递 JWT（JSON Web Token）或 Bearer Token。例如：Authorization: Bearer <token>

Timestamp: 时间戳，用于防止重放攻击。例如：Timestamp: 1631021641000

Nonce: 随机数，每次请求都不同，用于防止重放攻击。例如：Nonce: 1234567890

Signature: 签名，用于验证请求的完整性和真实性。例如：Signature: <signature>

- **签名机制**

签名过程通常包括以下步骤：

- 生成签名：
- 将请求参数、时间戳、随机数（Nonce）和密钥（Secret Key）按照一定顺序拼接。
- 对拼接后的字符串进行哈希加密（如 MD5 或 SHA256）。
- 例如：Signature = HMAC-SHA256(Parameter, SecretKey)
- 验证签名：
- 服务器接收到请求后，使用相同的密钥和算法对请求参数进行签名计算。
- 将计算出的签名与请求中的签名进行比对，如果一致，则请求有效。

- **数据结构示例**

请求 Header 示例

```
GET /api/v1/products HTTP/1.1
Host: example.com
```

```
Authorization: Bearer <token>
Timestamp: 1631021641000
Nonce: 1234567890
Signature: <signature>
```

签名计算示例

```
    const algorithm = 'HmacSHA256';
const secretKey = 'your-secret-key';
const timestamp = '1631021641000';
const nonce = '1234567890';
const data = 'param1=value1&param2=value2'; // 请求参数
const signatureString = `${data}${timestamp}${nonce}${secretKey}`;
const signature = crypto.createHmac(algorithm,
secretKey).update(signatureString).digest('hex');
```

如果数据交换的过程中包含文件可以采用 Base64 编码或二进制编码。

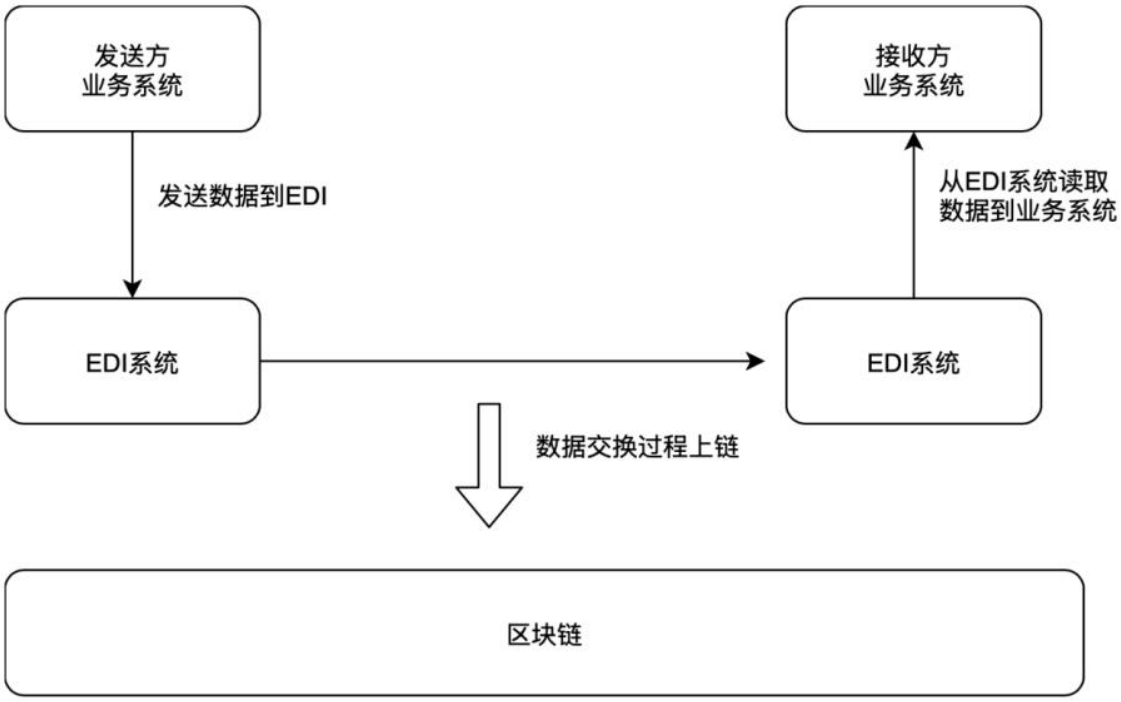
```
    # 假设我们要上传的文件名为 waybill_document.pdf
# 服务器端点为 /api/v1/orders/{orderId}/waybills

curl -X POST "http://example.com/api/v1/orders/{orderId}/waybills" \
     -H "Authorization: Bearer <token>" \
     -H "Content-Type: application/octet-stream" \
     -H "Waybill-Name: waybill_document.pdf" \
```

```
--data-binary "@waybill_document.pdf"
```

3.通过 EDI 数据交换标准进行点对点数据交换

电子数据交换（Electronic data interchange，缩写 EDI）是指按照同一规定的一套通用标准格式，将标准的经济信息通过通信网络传输，它广泛应用于贸易、运输、保险、银行和海关等行业，是一种用一种国际公认的标准格式，EDI 事务对于 B2B 流程至关重要，它们仍然是各种规模企业之间交换文档和事务的首选方式，在本协议中主要使用 EDI 方式来进行**“1 对 1”**的批量文件交换场景。



EDI 支持的网络层协议：

EDI 支持的网传输协议包括安全文件传输协议 (SFTP)、OFTP2、适用性声明 2（AS2，一种基于 HTTPS 的协议）、REST APIs、简单对象访问协议 (SOAP) 、ebXML 等。

EDI 数据交换的数据结构设计:

在设计EDI数据交换的数据结构时，需要遵循EDI标准，如UN/EDIFACT或ANSI X12，这些标准定义了电子商业文档的格式和结构。以下是一个基于UN/EDIFACT标准的EDI数据交换的数据结构示例：

1. EDI报文结构

EDI报文由三个核心部分组成：信封（envelope），段（segment）和数据元素（element）。

信封（Envelope）

EDI文档传输使用三个信封系统容纳交易集：

- **消息信封**：包含单个业务文档的信息。
- **组信封**：包含一组业务文档的信息。
- **交换信封**：包含整个EDI交换的信息。

段（Segment）

EDI事务集中的段（segment）由一组相似的数据元素（element）组成。每个段都有一个特定的标识符，如ST、BEG、N1等，用于指明随从的数据元素的类型。

数据元素（Element）

数据元素（element）是EDI事务集中的数据，例如，购买的商品、数量等。

2. 数据元素（Data Elements）

数据元素是EDI文件中最小的数据单位，例如公司名称、物品名称、数量、价格等。数据元素被区分为数字型、文本型和日期型，其定义包含数据类型、最小长度/最大长度、可用的代码值等。

3. 段（Segments）

段是一组相关的数据元素。例如，在采购订单中，一个段可能包含所有与买方相关的信息，如买方的识别号、名称和地址。

4. 复合数据元（Composite Data Elements）

复合数据元是由其他数据元组成的，其功能更强，包含的信息量更多。例如，一个复合数据元可能包含日期和时间信息。

5. EDIFACT 标准结构

EDIFACT 标准包括一系列涉及电子数据交换的标准、指南和规则，包括应用级语法规则、报文设计指南、数据元目录、代码目录、复合数据元目录、段目录和标准报文目录。

6. 示例：采购订单（ORDERS）

以下是一个简化的 EDIFACT ORDERS 报文示例，展示了如何结构化采购订单数据：

```
    UNH+1+ORDERS:D:96A:UN:EAN008'
BGM+380+281-2+++0+++123456'
DTM+137:20022120:102'
RFF+ON:12345678'
NAD+BY++Party A++++++'
NAD+SE++Party B++++++'
LIN+1+++1234567890123'
IMD+A+++Product A'
QTY+47:200'
PRI+AAA:1000'
UNS+S'
MOA+4:1000'
ALC+C+BAS:100'
MOA+4:200'
```

```
UNT+14+123456'
```

在这个示例中：

- UNH 是消息头，标识消息的开始。
- BGM 是业务分组消息，包含消息类型和相关参考号。
- DTM 是日期/时间/周期消息，包含日期信息。
- RFF 是参考消息，包含参考号码。
- NAD 是名称和地址消息，包含买卖双方的信息。
- LIN 是行项目消息，包含订单中的商品信息。
- IMD 是商品详情消息，包含商品描述。
- QTY 是数量消息，包含商品数量。
- PRI 是价格消息，包含商品价格。
- UNS 是段分隔消息，标识段的结束。
- MOA 是附加金额消息，包含附加费用信息。
- UNT 是消息尾，标识消息的结束。

3.3.4 数据存证技术方案

在分布式数据交互中通过其去中区块链其去心化和不可篡改的特性，为数据提供了一个安全可靠的存证解决方案。具体来说，区块链存证涉及以下几个关键步骤：

1. **哈希值存证：**将需要存证的数字文件计算出唯一的哈希值，并将该哈希值记录在区块链上，以此证明文件在某个时间点之前已经存在，并且内容未被篡改。

2. **利用区块链时间戳：**区块链作为一个分布式的公共账本，每个新区块都会获得一个时间戳，将数据记录在区块链上就相当于为其添加了一个公开可信的时间戳。

3. **数据存证模式：**应用将关键的业务凭证信息（如合同、支付凭证等关键业务信息）写入区块链，应用保存区块链交易 hash 存根，将区块链视为存证系统。当有举证需求时，根据区块链的交易 hash 查询之前存证的原始数据以证明业务凭证信息的存在性。

4. **数据存取模式：**应用将关键的业务数据写入区块链，多个参与方可根据业务数据属性在链上准实时查询检索业务数据，完成基于共享的可信的数据账本的分布式业务协作。

（1）计算数据哈希

可以采用 sha-256 或 SM3 等算法进行数据的哈希存证，示例如下

```
    package main


import (
    "crypto/sha256"
    "fmt"
    "encoding/hex"
)


func main() {
    // 需要存证的数据
    data := "block_header_data_and_transactions"
```

```
    // 计算哈希值
    hash := sha256.Sum256([]byte(data))

    // 将哈希值转换为十六进制字符串
    hashString := hex.EncodeToString(hash[:])

    fmt.Printf("Calculated Hash: %s\n", hashString)
}
```

（2）事件存证智能合约

需要进行事件存证的时候可以调用该合约，合约示例如下，可根据实际业务进行调整

```
    pragma solidity ^0.8.0;

contract EventNotarization {
    // 定义一个事件结构体，用于存储事件详情
    struct Event {
        uint256 id; // 事件的唯一标识符
        address creator; // 事件的发起人地址
        string originatingSystem; // 事件的发起系统
        string details; // 事件的详细信息
        uint256 timestamp; // 事件发生的时间戳
    }
```

```
    // 事件数组，用于存储所有事件
    Event[] public events;

    // 事件计数器，用于生成唯一 ID
    uint256 public nextId;

    // 事件被记录的事件日志
    event EventRecorded(uint256 id, address indexed creator, string
originatingSystem, string details, uint256 timestamp);

    // 用于记录新事件的函数
    function recordEvent(string memory _originatingSystem, string memory
_details) public {
        // 创建一个新的事件实例
        Event memory newEvent = Event({
            id: nextId,
            creator: msg.sender, // 获取调用者的地址作为事件的发起人
            originatingSystem: _originatingSystem, // 事件的发起系统
            details: _details, // 事件的详细信息
            timestamp: block.timestamp // 获取当前的区块时间戳作为事件发生时间
        });

        // 触发事件记录日志
        emit EventRecorded(newEvent.id, newEvent.creator,
newEvent.originatingSystem, newEvent.details, newEvent.timestamp);

        // 将新事件添加到数组中
```

```
        events.push(newEvent);

        // 增加 ID 计数器
        nextId++;
    }

    // 获取所有事件的函数
    function getEvents() public view returns (Event[] memory) {
        return events;
    }

    // 获取特定事件的函数
    function getEvent(uint256 _id) public view returns (Event memory) {
        require(_id < nextId, "Event ID does not exist.");
        return events[_id];
    }
}
```

1. **Event 结构体：**增加了 string originatingSystem 字段来存储事件的发起系统。
2. **recordEvent 函数：**现在接受两个参数，_originatingSystem 和_details，分别用于存储事件的发起系统和详细信息。
3. **EventRecorded 事件日志：**更新了事件日志，包括了 string originatingSystem，以便在前端过滤器中使用。
4. **getEvent 函数：**保持不变，用于根据 ID 检索特定事件。

（3）文件存证智能合约

需要进行文件存证的时候可以调用该合约，合约示例如下，可根据实际业务进行调整

```
    pragma solidity ^0.8.0;

contract FileNotarization {
    // 定义一个文件结构体，用于存储文件的哈希值和相关信息
    struct File {
        uint256 id; // 文件的唯一标识符
        address creator; // 文件的存证发起人地址
        string hash; // 文件的哈希值
        string originatingSystem; // 文件的发起系统
        uint256 timestamp; // 文件存证的时间戳
    }

    // 文件数组，用于存储所有文件的哈希值
    File[] public files;

    // 文件计数器，用于生成唯一 ID
    uint256 public nextId;

    // 文件存证的事件日志
    event FileNotarized(uint256 id, address indexed creator, string hash, string originatingSystem, uint256 timestamp);

    // 用于记录文件哈希的函数
```

```
    function notarizeFile(string memory _hash, string memory _originatingSystem)
public {
        // 创建一个新的文件实例
        File memory newFile = File({
            id: nextId,
            creator: msg.sender, // 获取调用者的地址作为文件的存证发起人
            hash: _hash, // 文件的哈希值
            originatingSystem: _originatingSystem, // 文件的发起系统
            timestamp: block.timestamp // 获取当前的区块时间戳作为文件存证时间
        });

        // 触发文件存证日志
        emit FileNotarized(newFile.id, newFile.creator, newFile.hash,
newFile.originatingSystem, newFile.timestamp);

        // 将新文件添加到数组中
        files.push(newFile);

        // 增加 ID 计数器
        nextId++;
    }

    // 获取所有文件的函数
    function getFiles() public view returns (File[] memory) {
        return files;
    }
```

```
    // 获取特定文件的函数
    function getFile(uint256 _id) public view returns (File memory) {
        require(_id < nextId, "File ID does not exist.");
        return files[_id];
    }
}
```

File 结构体：定义了文件的基本属性，包括 ID、存证发起人地址、哈希值、发起系统和时间戳。

files 数组：存储所有记录的文件哈希值。

nextId：用于生成每个文件的唯一 ID。

FileNotarized 事件日志：当新文件哈希被记录时，触发此日志。

notarizeFile 函数：允许用户记录一个新的文件哈希。它创建一个新的文件实例，并将其添加到文件数组中。

getFiles 函数：返回所有文件的列表。

getFile 函数：根据 ID 返回特定的文件。

（4）文件存证接口

方法：文件上链

方法描述：对贸易单证进行上链操作

方法名：save

参数名称	参数类型	是否必填	描述
hash	string	是	文件内容哈希
name	string	否	文件名称
business_id	string	是	业务 id
initiator	string	是	上链发起者
category	string	是	类别
type	string	否	单证类型
extend	object	否	扩展
auth_time	string	否	时间

（5）事件存证接口

方法：事件上链

方法描述：对事件进行上链操作

方法名：save

参数名称	参数类型	是否必填	描述
hash	string	是	事件内容哈希
name	string	否	事件名称

initiator	string	是	上链发起者
category	string	是	类别
type	string	是	事件类型
trigger_time	string	是	事件发生时间
extend	object	否	扩展

（6）存证核验接口

接口信息

基本信息

- 接口名称：Blockchain Evidence Verification
- 接口地址：../verify
- 请求方式：POST
- 接口版本：1.0

接口描述

此接口接受一个数据哈希值和一个区块链标识符，返回该数据是否已在指定区块链上存证的结果。

输入参数

参数名称	类型	必填	描述

dataHash	string	是	需要核验的数据的哈希值，对原始数据采用存入前相同的加密算法进行哈希加密
blockchainId	string	是	存证所在的区块链的唯一标识符

输出参数

参数名称	类型	描述
status	string	核验结果状态，verified 表示核验成功，unverified 表示核验失败
message	string	核验结果的描述信息
evidenceDetails	object	存证详情，仅在核验成功时返回

存证详情对象（evidenceDetails）

参数名称	类型	描述
blockNumber	integer	存证所在的区块编号

transactionId	string	存证的交易 ID
timestamp	datetime	存证的时间戳

请求示例

请求体

```
    {
  "dataHash":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
  "blockchainId": "mainnet"
}
```

cURL 命令

```
    curl -X POST https://api.blockchain.com/verify \
     -H 'Content-Type: application/json' \
     -d '{
           "dataHash":
"e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855",
           "blockchainId": "mainnet"
         }'
```

响应示例

成功响应

```
    {
  "status": "verified",
  "message": "The data hash has been successfully verified on the blockchain.",
  "evidenceDetails": {
    "blockNumber": 123456,
    "transactionId":
"0x123456789abcdef0123456789abcdef0123456789abcdef0123456789abcdef0",
    "timestamp": "2024-04-01T12:00:00Z"
  }
}
```

失败响应

```
    {
  "status": "unverified",
  "message": "The data hash could not be found on the blockchain."
}
```

3.4 数据标准转换层

3.4.1 通用贸易数据标准规范

通用贸易数据标准规范是在兼容 UN/CEFACT、WCO、单一窗口等标准的基础上制定，涵盖贸易单证分类、数据元素目录、代码表等标准化。目前已形成 100 多种贸易相关单证的通用贸易数据标准规范。这些规范有助于简化数据交换过程，降低数据整合成本，提升数据质量和准确性。

- 贸易单证分类兼容 UN/CEFACT 标准，包括采购订单、发票、报关单等单证的类型划分及定义。以下为部分内容

分类	单证名称	用途
商业单证（Commercial Documents）	商业发票（Commercial Invoice）	记录货物的详细信息，是卖方要求付款的凭证。
商业单证（Commercial Documents）	形式发票（Proforma Invoice）	用于报价或作为销售合同的一部分，提前告知买方商品的详情及费用。
商业单证（Commercial Documents）	装箱单（Packing List）	详细记录每个包装单位内的商品数量、重量和包装情况，便于运输和验收。
商业单证（Commercial Documents）	重量单（Weight List）	记录货物的毛重和净重，用于海关和运输参考。
运输单证（Transport Documents）	提单（Bill of Lading, B/L）	承运人签发的文件，证明货物已装船或已接收，是货物所有权的凭证。
运输单证（Transport Documents）	航空运单（Air Waybill, AWB）	用于空运货物的运输凭证。
运输单证（Transport Documents）	铁路运单	用于铁路运输的文件。

运输单证（Transport Documents）	公路运单	用于公路运输的文件。
运输单证（Transport Documents）	装船通知（Shipping Advice）	通知买方货物已装船的文件。
保险单证（Insurance Documents）	保险单（Insurance Policy）	保险公司出具的文件，证明货物已投保，保障货物在运输过程中的风险。
保险单证（Insurance Documents）	保险凭证（Insurance Certificate）	与保险单类似，但格式更简洁。
金融单证（Financial Documents）	信用证（Letter of Credit, L/C）	银行出具的保证在满足一定条件时支付货款的文件。
金融单证（Financial Documents）	汇票（Draft）	卖方要求买方付款的票据。
金融单证（Financial Documents）	本票（Promissory Note）	由出票人承诺在到期时无条件支付一定金额的票据。
海关及监管单证（Customs and Regulatory Documents）	出口申报单（Export Declaration）	出口商向海关提交的文件，用于申报出口货物。
海关及监管单证（Customs and Regulatory	进口申报单（Import Declaration）	进口商向海关提交的文件，用于申报进口货物。

Documents）		
海关及监管单证（Customs and Regulatory Documents）	原产地证书（Certificate of Origin）	证明货物原产地的文件，用于关税优惠等政策。
海关及监管单证（Customs and Regulatory Documents）	出口许可证	出口国政府签发的允许出口特定商品的文件。
海关及监管单证（Customs and Regulatory Documents）	进口许可证	进口国政府签发的允许进口特定商品的文件。
检验检疫单证（Inspection and Quarantine Documents）	质检证书（Inspection Certificate）	由第三方检验机构出具的文件，证明货物符合特定质量标准。
检验检疫单证（Inspection and Quarantine Documents）	装运前检验证书（Pre-shipment Inspection Certificate）	证明货物在装运前符合合同规定的标准和条件。
其他单证（Other Documents）	合同（Contract）	买卖双方签订的文件，明确交易条款和条件。
其他单证（Other Documents）	危险货物运输文件（Dangerous Goods Note）	用于运输危险货物的文件。

- 数据元素目录基于 ICC 国际商会关于关键贸易文件和数据元素的数字标准倡议（KTDDE）并结合单一窗口标准进行扩展，该倡议提供了一个关键贸易数据词汇表，包含遵循 MLETER 所需的关键数据元素。以下为部分内容

数据元素	唯一标识	描述
文档标识符	1004	识别特定文件的参考编号
预订参考编号	1016	承运人或其代理人指定的参考编号，用于识别特定货物，例如装货前预留货位时的预订参考编号
采购订单编号	1022	买家分配给订单的标识符
分运单文件标识符	1039	识别分运单的参考编号
海关申报文件，交易商指定	1097	交易员指定的用于识别申报的参考
危险货物申报标识	1115	识别危险货物申报单的参考编号
跟单信用证标识符	1172	识别跟单信用证的参考编号
运输合同文件/提单号码	1188	用于识别运输合同证明文件的参考编号
唯一托运编号	1202	识别特定一批货物的唯一参考

(UCR)		
合同编号	1296	买卖双方之间订立的合同标识符
发票号码	1334	用于识别发票的参考编号
货物申报编号	1426	海关指定或接受的用于识别货物申报的参考编号
货运代理参考编号	1460	货运代理指定的用于识别特定货物的参考编号
商品项目序列标识符	1496	区分一批货物中特定货物的序列号
EORI 编号	1R01	经济运营商注册和识别号码是从事欧盟货物进出口业务的企业的欧盟注册和识别号码
批准文号	1R02	机构批准文号（屠宰场、生产工厂、商店）
原产地证号码	1R03	识别原产地证书的参考编号
保单号码	1R04	用于识别保险单的参考号码
许可证/证书/单证册号码	1R05	相关发证机构为每份文件分配的唯一编号
日期		
签发日期	2007	文件签发日期，并酌情签署或以其他方式认证

订购日期	2011	订单日期
实际抵达日期	2107	运输工具到达的日期及/或时间。
最晚取货日期	2131	承运人可以提取一批货物的最晚日期和可选时间。
交货日期	2138	卖方和买方之间约定的将商品交付给买方的日期和时间（可选）
发货日期	2170	货物或预计发货或托运的日期和时间（可选）
预计出发时间	2195	交通工具预计出发的日期和/或时间
跟单信用证单据有效期结束日期	2211	信用证有效期届满的日期
跟单信用证单据签发日期	2237	信用证签发日期
实际出发日期	2281	交通工具出发的日期及或时间
货物装运日期	2347	货物即将或已经装载到运输工具上的日期和时间（可选）
预计到达时间	2349	运输工具预计到达的日期和/或时间
发票日期	2377	发票开具日期
付款到期日	2480	根据付款条款应向债权人提供到期金额的日期

许可证/证书/单证册到期日期	2D01	许可证/证书或单证册的有效期到期日期
当事人、地址、地点、国家		
买方	3002	商品或服务的销售对象。
仓库存管员	3004	一方将货物存放在仓库
收讫人	3006	发票开具给的一方。
卖方银行	3012	卖方指定接收付款的银行。
进口商	3020	办理进口申报的一方（或由海关清关代理或其他授权人员代为办理进口申报的一方）。这可能包括拥有货物的人或货物托运人
仓库管理员	3022	对入库货物负责的一方
出口商	3030	办理出口申报或由他人代为办理出口申报的当事人，且在申报被接受时为货物所有人或对货物拥有类似处置权利的当事人
托运路线	3050	货物运输所用的路线
目的地海关	3086	货物从海关过境制度放行的海关办事处
海关入境处	3088	货物进入目的地关境的海关

过境海关	3106	负责办理过境手续的海关
承运人（运输服务提供商）	3126	在指定地点之间提供货物运输的一方
收货人	3132	货物托运人
受保人	3136	受益于保险的一方。例如，在运输中，这通常是托运人。
发货至/交货方	3144	如果收货人与货物应交付给的一方不一致，例如集装箱将要放置或已经放置的地点。
仓库	3156	存放特定货物的仓库
货运代理	3170	承运货物的一方
运输设备操作员	3174	拥有、运营或控制运输设备的一方，例如集装箱
通知方	3180	被通知方
代理人	3196	授权代表
信用证申请人	3198	信用证申请人的名称和地址
信用证申请人代理银行	3234	除开证行以外，代表跟单信用证申请人行事的银行的名称和地址
可用跟单信用证	3242	可开具跟单信用证的银行名称和地址

的银行。		
跟单信用证受益人	3260	信用证受益人的名称和地址
发货/取货方	3282	货物将由承运人接收或已由承运人接收的地点，例如提货地点
跟单信用证付款人。	3290	信用证中任何汇票的开具人的名称和地址
托运人	3336	订货方按照运输合同规定托运货物
卖方	3346	向买方出售商品或服务的一方
跟单信用证偿付银行	3350	开证行指定的偿付银行的名称和地址
收款人	3370	应付款或已付款的一方
买方银行	3420	买方委托的付款银行
运费付款人	3470	运费支付责任方的姓名和地址
卖家的银行账号	3492	指定接收付款的银行账户标识符
保险公司	3P01	从事保险业务的公司详情
出票人/出票人	3P02	签发汇票的单位或个人（付款人）
许可证/证书/单证	3P03	签发许可证/证书/单证册的机构名称、签名和印

册签发者		章
管理/主管部门	3P04	管理机构/主管机关的名称、地址、国家印章/图章和所在国家
预先裁定申请人	3P05	申请预裁定的进口商、出口商、生产商或任何有正当理由的人的名称和地址
预先裁定申请接收人	3P06	负责签发预裁定的海关当局或指定机构
付款人	3P07	负有按照汇票规定支付义务的单位或个人（支付请求人）
担保人	3P08	作为消费税担保人的金融机构、保险公司或经批准的实体
消费税担保受益人	3P09	负责征收消费税的税务机关
位置		
货物交付地点（由承运人）	3000	货物交付给最终收货人的地点
目的地国家	3014	货物即将或已经送达的国家
紧急联系方式	3058	发生危险品紧急情况时联系人的姓名
货物出境海关办	3096	货物离境或准备离境的海关

公室		
原始装货地点	3099	海港、机场、货运站、火车站或货物首次装载到运输工具上的其他地点
付款地点	3108	付款地点或应付款地点的名称
海关担保处	3110	为过境货物的运输提供担保或保证的海关
发货	3150	发货地点或发货地点
帐户持有人	3192	账户持有人
地点或出发地	3214	运输工具计划出发或已经出发的港口、机场或其他类型的地点
货物目的地国家	3216	货物将交付给最终收货人或买方的国家名称
出口国	3220	货物最初从该国出口到进口国，中间没有发生任何商业交易。同义词：发货国。发货国：货物在关税同盟国家之间发货的国家
原产国	3238	根据海关关税或数量限制或任何与贸易有关的措施所规定的标准，生产或制造货物的国家名称
到达地点	3258	运输工具预计到达或已经到达的港口、机场或其他类型的地点
接收货物的地点	3348	承运人接收货物的地点

（承运人向托运人或其代理人接收货物的地点）		
基地港卸货地点	3356	按照运输合同规定从运输工具上卸下货物的地点或港口。货物可以在该地点或港口从主要运输工具上卸下，也可以不卸下
保险理赔员	3360	保险理赔员的姓名和地址
目的站名称	3392	海港、机场、货运站、火车站或其他从运输工具上卸下货物的地点
签发地点	3410	文件签发地点，以及在适当情况下签署或以其他方式认证的地点
转口国家样本	3L01	标本在进入本 CITES 文件签发国之前从哪个国家重新出口
样本原产国	3L02	标本采集自野外、圈养或人工培育的国家，但不再符合 CITES 规定豁免条件的植物标本除外
中转地点	3L03	此次物流运输运动的过境活动
条款、条件、指示		
运输合同文件条件	4002	参考印刷在文件上或单独提供的承运商的运输条件

操作说明	4078	一组处理说明的自由形式描述。例如应如何处理指定的货物、包裹或运输设备（集装箱）
加载说明	4080	关于在何处或如何将特定包裹或集装箱装载到运输工具上的指示
保险条件	4112	参考签发保险凭证的一般合同条款，和/或与所涉货物有关的具体条款措辞
认证	4192	官方认证、合法化或印章等的文本。
交易性质	4422	合同类型的文本表示
验证	4426	证明文件已经过认证的证明，在适当情况下注明认证方
送货说明	4492	有关货物的交货说明
CITES 附录参考	4C01	该物种被列入 CITES 公约附录编号（I、II 或 III）
安全信息	4C02	与货物相关的安全信息说明
担保条件	4C03	应支付消费税担保的情况
条款		
贸易条款条件描述	4052	自由形式的交货或运输条款描述（国际贸易术语）

贸易条款条件代码	4053	指定交货或运输条款的代码（国际贸易术语）
付款期限	4277	确定交易双方的付款条件（通用术语）
运输担保类型	4376	以现金、债券或书面担保形式做出的承诺的详细信息，以确保履行义务，例如根据过境程序
文件背书	4428	文件已签署的证明
付款方式	4467	指定付款方式的代码
CITES贸易交易的目的	4T01	T：商业，Z：动物园，G：植物园，Q：马戏团或巡回展览，S：科学，H：狩猎战利品，P：个人，M：医疗，E：教育，N：重新引入或引入野外，B：圈养繁殖或人工繁殖，L：执法/司法/法医

序号	英文名称	中文名称	标准的性质
1	Codes for representation of names of countries	国家名称的代码表示	建议书3号（GB/T 2659-2000）
2	Abbreviations of INCOTERMS	国际贸易术语解释通则缩写	建议书5号（GB/T 15423-1994）

3	Numerical representation of date, time, and periods of time	日期、时间和时间期限的数字表示	建议书 7 号（GB/T 7408-2005）
4	Alphabetical code for representation of currencies	表示货币的字母代码	建议书 9 号（GB/T 12406-2008）
5	Codes for ship’s name	船舶名称代码	建议书 10 号（GB/T 18366-2001）
6	Simpler shipping marks	简单运输标志	建议书 15 号（GB/T 18131-2000）
7	Codes for ports and other locations	口岸及相关地点代码	建议书 16 号（GB/T 15514-2008）
8	Payterms--Abbreviations for terms of payment	付款条款缩写	建议书 17 号（GB/T 18126-2010）
9	Code for modes of transport	运输方式代码	建议书 19 号（GB/T 6512-1998）
10	Codes for units of measure used in international Trade	国际贸易计量单位代码	建议书 20 号（GB/T 17295-2008）

11	Codes for types of cargo, packages and packaging materials	货物、包装以及包装类型代码	建议书 21 号（GB/T 16472-1996）
12	Freight cost code	运费代码	建议书 23 号（GB/T 17152-2008）
13	Trade and transport status codes	贸易和运输状态代码	建议书 24 号
14	Codes for types of means of transport	运输工具类型代码	建议书 28 号（GB/T 18804-2002）
15		国际贸易方式代码	GB/T 15421-2008
16		国际贸易付款方式分类与代码	GB/T 16962-2010
17		国际贸易合同代码编制规则	GB/T 16963-2010
18		中国及世界主要海运贸易港口代码	GB/T 7407-2008
19		HS 商品编码	
20		EDIFACT 代码表	GB/T 16833-2002

- 代码标准基于UN/CEFACT推出的代码标准化建议书，包括国家名称的代码表示、国际贸易术语解释通则缩写等。以下为部分内容

3.4.2 数据标准转换规则

同一类型贸易单证，由于存在多种不同的数据标准，这些标准在数据元素、代码表等方面存在差异，使得系统间的对接变得复杂。为解决这一问题，数据标准转换层提供一套转换规则，声明不同标准与通用贸易数据标准规范间的映射方式 ，包括对每一数据元素的中英文名称转换、对应代码的转换规则，以简化系统间的对接过程。

- 数据元素转换

对于同一数据元素在不同标准中的表现形式不一样的问题，可以通过数据元素映射将其转化为通用贸易数据标准数据要素，实现不同数据标准中数据元素的语义互操作，以确保在数据交换过程中保持对数据要素的一致性理解。

- 代码表转换

对于国家名称的代码表示等在不同标准中的表现形式不一样的问题，基于UN/CEFACT推出的代码标准化建议书，对不同标准间的代码进行转换。

3.4.3 数据适配器技术方案

数据交换过程通常涉及不同系统之间的数据交互，比如EDI（电子数据交换）或者不同应用程序之间的数据转换。数据适配器的作用是实现不同数据格式、协议或结构之间的转换，确保数据能够正确、高效地在系统间传输。

数据适配器由以下核心组件构成：

- **格式解析器**：解析输入数据格式（如CSV、数据库表、JSON/XML）及生成目标输出标准格式。

- **映射引擎**：执行字段间的转换逻辑，包括数据清洗、格式化和规则应用。
- **模板管理器**：存储和管理数据标准模板及用户自定义模板。
- **验证器**：确保输出数据符合目标格式的结构、类型和业务规则。
- **错误处理与日志**：记录转换过程中的错误并提供修复建议。

具体功能包括：

（1）格式解析与生成

- **输入解析**：
  - **结构化数据**（数据库、ERP）：通过 JDBC/ODBC 连接，使用 SQL 查询提取数据。
  - **半结构化数据**（XML/JSON）：采用 SAX/DOM 解析器（如 Java JAXB、Python xml.etree）。
  - **非结构化数据**（CSV/Excel）：使用 Apache POI 或 Pandas 进行解析。
- **目标格式生成**：
  - **X12/EDIFACT**：基于开源库生成符合标准的段、元素和分隔符。
  - **XML/JSON**：通过模板引擎动态生成，或直接构建 DOM 树。
  - **自定义格式**：用户通过 JSON Schema 或 YAML 定义模板，系统动态解析并填充数据。

示例：

场景：将 ERP 系统的订单（JSON）转换为 X12 850 格式。

输入数据：

```
    {
  "order_id": "1001",
  "customer_id": "CUST-123",
  "items": [
    {"sku": "A100", "qty": 2, "price": 50.0},
    {"sku": "B200", "qty": 1, "price": 30.0}
  ]
}
```

映射规则：

order_id → ST02（事务集控制号）。

items 数组聚合为多个 PO1 段，计算每项总价（qty*price）。

输出 X12：

```
    ST*850*0001
BEG*00*SA*1001**20231001
PO1*1*2*A100**50*EA*50*CP
PO1*2*1*B200**30*EA*30*CP
SE*4*0001
```

（2）数据映射引擎

- 映射规则配置：

- **可视化映射工具**：提供拖拽界面，将源字段与目标字段关联。
- **AI 辅助映射**：基于历史数据训练模型，推荐可能的字段映射关系（如“ERP.地址” → “X12.N4 段”）。
- **规则类型**：
  - **直接映射**：字段一对一复制。
  - **条件映射**：基于逻辑判断（如“若订单金额>1000，则标记为 VIP”）。
  - **计算映射**：执行公式（如总价=单价×数量 + 税费）。
  - **聚合映射**：合并多行数据（如多个订单项汇总为单个订单）。
- **脚本支持**：允许用户嵌入 Python/JavaScript 代码处理复杂逻辑（如地址标准化）。

- **规则存储**：
  - 映射规则以 JSON/YAML 格式存储于数据库，包含字段路径、转换逻辑、条件表达式。
  - 示例规则：

```
    {
  "source": "ERP.Order.Header.OrderDate",
  "target": "X12.850.DTM.02",
  "transform": "datetime_format('%Y-%m-%d', '%Y%m%d')",
  "condition": "source.Status == 'Approved'"
}
```

（3）模板管理

- **模板库：**
  - 预置行业标准模板（如 X12 850 订单、EDIFACT INVOIC）。
  - 用户可上传自定义模板（XML Schema、JSON Schema、X12 段定义）。
- **模板版本控制：**
  - 管理模板变更历史，支持回滚和差异对比。

（4）数据验证

- **结构验证：**
  - XML：通过 XSD 校验；JSON：通过 JSON Schema 校验。
  - X12/EDIFACT：校验段顺序、必填字段、元素长度和类型（如 Numeric、ID）。
- **业务规则验证：**
  - 自定义规则引擎（如 Drools）检查业务逻辑（如“订单号必须唯一”）。
- **错误反馈：**
  - 生成详细错误报告（如“DTM 段缺少日期”），定位到具体字段并提供修复建议。

（5）错误处理与日志

- **错误分级：**

- 致命错误（格式不符）：终止处理并通知用户。
- 警告（数据缺失）：记录日志并填充默认值（如空字段标记为 NULL）。

- **日志存储：**
  - 记录转换过程，支持异常反馈和可视化分析。

3.5 交易过程管理层

交易过程管理是“信贸链协议”的核心功能，作为一个分布式的可信贸易协作网络，买家可通过协议接口查询发布在信贸链上的商品和服务，同时也可以发布自己的需求；卖家通过自身的业务系统将自身的产品和服务信息发布到信贸链；交易双方通过智能合约进行匹配和执行交易，并将交易的过程和结果记录在区块链上。

3.5.1 流程说明

场景描述：

- 圈子管理：贸易主体可以创建圈子并邀请协作方加入圈子，加入圈子的协作方可以发布需求和服务进行交易。
- 需求发布：需求方发布明确的需求信息到圈子，包括产品、服务的详细描述，确保需求方的交易意图得到充分表达，同时提高需求的可见性和匹配效率。
- 供给发布：供给方发布产品、服务等资源的详细信息到链上，包括基础参数，确保供给信息透明公开、易于匹配。
- 供需匹配：系统基于智能合约按照地理位置、时效性、价格区间等多维度规则，智能推荐最佳匹配交易对象，优化供需对接的精准度。

- 发起交易：交易发起方向交易对手方及见证方（可选第三方或平台）发送交易请求，并协商交易细节，包括共识机制、支付方式和智能合约条款，最终达成一致并正式启动交易流程。

- 执行交易：在双方确认交易条件后，基于智能合约进行支付、交付、验收等操作，全程上链记录交易过程，确保操作透明、可追溯，并通过实时监控功能提示交易关键节点，支持异常情况的及时响应。

- 交易完成：交易结束后，所有参与方将交易的过程和结果记录在区块链上，并同步提供给交易双方及授权监管机构，为未来的审计、争议解决和法律依据提供坚实的基础。

为了实现以上场景，需研发相关功能作为支撑：

（1） 圈子管理

提供创建圈子、加入圈子服务以构建协作圈，相关服务的技术实现方案详见【 3.5.2 创建圈子合约】。

（2） 供给发布

提供服务/产品的发布服务，用户可以将服务发布给所有人可见，也可以发布到指定圈子，具体实现智能合约详见技术方案

（3）需求发布

提供需求发布服务，相关服务的技术实现方案详见【3.5.2 发布需求到信贸链】。

（4）供需匹配

提供商品与服务信息检索服务、需求和供给的匹配服务。系统基于智能合约按照地理位置、时效性、价格区间等多维度规则，智能推荐最佳匹配交易对象，优化供需对接的精准度。详见【3.5.2 商品和服务的匹配方案】。

（5）交易执行

交易执行过程采用 BSP 业务协作流程，详见《GB/T 44779-2024 国际贸易业务流程规范》 中的“购买—运输—支付参考数据模型”。

- **主要顶层过程**

  - 订立业务协议(销售)：买方向卖方发送产品或服务的报价请求。卖方回复或主动向潜在的买方提供报价。买方与选定的卖方进行商定，以达成合同协议的条款

  - 订单(销售)：买方确认所需的产品或服务，并根据合同协议发出订单。卖方收到订单并做出响应

  - 运输/边境清关：卖方按规定的交易条款发货。所有运输安排和要求均符合相关主管部门的规定。开具发票(要求付款)，买方收到产品或服务。开具发票(要求付款)遵循 GB/T 36368 和 GB/T 36371 的规定；

  - 支付(销售)：收到付款的请求。根据双方商定的条款，付款人进行支付，收款人接收付款。

- 在交易执行过程，我们将对交易事件进行存证，提供交易过程和交易结果的记录和核验服务。详见【3.5.2 交易执行中的技术方案】。

- 买方向卖方发送产品或服务的报价请求，会生成报价单。买方与选定的卖方根据报价单沟通确认购买意向。报价流程设计使用《GB/T 31874-2015 基于 ebXML》的“运输费用询价与报价”；报价单信息设计采用《 UN/EDIFACT D96A Message -Quotemessage》。

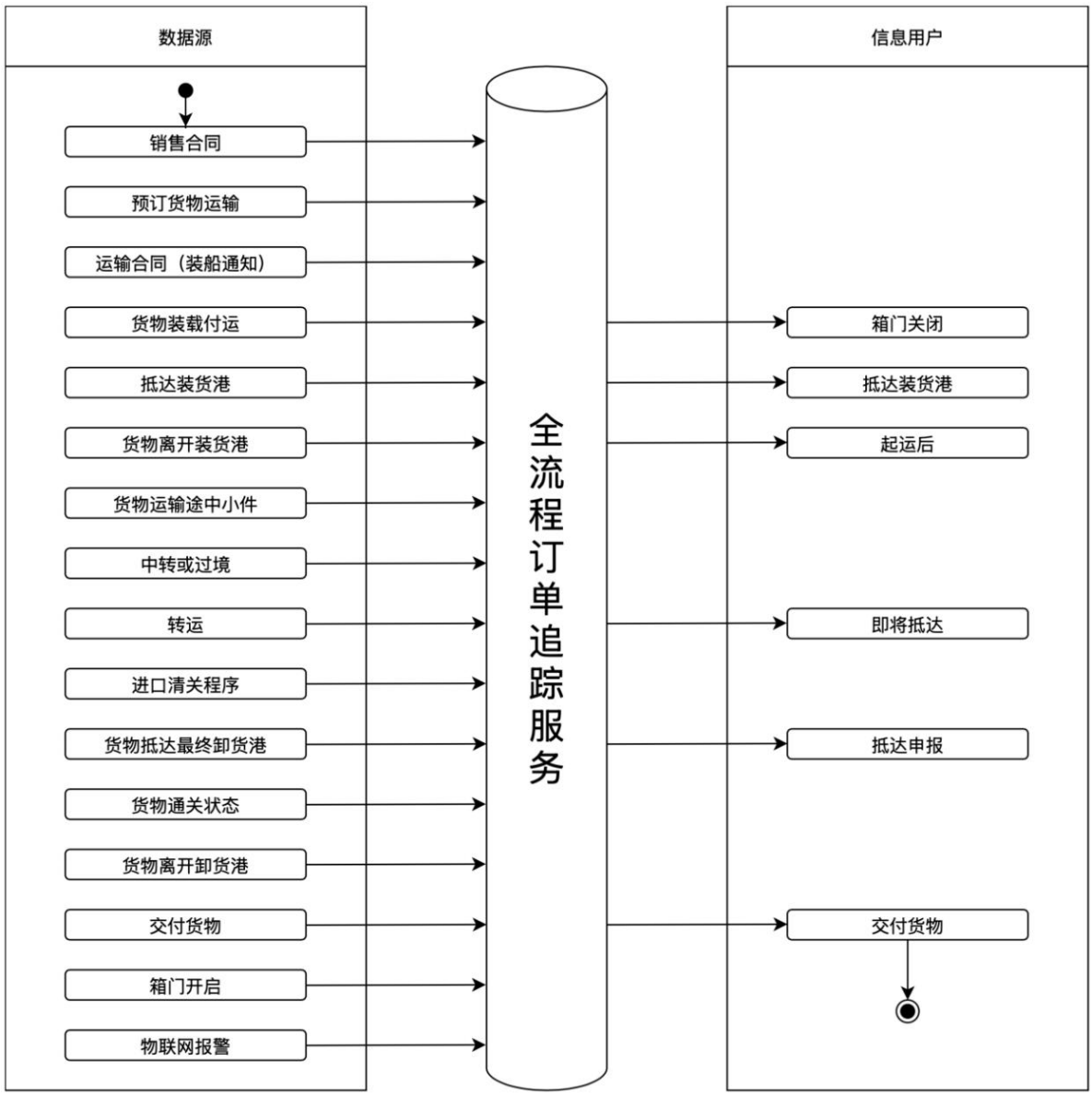
- 买方确认所需的产品或服务后发出订单，卖方收到订单并做出响应，下单后会生成唯一的订单 ID 用于后续运输、支付环节的全流程追踪。下单流程设计采

用《GB/T 39456-2020 国际贸易业务流程规范 订单处理》；订单 ID 的生成规则详见【3.5.5 订单号编码规则】。

（6）交易追踪

提供全流程交易追踪服务，记录和追踪订单全环节相关交易事件。

- 全流程订单追踪服务：连接贸易链条上的进出口贸易商、船公司、航空公司、码头、单一窗口等各参与方节点，通过可信贸易网络完成跨组织跨系统实时数据同步，构建全场景全环节数据链。工作原理是尽早向数据需求者提供来自受信任来源的高级数据。



- 添加数据：可通过连接器向全流程订单追踪服务添加信息。信息可作为完整的文档提交并集成到全流程订单追踪服务中，但文档经常重复在之前数据交换中已提交的信息。在这种情况下，信息可能是冗余的。冗余信息不应覆盖之前提交的信息。

- 信息更新：

  - 通过全流程订单追踪服务获取的数据都应是该时间点上真实的数据，不宜予以修改以免造数据不受信任。

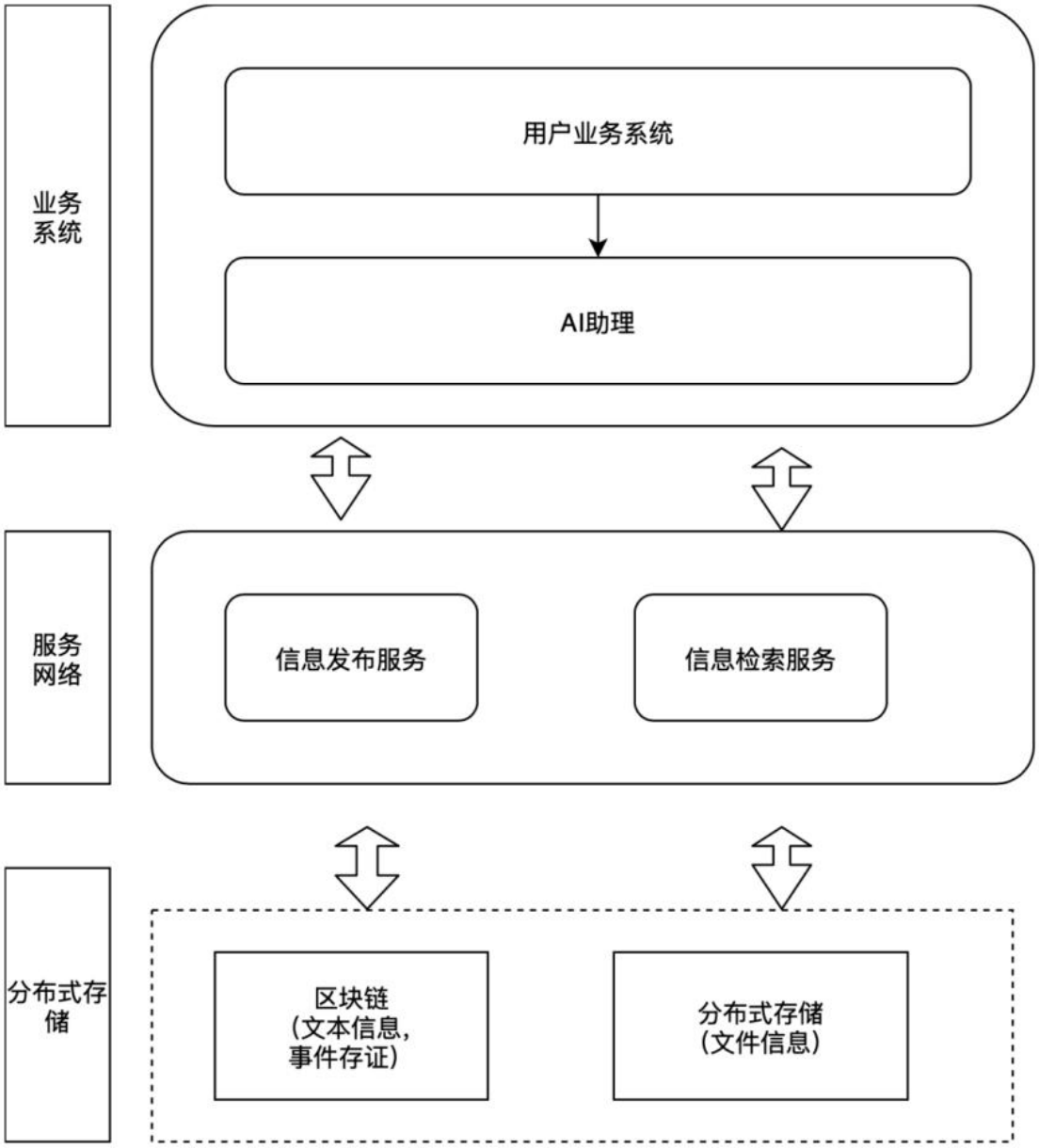
- 只有原始的执行者才能修改对应时间的数据以保持数据完整性。
- 在信息提供后发现数据不正确，则可重新发送信息并更新全流程订单追踪服务的数据，但新数据应带有时间戳并设置标记以表示是事件发生后的更改，以便向查看数据的人提供数据未被篡改的证明。

- 参与方：提供参与方标准相关服务，用于定义在订单全流程中参与方的角色以及对应的事件读写权限
  - 仅向参与对应订单的参与方提供相关的信息数据。
  - 根据参与各方角色来向参与方提供数据。
  - 可根据参与方角色定义事件的读写权限。
  - 相关数据不会暴露参与方身份，除非信息的接收方已获得授权来查看底层数据。
- 事件：提供事件标准相关服务，用于定义订单全流程中可执行的事件。

3.5.2 实现方案

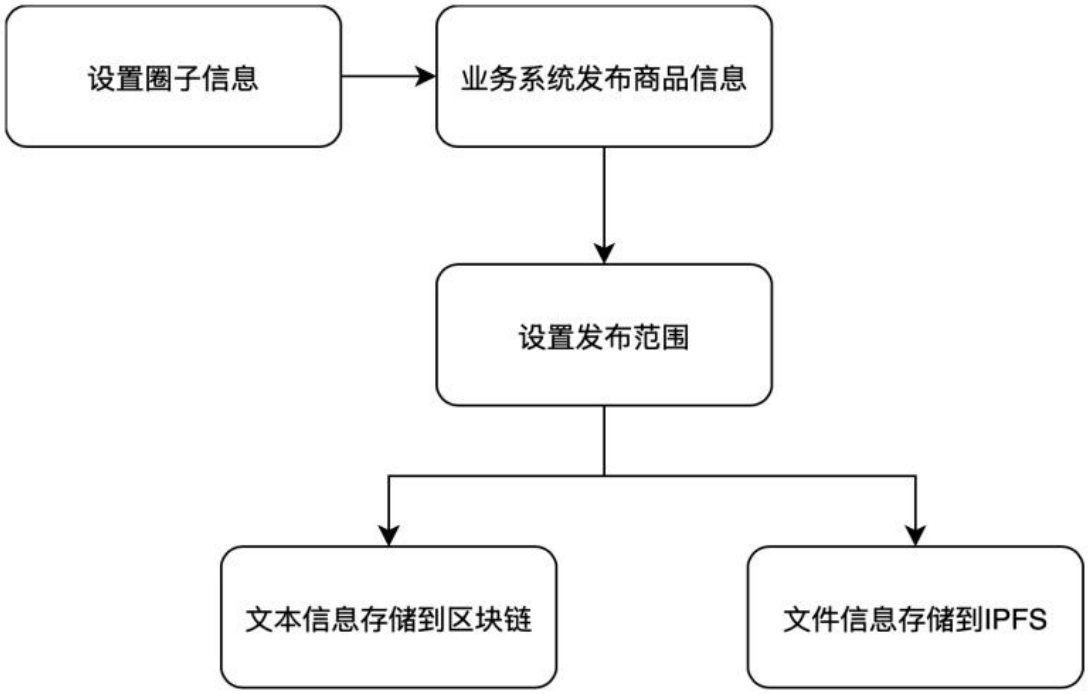
在新型电子商务的构想中，用户是基于自己的业务系统发布的商品和服务信息，商品和服务信息将被发布到分布式网络中，因此在交易过程管理从技术实现上来说主要有这样三部分，第一、商品和服务怎么发布到分布式网络；第二、在分布式网络中如何进行安全的服务信息检索；第三、买卖双方如何进行交易执行并完成交易。

本协议采用人工智能（AI）+区块链+IPFS 的技术方案进行商品和服务信息发布、匹配和交易跟踪，使用区块链来存储文本信息和交易事件，使用 IPFS 来存储文件信息如商品的图片或视频等以便让商品的信息更加丰富，使用 AI 助理来进行交易各方的自动匹配与交易跟踪。



（1）商品和服务发布方案

我们采用智能合约将商品和服务信息发送到区块链，用户可以选择将商品发布给所有人可见，也可以将商品发布给圈子可见。



创建圈子合约

```
    pragma solidity ^0.8.0;

contract CirclesContract {
    struct Circle {
        string id;
        string name;
        address[] members;
    }

    mapping(string => Circle) public circles;

    event CircleCreated(string name);
    event MemberAdded(string name, address member);
    event MemberRemoved(string name, address member);
```

```
    function createCircle(string memory _name) public {
        require(bytes(circles[_name].name).length == 0, "Circle already exists");
        circles[_name] = Circle({name, new address[](0)});
        emit CircleCreated(_name);
    }

    function addMember(string memory _name, address _member) public {
        require(bytes(circles[_name].name).length > 0, "Circle does not exist");
        circles[_name].members.push(_member);
        emit MemberAdded(_name, _member);
    }

    function removeMember(string memory _name, address _member) public {
        require(bytes(circles[_name].name).length > 0, "Circle does not exist");
        uint index = circles[_name].members.indexOf(_member);
        require(index != uint(-1), "Member not found");
        circles[_name].members[index] =
circles[_name].members[circles[_name].members.length - 1];
        circles[_name].members.pop();
        emit MemberRemoved(_name, _member);
    }

    function getCircleMembers(string memory _name) public view returns (address[]
memory) {
        require(bytes(circles[_name].name).length > 0, "Circle does not exist");
        return circles[_name].members;
```

```
    }
}
```

将商品或服务发布到圈子

```
    pragma solidity ^0.8.0;

import "./CirclesContract.sol";

contract MarketplaceContract {
   struct Item {
        uint id;
        string title;
        string description;
        uint price;
        address payable seller;
        bool isService;
        string ipfsHash; // IPFS 附件地址
        string circleName; // 关联的圈子名称
    }

    Item[] public items;

    event ItemAdded(uint id, string title, string description, uint price, address seller, bool isService, string ipfsHash, string circleName);

    function addItem(string memory _title, string memory _description, uint
```

```
_price, bool _isService, string memory _ipfsHash, string memory _circleName)
public {
        require(bytes(_circleName).length > 0, "Circle name must be provided");
        CirclesContract.Circle memory circle =
CirclesContract(circlesContract).circles(_circleName);
        require(circle.members.length > 0, "Circle must have members");
        uint id = items.length + 1;
        Item memory item = Item(id, _title, _description, _price,
payable(msg.sender), _isService, _ipfsHash, _circleName);
        items.push(item);
        emit ItemAdded(id, _title, _description, _price, msg.sender, _isService,
_ipfsHash, _circleName);
    }

    function getItem(uint _id) public view returns (Item memory) {
        Item storage item = items[_id - 1];
        require(item.circleName == "" || isCircleMember(item.circleName,
msg.sender), "Access denied");
        return item;
    }

    function isCircleMember(string memory _circleName, address _member) internal
view returns (bool) {
        CirclesContract circlesContract = CirclesContract(circlesContract);
        address[] memory members = circlesContract.getCircleMembers(_circleName);
        for (uint i = 0; i < members.length; i++) {
            if (members[i] == _member) {
```

```
                return true;
            }
        }
        return false;
    }

    function getAllItems() public view returns (Item[] memory) {
        return items;
    }
}
```

发布需求到信贸链

```
pragma solidity ^0.8.0;

contract DemandBoard {
    // 需求结构体
    struct Demand {
        uint id;
        string title;
        string description;
        address payable poster;
        bool isPublic;
        string[] circleNames;
    }

    // 存储需求的数组
```

```
    Demand[] public demands;

    // 事件声明，用于记录需求的发布
    event DemandPosted(uint id, string title, string description, address poster,
bool isPublic, string[] circleNames);

    // 发布需求
    function postDemand(string memory _title, string memory _description, bool
_isPublic, string[] memory _circleNames) public {
        uint id = demands.length + 1;
        demands.push(Demand(id, _title, _description, payable(msg.sender),
_isPublic, _circleNames));
        emit DemandPosted(id, _title, _description, msg.sender, _isPublic,
_circleNames);
    }

    // 获取特定需求的详细信息
    function getDemand(uint _id) public view returns (Demand memory) {
        Demand storage demand = demands[_id - 1];
        require(demand.isPublic || isCircleMember(demand.circleNames, msg.sender) ||
msg.sender == demand.poster, "Access denied");
        return demand;
    }

    // 检查用户是否在需求发布的圈子中
    function isCircleMember(string[] memory _circleNames, address _member)
private view returns (bool) {
```

```
        for (uint i = 0; i < _circleNames.length; i++) {
            // 这里需要一个外部合约 CirclesContract 来提供圈子成员的验证
            // 假设 CirclesContract 有一个函数 getCircleMembers(string memory)
returns (address[] memory)
            address[] memory members =
CirclesContract.circles(_circleNames[i]).getMembers();
            for (uint j = 0; j < members.length; j++) {
                if (members[j] == _member) {
                    return true;
                }
            }
        }
        return false;
    }

    // 获取所有需求
    function getAllDemands() public view returns (Demand[] memory) {
        return demands;
    }
}

// 假设的圈子合约，用于获取圈子成员
contract CirclesContract {
    mapping(string => address[]) public circles;

    function getMembers(string memory circleName) public view returns (address[]
memory) {
```

```
        return circles[circleName];
    }
}
```

将图文信息上传到 IPFS

```
    const IPFS = require('ipfs-http-client');
const fs = require('fs');

// 连接到本地的 IPFS 节点（默认地址和端口）
const ipfs = IPFS.create({ url: 'http://localhost:5001' });

async function uploadFileToIPFS(filePath) {
  try {
    // 读取文件内容
    const fileContent = fs.readFileSync(filePath);

    // 添加文件到 IPFS
    const added = await ipfs.add(fileContent);

    // 输出文件的 IPFS 哈希地址
    console.log(`File uploaded to IPFS with hash: ${added.path}`);
  } catch (error) {
    console.error('Error uploading file to IPFS:', error);
  }
}
```

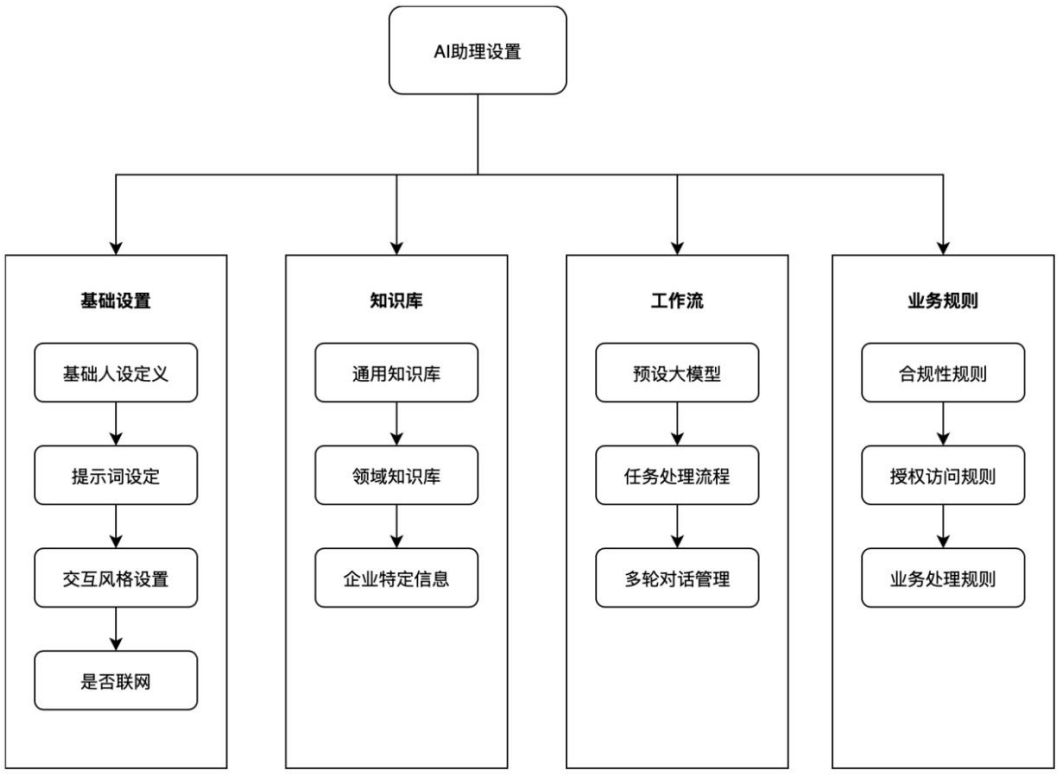
```
// 调用函数，上传文件
// 替换'path/to/your/file'为你的文件路径
uploadFileToIPFS('path/to/your/file');
```

（2） 商品和服务的匹配方案

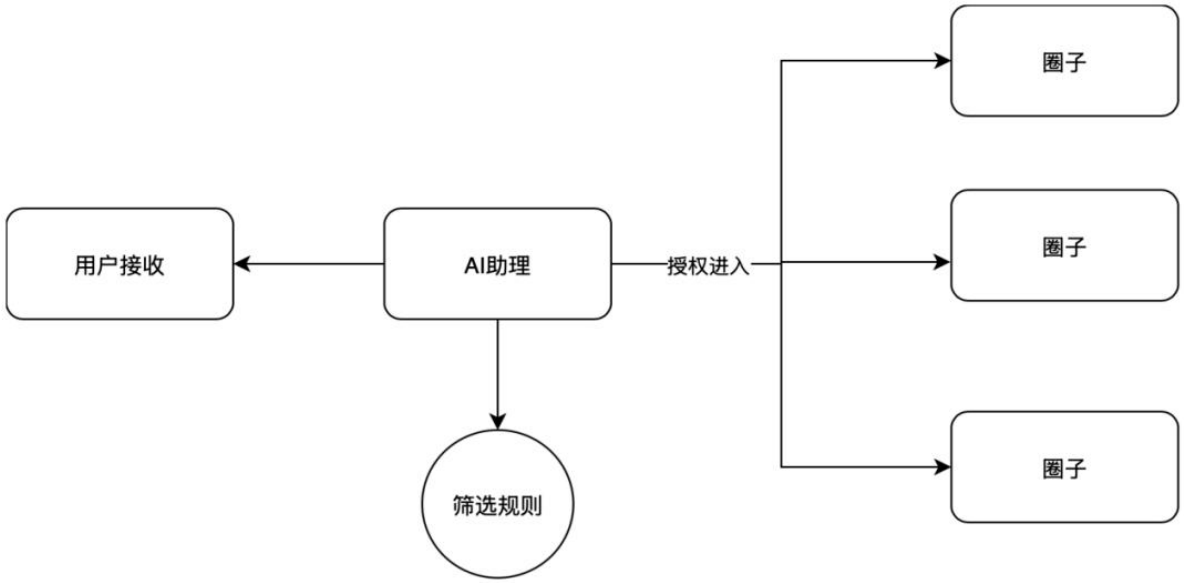
根据 B2B 贸易的特性，在商品和服务的匹配上我们提供了智能检索方案，帮助贸易参与方在大量的信息中筛选出符合自己需求的商品或服务。技术上主要是通过 AI 助理来进行实现，我们可以通过大模型、知识库、提示词工程、工作流、插件等多种技术手段进行对 AI 助手设置，AI 助理再基于按照地理位置、时效性、价格区间等多维度规则，智能推荐最佳匹配交易对象，优化供需对接的精准度。

AI 助理在用户授权后可以进入相应的圈子，查询和收集信息，可以通过聊天实时让 AI 助理搜索在线的商品和服务信息，也可以给 AI 助理制定任务，定时反馈商机信息。

AI 助理的设置



业务处理流程



（3）交易执行中的技术方案

事件权限管理方案

在新型电子商务场景下为了满足同一个交易不同事件用户的参与权限，我们设定如下权限控制办法：

- 以唯一订单号为标识，创建一个事件集合
- 参与到该集合的用户都关联到这个 ID 中
- 为每个用户设定不同的角色权限
- 用户可以根据业务情况继续添加事件
- 访问控制上先判断用户是否在事件集合中，再根据设定角色取得事件的读取/写入/拒绝访问等操作

访问控制合约示例

```
pragma solidity ^0.8.0;

contract DistributedEcommerceAccessControl {
    // 事件结构体
    struct Event {
        string eventName;
        address[] participants;
        mapping(address => Role) roles;
    }

    // 用户角色枚举
    enum Role {
        None,
        ReadOnly,
```

```
        ReadWrite,
        Deny
    }

    // 订单号与事件的映射
    mapping(bytes32 => Event) public events;

    // 订单号的事件集合
    mapping(bytes32 => bool) public eventExists;

    // 事件添加事件
    event EventAdded(bytes32 indexed orderId, string eventName);
    // 角色更新事件
    event RoleUpdated(bytes32 indexed orderId, address indexed user, Role role);

    // 添加新事件
    function addEvent(bytes32 _orderId, string memory _eventName) public {
        require(!eventExists[_orderId], "Event already exists for this order");
        events[_orderId] = Event({
            eventName: _eventName,
            participants: new address[](0)
        });
        eventExists[_orderId] = true;
        emit EventAdded(_orderId, _eventName);
    }

    // 添加用户到事件
```

```
    function addUserToEvent(bytes32 _orderId, address _user) public {
        require(eventExists[_orderId], "Event does not exist");
        require(!isUserInEvent(_orderId, _user), "User already added to the event");
        events[_orderId].participants.push(_user);
        emit EventAdded(_orderId, events[_orderId].eventName);
    }

    // 设置用户角色
    function setUserRole(bytes32 _orderId, address _user, Role _role) public {
        require(eventExists[_orderId], "Event does not exist");
        require(isUserInEvent(_orderId, _user), "User is not part of the event");
        events[_orderId].roles[_user] = _role;
        emit RoleUpdated(_orderId, _user, _role);
    }

    // 检查用户是否在事件集合中
    function isUserInEvent(bytes32 _orderId, address _user) public view returns
(bool) {
        for (uint i = 0; i < events[_orderId].participants.length; i++) {
            if (events[_orderId].participants[i] == _user) {
                return true;
            }
        }
        return false;
    }

    // 根据用户角色获取访问权限
```

```
    function checkAccess(bytes32 _orderId, address _user) public view returns
(bool) {
        require(eventExists[_orderId], "Event does not exist");
        require(isUserInEvent(_orderId, _user), "User is not part of the event");
        Role userRole = events[_orderId].roles[_user];
        if (userRole == Role.ReadOnly) {
            return true; // 只读权限
        } else if (userRole == Role.ReadWrite) {
            return true; // 读写权限
        } else {
            return false; // 拒绝访问
        }
    }
}
```

- **交易事件存证**：关键交易事件通过区块链进行存证，确保交易过程的不可篡改性和透明性，从而提高交易的可信度。

- **文件传输与存证**：交易单据和相关凭证通过信贸链网络传输，并进行 Hash 存证，确保文件的完整性和真实性，增强各方的信任。

这样的处理不仅保证了交易的安全性，还通过技术手段提升了交易的效率和可靠性。

交易事件存证智能合约示例

```
pragma solidity ^0.8.0;
```

```
contract TransactionEventRecorder {
    event TransactionRecorded(uint indexed id, address indexed seller, address
indexed buyer, string txHash, string description, string orderNumber);

    struct Transaction {
        uint id;
        address seller;
        address buyer;
        string txHash;
        string description;
        string orderNumber;
        bool isValid; // 标记交易记录是否有效
    }

    Transaction[] private transactions;
    uint private transactionId = 0; // 初始化为 0，确保与数组索引一致
    mapping(uint => bool) private transactionExists; // 用于校验交易 ID 的唯一性

    // 只有合同所有者才能调用此函数
    address private owner;

    constructor() {
        owner = msg.sender;
    }

    // 仅合同所有者可以记录交易
    modifier onlyOwner() {
```

```
        require(msg.sender == owner, "Only owner can record transactions");
        _;
    }

    function recordTransaction(address _seller, address _buyer, string memory
_txHash, string memory _description, string memory _orderNumber) public
onlyOwner {
        require(!transactionExists[transactionId], "Transaction ID already exists");

        transactionId++;
        Transaction memory transaction = Transaction(transactionId, _seller,
_buyer, _txHash, _description, _orderNumber, true);
        transactions.push(transaction);

        transactionExists[transactionId] = true;
        emit TransactionRecorded(transactionId, _seller, _buyer, _txHash,
_description, _orderNumber);
    }

    function getTransaction(uint _id) public view returns (Transaction memory) {
        require(_id <= transactions.length, "Transaction ID out of range");
        return transactions[_id - 1]; // 返回数据的方式可能存在其他问题，需要检
查
    }

    function getAllTransactions() public view returns (Transaction[] memory) {
        return transactions;
```

```
    }

    // 额外功能：允许合同所有者更新交易记录
    function updateTransaction(uint _id, string memory _description) public
onlyOwner {
        require(_id <= transactions.length, "Transaction ID out of range");
        Transaction storage transaction = transactions[_id - 1];
        transaction.description = _description;
        transaction.isValid = true; // 更新后仍有效
    }

    // 额外功能：允许合同所有者删除交易记录
    function deleteTransaction(uint _id) public onlyOwner {
        require(_id <= transactions.length, "Transaction ID out of range");
        Transaction storage transaction = transactions[_id - 1];
        transaction.isValid = false; // 标记为无效
    }
}
```

交易文件存证智能合约示例

```
    pragma solidity ^0.8.0;

contract FileProofNotary {
    // 定义文件存证结构体
    struct FileProof {
        uint id;
```

```
        string fileName;
        string fileHash; // 文件的哈希值
        address sender; // 发送者地址
        address receiver; // 接收者地址
        bool isSent; // 标记文件是否已发送
        bool isReceived; // 标记文件是否已接收
    }

    // 存储所有文件存证的数组
    FileProof[] public fileProofs;

    // 事件声明，用于记录文件发送和接收
    event FileSent(uint id, string fileName, string fileHash, address sender,
address receiver);
    event FileReceived(uint id, string fileName, address receiver);

    // 发送文件
    function sendFile(string memory _fileName, string memory _fileHash, address
_receiver) public {
        uint id = fileProofs.length + 1;
        FileProof memory fileProof = FileProof(id, _fileName, _fileHash,
msg.sender, _receiver, true, false);
        fileProofs.push(fileProof);
        emit FileSent(id, _fileName, _fileHash, msg.sender, _receiver);
    }

    // 接收文件
```

```
    function receiveFile(uint _id) public {
        FileProof storage fileProof = fileProofs[_id - 1];
        require(fileProof.sender == msg.sender || fileProof.receiver == msg.sender,
"Only sender or receiver can mark as received");
        require(!fileProof.isReceived, "File already marked as received");
        fileProof.isReceived = true;
        emit FileReceived(_id, fileProof.fileName, fileProof.receiver);
    }

    // 获取特定文件存证的详细信息
    function getFileProof(uint _id) public view returns (FileProof memory) {
        return fileProofs[_id - 1];
    }

    // 获取所有文件存证
    function getAllFileProofs() public view returns (FileProof[] memory) {
        return fileProofs;
    }
}
```

3.5.3 数据结构

（1）Circle 表

字段名称	字段类型	字段描述

circle_id	INT	圈子的唯一标识符
name	VARCHAR(255)	圈子的名称
circle_type	VARCHAR(50)	圈子的类型
status	VARCHAR(50)	圈子的状态
member_count	INT	圈子的成员数量
description	TEXT	圈子的描述
owner_id	INT	圈子的拥有者 ID
parent_circle_id	INT	父圈子的 ID（如果圈子是子圈子的话,如果是一级圈子，则该字段为 0）
created_by	INT	圈子的创建者 ID
created_at	DATETIME	创建时间
updated_at	DATETIME	更新时间

（2）User 表

字段名称	数据类型	描述
user_id	INT	用户的唯一标识符

user_type	VARCHAR(50)	用户的类型（如个人用户、企业用户等）
contact_info	VARCHAR(255)	用户的联系方式（如电话、地址等）
bio	TEXT	用户的个人简介
is_verified	BOOLEAN	用户的认证状态（如是否验证）
username	VARCHAR(255)	用户名
email	VARCHAR(255)	邮箱
created_at	DATETIME	注册时间

（3）Role 表

字段名称	数据类型	描述
role_id	INT	角色的唯一标识符
name	VARCHAR(255)	角色名称
permissions	TEXT	角色的权限列表

（4）CircleRole 表

字段名称	数据类型	描述
circle_role_id	INT	圈子角色的唯一标识符
circle_id	INT	圈子 ID
user_id	INT	用户 ID
role_id	INT	角色 ID

（5）订单表 order

字段名称	字段类型	字段描述
OrderID	VARCHAR	订单的唯一标识符
OrderDate	DATE	订单日期
BuyerID	VARCHAR	买家的唯一标识符
SellerID	VARCHAR	卖家的唯一标识符
PaymentTerms	VARCHAR	支付条款详情
DeliveryDate	DATE	交货日期
Currency	VARCHAR	订单使用的货币类型
TotalAmount	DECIMAL	订单总金额
ControlTotal	DECIMAL	用于检查的控制总金额

Status	VARCHAR	订单状态（如：待确认、已确认、已发货等）
Remarks	TEXT	订单的额外说明或备注
PaymentID	INT	支付信息的唯一标识符，外键关联支付信息表
ShippingID	INT	运输信息的唯一标识符，外键关联运输信息表
TaxID	INT	税务信息的唯一标识符，外键关联税务信息表
ContactID	INT	联系人信息的唯一标识符，外键关联联系人信息

（6）订单明细表 Order Detail Table

字段名称	字段类型	字段描述
LineItemID	INT	行项目的唯一标识符（自增）
OrderID	VARCHAR	订单的唯一标识符（外键）
ProductID	VARCHAR	产品的唯一标识符

Quantity	DECIMAL	订购数量
UnitPrice	DECIMAL	单位价格
LineTotal	DECIMAL	行项目总金额
ProductDescription	TEXT	产品的描述
DeliveryLocation	VARCHAR	交货地点
TaxRate	DECIMAL	税率
TaxAmount	DECIMAL	税额
AllowanceOrCharge	DECIMAL	折扣或附加费
PaymentDueDate	DATE	支付截止日期
ShippingDetails	TEXT	发货详情（如：运输方式、预计到达时间等）
TrackingNumber	VARCHAR	跟踪号码（如果有）
DocumentID	INT	文档要求的唯一标识符，外键关联文档要求

（7）支付信息表 Payment Information Table

字段名称	数据类型	描述

PaymentID	INT	支付信息的唯一标识符（自增）
OrderID	VARCHAR	关联到订单表的 OrderID（外键）
PaymentTerms	VARCHAR	支付条款，如预付、货到付款等
PaymentDueDate	DATE	支付截止日期
PaymentMethod	VARCHAR	支付方式，如信用卡、电汇、支票等

（8）运输信息表 Shipping Information Table

字段名称	数据类型	描述
ShippingID	INT	运输信息的唯一标识符（自增）
OrderID	VARCHAR	关联到订单表的 OrderID（外键）
CarrierID	VARCHAR	承运人 ID，从 NAD 段中提取
ModeOfTransport	VARCHAR	运输方式，如空运、海运、陆运等

ExpectedDeliveryDate	DATE	预计到达日期
TrackingNumber	VARCHAR	运输跟踪号码

（9）税务信息表 Tax Information Table

字段名称	数据类型	描述
TaxID	INT	税务信息的唯一标识符（自增）
OrderID	VARCHAR	关联到订单表的 OrderID（外键）
TaxType	VARCHAR	税种类型，如增值税、消费税等
TaxRate	DECIMAL	税率
TaxAmount	DECIMAL	税额
ExemptionDetails	TEXT	免税条款或其他税务相关的说明

（10）联系人信息表 Contact Information Table

字段名称	数据类型	描述
ContactID	INT	联系人信息的唯一标识符（自增）

OrderID	VARCHAR	关联到订单表的 OrderID（外键）
PartyRole	VARCHAR	联系人的角色，如买家代表、卖家代表等
ContactName	VARCHAR	联系人姓名
PhoneNumber	VARCHAR	联系电话
EmailAddress	VARCHAR	电子邮件地址

（11）产品表 Product Table

字段名称	数据类型	描述
ProductID	VARCHAR	产品的唯一标识符，主键
ProductName	VARCHAR	产品名称
IMDDescription	TEXT	产品描述，对应 IMD 段
AdditionalID	VARCHAR	附加产品 ID，如 HS 编码，对应 PIA 段
Quantity	DECIMAL	产品数量，对应 QTY 段
UnitPrice	DECIMAL	单位价格，对应 PRI 段
UnitOfMeasure	VARCHAR	单位，描述产品计量单

		位，如件、箱等
MeasurementDetails	TEXT	物理测量信息，如尺寸、重量等，对应 MEA 段
CategoryID	VARCHAR	产品类别 ID，外键关联产品类别表
Brand	VARCHAR	品牌名称
Manufacturer	VARCHAR	制造商名称
SKU	VARCHAR	库存单位（SKU）
Barcode	VARCHAR	条形码或 UPC 码
TaxClassID	VARCHAR	税务类别 ID，外键关联税务类别表
ShippingDetails	TEXT	发货细节，如包装类型、堆叠限制等，可能涉及 PAC、MEA 等段
ComplianceInfo	TEXT	合规信息，如安全标准、认证等，可能涉及 RCS 段
AdditionalInfo	TEXT	其他附加信息，如产品特性、使用说明等

（12）需求表 Requirements Table

字段名称	数据类型	描述
RequirementID	INT	需求的唯一标识符，自增主键
Title	VARCHAR	需求的标题
Description	TEXT	需求的详细描述，对应 FTX 段
Quantity	DECIMAL	需求的数量，对应 QTY 段
PriceRange	VARCHAR	价格范围，如“1000-5000”，对应 PRI 段
RequiredDate	DATE	期望的完成或交付日期，对应 DTM 段
Location	VARCHAR	需求的地理位置或交付地点，对应 LOC 段
Currency	VARCHAR	预算或价格的货币类型，对应 CUX 段
ReferenceNumber	VARCHAR	需求的参考编号，如需求 ID 或合同编号，对应 RFF 段
LineItemNumber	INT	行项目编号，如果需求是

		针对具体的产品或服务项，对应 LIN 段
ScheduleDetails	TEXT	需求的时间安排细节，如期望的交付时间表，对应 SCC 段
ComplianceInfo	TEXT	合规信息或特定要求，如安全标准、认证等，可能涉及 RCS 段
AdditionalInfo	TEXT	其他附加信息，如特殊条件或备注

（13）需求应答信息表 Requirements_response

字段名称	数据类型	描述
responseId	string	需求应答的唯一标识符。
demandId	string	相关需求的唯一标识符。
providerId	string	提供应答的供应商 ID。
responseDate	string	应答日期，格式为 YYYY-MM-DD。
productDetails	object	提供的产品详细信息。
- productName	string	产品名称。

- productDescription	string	产品描述。
- productPrice	float	产品价格。
- productQuantity	int	可提供的产品数量。
- productSpecs	object	产品规格，可能包含多个规格项。
- specName	string	规格名称。
- specValue	string	规格值。
availability	string	产品的可用性描述，如“现货”、“需预定”等。
leadTime	string	产品的交货期，如“2-3周”。
warranty	string	产品的保修政策。
paymentTerms	string	支付条款，如“预付款”、“货到付款”等。
contactInfo	object	供应商的联系信息。
contactName	string	联系人姓名。
contactPhone	string	联系人电话。

contactEmail	string	联系人邮箱。
additionalInfo	string	其他补充信息或特殊说明。

（14）事件记录表 Events 表

字段名	数据类型	描述	约束条件
EventID	string	事件 ID	主键，非空
OrderID	string	订单 ID，以订单号串联起同一订单下的所有事件	非空
EventCreater	string	事件提交人	非空
EventType	string	事件类型	非空
EventTime	DATETIME	事件的生成时间	非空
RecordTime	DATETIME	事件被记录时间	非空
ReadPoint	string	事件生成地点	可空
BizLocation	string	业务执行地点	可空
ActionType	string	事件的动作类型	可空
BizStep	string	业务步骤	可空

Extension	string	扩展信息	可空

（15）EventParticipants 表

字段名	数据类型	描述	约束条件
ParticipantID	string	参与者 ID	主键，非空
UserID	string	用户 ID	非空
EventID	string	事件 ID	非空，外键关联到 ProductEvents

（16）EventRoles 表

字段名	数据类型	描述	约束条件
RoleID	INT	角色 ID	主键，非空
RoleName	string	角色名称	非空

（17）EventAccess 表

字段名	数据类型	描述	约束条件
AccessID	string	访问控制 ID	主键，非空
EventID	string	事件 ID	非空，外键关联到 ProductEvents
ParticipantID	string	参与者 ID	非空，外键关联到

			EventParticipants
RoleID	string	角色 ID	非空，外键关联到 EventRoles

3.5.4 接口设计

此类接口主要是应用层面的数据接口，底层实现逻辑已经在技术方案中进行了描述。

（1）商品发布接口

- 接口概述

该接口用于企业用户（卖家）在 B2B 电商平台上发布新的商品信息，供其他企业用户（买家）浏览和购买。该接口的底层逻辑中需要将文本信息上链到区块链，将文件信息上传到 IPFS。

- 接口信息

接口地址：POST /api/products

接口版本：v1.0

- 请求参数

请求体格式

```
  {
"supplierId": "供应商 ID",
"productName": "商品名称",
"productCode": "商品编码",
```

```
  "category": "商品分类",
  "description": "商品描述",
  "price": "价格",
  "quantity": "库存数量",
  "minOrderQuantity": "最小订购数量",
  "moq": "最小起订量",
  "files": "商品图片 ipfsHash 地址，如果存在图片或视频等文件需要先将其传输到
ipfs 上",
  "specs": "商品规格",
  "warranty": "保修政策",
  "shippingDetails": "物流详情",
  "IsPublic": "是否全部公开",
  "circles":"关联的圈子名称"
}
```

参数说明

supplierId (string): 发布商品的供应商 ID，必填。

productName (string): 商品名称，必填。

productCode (string): 商品编码，用于内部管理，必填。

category (string): 商品分类，必填。

description (string): 商品详细描述，必填。

price (float): 商品价格，必填。

quantity (int): 当前库存数量，必填。

minOrderQuantity (int): 最小订购数量，可选。

moq (int): 最小起订量，可选。

files (array[string]): 商品图片等详细介绍资料列表，可选。

specs (object): 商品规格，可选。

warranty (string): 保修政策，可选。

shippingDetails (string): 物流详情，可选。

IsPublic (bool): 是否公开给全网，可根据情况增加。

circles (array[string]): 当不公开给全网时，需要填写需要发布到的圈子 id 列表。

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
    {
  "productId": "prod_789",
  "message": "商品发布成功"
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

- 接口使用示例

请求示例

```
    POST /api/products
Host: example.com
Content-Type: application/json
Authorization: Bearer <supplier_token>

{
  "supplierId": "supp_123",
  "productName": "高端工业机器人",
  "productCode": "IR-2024",
  "category": "工业自动化",
  "description": "适用于高精度制造流程的机器人",
  "price": 15000.00,
  "quantity": 50,
  "minOrderQuantity": 5,
  "moq": 10,
  "images": ["http://example.com/images/product1.jpg"],
  "specs": {
    "weight": "500kg",
```

```
    "dimensions": "1m x 1m x 2m"
  },
  "warranty": "一年质保",
  "shippingDetails": "海运或空运",
  "paymentTerms": "预付款 50%，余款发货前支付"
}
```

响应示例

```
    {
  "productId": "prod_789",
  "message": "商品发布成功"
}
```

（2）服务发布接口

- 接口概述

该接口用于将服务信息发布到新型电子商务系统中，以便用户可以浏览和购买服务。

- 接口信息

接口地址：POST /api/services

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "name": "服务名称",
  "description": "服务描述",
  "price": 价格,
  "duration": "服务时长（分钟）",
  "category": "服务分类",
  "availability": "服务可用性（如：周一至周五 9:00-17:00）",
  "IsPublic": "是否全部公开",
  "circles":"关联的圈子名称"
}
```

参数说明

name (string): 服务名称，必填。

description (string): 服务描述，必填。

price (float): 服务价格，必填。

duration (int): 服务时长（分钟），必填。

category (string): 服务分类，必填。

availability (string): 服务可用性，必填。

IsPublic (bool): 是否公开给全网，可根据情况增加。

circles (array[string]): 当不公开给全网时，需要填写需要发布到的圈子 id 列表。

请求头:Content-Type: application/json

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
    {
  "message": "服务发布成功",
  "serviceId": "生成的服务 ID"
}
```

响应头：Location: /api/services/<serviceId>

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

（3）需求发布接口

- 接口概述

该接口用于允许买家在新型电子商务系统中发布自己的需求，以便商家能够查看并提供相应的服务或商品。

- 接口信息

接口地址：POST /api/demands

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "title": "需求标题",
  "description": "需求详细描述",
  "category": "需求分类",
  "location": "需求地点",
  "budget": "预算范围",
  "deadline": "截止日期",
  "contactInfo": {
    "name": "联系人姓名",
    "phone": "联系人电话",
    "email": "联系人邮箱"
  },
  "IsPublic": "是否全部公开",
  "circles":"关联的圈子名称"
}
```

参数说明

title (string): 需求标题，必填。

description (string): 需求详细描述，必填。

category (string): 需求分类，必填。

location (string): 需求地点，必填。

budget (string): 预算范围，必填。

deadline (string): 截止日期，格式为 YYYY-MM-DD，必填。

contactInfo (object): 联系信息，必填。

name (string): 联系人姓名。

phone (string): 联系人电话。

email (string): 联系人邮箱。

IsPublic (bool): 是否公开给全网，可根据情况增加。

circles (array[string]): 当不公开给全网时，需要填写需要发布到的圈子 id 列表。

请求头:Content-Type: application/json

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
{
"demandId": "需求 ID",
```

```
  "message": "需求发布成功"
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

（4） 商品检索

- 接口概述

该接口用于允许用户根据关键词或其他条件检索商品信息。调用改方法前需要先调用`可搜索加密方法`，对查询内容进行解密。

- 接口信息

接口地址：GET /api/products/search

接口版本：v1.0

- 请求参数

查询参数

keyword (string): 检索关键词。

category (string): 商品分类。

priceRange (string): 价格范围，格式为最低价格-最高价格。

sort (string): 排序方式，例如 price_asc 表示按价格升序排列。

page (int): 页码。

limit (int): 每页显示的商品数量。

请求头 Content-Type: application/json

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
  {
"total": 100,
"page": 1,
"limit": 10,
"products": [
  {
    "id": "产品 ID",
    "name": "商品名称",
    "description": "商品描述",
    "price": 价格,
    "stock": 库存,
    "category": "商品分类"
```

```
    },
    // 其他商品信息...
  ]
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

（5）商品询价

- 接口概述

该接口用于允许用户对特定商品进行询价，获取商品的价格信息或者获取更优惠的价格。

- 接口信息

接口地址：POST /api/products/{productId}/inquiry

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "userId": "用户 ID",
  "quantity": 数量,
  "requestedPrice": "请求的价格",
  "contactInfo": {
    "name": "联系人姓名",
    "phone": "联系人电话",
    "email": "联系人邮箱"
  },
  "additionalInfo": "其他信息"
}
```

参数说明

userId (string): 发起询价的用户 ID，必填。

quantity (int): 询价的商品数量，必填。

contactInfo (object): 联系信息，可选，询价人可自行选择是否需要留下联系方式。

name (string): 联系人姓名。

phone (string): 联系人电话。

email (string): 联系人邮箱。

requestedPrice (float): 用户请求的价格，可选。

additionalInfo (string): 其他有助于报价的信息，如购买时间、特殊要求等，可选。

请求头

Content-Type: application/json

Authorization: Bearer <token>

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
    {
  "status": "询价状态",
  "message": "询价成功，等待回复",
  "inquiryId": "询价 ID",
  "bestOffer": {
    "price": "报价",
    "validUntil": "有效期至",
    "terms": "报价条件"
  }
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

商品信息不存在

```
    {
  "error": "商品不存在",
  "message": "请求的商品 ID 不存在"
}
```

- 接口使用示例

请求示例

```
    POST /api/products/12345/inquiry
Host: example.com
Content-Type: application/json
Authorization: Bearer <user_token>

{
  "userId": "user_123",
  "quantity": 10,
  "requestedPrice": 250.0,
  "additionalInfo": "需要在下周一前交货"
```

```
}
```

响应示例

```
    {
  "status": "询价已提交",
  "message": "询价成功，等待回复",
  "inquiryId": "inq_001",
  "bestOffer": {
    "price": 245.0,
    "validUntil": "2024-05-01",
    "terms": "报价有效期内有效，需预付 50%定金"
  }
}
```

（6） 商品下单

- 接口概述

该接口用于处理用户的下单请求，创建一个新的订单并返回订单详情。该接口底层需要调用数据存证接口，我们需要将下单过程存储到区块链中。

- 接口信息

接口地址：POST /api/orders

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "userId": "用户 ID",
  "productId": "商品 ID",
  "quantity": 数量,
  "shippingAddress": {
    "addressLine1": "街道地址 1",
    "addressLine2": "街道地址 2",
    "city": "城市",
    "state": "州/省",
    "postalCode": "邮政编码",
    "country": "国家"
  },
  "paymentMethod": "支付方式",
  "notes": "订单备注"
}
```

参数说明

userId (string): 下单用户的 ID，必填。

productId (string): 购买的商品 ID，必填。

quantity (int): 购买商品的数量，必填。

shippingAddress (object): 收货地址信息，必填。

addressLine1 (string): 街道地址 1。

addressLine2 (string): 街道地址 2。

city (string): 城市。

state (string): 州/省。

postalCode (string): 邮政编码。

country (string): 国家。

paymentMethod (string): 支付方式，必填。

notes (string): 订单备注，可选。

请求头

Content-Type: application/json

Authorization: Bearer <token>

- 响应参数

成功响应

响应状态码：201 Created

响应体：

```
  {
"orderId": "订单 ID",
"userId": "用户 ID",
"productId": "商品 ID",
"quantity": 数量,
"totalPrice": "总价",
```

```
  "shippingAddress": {
    "addressLine1": "街道地址 1",
    "addressLine2": "街道地址 2",
    "city": "城市",
    "state": "州/省",
    "postalCode": "邮政编码",
    "country": "国家"
  },
  "paymentMethod": "支付方式",
  "status": "订单状态",
  "notes": "订单备注",
  "createdAt": "创建时间"
}
```

错误响应

参数错误：

```
  {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

商品信息不存在

```
  {
  "error": "商品不存在",
```

```
  "message": "请求的商品 ID 不存在"
}
```

认证失败：

```
    {
  "error": "认证失败",
  "message": "无效的认证信息"
}
```

- 4. 接口使用示例

请求示例

```
    POST /api/orders
Host: example.com
Content-Type: application/json
Authorization: Bearer <user_token>

{
  "userId": "user_123",
  "productId": "prod_456",
  "quantity": 2,
  "shippingAddress": {
    "addressLine1": "123 Main St",
    "addressLine2": "",
    "city": "Anytown",
```

```
    "state": "CA",
    "postalCode": "12345",
    "country": "USA"
  },
  "paymentMethod": "Credit Card",
  "notes": "请尽快发货"
}
```

响应示例

```
    {
  "orderId": "ord_789",
  "userId": "user_123",
  "productId": "prod_456",
  "quantity": 2,
  "totalPrice": 599.98,
  "shippingAddress": {
    "addressLine1": "123 Main St",
    "addressLine2": "",
    "city": "Anytown",
    "state": "CA",
    "postalCode": "12345",
    "country": "USA"
  },
  "paymentMethod": "Credit Card",
  "status": "Pending",
```

```
  "notes": "请尽快发货",
  "createdAt": "2024-04-01T12:00:00Z"
}
```

（7） 商品发货回执

- 接口概述

该接口用于卖方在收到买方货款并完成商品发货后，将发货单和物流订单号等信息发送给买方，以便买方跟踪商品物流状态。

- 接口信息

接口地址：POST /api/orders/{orderId}/shipment

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "carrier": "物流公司名称",
  "trackingNumber": "物流订单号",
  "shipmentDate": "发货日期",
  "expectedDeliveryDate": "预计送达日期",
  "shipmentDetails": "发货详细说明",
  "shipmentImage": "发货单图片链接"
}
```

参数说明

carrier (string): 物流公司名称，必填。

trackingNumber (string): 物流订单号，用于买方跟踪物流状态，必填。

shipmentDate (string): 发货日期，格式为 YYYY-MM-DD，必填。

expectedDeliveryDate (string): 预计送达日期，格式为 YYYY-MM-DD，必填。

shipmentDetails (string): 发货详细说明，如包装类型、发货方式等，可选。

shipmentImage (string): 发货单图片链接，可选。

请求头

Content-Type: application/json

Authorization: Bearer <token>

- 响应参数

成功响应

响应状态码：200 OK

响应体：

```
  {
"message": "发货信息已更新",
"orderId": "订单 ID",
"carrier": "物流公司名称",
"trackingNumber": "物流订单号",
"shipmentDate": "发货日期",
```

```
  "expectedDeliveryDate": "预计送达日期"
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

订单不存在

```
    {
  "error": "订单不存在",
  "message": "请求的订单 ID 不存在"
}
```

认证失败：

```
    {
  "error": "认证失败",
  "message": "无效的认证信息"
}
```

- 接口使用示例

请求示例

```
    POST /api/orders/ord_789/shipment
Host: example.com
Content-Type: application/json
Authorization: Bearer <seller_token>

{
  "carrier": "Speedy Express",
  "trackingNumber": "123456789",
  "shipmentDate": "2024-04-05",
  "expectedDeliveryDate": "2024-04-12",
  "shipmentDetails": "商品已安全包装，通过空运发货",
  "shipmentImage": "http://example.com/shipments/shipment123.jpg"
}
```

响应示例

```
    {
  "message": "发货信息已更新",
  "orderId": "ord_789",
  "carrier": "Speedy Express",
  "trackingNumber": "123456789",
  "shipmentDate": "2024-04-05",
  "expectedDeliveryDate": "2024-04-12"
```

```
}
```

（8） 需求应答

- 接口概述

该接口用于卖家在查询到买家提出的需求后，表示愿意接洽，并提供自己的初步产品信息和联系方式给买家。

- 接口信息

接口地址：POST /api/demands/{demandId}/responses

接口版本：v1.0

- 请求参数

请求体格式

```
    {
  "providerId": "卖家 ID",
  "productName": "产品名称",
  "productDescription": "产品描述",
  "productPrice": "产品价格",
  "availability": "产品可用性",
  "contactName": "联系人姓名",
  "contactPhone": "联系人电话",
  "contactEmail": "联系人邮箱"
}
```

参数说明

providerId (string): 卖家的唯一标识符，必填。

productName (string): 提供的产品名称，必填。

productDescription (string): 提供的产品描述，必填。

productPrice (float): 提供的产品价格，必填。

availability (string): 产品的可用性或交付时间，必填。

contactName (string): 联系人姓名，必填。

contactPhone (string): 联系人电话，必填。

contactEmail (string): 联系人邮箱，必填。

请求头

Content-Type: application/json

Authorization: Bearer <token>

- 响应参数

成功响应

响应状态码：201 Created

响应体：

```
{
"message": "需求应答已提交",
"responseId": "应答 ID",
```

```
  "providerId": "卖家 ID",
  "productName": "产品名称",
  "contactName": "联系人姓名",
  "contactPhone": "联系人电话",
  "contactEmail": "联系人邮箱"
}
```

错误响应

参数错误：

```
    {
  "error": "参数错误",
  "message": "具体错误信息"
}
```

需求不存在

```
    {
  "error": "需求不存在",
  "message": "请求的需求 ID 不存在"
}
```

认证失败：

```
    {
  "error": "认证失败",
```

```
  "message": "无效的认证信息"
}
```

- 接口使用示例

请求示例

```
    POST /api/demands/dem_123/responses
Host: example.com
Content-Type: application/json
Authorization: Bearer <seller_token>

{
  "providerId": "prov_456",
  "productName": "高级智能手表",
  "productDescription": "最新型号，健康监测功能，长效电池",
  "productPrice": 299.99,
  "availability": "现货，预计 3 个工作日内发货",
  "contactName": "张三",
  "contactPhone": "123-456-7890",
  "contactEmail": "zhangsan@example.com"
}
```

响应示例

```
    {
  "message": "需求应答已提交",
```

```
  "responseId": "res_789",
  "providerId": "prov_456",
  "productName": "高级智能手表",
  "contactName": "张三",
  "contactPhone": "123-456-7890",
  "contactEmail": "zhangsan@example.com"
}
```

3.5.5 订单号编码规则

我们使用 UN/EDIFACT 的 BGM 段作为订单号的编码规则，它提供了一种标准化的方式来标识和处理订单信息。以下是基于 BGM 段设计订单号的编码规则：

（1）订单号编码结构

```
[BGM-010]-[BGM-020]-[BGM-030]-[BGM-040]
```

- **BGM-010**：文档/消息名称
- **BGM-020**：文档/消息编号
- **BGM-030**：消息功能编码
- **BGM-040**：响应类型编码

（2）编码规则详细说明

BGM-010：文档/消息名称

- 编码：1001（Document/message name, coded）

- 代码列表限定符：1131（Code list qualifier）
- 代码列表负责机构：3055（Code list responsible agency, coded）
- 名称：1000（Document/message name）

示例：如果订单是标准销售订单，可以编码为：

- 1001：ORDR
- 1131：UN
- 3055：6
- 1000：Order

BGM-020：文档/消息编号

- 编码：1004（Document/message number）

格式：an..35（字母和数字的组合，长度最多35位）

示例：自动生成的唯一订单编号，如：20241117-001

BGM-030：消息功能编码

- 编码：1225（Message function, coded）

格式：an..3（字母和数字的组合，长度最多3位）

示例：如果这是一个新订单，可以编码为：NEW

BGM-040：响应类型编码

- 编码：4343（Response type, coded）

格式：an..3（字母和数字的组合，长度最多3位）

示例：如果这个订单不需要响应，可以编码为：NOR

（3）完整的订单号示例

```
ORDR-UN-6-Order-20241117-001-NEW-NOR
105-42-6-105-20241117-001-2-NA
```

订单号解析

BGM-010：105-42-6-105

- 编码：1001-105 （Purchase order 采购订单）
- 代码列表限定符：1131-42 （Business function 用于业务功能相关的代码）
- 代码列表负责机构：3055-6 （UN/ECE 联合国 - 欧洲经济委员会）
- 名称：1000-105 （Purchase order 采购订单）

BGM-020：20241117-001

- 订单编码： 1225-20241117-001 （自动生成的唯一订单编号）

BGM-030：2

- 编码：1225-2 创建订单 （Addition 添加）

BGM-040：NA

- 编码：4343-NA （No acknowledgement needed 无需确认）

4、应用示例

“信贸链协议”在贸易领域具有广泛的应用场景，各方可以利用它快速构建自己的“朋友圈”，比如打造新型电子商务网络、行业级供应链协作网络、智慧口岸物流服务网络等，推动不同业务协作场景实现数字化、标准化和智能化。下面举几个如何利用“信贸链协议”构建应用场景的例子：

4.1 二手产品新型电子商务网络

4.1.1 行业痛点分析

当前二手产品交易市场存在以下核心问题：

- 缺乏信用机制

买卖双方身份难验证，虚假交易、假冒商品屡见不鲜，缺乏可信的信用评价体系。

- 二手物品认证困难

由于二手物品已经经过使用，折旧情况如何对商品价值影响较大，如果没有一个权威的机构进行认证，二手物品的价值很难得到认可，但认证后的真实性认定和核验也较为困难。

- 信息孤岛

不同平台数据标准不统一，商品信息分散且难以互通，导致交易效率低下。

- 交易风险高

支付欺诈、物流责任不清、售后纠纷频发，缺乏全流程可追溯机制。

- 匹配效率低

传统交易平台依赖人工搜索，供需匹配精度不足。

4.1.2 解决思路

基于“信贸链协议”的分布式技术架构，构建一个开放、中立、安全的二手产品交易网络，通过信贸链的存证能力提升各方信任，通过AI的智能匹配机制提升了交易的效率，具体思路如下：

- 可信身份认证

利用数字身份握手层（DID）为买卖双方、第三方检测机构、物流服务商等实体分配唯一身份标识，确保参与者真实可信。

- 物品情况认定

通过第三方机构对物品的情况进行认定，并上传检查报告，将商品DID和检测报告进行绑定，并支持溯源。

- 智能匹配与执行

借助交易过程管理层的AI技术分析用户需求与商品特征，通过AI+智能合约自动化撮合交易，优化资源配置，提升了交易效率，为供需双方都提供了更低的交易成本。

- 全流程闭环

通过分布式的二手产品交易网络连接各方，融合检测整备机构、国内/国际物流公司、金融保险、海外仓运营企业、法财税等第三方专业服务，形成检测整备、物流运输、智能通关等全链条、开放共享的再生资源数字化服务生态，通过数据交换存证层实现异构系统互联，形成全流程不可篡改的交易记录、全程可追溯，明确各方责任，形成全流程闭环。

4.1.3 网络搭建：基于信贸链五层协议的技术实现

- **数字身份握手层**

供应链上下游企业注册到二手产品新型电子商务网络中，包括买卖双方、第三方检测机构、物流服务商，分配唯一的分布式身份标识（DID），确保身份可信。通过可验证凭证（VC）增强企业资质可信度（如进出口许可、产品质量认证）。

- **跨链流转与同步层**

二手电商环境，不同平台可能基于不同的区块链网络运行，例如国外部分企业使用以太坊，国内部分企业使用长安链。利用跨链通信协议，打破这种“链岛”现象，实现二手商品信息与检验检测报告在不同链之间的自由流动。

- **数据交换与存证层**

通过加密技术实现交易单证（如合同、检测报告）的安全交换，并为每份文件生成唯一数据指纹上链存证。

提供链上核验功能，参与方可实时验证二手物品检测报告、运输记录的真实性。

- **数据标准转换层**

参考 UN/CEFACT 标准，定义二手商品核心字段（如品牌、型号、成色、保修状态），统一不同平台的数据格式与标准，并为不同平台开发适配器（Adapter），确保数据能够被顺利交换与共享。

- **交易过程管理层**

圈子创建：大型链主企业可以创建圈子，包括自定义圈子名称、成员角色（如供应商、制造商、物流公司）及权限设置，并邀请上下游合作伙伴加入圈子。

商品发布： 卖家调用智能合约发布商品信息，可选择公开或定向推送至特定圈子。

供需匹配： AI 引擎基于用户画像（历史交易、偏好）和商品特征，使用协同过滤算法推荐匹配结果。

交易执行：买卖双方确认订单后，触发智能合约锁定资金，物流信息上链追踪。

4.1.4 业务流程示例：二手手机交易全流程

- 商品检测及发布
  - 卖家填写手机详细信息（如型号、成色、价格、使用情况）并提交第三方质检。
  - 第三方检验手机质量后将检测结果（如成色等级、故障情况）生成质检报告并上传至 IPFS 存储。
  - 卖家通过智能合约发布商品，并指定发布圈子，如果圈子中的用户使用了不同的区块链底座，将自动进行跨链商品信息同步。
- 智能撮合
  - AI 分析买家搜索关键词“iPhone 12 95 新”，匹配卖家商品。
  - 基于输入的需求与信用评分（链上历史交易数据）推荐最优卖家。
- 商品查验
  - 买家在线查看商品信息，并查看卖家的历史信用情况。
  - 并查看商品的检验报告，并查看检测机构和检测人的信息。
- 交易确认

- 买家下单通过网络发送采购信息给卖家。
- 卖家收到订单后确认发货信息，并安排物流服务。

- 物流追踪
  - 每个物流节点（揽收、运输、签收）信息实时上链。
  - 买家可以实时查询物流状态。

- 资金结算及交易存证
  - 买家确认收货，资金结算完成后，交易数据生成全局哈希值并永久存证至区块链。

4.1.5 总结

本方案通过“信贸链协议”五层架构，构建了一个高效、透明、安全的二手产品交易网络，解决了身份验证、数据孤岛、交易风险等核心痛点。开发者可基于开源代码（如慧贸 OS）快速部署节点，结合智能合约与 AI 技术实现业务闭环。未来可通过扩展跨链生态、优化隐私计算算法，进一步提升网络性能与用户体验。

4.2 农产品供应链协作网络

4.2.1 行业痛点分析

- **数据孤岛与标准不统一**
  - 农产品供应链各环节（如海关、质检、支付）数据格式与标准差异大，导致信息割裂，难以实现全流程透明化。例如，物流状态与支付凭证无法实时互通，影响交易效率。

- **供应链协同效率低下:**

  - 信息传递和业务协作主要依赖人工沟通和纸质单据，效率低下，易出错。生产、加工、物流、销售环节信息不共享，物流调度优化困难，易造成运输延迟、损耗增加。

  - 面对市场需求变化或突发事件（如自然灾害、疫情），供应链响应速度慢，难以快速调整生产和供应计划。

- **跨境贸易合规复杂度高**

  - 跨境农产品贸易涉及多国监管体系，传统追溯系统难以实现跨境互认和信息共享，难以动态整合多国合规要求。

  - 现有追溯系统信息采集环节不足，追溯信息链条断裂，无法满足消费者全面了解产品信息的需求。

4.2.2 解决思路

- **打造可信透明的信息共享平台:**

  - 将农产品生产、加工、质检、物流、销售等环节的关键数据上链存证，确保数据不可篡改和可追溯。

  - 鼓励农产品供应链各参与者（ 生产企业、监管部门、消费者）共同参与平台建设和数据维护，提升数据可信度和公信力。

  - 基于 DID 身份认证和权限管理机制，实现数据分级授权共享，保障商业秘密和用户隐私。

  - 提供农产品全流程可视化追溯信息查询服务。

- **AI 赋能的动态决策体系：**

  - 基于链上实时交易数据（如商品检索接口的访问量、询价接口报价趋势），通过 AI 模型预测区域市场需求，动态调整生产计划与仓储布局。

- **建立智能高效的协同机制**:

  - 利用智能合约自动化执行交易合同、质量检测、支付结算等环节，减少人工干预，提高效率和降低风险。

  - 利用 AI 技术分析市场数据、气候数据、销售数据等，实现农产品供需精准预测，指导生产和销售计划。

  - 结合物联网、AI 等技术，实现农产品物流全过程监控和智能调度，优化运输路线、降低损耗、提升效率。

4.2.3 网络搭建过程：基于信贸链五层协议的技术实现

- **数字身份握手层**

  - 为参与方（企业、政府机构、产品、硬件设备等）分配唯一且不可篡改的数字身份，确保身份真实性和可追溯性。例如，农产品生产商、物流方、监管机构均可在网络中完成身份注册与验证。

  - 支持组织身份、员工身份、产品身份（如农产品批次编码）的链上管理。

- **跨链流转与同步**

  - 支持农产品交易中涉及的跨境支付、物流数据跨链同步，确保多链环境下数据一致性。

- **数据交换与存证层**

  - 通过加密技术实现交易单证（如合同、质检报告）的安全交换，并为每份文件生成唯一数据指纹上链存证.

  - 提供链上核验功能，参与方可实时验证农产品原产地证明、运输记录的真实性。

- **数据标准转换层**

  - 自动转换不同系统间的数据格式差异，例如将国际标准 UN/EDIFACT 与国内标准 GB/T 36371 的发票数据进行适配，降低集成复杂度。

  - 支持农产品贸易中多语言、多计量单位的标准化处理。

- **交易过程管理层**

  - 基于智能合约自动化执行交易流程（如支付、验收），全程上链记录关键节点（如订单确认、清关完成）。

  - 通过全流程订单追踪服务实现全流程状态实时追踪，例如实时监控农产品运输路径节点并标记异常事件。

4.2.4 业务流程示例：农产品交易全流程

- **销售下单**

  - 买方通过平台发布采购需求，智能合约自动生成包含产品规格、价格及交付条款的电子订单，并分配唯一订单 ID 用于全流程追踪。供需双方通过加密通信确认合同细节，电子签名与订单哈希值同步上链存证。

- **工厂采购**

- 系统基于智能合约匹配最优供应商，采购订单数据元素（如商品编码、计量单位）自动转换为 EDIFACT 或 X12 等国际标准格式，区块链记录供应商资质核验结果及原材料质检报告。

- **物流运输**

- 根据货物体积、时效要求智能推荐运输方案，订舱委托书与装箱单等通过 IPFS 加密存储，文件哈希值上链供承运方核验。船期动态实时更新至区块链节点。

- 集装箱进场信息与码头作业进度实时同步，电子装船通知单关联提单号并触发报关流程。异常事件（如货损）由 AI 识别并写入区块链告警日志。

- **通关放行**

- AI 自动填充报关单字段，校验 HS 编码与贸易术语（如 FOB/CIF）合规性，发票、原产地证等文件哈希值经智能合约比对上链数据，申报状态变更实时推送。

- 区块链存证的放行指令与船舶离港数据自动关联，电子舱单信息通过跨链协议同步至目的港监管系统。货物位置通过 GPS 数据上链实现动态可视化。

- **提单签发**

- 电子提单基于 UN/CEFACT 标准生成，承运方私钥签名后存证至区块链。提单转让记录与所有权变更通过智能合约自动执行，避免纸质单据冒用风险。

- **清关提货**

- 进口商凭电子提单发起清关，AI 自动匹配预申报数据并计算应缴税费。海关放行指令触发智能合约释放货物，提货人身份经 DID 验证后完成交付。

- **收付汇结算**

- 智能合约根据提单签收状态自动发起付款请求，外汇结算数据符合 GB/T 36371 标准并加密传输至银行系统。信用证条款与资金流向全程上链。

4.2.5 总结

通过“信贸链”协议与智能合约的结合，二手手机交易全流程实现了从商品发布到资金结算的全场景数字化与智能化管理。具体优势如下：

高效性：智能合约与 AI 技术大幅减少了人工干预，缩短了交易周期。

透明性：区块链技术确保了商品信息和交易记录的透明可查，增强了买卖双方的信任。

安全性：数据上链存证和隐私保护机制确保了敏感信息的安全性。

可追溯性：全程物流数据的实时记录与存证为后续追溯提供了可靠依据。

智能化：AI 算法优化了供需匹配与物流路径规划，提升了整体效率。

4.3 空港口岸物流服务网络

4.3.1 行业痛点分析

- **数据孤岛问题**
  - 当前空港口岸物流涉及多个部门（如海关、航空公司、物流公司等），各部门间数据标准不统一，信息共享不畅，协同效率不高。
- **通关效率低下**
  - 传统的通关流程依赖人工审核和纸质文件，耗时长且容易出错。

- 缺乏智能化的预测和分析工具，无法提前预判货物通关时间，导致口岸拥堵和延误。

- **缺乏信任机制**

- 由于数据分散且难以验证，企业与政府部门之间缺乏信任，容易出现虚假申报、走私等问题。

- 跨境物流中各方协作困难，责任归属不明确。

- **物流路径优化不足**

- 缺乏对货物运输路径的实时监控和动态调整能力，导致运输时间长、成本高。

- 仓储和配送资源分配不合理，容易出现资源浪费或供需失衡。

- **监管与合规挑战**

- 政府部门难以实时监控物流全过程，难以及时发现违规行为（如货物走私、瞒报等）。

- 缺乏统一的监管标准和追溯机制，导致跨境物流中的风险难以控制。

- **资源浪费与成本高昂**

- 因信息不对称和流程低效，企业常面临货物滞港、仓储积压等问题，增加了运营成本。

- 缺乏对物流全链条的可视化管理，难以优化资源配置。

4.3.2 解决思路

基于“信贸链协议”的技术架构，结合“单一窗口”模式，构建一个开放、可信、高效的空港口岸物流服务网络。具体思路如下：

- **数据互通与共享**
  - 利用“信贸链协议”实现跨部门、跨区域的数据互通，统一数据标准，消除信息孤岛。
  - 将海关、税务、物流公司等部门的数据上链存证，确保数据真实性和可追溯性。
- **智能化通关流程**
  - 通过AI技术对历史通关数据进行分析，优化通关流程并预测货物通关时间。
  - 引入智能合约自动化处理报关、检疫等环节，减少人工干预。
- **可信身份认证**
  - 为海关、物流公司、企业等参与方分配唯一的分布式身份标识（DID），确保参与方身份真实可信。
  - 第三方机构（如行业协会）颁发可验证凭证（VC），增强企业资质可信度。
- **物流路径优化**
  - 利用AI算法分析实时交通数据和历史运输数据，动态优化物流路径，减少运输时间和成本。

- 实时监控货物状态（如温度、湿度）并记录至区块链，确保货物安全。

- **全流程可视化监管**

  - 从货物出库到最终交付，每个环节的信息实时上链，监管部门可随时查看物流状态。

  - 异常情况（如货物延误、损坏）自动触发预警机制，并通知相关方及时处理。

- **降低运营成本**

  - 通过数据共享和流程自动化减少重复工作，降低企业运营成本。

  - 动态调整仓储和配送资源，避免资源浪费。

4.3.3 网络搭建过程：基于信贸链五层协议的技术实现

- **数字身份握手层**

  - 企业通过“单一窗口”平台提交资质信息（如营业执照、税务登记证），生成 DID 标识符。

  - 海关、商检等部门通过智能合约验证企业资质，并颁发 VC（如“进出口资质凭证”）。

  - DID 标识与 VC 绑定后上链存证，确保身份可信。

- **跨链流转与同步层**

  - 空港口岸物流涉及多个区域和平台，不同平台可能基于不同的区块链网络运行。利用跨链通信协议，打破这种“链岛”现象，实现资产（如电子提

单、运输单）和信息（如货物状态、通关数据、检验证明）在不同链之间的自由流动。

- **数据交换与存证层**

  - 空港口岸物流涉及海关、商检、税务、物流公司等多个组织，通过开发各组织相应系统连接器，利用端对端数据交换技术，打破信息孤岛，实现各组织间的数据互通与共享。

  - 在数据共享过程中，采用国密算法对数据进行加密，确保敏感信息不被泄露。

  - 对关键物流数据（如报关单、运输记录）生成唯一的数据指纹（如SHA-256 哈希值），通过智能合约记录至区块链，确保其不可篡改性和可追溯性。

- **数据标准转换层**

  - 不同组织可能使用不同的数据格式和标准，如海关与企业之间的报关单可能采用不同的格式（如 Excel、PDF、Json），通过统一数据标准（如基于UN/CEFACT（联合国贸易便利化和电子业务中心）等国际标准，定义空港口岸物流的核心数据字段（如货物编号、运输方式、通关状态））和为不同系统开发数据适配器，消除不同系统间的语义差异，并根据业务需求动态调整数据转换规则，适应不断变化的物流场景，确保物流数据能够高效、准确地共享与交换。

- **交易过程管理层**

  - 通过将货物从出库到交付的每个环节信息实时上链，监管部门通过统一业务编号查看物流状态并处理异常情况，实现物流全过程的透明化监管。

4.3.4 业务流程示例：冷链食品进口

冷链食品进口涉及多个环节（如预报、查验、放行、物流运输、交付），且需要海关、物流公司、企业等多方协作。通过“信贸链协议”与“单一窗口”平台的结合，进口冷链食品的通关流程将更加高效、透明和可追溯。以下是业务流程示例：

- **货物抵港预报与智能审核**

  - 企业通过“单一窗口”平台提交进口冷链食品的预报信息（如货物名称、数量、重量、原产地、运输方式、预计到港时间）。

  - 海关后台系统接收预报信息并完成预审，AI 引擎分析预报信息（如货物类型、原产国）并预测通关时间（ETA），提前安排查验资源，生成查验任务分配至查验部门。

- **货物查验与记录**

  - 货物抵达港口后，物流公司完成卸载并将货物运送至查验场地。

  - 查验设备（如温度传感器、X 光机）实时采集货物状态（如温度、包装完整性）并上传至区块链。

  - AI 算法分析货物图像（如 X 光扫描图）和历史数据，辅助海关人员快速识别潜在问题（如包装破损、夹藏物品），海关人员根据智能合约提示的查验重点（如重点关注货物批次、原产国）进行实物查验。

  - 查验结果（如合格/不合格）通过区块链实时记录，并生成唯一的数据指纹。

- **通关放行与物流调度**

- 海关系统通过智能合约自动比对预报信息、查验结果与通关规则（如检疫标准、关税政策）。
- 如需查验，智能合约触发查验通知通知收发货人。
- 收发货人在单一窗口平台提交提货预约，并联系物流公司安排运输计划。

- **货物运输与监控**

- 物流公司通过物联网设备实时监控货物状态（如温度、湿度、震动）并上传至区块链。
- 企业可以实时查看货物运输状态，并提前安排仓储和配送资源。
- 若货物状态异常（如温度超标），系统自动触发预警并通知相关方（如企业、物流公司）。

- **货物交付与结算**

- 货物送达后，企业确认收货并通过智能合约完成收货流程并通知海外卖家。
- 智能合约记录交易完成状态，并生成全局交易哈希存证。
- 监管部门可以查询货物从抵港到交付的全生命周期数据，确保合规性。

4.3.5 总结

本方案通过“信贸链协议”与“单一窗口”相结合，构建了一个高效、透明、可信的空港口岸物流服务网络。方案解决了数据孤岛、通关效率低、物流路径优化、货物抵港预报与智能审核不足等核心痛点，并通过智能化和自动化手段提升了整体效率

与用户体验。未来可通过扩展跨链生态、优化 AI 算法进一步提升网络性能与服务能力，推动传统物流向数字化、智能化转型。

5、附录

5.1 词汇

（1）区块链

区块链是一种分布式账本技术，它允许多个参与者共同维护一个不断增长的数据记录列表，这些记录被称为区块。每个区块包含了一定时间内的交易信息，并通过密码学方法相互链接。区块链的特点是去中心化、不可篡改和透明性，这些特性使其在加密货币、供应链管理、智能合约等多个领域具有广泛的应用潜力。

（2）智能合约

智能合约是一种运行在区块链上的程序，它能够在满足预设条件时自动执行合约条款。智能合约的代码定义了合约的规则和条款，当这些条件被触发时，合约能够自动执行，无需中介参与。智能合约的应用可以简化交易流程，降低成本，并提高效率。

（3）Token

在区块链领域，Token 是一种数字资产，它可以代表各种资产或权益，如货币、股票、忠诚度积分等。Token 可以在区块链上发行、流通和交易，它们可以是同质化的（如加密货币），也可以是非同质化的（如代表独特资产的 NFT）。

（4）IPFS

IPFS 是一个旨在创建一个分布式、去中心化、版本化和点对点的文件存储和共享网络的协议。它通过内容寻址来识别和检索文件，每个文件或数据块都有一个唯一的哈希值（CID），这使得 IPFS 能够提供数据的持久存储和快速检索。IPFS 的目标是构建一个更加开放、快速和安全的互联网，它在数据存储、内容分发和网络架构方面具有革命性的潜力。

（5）DID

去中心化标识符 (DID) 协议是一种标识符，可实现可验证的去中心化数字身份。DID 充当唯一标识符。然后，可验证凭证用于存储和表示与该加密身份绑定的机器可

读凭证。万维网联盟 (W3C) 开发了可验证凭证数据模型，该模型为任何主体（即个人、公司、实物或数字商品，甚至文档）提供了一种在互联网上表达身份凭证的标准方法。

（6）共识算法

共识算法（Consensus Algorithm）是分布式系统中用于确保多个节点在存在网络延迟、节点故障、恶意行为等异常情况下，仍能就某一状态或数据达成一致的机制。它在区块链和分布式系统中尤为重要，因为这些系统依赖于多个节点之间的协作来维护数据的一致性和可靠性。如 DPoS（委托权益证明）、PoS、PBFT、TBFT 等。

5.2 遵循和兼容的国内国际标准

5.2.1 遵循和兼容的国际标准

序号	发布组织	英文名称	中文名称
1	UN/CEFACT 联合国贸易便利化与电子业务中心	NATIONAL TRADE FACILITATION BODIES	全国性贸易便利化机构
2		Unique identification code methodology	唯一标识编码方法（UNIC）
3		Measures facilitate maritime transport documents procedures	海运单证简化程序措施
4		Facilitation of identified legal problems in import clearance procedures	在进口清关程序中确定法律问题的简化措施
5		Authentication of Trade Documents by	用非签署方式对贸

		means other than signature	易单证认证
6		Facilitation Measures related to international Trade procedures	有关国际贸易便利化措施
7		Pre-shipment inspection	装运前检验
8		Establishing a single window	建立单一窗口
9		Data Simplification and Standardization for International Trade	国际贸易数据简化与标准化
10		Establishing a legal framework for international trade Single Window	建立国际贸易单一窗口的法律框架
11		Codes for representation of names of countries	国家名称的代码表示
12		Abbreviations of INCOTERMS	国际贸易术语解释通则缩写
13		Numerical representation of date, time, and periods of time	日期、时间和时间期限的数字表示
14		Alphabetical code for representation of currencies	表示货币的字母代码
15		Codes for ship’s name	船舶名称代码
16		Simpler shipping marks	简单运输标志

17		Codes for ports and other locations	口岸及相关地点代码
18		Payterms-- Abbreviations for terms of payment	付款条款缩写
19		Code for modes of transport	运输方式代码
20		Codes for units of measure used in international Trade	国际贸易计量单位代码
21		Codes for types of cargo, packages and packaging materials	货物、包装以及包装类型代码
22		Freight cost code	运费代码
23		Trade and transport status codes	贸易和运输状态代码
24		Codes for types of means of transport	运输工具类型代码
25		United Nations Layout Key for Trade Documents	联合国贸易单证样式
26		Location of codes in trade documents	贸易单证中代码的位置
27		Codes for representation of names of countries	国家名称的代码表示

28		NATIONAL TRADE FACILITATION BODIES	全国性贸易便利化机构
29		Abbreviations of INCOTERMS	国际贸易术语解释通则缩写
30		Aligned invoice Layout Key for international Trade	国际贸易套合式发票样式
31		Numerical representation of date, time, and periods of time	日期、时间和时间期限的数字表示
32		Unique identification code methodology	唯一标识编码方法（UNIC）
33		Alphabetical code for representation of currencies	表示货币的字母代码
34		Codes for ship’s name	船舶名称代码
35		Documentary aspects of the international transport of dangerous goods	国际危险品运输文件
36		Measures facilitate maritime transport documents procedures	海运单证简化程序措施
37		Facilitation of identified legal problems in import clearance procedures	在进口清关程序中确定法律问题的简

			化措施
38		Authentication of Trade Documents by means other than signature	用非签署方式对贸易单证认证
39		Simpler shipping marks	简单运输标志
40		Codes for ports and other locations	口岸及相关地点代码
41		Payterms-- Abbreviations for terms of payment	付款条款缩写
42		Facilitation Measures related to international Trade procedures	有关国际贸易便利化措施
43		Code for modes of transport	运输方式代码
44		Codes for units of measure used in international Trade	国际贸易计量单位代码
45		Codes for types of cargo, packages and packaging materials	货物、包装以及包装类型代码
46		Layout Key for standard consignment instructions	标准托运指示单证样式
47		Freight cost code	运费代码
48		Trade and transport status codes	贸易和运输状态代

			码
49		UN/EDIFACT	行政、商业和运输业电子数据交换
50		Commercial use of interchange agreements for EDI	电子数据交换用商用交换协议
51		Pre-shipment inspection	装运前检验
52		Codes for types of means of transport	运输工具类型代码
53		UN/EDIFACT-ISO 9735Electronic data interchange for administration, commerce and transport (EDIFACT)--Application level syntax rules	行政、商业和运输业电子数据交换（EDIFACT）应用级语法规则
54		UN/EDIFACT Directories(UNTDID)	联合国贸易数据交换目录
55		Business Requirement Specification (BRS)	业务需求规范
56		Requirement Specification Mapping (RSM)	需求规范映射
57		Core Components Library(UN/CCL)	核心构件库
58		UNTDED-ISO7372Trade data elements directory	贸易数据元目录
59		XML Schema	XML Schema

60		UN/CEFACT Modelling Methodology	UN/CEFACT 建模方法技术规范
61		UN/CEFACT Core Components Technical specifications	UN/CEFACT 核心构件技术规范
62		UN/CEFACT XML Naming and Design Rules	UN/CEFACT XML 命名和设计规则
63		UN/CEFACT UML Profile for Core Components	UN/CEFACT 核心构件的 UML 轮廓
64		UN/CEFACT CCTS Data Type Catalogue	UN/CEFACT 核心构件数据类型目录
65	IMDA	TradeTrust	通商互信标准框架
66	ICC	Uniform Rules for Digital Trade Transactions VERSION 1.0	数字化贸易交易统一规则（URDTT）1.0 版
67		Key Trade Documents and Data Elements Framework	关键贸易单据和数据元素(KTDDE)
68	GLEIF	Legal Entity Identifier	全球法人识别编码(LEI)
69	国际航空运输协会 （IATA）	Dangerous Goods Regulations Certificate	危险品法规（DGR）

70	RosettaNet	RosettaNetImplementation Framework	RosettaNet 实施框架(RNIF)
71	ANSI	ANSI（ANSI ASC X12）	EDI（电子数据交换）文档标准

5.2.2 遵循和兼容的国内标准

序号	标准名称	标准号
1	国际贸易方式代码	GB/T 15421-2008
2	世界各国和地区名称代码	GB/T 2659-2000
3	数据和交换格式信息交换日期和时间表示法	GB/T 7408-2005
4	表示货币和资金的代码	GB/T 12406-2008
5	国际贸易交货条款代码	GB/T 15423-1994
6	运输方式代码	GB/T 6512-1998
7	国际贸易运输船舶名称与代码编制原则	GB/T 18366-2001
8	国际贸易用标准运输标志	GB/T 18131-2000
9	中华人民共和国口岸及相关地点代码	GB/T 15514-2008

10	中国及世界主要海运贸易港口代码	GB/T 7407-2008
11	国际贸易付款方式分类与代码	GB/T 16962-2010
12	国际贸易付款条款的缩略语--PAYTERMS	GB/T 18126-2010
13	国际贸易合同代码编制规则	GB/T 16963-2010
14	国际贸易计量单位代码	GB/T 17295-2008
15	货物类型、包装类型和包装材料类型代码	GB/T 16472-1996
16	运费代码(FCC)运费和其他费用的统一描述	GB/T 17152-2008
17	运输工具类型代码	GB/T 18804-2002
18	国际贸易术语解释通则缩写代码	GB/T 15423-1994
19	EDIFACT 代码表	GB/T 16833-2002
20	HS 商品编码	
21	国际贸易单证样式	GB/T 14392-2009
22	贸易单证中代码的位置	GB/T 14393-2008
23	格式设计 基本样式	GB/T 16832-1997
24	国际贸易单证格式标准编制规则	GB/T 17298-2009
25	国际贸易出口单证格式 第 1 部分： 商业发票	GB/T 15310.1-2009

26	国际贸易出口单证格式 第2部分：装箱单	GB/T 15310.2-2009
27	国际贸易出口单证格式 第3部分：装运通知	GB/T 15310.3-2009
28	第4部分：中华人民共和国出口货物原产地证书	GB/T 15310.4-1994
29	进出口许可证格式 第1部分：进口许可证格式	GB/T 15311.1-2008
30	进出口许可证格式 第2部分：出口许可证格式	GB/T 15311.2-2008
31	贸易数据交换 贸易数据元目录 数据元	GB/T 15191-1997
32	EDIFACT 语法规则第1部分:公用的语法规则	GB/T 14805.1-2007
33	EDIFACT 语法规则第2部分:批式 EDI 语法规则	GB/T 14805.2-2007
34	EDIFACT 语法规则第3部分:交互式 EDI 语法规则	GB/T 14805.3-2007
35	EDIFACT 语法规则第4部分:批式 EDI 服务报告报文	GB/T 14805.4-2007
36	EDIFACT 语法规则第5部分:批式 EDI 安全规则	GB/T 14805.5-2007
37	EDIFACT 语法规则第6部分:安全鉴别和确认报文	GB/T 14805.6-2007
38	EDIFACT 语法规则第7部分:批式 EDI 保密规则	GB/T 14805.7-2007

39	EDIFACT 语法规则第 8 部分:EDI 中的相关数据	GB/T 14805.8-2007
40	EDIFACT 语法规则第 9 部分:密钥和证书管理报文	GB/T 14805.9-2007
41	EDIFACT 语法规则第 10 部分:语法服务目录	GB/T 14805.10-2007
42	EDIFACT 报文设计规则	GB/T 15947-2001
43	EDIFACT 段目录	GB/T 15634-2008
44	EDIFACT 复合数据元目录	GB/T 15635-2008
45	EDIFACT 代码表	GB/T 16833-2002
46	EDIFACT 数据元目录	GB/T 17699-2008
47	EDI 交换协议样本	GB/T 17629-1998
48	中华人民共和国进口许可证报文	GB/T 17302.1-2008
49	中华人民共和国进出口许可证报文	GB/T 17302.2-2008
50	发票报文 第 1 部分 联合国标准发票报文	GB/T 17303.1-1998
51	发票报文 第 2 部分 国际贸易商业发票报文	GB/T 17303.2-1998
52	参与方信息报文	GB/T 18130-2000
53	应用错误与确认报文	GB/T 18128-2000

54	一般用途报文	GB/T 18783-2002
55	订购单变更请求报文	GB/T 17536-1998
56	订购单应答报文	GB/T 17537-1998
57	国际物流政府管理报文	GB/T 17703.2-1999
58	销售数据报告报文	GB/T 17705-1999
59	收货通知报文	GB/T 17231-1998
60	发货通知报文	GB/T 17232-1998
62	订购单报文	GB/T 17233-1998
62	基于 XML 的电子商务 技术体系结构	GB/T 19256.1 -2003
63	基于 XML 的电子商务 协同规程轮廓与协议规范	GB/T 19256.2 -2006
64	基于 XML 的电子商务 消息服务规范	GB/T 19256.3 -2006
65	基于 XML 的电子商务 注册系统信息模型规范	GB/T 19256.4 -2006
66	基于 XML 的电子商务 注册服务规范	GB/T 19256.5 -2006
67	基于 XML 的电子商务 业务过程规范模式	GB/T 19256.6 -2006
68	基于 XML 的电子商务 业务过程构件设计规则	GB/T 19256.7 -2007
69	基于 XML 的电子商务 报文设计规则	GB/T 19256.8 -2007

70	基于 XML 的电子商务 核心构件与业务信息实体规范	GB/T 19256.9 -2006