

Agent2Agent (A2A) 协议

📅 2025年4月12日 ⌚ 7 分钟阅读

#AI #google #A2A #技术

本文介绍了Google公司A2A协议详细解读。

A2A: Google如何用"Kubernetes式思维"重新定义多智能体系统?

在2025年这个被业界称为"多智能体系统(MAS)元年"的时代，Google再次展现了其作为开源界超级大佬的前瞻性，推出了A2A(Agent-to-Agent)框架——这个可能彻底改变MAS生态的游戏规则改变者。

A2A 协议是一种旨在实现人工智能代理之间无缝通信和协作的开放标准。它定义了一套通用的消息传递格式和交互模式，使得不同的AI代理能够相互发现、协商能力、执行任务并共享结果，从而更有效地完成复杂的最终用户请求。该协议旨在促进构建更强大、更通用的代理系统，这些系统可以跨越不同的环境和平台协同工作。

从碎片化到统一：A2A的颠覆性设计理念

想象一下Kubernetes对微服务世界的革命性影响，A2A对MAS领域带来的正是这种级别的范式转变。当前市场上的MAS框架——无论是AutoGen、LangChain、CAMEL还是MetaGPT——都像是一座座孤岛，各自使用专有的通信协议，导致不同框架的智能体根本无法相互发现和协作。这就像早期的容器编排系统，每家都有自己的解决方案，直到Kubernetes出现才统一了江湖。

A2A的核心创新在于它提供了一个**统一的通信协议和交互标准**，并配备了完整的**Agent Orchestrator和管理平台**。这相当于为MAS世界带来了K8s式的集群管理能力，让开发者能够在一个平台上管理和监控所有智能体，无论它们原本属于哪个框架。

目录

文章信息

字数

阅读时间

发布时间

更新时间

标签

#AI #google #A2A #技

MCP协议：A2A生态的"CNI/CSI/CRI"

特别值得关注的是A2A从Day 0就支持的MCP协议——这堪称MAS领域的"基础设施插件标准"。在Kubernetes中，我们有CNI(网络)、CSI(存储)、CRI(容器运行时)等标准接口；而在A2A生态中，MCP协议扮演着类似的角色，负责智能体与各种服务、工具和信息源之间的标准化通信。

这种设计的美妙之处在于它的**可扩展性**：任何将自己的Agent或MAS系统包装成MCP Server的实现，都能无缝接入A2A生态。但Google的野心显然不止于此——MCP只是A2A支持的众多协议之一，其终极目标是成为连接所有MAS框架的"万能胶水"。

2025：MAS元年的天时地利

Google选择在2025年推出A2A绝非偶然。随着Deep Research、PC/Web Operator、Claude Desktop、Manus等创新产品的爆发式增长，MAS技术确实迎来了它的高光时刻。Google联合50家厂商共同推进A2A生态，这种"联盟式"打法不仅展现了其市场号召力，更是一种精心策划的生态占领策略。

相比之下，国内虽然也有ANP等类似尝试，但在影响力和生态建设上确实存在明显差距。作为开源界的"优等生"，Google再次证明了自己定义行业标准的能力——就像当年Android统一移动操作系统、Kubernetes统一容器编排一样，A2A很可能会成为MAS领域的事实标准。

未来展望：当每个Agent都成为A2A公民

A2A的出现预示着MAS发展将进入新阶段：

开发效率革命：再也不用为不同框架的兼容性头疼

资源利用率提升：跨系统的Agent协作成为可能

创新加速：开发者可以专注于业务逻辑而非底层通信

这不禁让人想起Kubernetes早期的发展轨迹——从被质疑到被接受，再到成为行业标配。A2A是否也会沿着同样的路径发展？在MAS元年的背景下，答案很可能是肯定的。

作为技术人，我们或许正在见证一个新时代的开端——当A2A让每个智能体都能自由沟通协作时，真正的分布式人工智能才算是迈出了坚实的一步。Google这次又走在了前面，而我们要做的，就是准备好迎接这场由A2A带来的MAS生态大统一。

A2A 协议是如何工作的？

A2A 协议主要围绕客户端代理和远程代理之间的交互。客户端代理发起任务，而远程代理则执行这些任务并提供结果。这个过程通常包括以下几个关键步骤：**能力发现**（通过代理卡广告代理的能力）、**任务管理**（使用任务对象跟踪任务的生命周期和状态）、**协作**（代理之间交换包含内容和指令的消息）以及 **用户体验协商**（代理协商内容格式和用户界面能力）。对于长时间运行的任务，代理可以通过轮询或推送通知来保持同步。任务的结果以“工件”的形式返回。

什么是“代理卡 (Agent Card)”？它的作用是什么？

代理卡是一个 JSON 格式的文档，远程代理使用它来宣传自身的能力、技能以及认证机制。客户端代理通过查询代理卡的 URL（通常托管在 `/well-known/agent.json` 路径下）来发现潜在的代理。代理卡包含了代理的名称、描述、URL、提供者信息、版本、文档链接、支持的能力（如流式传输和推送通知）、认证要求、默认输入输出模式以及它所拥有的技能列表。客户端代理利用代理卡的信息来识别最适合执行特定任务的代理，并了解如何与其进行通信。

A2A 协议中“任务 (Task)”的概念是什么？

在 A2A 协议中，“任务”是一个有状态的实体，它允许客户端代理和远程代理为了达成特定的结果而进行协作。一个任务由客户端代理创建，其状态由远程代理管理。任务可以立即完成，也可以是长时间运行的。在任务的生命周期中，客户端和远程代理可以通过交换“消息 (Message)”来沟通上下文、指令和状态，而远程代理则将任务的执行结果以“工件 (Artifact)”的形式发送给客户端。任务还可以包含一个可选的会话 ID (sessionId)，用于将多个相关的任务组织在一起。

A2A 协议是如何处理长时间运行的任务和状态更新的？

对于需要较长时间才能完成的任务，A2A 协议支持多种机制来跟踪和获取状态更新。客户端代理可以定期**轮询 (polling)** 远程代理以获取最新的任务状态和工件。此外，如果远程代理支持，它可以利用 **服务器发送事件 (SSE)** 建立持久连接，实时向客户端推送状态更新和工件。A2A 协议还定义了 **推送通知 (push notifications)** 机制，即使在客户端与代理断开连接的情况下，代理也可以通过外部通知服务向客户端发送更新通知。

A2A 协议中“消息 (Message)”和“工件 (Artifact)”有什么区别？

在 A2A 协议中，“消息 (Message)”和“工件 (Artifact)”代表了代理之间交换的不同类型的内容。“消息”包含的是任何非最终结果的内容，例如代理的思考过程、用户上下文、指令、错误或状态信息。一个消息可以包含多个“部分 (Part)”，每个部分可以有不同的内容类型（如文本、文件或数据）。“工件 (Artifact)”则是任务的最终结果，它是不可变的，可以命名，并且也可以包含多个“部分”。例如，一个生成网页的任务可能会产生一个 HTML 工件和一个图像工件。

A2A 协议如何考虑企业级应用的安全性和认证？

A2A 协议强调与现有企业安全基础设施的无缝集成，而不是发明新的安全标准。它将企业级代理视为标准的基于 HTTP 的应用程序，并依赖于企业标准的认证、授权、数据隐私、追踪和监控机制。协议本身通过代理卡传递认证需求（方案和凭据），而实际的凭据协商和传输（例如 OAuth 令牌）则发生在 A2A 协议之外，通常通过 HTTP 头部进行。A2A 服务器需要验证客户端和用户的身份，并基于技能和工具进行细粒度的授权。对于推送通知，A2A 也提供了一些安全建议，例如验证通知 URL 和使用非对称或对称密钥进行签名验证。

Agent2Agent (A2A) 协议学习指南

核心概念

Agent (代理)：能够自主执行任务并与其他 Agent 或用户通信的软件实体。

Client Agent (客户端代理)：负责发起和管理任务，并与远程 Agent 交互的 Agent。

Remote Agent (远程代理)：接收并处理来自客户端 Agent 的任务请求，并返回结果或执行操作的 Agent。

Agent Card (代理卡片)：JSON 格式的元数据，描述了 Agent 的名称、描述、URL、提供者、版本、能力、技能、认证方式以及支持的输入/输出模式等信息。用于 Agent 的发现和能力声明。

Task (任务)：客户端 Agent 请求远程 Agent 完成的特定工作单元。Task 具有状态和生命周期，可以立即完成或长时间运行。

Artifact (工件) : 远程 Agent 执行 Task 后生成的结果, 可以是文本、文件或其他数据形式。Artifact 是不可变的, 并且可以包含多个 Part。

Message (消息) : 客户端 Agent 和远程 Agent 之间交换的非 Artifact 内容, 包括状态更新、指令、错误信息、元数据等。Message 也可以包含多个 Part。

Part (部分) : 构成 Message 或 Artifact 的基本内容单元, 具有特定的内容类型 (如 text、file、data) 和可选的元数据。

Skill (技能) : Agent 具备的特定能力或功能, Agent Card 中会列出 Agent 支持的技能, 包括 ID、名称、描述、标签、示例、输入/输出模式等。

Streaming (流式传输) : 一种通信模式, 允许远程 Agent 在 Task 执行过程中逐步发送状态更新和 Artifact 的 Part, 而无需等待 Task 完全完成。

Push Notification (推送通知) : 一种机制, 允许远程 Agent 在 Task 状态发生变化时, 通过外部通知服务通知客户端 Agent, 即使客户端 Agent 未主动连接。

Agent 发现

Open Discovery (开放发现) : 推荐 Agent 在其域名下的 /.well-known/agent.json 路径托管 Agent Card, 客户端通过 DNS 解析域名并发送 HTTP GET 请求获取。

Curated Discovery (Registry-Based) (策展发现/基于注册表的发现) : 通过中心化的 Agent 目录或注册表来发现 Agent, 注册表由管理员维护和管理。

Private Discovery (API-Based) (私有发现/基于 API 的发现) : 通过自定义的 API 接口来交换和发现 Agent Card。

Securing Agent Cards (保护代理卡片) : Agent Card 可能包含敏感信息, 可以通过身份验证和授权机制进行保护, 例如 mTLS。

核心对象

Task Object (任务对象) : 包含 Task 的唯一 ID、可选的 Session ID、当前状态 (TaskState)、可选的历史记录 (history)、相关的 Artifacts、元数据 (metadata) 等。

Artifact Object (工件对象) : 包含工件的名称、描述、Parts 数组、索引、是否追加、是否为最后一块以及可选的元数据。

Message Object (消息对象) : 包含消息的角色 (role - user 或 agent)、Parts 数组以及可选的元数据。

Part Objects (部分对象) :

TextPart: 包含类型 "text" 和文本内容 "text"。

FilePart: 包含类型 "file" 和文件内容对象 "file" (包含 name, mimeType, bytes 或 uri)。

DataPart: 包含类型 "data" 和任意 JSON 数据 "data"。

PushNotificationConfig Object (推送通知配置对象) : 包含推送通知服务的 URL、可选的令牌 (token) 和认证信息 (authentication - schemes)。

核心方法 (JSON-RPC)

tasks/send: 客户端发送消息以创建新 Task、恢复中断的 Task 或重新打开已完成的 Task。

tasks/get: 客户端检索 Task 的 Artifacts 和可选的历史记录。

tasks/cancel: 客户端取消先前提交的 Task。

tasks/sendSubscribe: 客户端发送消息并订阅 Task 的流式更新。

tasks/resubscribe: 断开连接的客户端重新订阅支持流式传输的远程 Agent 以接收 Task 更新。

tasks/pushNotification/set: 客户端设置接收 Task 状态更改的推送通知配置。

tasks/pushNotification/get: 客户端检索当前为 Task 配置的推送通知配置。

企业就绪

A2A 旨在与现有的企业安全基础设施无缝集成，不创造新的安全标准。

依赖标准的 HTTP 协议进行传输，利用 TLS 进行安全通信。Agent 通过数字证书进行服务器身份验证。客户端和用户身份通过协议外的机制进行管理和传递，通常通过 HTTP 头部。

推荐 Agent 基于技能 (Skills) 和工具 (Tools) 进行授权管理。

推送通知

支持 Agent 在连接和断开连接状态下向客户端发送 Task 更新。

客户端可以通过 Agent Card 查询 Agent 是否支持推送通知。

Agent 在适当的时机发送通知，例如 Task 进入最终状态或需要用户输入时。

Agent 需要验证客户端提供的推送通知 URL 的安全性，例如通过发送挑战请求。通知接收方需要验证接收到的通知的真实性，例如使用非对称密钥 (JWT + JWKS) 或对称密钥进行签名验证。建议实施重放攻击预防机制（例如检查事件时间戳）和密钥轮换机制。

典型流程

Discovery (发现) : 客户端从服务器的 `/.well-known/agent.json` URL 获取 Agent Card。

Initiation (初始化) : 客户端发送包含初始用户消息和唯一 Task ID 的 `tasks/send` 或 `tasks/sendSubscribe` 请求。

Processing (处理) : (Streaming): 服务器作为 Task 进行发送 SSE 事件 (状态更新、工件) 。

(Non-Streaming): 服务器同步处理 Task 并在响应中返回最终的 Task 对象。

Interaction (Optional) (交互 - 可选) : 如果 Task 进入 `input-required` 状态, 客户端使用相同的 Task ID 通过 `tasks/send` 或 `tasks/sendSubscribe` 发送后续消息。

Completion (完成) : Task 最终达到终端状态 (`completed`, `failed`, `canceled`)。

简答题 (Quiz)

什么是 Agent Card? 它在 A2A 协议中扮演什么角色?

简述 A2A 协议中 Task 的生命周期, 并列举至少三种可能的 Task 状态。

Artifact 和 Message 在 A2A 协议中有什么区别? 它们各自包含哪些基本元素?

解释 A2A 协议中的“Streaming”特性及其优势。

描述 A2A 协议中推荐的 Agent 发现机制——“Open Discovery”是如何工作的?

A2A 协议是如何处理 Agent 之间的身份验证的? 是否在协议层面定义了具体的认证方式?

什么是 Skill? Agent Card 中如何描述 Skill? Skill 在授权管理中有什么作用?

简述 A2A 协议中推送通知的基本流程。Agent 在发送推送通知时需要考虑哪些安全因素?

描述 `tasks/send` 和 `tasks/get` 这两个核心方法的主要功能。

在 A2A 协议中, 如果一个 Task 需要用户提供额外的输入才能继续执行, Agent 会如何通知 Client? Client 又该如何响应?

简答题答案 (Answer Key)

Agent Card 是一个 JSON 格式的元数据文档, 用于描述 Agent 的基本信息、能力、技能和认证方式。它在 A2A 协议中扮演着 Agent 广告自身能力、供客户端发现和选择合适 Agent 的关键角色。

Task 由客户端创建并发送给远程 Agent，经历不同的状态，如 submitted（已提交）、working（工作中）、input-required（需要输入）、completed（已完成）、failed（失败）、canceled（已取消）等。Task 的状态由远程 Agent 决定。

Artifact 是远程 Agent 执行 Task 后生成的结果，代表任务的产出，是不可变的。Message 是 Agent 之间交换的非结果内容，用于通信状态、指令、错误等。两者都可以包含多个 Part 作为内容单元。

“Streaming”是一种流式传输模式，允许远程 Agent 在 Task 执行过程中逐步发送状态更新 (TaskStatusUpdateEvent) 和 Artifact 的部分内容 (TaskArtifactUpdateEvent)。这种方式可以提高响应速度，尤其对于长时间运行或生成大量数据的任务。

“Open Discovery”推荐 Agent 在其服务器域名下的 /.well-known/agent.json 路径托管 Agent Card。客户端通过 DNS 解析得到服务器 IP 地址，然后向该特定路径发送 HTTP GET 请求来获取 Agent Card。

A2A 协议本身并不定义具体的 Agent 身份验证方式。它通过 Agent Card 的 authentication 字段声明 Agent 支持的认证方案（如 OAuth2、Bearer 等），具体的认证流程需要在协议之外进行，认证凭据通常通过 HTTP 头部传递。

Skill 是 Agent 具备的特定能力或功能。Agent Card 的 skills 数组中包含了 Skill 的 ID、名称、描述、标签、输入/输出模式等信息。在授权管理中，Agent 可以基于 Skill 来限制客户端的访问权限，例如只允许具有特定 OAuth Scope 的客户端调用某个技能。

推送通知的基本流程是客户端在发送 Task 时或通过 tasks/pushNotification/set 方法配置推送通知的 URL 和认证信息，远程 Agent 在 Task 状态变化时向该 URL 发送通知。Agent 需要验证 URL 的合法性，并考虑使用签名等方式确保通知的安全性。

tasks/send 方法用于客户端向远程 Agent 发送消息，可以创建新的 Task、恢复中断的 Task 或重新打开已完成的 Task。tasks/get 方法用于客户端向远程 Agent 请求获取指定 Task 的 Artifacts，还可以选择获取 Task 的历史消息。

如果 Task 进入 input-required 状态，远程 Agent 会在 Task 的 status 字段中将 state 设置为 input-required，并且通常会在 message 字段中包含一个描述需要用户提供的具体信息的 Message 对象。客户端需要解析这个 Message，获取所需信息，然后通过 tasks/send 或 tasks/sendSubscribe 方法发送包含必要输入的新消息以恢复 Task 的执行。

论述题

探讨 A2A 协议在构建复杂的、跨多个 AI Agent 协作的智能应用方面的潜力与挑战。请结合 Agent 发现、任务管理和数据交换等方面进行分析。

分析 A2A 协议中 Agent Card 的设计对于实现 Agent 的互操作性和可发现性的重要性。你认为

Agent Card 未来可能需要包含哪些额外的信息？比较和对比 A2A 协议中同步 (non-streaming) 和异步 (streaming 和 push notification) 的任务处理方式，并讨论在不同应用场景下选择哪种方式更为合适。讨论 A2A 协议在企业环境中的应用前景，并分析其在企业就绪性（安全性、身份验证、授权、监控等方面）方面需要考虑的关键因素。基于你对 A2A 协议的理解，设计一个具体的应用场景，描述 Client Agent 如何利用 A2A 协议与多个 Remote Agent 协作完成一个复杂的任务。请详细说明 Agent 之间的交互流程、数据格式和可能的错误处理机制。

术语表

A2A (Agent-to-Agent): Agent 之间的通信协议的简称。

API (Application Programming Interface): 应用程序编程接口，允许不同的软件组件进行交互的一组规则和规范。

Artifact (工件): Task 执行后生成的结果数据。

Agent (代理): 能够自主执行任务并与其他系统交互的软件实体。

Agent Card (代理卡片): 描述 Agent 元数据的 JSON 文档。

Authentication (身份验证): 验证用户或 Agent 身份的过程。

Authorization (授权): 确定已验证身份的用户或 Agent 是否具有执行特定操作或访问特定资源的权限。

Client (客户端): 在 A2A 协议中，通常指发起 Task 请求的 Agent。

JSON (JavaScript Object Notation): 一种轻量级的数据交换格式。

JSON-RPC: 一种无状态的、轻量级的远程过程调用 (RPC) 协议，使用 JSON 作为数据格式。

Metadata (元数据): 描述数据的数据。

Message (消息): Agent 之间交换的通信内容，不包括最终结果 (Artifact)。

OAuth (Open Authorization): 一种开放标准的授权协议，允许用户授权第三方应用访问其在另一服务上存储的信息，而无需将凭据泄露给第三方应用。

OpenAPI: 一种用于描述、生产、消费和可视化 RESTful Web 服务的规范格式。

Part (部分): 构成 Message 或 Artifact 的内容单元。

Push Notification (推送通知): 一种将信息从服务器主动发送到客户端的技术。

Remote Agent (远程代理): 接收并处理来自客户端的 Task 请求的 Agent。

Schema (模式): 定义数据结构和约束的蓝图。

SDK (Software Development Kit): 软件开发工具包, 包含开发应用程序所需的工具、库和文档。

Server-Sent Events (SSE): 一种服务器推送技术, 允许服务器通过 HTTP 连接单向地向客户端发送事件流。

Skill (技能): Agent 具备的特定能力或功能。

Streaming (流式传输): 逐步传输数据的过程。

Task (任务): 客户端请求远程 Agent 执行的工作单元。

TLS (Transport Layer Security): 传输层安全协议, 用于在网络通信中提供加密和数据完整性。

URL (Uniform Resource Locator): 统一资源定位符, 用于标识互联网上的资源。

URI (Uniform Resource Identifier): 统一资源标识符, 用于标识资源。

Agent2Agent (A2A) 协议简报

概述

Agent2Agent (A2A) 协议是一项由 Google 发起的开源项目, 旨在标准化 AI 代理之间的通信方式, 实现跨系统、跨平台的代理互操作性。该协议定义了一套通用的消息格式、交互模式和发现机制, 使得不同的 AI 代理能够协同工作, 共同完成复杂的任务, 从而提升生产力, 自动化流程并增强用户体验。

核心概念与主题

代理发现 (Agent Discovery):

A2A 协议通过标准化“代理卡 (Agent Card)”的 JSON 格式来描述代理的能力、技能和认证机制, 使得客户端代理能够找到最适合执行特定任务的远程代理。

开放发现 (Open Discovery): 推荐企业将代理卡托管在 <https://DOMAIN/.well-known/agent.json> 这个标准路径下, 客

户端可以通过 DNS 解析域名并发送 HTTP GET 请求获取代理卡。这使得 Web 爬虫和应用程序能够轻松发现代理。

策展发现 (Curated Discovery - Registry-Based): 设想存在由管理员维护的公司或团队特定的代理注册中心 (“代理目录”或私有市场)。协议正在考虑增加对注册中心的支持。

私有发现 (Private Discovery - API-Based): 存在通过自定义 API 交换代理卡的私有“代理商店”或专有代理。A2A 目前不将私有发现 API 作为其关注点。

安全: 代理卡可能包含敏感信息, 因此建议实施认证和授权机制来保护代理卡, 例如使用 mTLS 限制对特定客户端的访问。注册中心和私有发现 API 也应需要身份验证。

任务管理 (Task Management):

A2A 的通信以任务完成为导向。任务 (Task) 是一个有状态的实体, 允许客户端和远程代理协作以实现特定结果并生成结果 (Artifacts, 制品)。

任务由客户端创建, 状态由远程代理决定。客户端可以选择性地设置 sessionId 将多个任务关联到同一个会话。

远程代理可以立即完成请求、安排后续工作、拒绝请求、协商不同的模式、请求更多信息或委托给其他代理和系统。

即使在完成目标后, 客户端也可以在同一任务的上下文中请求更多信息或更改 (例如, “画一只兔子”, 然后“把它变成红色”)。

任务用于传输 Artifacts (结果) 和 Messages (想法、指令等)。任务维护状态和可选的历史记录。

Task 对象关键属性: id (唯一标识符), sessionId, status (状态), history (消息历史), artifacts (生成的结果), metadata (元数据)。

协作 (Collaboration):

代理之间通过 Messages 进行通信, 消息可以包含代理的想法、用户上下文、指令、错误、状态或元数据。

Artifacts 是代理执行任务后生成的结果, 是不可变的, 可以命名, 并且可以包含多个 Parts (部分)。流式响应可以向现有 Artifacts 追加 Parts。

Part 是消息或 Artifact 中交换的完整内容片段, 每个 Part 都有自己的内容类型和元数据。支持的 Part 类型包括 text (文本, TextPart), file (文件, FilePart, 支持 bytes - base64 编码或 uri), data (任意数据, DataPart)。

Message 对象关键属性: role (“user” 或 “agent”), parts (内容片段数组), metadata (元数据)。

Artifact 对象关键属性: name, description, parts, metadata, index (用于流式传输), append (是否追加), lastChunk (是否最后一块)。

用户体验协商 (User Experience Negotiation):

每个消息都包含 parts，允许客户端和远程代理协商所需内容的正确格式。

在任务生命周期内支持动态 UX 协商，例如代理在对话中途添加音频/视频。

流式支持 (Streaming Support):

对于支持流式传输的客户端和远程代理，客户端可以使用 tasks/sendSubscribe 在创建新任务时建立流式连接（基于 Server-Sent Events - SSE）。

远程代理可以发送 TaskStatusUpdateEvent (状态更新或指令/请求) 和 TaskArtifactUpdateEvent (流式传输结果)。

TaskArtifactUpdateEvent 可以向现有 Artifacts 追加新的 Parts。

代理需要在流结束或需要额外用户输入时设置 final: true 属性。

断开连接的客户端可以使用 tasks/resubscribe 重新订阅以接收任务更新。

非文本媒体 (Non-textual Media):

A2A 支持在消息和 Artifacts 中包含非文本数据，例如图像、视频等，通过 file 类型的 Part 实现，可以包含 base64 编码的 bytes 或 uri。

结构化输出 (Structured Output):

客户端或代理可以请求对方提供结构化输出。这可以通过在 Part 的元数据中指定 mimeType 为 application/json 并包含 schema 来实现。

推送通知 (Push Notifications):

A2A 支持安全的通知机制，代理可以通过 PushNotificationService 在连接断开的情况下通知客户端更新。

客户端可以在代理卡中查看代理是否支持 pushNotifications 功能。

客户端可以在 tasks/send 或通过 tasks/pushNotification/set 方法设置任务的推送通知配置，包括 url 和 authentication (例如 bearer token)。

安全: 代理不应盲目信任客户端提供的推送通知 URL，建议通过发送带有 validationToken 的 GET 挑战请求来验证 URL 的有效性。还可以要求通知服务使用预先确定的密钥签名

validationToken 来进一步验证身份。通知接收者也应验证通知的真实性，例如检查 JWT 签名或使用对称密钥进行验证。建议实施重放保护机制，例如检查事件时间戳。密钥轮换对于保证安全性也很重要。

错误处理 (Error Handling):

服务器在处理客户端请求遇到错误时，会以 ErrorMessage 格式进行响应，其中包含 code (错误代码), message (错误描述) 和可选的 data。

A2A 采用标准的 JSON-RPC 错误代码，并定义了一些特定于 A2A 的错误代码，例如 -32001 (Task not found), -32003 (Push notifications not supported), -32005 (Incompatible content types)。

企业就绪 (Enterprise Readiness):

A2A 旨在无缝集成到现有企业基础设施中，不发明新的安全标准，而是依赖于标准的 HTTP 安全机制。

传输层安全 (Transport Level Security): 强制使用 TLS (版本 1.2 或更高版本) 进行通信，并支持行业标准 TLS 密码套件。

服务器身份 (Server Identity): 服务器通过由知名证书颁发机构签名的数字证书在 TLS 握手期间提供其身份，客户端应验证服务器身份。

客户端和用户身份 (Client and User Identity): A2A 模式中没有用户或客户端标识符的概念。相反，A2A 通过协议传达认证要求（方案和凭据）。客户端负责（在 A2A 之外）与适当的身份验证机构协商并检索/存储凭据材料（如 OAuth 令牌），这些材料将通过 HTTP 标头而不是 A2A 负载传递。建议客户端在请求中始终提供客户端身份（代表客户端代理）和用户身份（代表其用户）。A2A 支持在 INPUT-REQUIRED 状态下请求客户端提供额外的身份验证信息。

客户端认证 (Authenticating Clients): 服务器应发布其支持的身份验证方案（例如通过 HTTP 401 响应中的 WWW-Authenticate 标头或 OIDC 发现文档）。服务器可以根据用户和应用程序身份验证请求，并使用标准的 HTTP 响应代码拒绝或质询请求。

授权和数据隐私 (Authorization and Data Privacy): 服务器应基于用户和应用程序身份授权请求。建议代理至少在两个维度上管理访问: Skills (按技能授权，例如使用 OAuth 范围限制对特定技能的访问) 和 Tools (通过工具限制对敏感数据或操作的访问，代理需要基于应用程序+用户权限授权访问工具)。

核心对象 (Core Objects):

AgentCard: 描述代理的元信息、能力、技能和认证需求。

Task: 代表客户端和远程代理之间为完成特定目标而进行的交互。

Artifact: 任务完成后的结果。

Message: 客户端和代理之间交换的任何非 Artifact 内容。

Part: Message 或 Artifact 的组成部分，包含实际内容和类型信息。

PushNotificationConfig: 配置任务的推送通知。

通信流程 (Typical Flow):

发现 (Discovery): 客户端从服务器的 well-known URL 获取代理卡。

启动 (Initiation): 客户端发送包含初始用户消息和唯一任务 ID 的 tasks/send 或 tasks/sendSubscribe 请求。

处理 (Processing):(流式): 服务器在任务进行过程中发送 SSE 事件 (状态更新、制品) 。

(非流式): 服务器同步处理任务并在响应中返回最终的任务对象。

交互 (Interaction - 可选): 如果任务进入 input-required 状态, 客户端使用相同的任务 ID 通过 tasks/send 或 tasks/sendSubscribe 发送后续消息。

完成 (Completion): 任务最终达到终端状态 (completed, failed, canceled) 。

未来计划 (What's next)

协议增强:正式将授权方案和可选凭据直接包含在 AgentCard 中。

研究 QuerySkill() 方法以动态检查不支持或未预料到的技能。

支持任务 内部 的动态 UX 协商 (例如, 代理在对话中途添加音频/视频) 。

探索扩展对客户端发起方法 (超出任务管理) 的支持。

改进流式传输可靠性和推送通知机制。

示例与文档增强:简化 "Hello World" 示例。

包含与不同框架集成或展示特定 A2A 功能的更多代理示例。

为常见的客户端/服务器库提供更全面的文档。

从 JSON Schema 生成人类可读的 HTML 文档。

参与贡献 (Contributing)

A2A 协议是一个开放源代码项目，欢迎社区贡献。可以通过以下方式参与：

阅读贡献指南。

在 GitHub Discussions 中提问。

在 GitHub Issues 中提供协议改进反馈。

使用 Google 表单发送私有反馈。

合作伙伴反馈 (Feedback from our A2A partners)

许多行业领先的公司（如 DataStax, Datadog, Elastic, JFrog, LabelBox, LangChain, MongoDB, Neo4j, New Relic, Pendo, PayPal, SAP, Salesforce, Supertab, UKG, Weights & Biases 以及 EPAM, HCLTech, KPMG, Quantiphi, TCS 等服务合作伙伴）都对 A2A 协议表示了积极的支持和参与，认为 A2A 是实现 AI 系统真正互操作性的重要一步，能够克服当前的集成挑战，推动下一代智能代理应用程序的发展。

结论

Agent2Agent (A2A) 协议为构建可互操作的 AI 代理生态系统奠定了基础。通过标准化代理发现、任务管理和通信流程，A2A 有望促进更强大、更灵活的代理系统的发展，从而在各个领域实现更高的自动化水平和更智能的解决方案。该协议的开源性质和积极的社区参与将是其未来成功的关键。

ADK

参考

[Announcing the Agent2Agent Protocol \(A2A\)](#)

[Building the industry's best agentic AI ecosystem with partners](#)

[A2A Github](#)

[Google's Agent2Agent \(A2A\) Protocol: A New Era of AI Agent Collaboration and Its Organizational Impact](#)

[A2A 协议中文文档](#)

[A2A 协议英文文档](#)

分享这篇文章



相关文章推荐

AI Agent Gateway

AI Agent Gateway

Llama 4 模型系列

本文介绍了Llama 4 模型系列详...

Chain of Draft 论文..

本文介绍了Chain of Draft (CoD...