目前，AI 领域有三大巨头在 Agent 生态中积极布局：

- **Anthropic**：推出了 **MCP**，旨在标准化 AI 大模型与外部工具和数据源的交互
- **Google**：推出了 **A2A**，专注于 Agent 之间的通信和协作。
- **OpenAI**：早在2023年就推出了 **Function Calling**，为大模型提供了工具调用功能。

这三大巨头的举措，仿佛是在为 AI Agent 的发展铺设一条从个体到集体的进化之路。从大模型本身，到为大模型添加工具调用功能，再到大模型与工具的交互标准，最后到 AI Agent 之间的通信协议，这一系列的发展就像是**为一个聪明的大脑逐步武装四肢，赋予多种能力，最终使其能够协作完成复杂任务，形成一个高效的团队。**

接下来，我们对 MCP、A2A 和 Function Calling 进行全面的解读与对比，探讨它们之间的具体区别以及如何实现合作。

1、Function Calling：直接但缺乏扩展性

Function Calling 是由 OpenAI 等公司推动的一种技术，它允许大语言模型（LLM）通过自然语言指令与外部工具和服务进行交互，从而将自然语言转换为具体的 API 调用。这一技术解决了大语言模型在训练完成后知识更新停滞的问题，使大模型能够获取实时信息，比如：当前的天气、股市收盘点数等。

第一、工作原理

Function Calling 的工作原理可以通过**以下4个步骤来理解**：

1、识别需求：大模型识别出用户的问题需要调用外部 API 来获取实时信息。比如：用户询问“今天北京的天气如何？”大模型会识别出这是一个关于实时天气的问题。

2、选择函数：大模型从可用的函数库中选择合适的函数。在这个例子中，大模型会选择 get_current_weather 函数。

3、准备参数：大模型准备调用函数所需的参数。例如：

```
{  "location": "北京",  "unit": "celsius"}
```

3、调用函数：AI 应用使用这些参数调用实际的天气 API，获取北京的实时天气数据。

4、整合回答：大模型将获取的数据整合成一个完整的回答，比如：“根据最新数据，北京今天的天气晴朗，当前温度23°C，湿度45%，微风。今天的最高温度预计为26°C，最低温度为18°C。”

第二、对开发者的好处

对于开发者来说，使用 LLM 的 Function Calling 入门相对容易。开发者只需按照 API 的要求定义函数规格（通常是 JSON 格式），并将其随 Prompt 请求发送给大模型。大模型会根据需要调用这些函数，整个逻辑相当直观。因此，对于单一大模型、少量功能的简单应用，**Function Calling 的实现非常直接，几乎可以“一键”将大模型输出对接到代码逻辑中。**

第三、局限性

然而，Function Calling 也有**一些局限性**：

缺乏跨大模型的一致性：每个 LLM 供应商的接口格式略有差异，这使得开发者在支持多个大模型时需要为不同的 API 做适配，或者使用额外的框架来处理这些差异。

平台依赖性：Function Calling 通常依赖于特定的平台或框架，这限制了其在不同环境中的通用性。

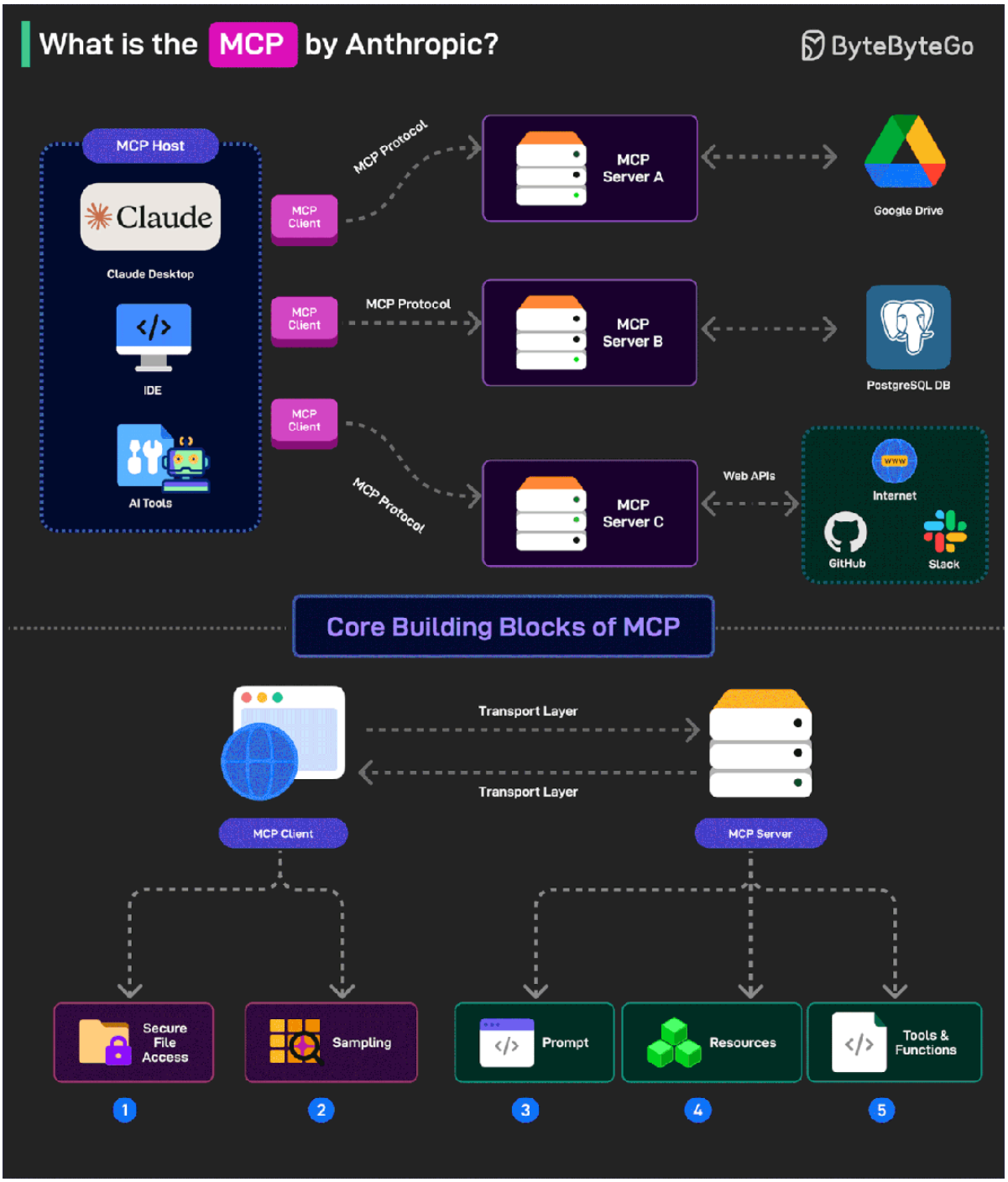
扩展性有限：虽然 Function Calling 能够解决特定问题，但在面对更复杂的任务时，其扩展性可能会受到限制。开发者可能需要为每个新功能编写新的函数，并确保这些函数与模型的交互逻辑兼容。

第四、总结

Function Calling 是一种强大的工具，它为大语言模型提供了与外部工具和服务交互的能力，从而解决了大模型知识更新停滞的问题。然而，它的局限性在于缺乏跨模型的一致性和平台依赖性。尽管如此，Function Calling 仍然是一个重要的技术，尤其是在需要快速实现特定功能时。未来，随着技术的不断发展，我们期待看到更多能够克服这些局限性的解决方案。

2、MCP：构建 AI 应用与外部工具的桥梁

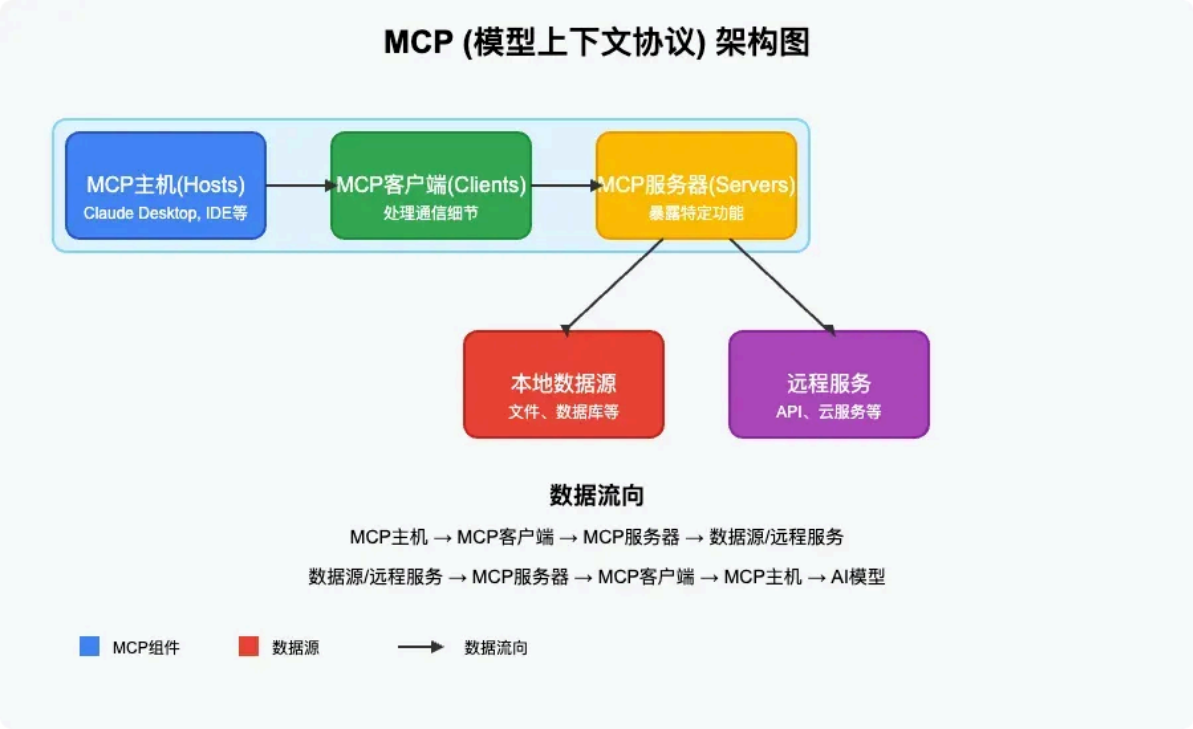
MCP（Model Context Protocol）是由 Anthropic 公司提出的一种协议，旨在解决不同大语言模型（LLM）与不同外部工具集成的标准化问题。通过MCP，开发者能够以一种统一的方式将各种数据源和工具连接到 AI 大模型，从而提升大模型的实用性和灵活性。



目前，MCP 生态已经得到了广泛的支持，包括 Anthropic 的 Claude 系列、OpenAI 的 GPT 系列、Meta 的 Llama 系列、DeepSeek、阿里的通义系列以及 Anysphere 的 Cursor 等主流模型均已接入 MCP 生态。

第一、MCP 的架构设计

MCP 采用了客户端-服务器架构，主要包括以下几个核心组件：



1、MCP 主机（Hosts）

角色：这是需要访问数据的程序，例如Claude Desktop、各种IDE或AI工具。

功能：它们是MCP生态系统的入口点，负责向用户提供AI功能，并作为用户与AI模型之间的桥梁。

2、MCP 客户端（Clients）

角色：这些是协议客户端，负责维持与 MCP 服务器的1:1连接。

功能：它们处理通信细节，确保主机和服务器之间的数据传输顺畅，从而实现高效的数据交互。

3、MCP 服务器（Servers）

角色：这些是轻量级程序，每个服务器都通过标准化的 Model Context Protocol 暴露特定功能。

功能：服务器是 MCP 的核心，它们连接 AI 大模型与实际数据源，使模型能够访问和操作数据。

4、数据源

本地数据源：包括您计算机上的文件、数据库和服务，MCP 服务器可以安全地访问这些资源。

远程服务：通过互联网可用的外部系统（比如：通过 API），MCP 服务器可以连接这些系统，从而扩展模型的能力。

第二、MCP 的优势

统一性：MCP 提供了一个统一的协议标准，使得不同 AI 大模型能够以一致的方式连接到各种数据源和工具，从而避免了平台依赖性问题。

安全性：通过 MCP，数据的传输和访问过程更加安全，敏感数据可以保留在本地，无需全部上传到云端。

灵活性：MCP 支持多种数据源和工具的连接，无论是本地资源还是远程服务，都可以轻松集成到AI 应用中。

生态丰富：MCP 生态已经得到了广泛的支持，开发者可以利用现有的MCP服务器和工具，快速构建和部署AI应用。

第三、总结

MCP 通过其客户端-服务器架构和标准化的协议，为 AI 大模型与外部工具和数据源的集成提供了一个高效、安全且灵活的解决方案。它不仅解决了不同大模型与工具之间的兼容性问题，还为开发者提供了一个丰富的生态系统，使得AI应用的开发和部署变得更加简单和高效。

三、3、A2A：助力 Agent 间的通信与协同

谷歌最新推出的 A2A（Agent2Agent）开放协议，专注于解决不同 Agent 之间的通信和协同问题，旨在构建一个更加灵活和高效的多 Agent 系统。

要深入理解A2A协议，我们首先需要掌握几个关键概念：

第一、关键概念

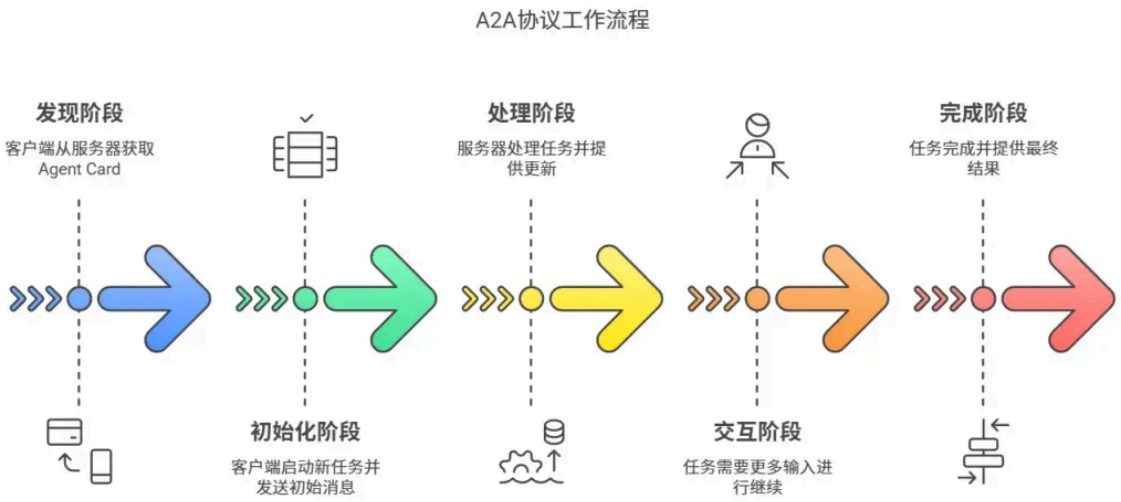
A2A Client：类似于点餐的顾客，负责向 A2A Server 发送请求，启动任务。

A2A Server：类似于餐厅的服务员和厨师团队，负责处理请求并返回响应，告知任务的状态。

任务状态：任务在执行过程中可能会经历多个状态，比如：已提交、处理中、需要输入等，最终完成或失败。

第二、典型工作流程

A2A 协议的典型工作流程可以分为以下**几个步骤**：



1、请求发送：A2A Client 向 A2A Server 发送请求，启动一个任务。这个请求包含了任务的详细信息和所需的操作。

2、请求处理：A2A Server 接收到请求后，开始处理任务，并返回一个初始响应，告知任务的当前状态。

3、状态更新：任务在执行过程中会经历多个状态变化。A2A Server 会定期更新任务状态，并将这些状态信息反馈给 A2A Client。

4、任务完成或失败：任务最终会完成或失败。A2A Server 会将最终结果返回给 A2A Client，告知任务的执行结果。

第三、A2A 的优势

灵活性：A2A 协议允许不同 Agent 之间的动态通信和协同，使得系统能够灵活应对各种复杂任务。

扩展性：通过标准化的通信机制，A2A 协议支持多 Agent 系统的扩展，可以轻松添加新的 Agent 或服务。

任务管理：A2A 协议提供了丰富的任务状态管理功能，使得任务的执行过程更加透明和可控。

协同能力：A2A 协议促进了 Agent 之间的协作，使得多个 Agent 可以共同完成复杂的任务，提高系统的整体效率。

第四、总结

A2A 协议通过其灵活的通信机制和强大的任务管理功能，为不同 Agent 之间的协同工作提供了一个高效、透明的解决方案。它不仅解决了 Agent 之间的通信问题，还提升了多 Agent 系统的整体性能和扩展性。随着技术的不断发展，A2A 协议有望在更多领域得到广泛应用，推动 AI 技术的发展。

4、MCP vs Function Calling vs A2A 关系

第一、MCP ↔ Function Calling 关系：设计理念与应用场景的差异

尽管 MCP 和 Function Calling 都旨在促进大语言模型（LLM）与外部工具和服务的交互，但它们在设计理念和应用场景上存在显著差异，尤其是在可扩展性方面。

1、Function Calling 的局限性

Function Calling 由于缺乏统一标准，不同 LLM 需要各自的函数定义格式。如果有 M 个不同 LLM 应用和 N 个不同工具/服务，理论上可能需要实现 M×N 次重复的对接工作。此外，Function Calling 本身并不直接支持多步调用组合，大模型只能一次调用一个函数，获取结果后如果需调用下一个函数，需要由应用逻辑将结果馈入大模型下一轮对话，再触发下一个函数调用。虽然在原理上可以实现函数输出作为输入形成链条，但这一切需要开发者在应用层精心编排，大模型自身缺乏对跨调用流程的全局观。

2、MCP 的扩展性优势

MCP 的扩展性则通过统一的接口标准，将复杂的 M（个模型）×N（个外部工具对接）问题转化为 M+N 的问题。工具创建者只需为每个工具/系统实现一次 MCP Server，应用开发者只需为每个应用实现一次 MCP Client，各自遵循通用协议即可协同工作，扩展新功能的边际成本大幅降低。

第二、MCP ↔ A2A 关系：能力互补

那么，为什么在有了 MCP 之后，还需要 A2A 来协作不同 Agent 呢？对比 MCP 与 A2A，可以发现两者的关系更多是一种能力的互补：MCP 让 Agent 能够使用工具，而 A2A 让 Agent 能够与其他 Agent 协作。**一个解决“做什么”，一个解决“与谁合作”。**

背后的逻辑就像上班，有的同事（Agent）擅长研发汽车发动机，有的同事（Agent）擅长组装。所有人通过一个流水线串联共同完成一个项目，一定比一个同事（Agent）独自研发汽车，然后再组装并营销的效率更高。

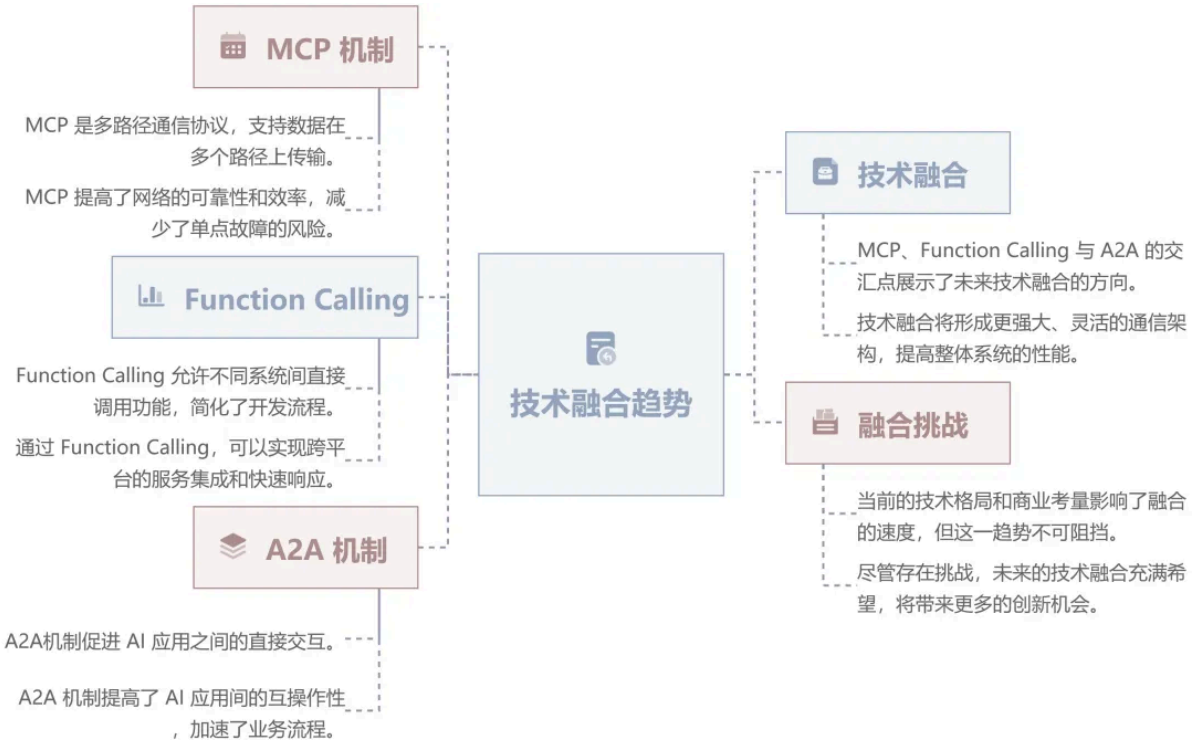
第三、A2A *↔* Function Calling 关系：能力协同

A2A 可以支持 Agent 之间的通信，而每个 Agent 可以通过 Function Calling 调用外部工具。

这种结合可以实现复杂的任务分配和协作，提升系统的整体性能。

第四、未来趋势：技术融合

长期来看，我们可能会看到这三大通信机制（Function Calling、MCP、A2A）逐渐融合的趋势。不过，目前 OpenAI 和 Anthropic 尚未支持 A2A。这可能是因为，尽管大家在技术布道时都有自己的理念，但最终如何选择取决于商业决策。然而，从长期来看，技术融合之路势在必行。


MCP 机制
MCP 是多路径通信协议，支持数据在多个路径上传输。
MCP 提高了网络的可靠性和效率，减少了单点故障的风险。
Function Calling
Function Calling 允许不同系统间直接调用功能，简化了开发流程。
通过 Function Calling，可以实现跨平台的服务集成和快速响应。
A2A 机制
A2A机制促进 AI 应用之间的直接交互。
A2A 机制提高了 AI 应用间的互操作性，加速了业务流程。
技术融合趋势
技术融合
MCP、Function Calling 与 A2A 的交汇点展示了未来技术融合的方向。
技术融合将形成更强大、灵活的通信架构，提高整体系统的性能。
融合挑战
当前的技术格局和商业考量影响了融合的速度，但这一趋势不可阻挡。
尽管存在挑战，未来的技术融合充满希望，将带来更多的创新机会。