Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

Get Started

MCP 简介

开始使用 Model Context Protocol (MCP)

Model Context Protocol (MCP) 是一个开放协议，它使 LLM 应用与外部数据源和工具之间的无缝集成成为可能。无论你是构建 AI 驱动的 IDE、改善 chat 交互，还是构建自定义的 AI 工作流，MCP 提供了一种标准化的方式，将 LLM 与它们所需的上下文连接起来。

On this page
开始使用 MCP
开发工具
探索 MCP
贡献

开始使用 MCP

选择最适合你需求的路径：

快速入门
体验 MCP 的最快方式 — 将示例服务器连接到 Claude Desktop

构建你的第一个服务器 (Python)
在 Python 中创建一个简单的 MCP 服务器以了解基础知识

构建你的第一个服务器 (TypeScript)
在 TypeScript 中创建一个简单的 MCP 服务器以了解基础知识

开发工具

构建和调试 MCP 服务器的必备工具：

Debug 指南
了解如何有效地调试 MCP 服务器和集成

MCP Inspector
使用我们的交互式调试工具来测试和检查你的 MCP 服务器

探索 MCP

深入了解 MCP 的核心概念和功能：

核心架构
了解 MCP 如何连接 clients、servers 和 LLMs

Resources
从你的 server 向 LLMs 暴露数据和内容

Prompts
创建可复用的提示模板和工作流

Tools
让 LLM 通过你的 server 调用工具

Sampling
让你的 server 调用 LLM

传输层
了解 MCP 的通信机制

贡献

想贡献代码？在 GitHub 上查看 **@modelcontextprotocol**，加入我们不断增长的开发者社区，与 MCP 一起构建。

Was this page helpful? Yes No

Documentation
Python SDK
TypeScript SDK
概览

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

Get Started

快速入门

在 5 分钟内开始使用 MCP

MCP 是一个协议，它可以在 host 应用（如 **Claude Desktop**）和本地服务之间建立安全连接。在本快速入门指南中，你将学习如何：

- 设置本地 SQLite 数据库
- 通过 MCP 将 Claude Desktop 连接到该数据库
- 安全地查询和分析你的数据

> 虽然本指南重点使用 Claude Desktop 作为 MCP 主机示例，但该协议是开放的，可以被任何应用程序集成。例如，AI 工具开发者都可参考可以使用 MCP 以连接其现有系统的集成。

> Claude Desktop 的 MCP 支持目前处于开发者预览阶段。仅支持连接到运行在你机器上的本地 MCP 服务器。由于不同应用 MCP 思想，此集成仅在 Claude Desktop 应用程序中可用，而不适用于 Claude 的网页界面 (claude.ai)。

MCP 如何工作

MCP (Model Context Protocol) 是一个开放协议，它可以在 AI 应用程序与本地或远程资源之间实现安全、受控的交互。让我们来解析一下它的工作原理，然后看看我们将在本指南中如何使用它。

通用架构

MCP 的核心是一个 client-server 架构，host 应用程序可以连接到多个服务器：

- **MCP Hosts**: 像 Claude Desktop、IDEs 或 AI 工具这样的程序，它们希望通过 MCP 访问资源
- **MCP Clients**: 维护与服务器 1:1 连接的协议客户端
- **MCP Servers**: 轻量级程序，通过标准化的 Model Context Protocol 暴露特定功能
- **Local Resources**: 你的计算机资源（数据库、文件、服务），MCP 服务器可以安全地访问这些资源
- **Remote Resources**: 通过互联网可用的资源（例如，通过 APIs），MCP 服务器可以连接到这些资源

在本指南中

对于本快速入门，我们将使用 SQLite 实现一个集中示例：

1. Claude Desktop 作为我们的 MCP 客户端
2. 一个 SQLite MCP 服务器提供安全的数据库访问
3. 你的本地 SQLite 数据库存储实际数据

SQLite MCP 服务器将你的本地 SQLite 数据库之间的通信完全发生在你的机器上 — 你的 SQLite 数据库不会暴露在互联网上。Model Context Protocol 确保 Claude Desktop 只能通过定义良好的接口执行批准的数据库操作，这为你提供了一种安全的方式，让 Claude 分析和交互你的本地数据，同时完全控制它可以访问的内容。

前提条件

- macOS 或 Windows
- 安装最新版本的 **Claude Desktop**
- `uv` 0.4.18 或更高版本（`uv --version` 检查）
- Git（`git --version` 检查）
- SQLite（`sqlite3 --version` 检查）

▸ 安装前提条件 (macOS)

▸ 安装前提条件 (Windows)

安装

macOS　Windows

1. **创建示例数据库**

   创建一个简单的 SQLite 数据库进行测试：

```
# 创建一个新的 SQLite 数据库
sqlite3 ~/test.db <<EOF
CREATE TABLE products (
  id INTEGER PRIMARY KEY,
  name TEXT,
  price REAL
);

INSERT INTO products (name, price) VALUES
  ('Widget', 17.99),
  ('Gadget', 29.99),
  ('Slime', 39.99),
  ('Smart Watch', 199.99),
  ('Wireless Earbuds', 89.99),
  ('Portable Charger', 24.99),
  ('Bluetooth Speaker', 79.99),
  ('Phone Stand', 15.99),
  ('Laptop Sleeve', 34.99),
  ('Mini Drone', 299.99),
  ('LED Desk Lamp', 45.99),
  ('Keyboard', 129.99),
  ('Mouse Pad', 12.99),
  ('USB Hub', 49.99),
  ('Webcam', 89.99),
  ('Screen Protector', 9.99),
  ('Travel Adapter', 27.99),
  ('Gaming Headset', 159.99),
  ('Fitness Tracker', 119.99),
  ('Portable SSD', 179.99);
EOF
```

2. **配置 Claude Desktop**

   在文本编辑器中打开 `~/Library/Application Support/Claude/claude_desktop_config.json` 中的 Claude Desktop 应用配置。

   例如，如果你安装了 **VS Code**：

```
code ~/Library/Application\ Support/Claude/claude_desktop_config.json
```

   添加以下配置（将 YOUR_USERNAME 替换为你的实际用户名）：

```
{
  "mcpServers": {
    "sqlite": {
      "command": "uvx",
      "args": ["mcp-server-sqlite", "--db-path", "/Users/YOUR_USERNAME/test
    }
  }
}
```

   这告诉 Claude Desktop：

   1. 有一个名为 "sqlite" 的 MCP 服务器
   2. 通过运行 `uvx mcp-server-sqlite` 启动它
   3. 将其连接到你的测试数据库

   保存文件，并重新启动 **Claude Desktop**。

测试

让我们验证一切正常。尝试向 Claude Desktop 发送以下提示：

```
你能连接到我的 SQLite 数据库并告诉我有哪些产品及其价格吗？
```

Claude Desktop 将会：

1. 连接到 SQLite MCP 服务器
2. 查询你的本地数据库
3. 格式化并展示结果

Claude Desktop 成功使用我们的 SQLite 数据库 🎉

背后发生了什么？

当你使用 MCP 与 Claude Desktop 交互时：

1. **服务器发现**：Claude Desktop 在启动时连接到你配置的 MCP 服务器
2. **协议握手**：当你询问数据时，Claude Desktop：
   - 确定哪个 MCP 服务器可以提供帮助（在本例中是 sqlite）
   - 通过协议协商能力
   - 从 MCP 服务器请求数据或操作
3. **交互流程**：

4. **安全性**：
   - MCP 服务器仅暴露特定、受控的功能
   - MCP 服务器在你的机器上本地运行，它们访问的资源不会暴露在互联网上
   - Claude Desktop 需要用户确认以进行敏感操作

尝试这些示例

现在 MCP 正常工作，尝试这些超越单超强大的示例：

▸ 基本查询

▸ 数据分析

▸ 复杂操作

添加更多功能

想给 Claude Desktop 更多本地集成功能？将这些服务器添加到你的配置中：

> 请注意，许多 MCP 服务器需要在你的机器上安装 **Node.js**。

▸ 文件系统访问

▸ PostgreSQL 连接

更多 MCP 客户端

虽然本指南演示了使用 Claude Desktop 作为客户端的 MCP，但还有其他支持 MCP 集成的应用程序：

Zed Editor
一个高性能的、多用户代码编辑器，内置 MCP 支持以实现 AI 驱动的代码辅助

Cody
代码智能平台，具有 MCP 集成功能，以增强代码搜索和分析能力

每个主机应用程序可能以不同的方式实现 MCP 功能或支持不同的功能。请查看它们各自的文档以获取特定的设置说明和支持的功能。

故障排除

▸ Claude Desktop 中没有显示任何内容？

▸ MCP 或数据库错误？

下一步

构建你的第一个 MCP 服务器
创建你自己的 MCP 服务器，为你的 LLM 客户端提供新功能。

探索示例
浏览我们的示例服务器集合，看看有什么可能。

Was this page helpful?　Yes　No

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

Get Started

MCP 客户端

支持 MCP 的应用列表

此页面概述了支持 Model Context Protocol (MCP) 的应用程序。每个客户端可能支持不同的 MCP 功能子集，从而允许与 MCP 服务器的不同级别的集成。

功能支持表格

客户端	Resources	Prompts	Tools	Sampling	Roots	备注
Claude Desktop App	✅	✅	✅	❌	❌	完全支持
Zed	❌	✅	❌	❌	❌	提示以 / 命令的形式出现
Sourcegraph Cody	✅	❌	❌	❌	❌	通过 OpenCTX 支持 resources
Firebase Genkit	⚠️	✅	✅	❌	❌	通过 tools 支持列出和查找 resources
Continue	✅	✅	✅	❌	❌	完全支持
GenAIScript	❌	❌	✅	❌	❌	支持 tools

客户端详情

Claude Desktop 应用

Claude Desktop 为 MCP 提供全面支持，实现与本地工具和数据源的深度集成。

主要特性：

- 完全支持 resources，允许附加本地文件和数据
- 支持 prompt 模板
- Tool 集成以执行命令和脚本
- 本地 server 连接以增强隐私和安全性

> ⓘ Claude.ai 网页版暂不支持 MCP。MCP 特性仅在桌面应用中可用。

Zed

Zed 是一个高性能代码编辑器，内置 MCP 支持，专注于 prompt 模板和 tool 集成。

主要特性：

- 提示模板在编辑器中以 / 命令的形式出现
- Tool 集成以增强编码工作流
- 与编辑器功能和工作区上下文的紧密集成
- 不支持 MCP resources

Sourcegraph Cody

Cody 是 Sourcegraph 的 AI 编码助手，通过 OpenCTX 实现 MCP。

主要特性：

- 支持 MCP resources
- 与 Sourcegraph 的 code intelligence 集成
- 使用 OpenCTX 作为抽象层
- 计划在未来支持更多 MCP 特性

Firebase Genkit

Genkit 是 Firebase 的 SDK，用于在应用中构建和集成 GenAI 功能。**genkitx-mcp** 插件使得可以将 MCP 服务器作为 client 使用，或从 Genkit tools 和 prompts 创建 MCP server。

主要特性：

- 客户端支持 tools 和 prompts（部分支持 resources）
- Rich discovery with support in Genkit's Dev UI playground
- 与 Genkit 的现有 tools 和 prompts 无缝互操作
- 适用于来自顶级提供商的各种 GenAI 模型

Continue

Continue 是一个开源 AI 代码助手，支持所有 MCP 功能。

主要特性

- 输入 "@" 调用 MCP resources
- Prompt 模板以 / 命令的形式调用
- 在聊天中直接使用内置工具和 MCP 工具
- 支持 VS Code 和 JetBrains IDE，与任何 LLM 一起使用

GenAIScript

使用 **GenAIScript**（在 JavaScript 中）以编程方式为 LLM 组装 prompts。在 JavaScript 中编排 LLM、tools 和 resources。

主要特性：

- JavaScript 工具箱用于处理提示
- 便于使用和提高生产力的抽象层
- 无缝集成 Visual Studio Code

为你的应用添加 MCP 支持

如果你已为你的应用程序添加了 MCP 支持，我们鼓励你提交一个 pull request 将其添加到此列表中。MCP 集成可以为你的用户提供强大的上下文 AI 功能，并使你的应用程序成为不断增长的 MCP 生态的一部分。

支持 MCP 的好处：

- 使用户能够带入他们自己的上下文和 tools
- 加入一个不断增长的可互操作 AI 应用程序生态
- 为用户提供灵活的集成选项
- 支持 local-first 的 AI 工作流

要开始在你的应用程序中实现 MCP，请查看我们的 **Python** 或 **TypeScript** SDK 文档。

更新和更正

此列表由社区维护。如果你发现任何不准确之处或希望更新有关应用程序中 MCP 支持的信息，请提交一个 pull request 或 **在我们的文档库中打开一个问题**。

On this page
功能支持表格
客户端详情
Claude Desktop 应用
Zed
Sourcegraph Cody
Firebase Genkit
Continue
GenAIScript
为你的应用添加 MCP 支持
更新和更正

Was this page helpful? Yes No

Python

Create a simple MCP server in Python in 15 minutes

Let's build your first MCP server in Python! We'll create a weather server that provides current weather data via a National Weather Service API. We'll provide using tools.

Prerequisites

- The following steps are for macOS. Guides for other platforms are coming soon.

1. **Install Python**

   You'll need Python 3.10 or higher.

```
python --version # Should be 3.10 or higher
```

2. **Install uv via homebrew**

   For Mac, Linux, and Windows, see the uv docs.

```
brew install uv
uv --version # Should be 0.4.18 or higher
```

3. **Create a new project using the MCP project creator**

```
uv dlx create-mcp-server --name weather_service
cd weather_service
```

4. **Install additional dependencies**

```
uv add httpx python-dotenv
```

5. **Set up environment**

   Create .env:

```
OPENWEATHER_API_KEY=your_key_here
```

Create your server

1. **Add the base imports and setup**

   weather_service/src/weather_service/server.py:

```
import sys
import os
import logging
from dotenv import load_dotenv, find_dotenv
from mcp.server.fastmcp import FastMcp
from typing import Any, Literal, Optional
from typing import dict
import time

import requests
import datetime
import os
from dotenv import load_dotenv
from mcp.server.stdio import
from mcp.types import (
    Resource,
    Tool,
    TextContent,
    ImageContent,
    EmbeddedResource,
    LoggingLevel
)
from pydantic import AnyUrl

# Load environment variables
load_dotenv()

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger("weather-server")

# Constants and configuration
OPENWEATHER_API_KEY = os.getenv("OPENWEATHER_API_KEY")
BASE_URL = "https://api.openweathermap.org/data/2.5"
DEFAULT_CITY = "London"
CACHE_DURATION = 300  # 5 minutes in seconds
USER_AGENT = "weather-app/1.0"

# Validate required environment variables
if not OPENWEATHER_API_KEY:
    raise ValueError("OPENWEATHER_API_KEY environment variable required")

# Initialize FastMcp server
mcp = FastMcp("weather", "Tools for accessing weather information")
DEFAULT_CITY = "London"
CACHE_DURATION = 300  # 5 minutes in seconds
USER_AGENT = "weather-app/1.0"

# Log that we're starting
logger.info("Starting weather server")
```

2. **Add weather fetching functionality**

   Add this functionality:

```
# Simple cache implementation
weather_cache = {}

def get_cache_key(city, units):
    return f"{city|.lower()}:{units}"

def is_cache_valid(timestamp):
    return time.time() - timestamp < CACHE_DURATION

async def fetch_weather(city=DEFAULT_CITY, units="metric"):
    cache_key = get_cache_key(city, units)
    if cache_key in weather_cache:
        cached_data, timestamp = weather_cache[cache_key]
        if is_cache_valid(timestamp):
            logger.info(f"Using cached data for {city}")
            return cached_data

    logger.info(f"Fetching weather data for {city}")
    try:
        params = {
            "q": city,
            "appid": OPENWEATHER_API_KEY,
            "units": units,
            "exclude": "minutely,hourly,daily"
        }
        headers = {
            "User-Agent": USER_AGENT
        }

        response = requests.get(BASE_URL, params=params, headers=headers)
        response.raise_for_status()
        data = response.json()

        weather_cache[cache_key] = (data, time.time())
        return data
    except Exception as e:
        logger.error(f"Error fetching weather data: {str(e)}")
        raise ValueError(f"Failed to fetch weather data: {str(e)}")

def format_temperature(temp, units="metric"):
    if units == "metric":
        return f"{temp|.1f|°C"
    else:
        return f"{temp|.1f|°F"

def format_wind_speed(speed, units="metric"):
    if units == "metric":
        return f"{speed|.1f| m/s"
    else:
        return f"{speed|.1f| mph"

def format_weather_response(data, units="metric"):
    try:
        weather = data["weather"][0]["description"].capitalize()
        temp = data["main"]["temp"]
        feels_like = data["main"]["feels_like"]
        humidity = data["main"]["humidity"]
        wind_speed = data["wind"]["speed"]
        city = data["name"]
        country = data["sys"]["country"]

        return f"""Weather in {city}, {country}
Condition: {weather}
Temperature: {format_temperature(temp, units)}
Feels like: {format_temperature(feels_like, units)}
Humidity: {humidity}%
Wind speed: {format_wind_speed(wind_speed, units)}"""
    except KeyError as e:
        logger.error(f"Error formatting weather data: {str(e)}")
        raise ValueError("Received incomplete weather data")
```

3. **Implement resource handlers**

   Add these resource related handlers to our main function:

```
async def list_resources() -> list[Resource]:
    logger.info("Listing resources")
    return [
        Resource(
            uri=AnyUrl("weather://current/{city}"),
            name="Current Weather",
            description="Real-time weather data for a city",
            mimeType="application/json",
            description="Current weather data"
        ),
        Resource(
            uri=AnyUrl("weather://forecast/{city}"),
            name="Weather Forecast",
            description="7-day weather forecast for a city",
            mimeType="application/json",
            description="Weather forecast data"
        )
    ]

@mcp.resource_handler(lambda uri: uri.scheme == "weather")
async def read_resource(uri):
    logger.info(f"Reading resource: {uri}")
    city = uri.path.strip('/')
    if not city:
        raise ValueError("City name required")

    try:
        if uri.path == f"/current/{city}":
            data = fetch_weather(city, "metric")
            return {
                "temperature": data["main"]["temp"],
                "feels_like": data["main"]["feels_like"],
                "humidity": data["main"]["humidity"],
                "wind_speed": data["wind"]["speed"],
                "description": data["weather"][0]["description"],
                "city": data["name"],
                "country": data["sys"]["country"]
            }
        elif uri.path == f"/forecast/{city}":
            data = fetch_weather(city, "metric")
            return {
                "city": data["name"],
                "country": data["sys"]["country"],
                "forecast": [
                    {
                        "date": "today",
                        "temperature": data["main"]["temp"],
                        "description": data["weather"][0]["description"],
                        "humidity": data["main"]["humidity"],
                        "wind_speed": data["wind"]["speed"]
                    }
                ]
            }
        raise ValueError(f"Unknown resource: {uri}")
    except Exception as e:
        logger.error(f"Error reading resource: {str(e)}")
        raise ValueError(f"Error processing resource: {str(e)}")
```

4. **Implement tool handlers**

   Add these tool related functions:

```
async def list_tools() -> list[Tool]:
    logger.info("Listing tools")
    return [
        Tool(
            name="get_current_weather",
            description="Get weather data for a city",
            inputSchema={
                "type": "object",
                "properties": {
                    "city": {
                        "type": "string",
                        "description": "City name"
                    },
                    "units": {
                        "type": "string",
                        "enum": ["metric", "imperial"],
                        "description": "Temperature units"
                    }
                },
                "required": ["city"]
            }
        )
    ]

@mcp.call_tool()
async def call_tool(name: str, arguments: dict | None=None):
    logger.info(f"Calling tool: {name} with arguments: {arguments}")
    if not arguments:
        raise ValueError("Missing arguments")

    if name == "get_current_weather":
        city = arguments.get("city")
        units = arguments.get("units", "metric")
        if not city:
            raise ValueError("City parameter required")

        try:
            data = fetch_weather(city, units)
            formatted_response = format_weather_response(data, units)
            return [TextContent(type="text", text=formatted_response)]
        except ValueError as e:
            return [TextContent(type="text", text=f"Error: {str(e)}")]
    else:
        raise ValueError(f"Unknown tool: {name}")
```

5. **Add the main function**

   Add this to the end of weather_service/src/weather_service/server.py:

```
async def main():
    logger.info("Starting weather server")
    try:
        async with stdio_server() as (read_stream, write_stream):
            logger.info("Server running")
            await mcp.run(
                read_stream,
                write_stream,
                mcp.create_initialization_options()
            )
    except Exception as e:
        logger.error(f"Server error: {str(e)}")
        raise

if __name__ == "__main__":
    import asyncio
    asyncio.run(main)
```

6. **Check your entry point in __init__.py**

   Verify that the end of weather_service/src/weather_service/__init__.py looks like this:

```
from .server import mcp

def main():
    """Main entry point for the package."""
    mcp.run(transport='stdio')

# Optionally expose other important items at package level
__all__ = ['main', 'mcp']
```

Connect to Claude Desktop

1. **Update Claude config**

   Edit ~/Library/Application Support/Claude/claude_desktop_config.json:

```
{
  "preferences": {
    "quickAuxiliarySearchCommand": "alt+space",
    "quickAssistant": {},
    "keyboardSuggestions": true,
    "sparkleConfig": {
      "feed": "https://claude.ai/api/desktop/darwin/universal/sparkle/stable.rss",
      "lastCheck": 1740000000000,
      "lastUpdate": 1740000000000,
      "autoInstallUpdates": true
    },
    "ccd-version": "1.0.0",
    "ccd-hash": "abcdef123456",
    "ccd-date": "2025-02-20T12:00:00.000Z",
    "ccd-channel": "stable",
    "ccd-platform": "darwin-arm64",
    "ccd-electron": "34.0.0-beta.7-darwin.20241107.1",
    "ccd-node": "20.18.1",
    "ccd-v8": "13.4.114.21-electron.0.beta.1-11-g7f5a8b3c4d5e",
    "ccd-os": "24.0.0",
    "ccd-arch": "arm64",
    "ccd-electron-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version": "34.0.0-beta.7",
    "ccd-node-version": "20.18.1",
    "ccd-v8-version": "13.4.114.21",
    "ccd-os-version": "24.0.0",
    "ccd-arch-version": "arm64",
    "ccd-electron-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "9876543210fedcba0987654321abcdef01234567",
    "ccd-v8-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "fedcba9876543210abcdef0123456789abcdef01",
    "ccd-os-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "0123456789abcdefghijklmnopqrstuvwxyz",
    "ccd-arch-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version": "abcdefghijklmnopqrstuvwxyz0123456789",
    "ccd-electron-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "1a2b3c4d5e6f7890abcdef1234567890abcdef12",
    "ccd-node-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha-version-sha": "9876543
```

Testing

1 **Create test file**

Create `tests/weather_test.py`:

```python
import pytest
import os
from unittest.mock import patch, Mock
from datetime import datetime
import json
from pydantic import AnyUrl
os.environ["OPENWEATHER_API_KEY"] = "TEST"

from weather_service.server import (
    fetch_weather,
    read_resource,
    call_tool,
    list_resources,
    list_tools,
    DEFAULT_CITY
)

@pytest.fixture
def anyio_backend():
    return "asyncio"

@pytest.fixture
def mock_weather_response():
    return {
        "main": {
            "temp": 20.5,
            "humidity": 65
        },
        "weather": [
            {"description": "scattered clouds"}
        ],
        "wind": {
            "speed": 3.6
        }
    }

@pytest.fixture
def mock_forecast_response():
    return {
        "list": [
            {
                "dt_txt": "2024-01-01 12:00:00",
                "main": {"temp": 18.5},
                "weather": [{"description": "sunny"}]
            },
            {
                "dt_txt": "2024-01-02 12:00:00",
                "main": {"temp": 17.2},
                "weather": [{"description": "cloudy"}]
            }
        ]
    }

@pytest.mark.anyio
async def test_fetch_weather(mock_weather_response):
    with patch('requests.Session.get') as mock_get:
        mock_get.return_value.json.return_value = mock_weather_response
        mock_get.return_value.raise_for_status = Mock()

        weather = await fetch_weather("London")

        assert weather["temperature"] == 20.5
        assert weather["conditions"] == "scattered clouds"
        assert weather["humidity"] == 65
        assert weather["wind_speed"] == 3.6
        assert "timestamp" in weather

@pytest.mark.anyio
async def test_read_resource():
    with patch('weather_service.server.fetch_weather') as mock_fetch:
        mock_fetch.return_value = {
            "temperature": 20.5,
            "conditions": "clear sky",
            "timestamp": datetime.now().isoformat()
        }

        uri = AnyUrl("weather://London/current")
        result = await read_resource(uri)

        assert isinstance(result, str)
        assert "temperature" in result
        assert "clear sky" in result

@pytest.mark.anyio
async def test_call_tool(mock_forecast_response):
    class Response():
        def raise_for_status(self):
            pass

        def json(self):
            return mock_forecast_response

    class AsyncClient():
        async def __aenter__(self):
            return self

        async def __aexit__(self, *exc_info):
            pass

        async def get(self, *args, **kwargs):
            return Response()

    with patch('httpx.AsyncClient', new=AsyncClient) as mock_client:
        result = await call_tool("get_forecast", {"city": "London", "days":

        assert len(result) == 1
        assert result[0].type == "text"
        forecast_data = json.loads(result[0].text)
        assert len(forecast_data) == 1
        assert forecast_data[0]["temperature"] == 18.5
        assert forecast_data[0]["conditions"] == "sunny"

@pytest.mark.anyio
async def test_list_resources():
    resources = await list_resources()
    assert len(resources) == 1
    assert resources[0].name == f"Current weather in {DEFAULT_CITY}"
    assert resources[0].mimeType == "application/json"

@pytest.mark.anyio
async def test_list_tools():
    tools = await list_tools()
    assert len(tools) == 1
    assert tools[0].name == "get_forecast"
    assert "city" in tools[0].inputSchema["properties"]
```

2 **Run tests**

```
uv add --dev pytest
uv run pytest
```

Troubleshooting

Installation issues

```
# Check Python version
python --version

# Reinstall dependencies
uv sync --reinstall
```

Type checking

```
# Install mypy
uv add --dev pyright

# Run type checker
uv run pyright src
```

Next steps

Architecture overview
Learn more about the MCP architecture

Python SDK
Check out the Python SDK on GitHub

Was this page helpful? Yes No

Powered by Mintlify

Documentation
Home Page
Quickstart
Transcribe Q&A
FAQ

Get Started
MCP 101
SDK 101
MCP 201

Your First MCP Server
Python
TypeScript

FIELD
Debugging
Inspector

MCS
MCS SDK
Resources
Primers
Tools
Sampling
中国区

On this page
Prerequisites
Create your server
Connect to Claude Desktop
Try it out!
Understanding the code
Best practices
Available transports
Troubleshooting
Build errors
Runtime errors
Type errors
Next steps

TypeScript

Create a simple MCP server in TypeScript in 15 minutes.

Let's build your first MCP server in TypeScript! We'll create a weather server that provides current weather data as a resource and Check North Korea's trading tools.

This guide uses the OpenWeatherMap API. You'll need a free API key from OpenWeatherMap.

Prerequisites

1. **Install Node.js**

   You'll need Node.js 18 or higher:

```
node --version
```

2. **Create a new project**

   You can use our **create-typescript-server** tool to bootstrap a new project:

```
npx @modelcontextprotocol/create-server weather-server
cd weather-server
```

3. **Install dependencies**

```
npm install --save axios dotenv
```

4. **Set up environment**

   Create `.env`:

```
OPENWEATHER_API_KEY=your_api_key_here
```

   Make sure to add your environment file to `.gitignore`:

```
.env
```

Create your server

1. **Define types**

   Create a file `src/types.ts`, and add the following:

```
import {CallToolRequestSchema} {
  name: {
    type: "string",
    enum: ["get_weather"],
  },
  arguments: {
    type: "object",
    properties: {
      city: {
        type: "string",
        description: "City name",
      },
      units: {
        type: "string",
        enum: ["metric", "imperial"],
        description: "Temperature units",
      },
    },
    required: ["city"],
  },
},
],
};
});

// Define the tool for forecast
server.setRequestHandler(ListToolsRequestSchema, async () => {
  return {
    tools: [
      {
        name: "get_forecast",
        description: "Get weather forecast for a city",
        inputSchema: {
          type: "object",
          properties: {
            city: {
              type: "string",
              description: "City name",
            },
            days: {
              type: "number",
              description: "Number of days (1-5)",
              minimum: 1,
              maximum: 5,
            },
          },
          required: ["city"],
        },
      },
    ],
  };
});

// Handle forecast tool calls
server.setRequestHandler(CallToolRequestSchema, async (request) => {
  if (request.params.name !== "get_forecast") {
    throw new McpError(
      ErrorCode.MethodNotFound,
      `Unknown tool: ${request.params.name}`
    );
  }

  const {city, days = 3} = request.params.arguments as {
    city: string;
    days?: number;
  };

  try {
    const apiKey = process.env.OPENWEATHER_API_KEY!;
    if (!apiKey) {
      throw new McpError(
        ErrorCode.InternalError,
        "OpenWeather API key not configured"
      );
    }

    // Get forecast data
    const response = await axios.get(
      `http://api.openweathermap.org/data/2.5/forecast?q=${city}&cnt=${days * 8}&units=metric&appid=${apiKey}`
    );

    const forecasts = response.data.list;
    const cityName = response.data.city.name;

    let result = `Forecast for ${cityName}:\n\n`;
    const daily = {};
    forecasts.slice(0, days * 8).forecast_items.length; i += 8) {
    const dayForecast = forecasts[i];
    const date = new Date(dayForecast.dt * 1000).toLocaleDateString();
    daily[date] = {
      temp: dayForecast.main.temp,
      description: dayForecast.weather[0].description,
      humidity: dayForecast.main.humidity
    };
  }

  for (const [date, data] of Object.entries(daily).slice(0, days).forecast_items.length; i += 8) {
    const dayForecast = forecasts[i];
    const date = new Date(dayForecast.dt * 1000).toLocaleDateString();
    daily
```

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

开发工具

Debugging

A comprehensive guide to debugging Model Context Protocol (MCP) integrations

Effective debugging is essential when developing MCP servers or integrating them with applications. This guide covers the debugging tools and approaches available in the MCP ecosystem.

This guide is for macOS. Guides for other platforms are coming soon.

Debugging tools overview

MCP provides several tools for debugging at different levels:

1. **MCP Inspector**
   - Interactive debugging interface
   - Direct server testing
   - See the **Inspector guide** for details
2. **Claude Desktop Developer Tools**
   - Integration testing
   - Log collection
   - Chrome DevTools integration
3. **Server Logging**
   - Custom logging implementations
   - Error tracking
   - Performance monitoring

Debugging in Claude Desktop

Checking server status

The Claude.app interface provides basic server status information:

1. Click the 🔌 icon to view:
   - Connected servers
   - Available prompts and resources
2. Click the 🔨 icon to view:
   - Tools made available to the model

Viewing logs

Review detailed MCP logs from Claude Desktop:

```
# Follow logs in real-time
tail -n 20 -f ~/Library/Logs/Claude/mcp*.log
```

The logs capture:

- Server connection events
- Configuration issues
- Runtime errors
- Message exchanges

Using Chrome DevTools

Access Chrome's developer tools inside Claude Desktop to investigate client-side errors:

1. Enable DevTools:

```
jq '.allowDevTools = true' ~/Library/Application\ Support/Claude/developer_settings
.tmp && mv tmp.json ~/Library/Application\ Support/Claude/developer_settings.json
```

2. Open DevTools: `Command-Option-Shift-i`

Note: You'll see two DevTools windows:

- Main content window
- App title bar window

Use the Console panel to inspect client-side errors.

Use the Network panel to inspect:

- Message payloads
- Connection timing

Common issues

Environment variables

MCP servers inherit only a subset of environment variables automatically, like `USER`, `HOME`, and `PATH`.

To override the default variables or provide your own, you can specify an `env` key in `claude_desktop_config.json`:

```
{
  "mcpserver": {
    "command": "mcp-server-mysqq",
    "env": {
      "MYAPP_API_KEY": "some_key",
    }
  }
}
```

Server initialization

Common initialization problems:

1. **Path Issues**
   - Incorrect server executable path
   - Missing required files
   - Permission problems
2. **Configuration Errors**
   - Invalid JSON syntax
   - Missing required fields
   - Type mismatches
3. **Environment Problems**
   - Missing environment variables
   - Incorrect variable values
   - Permission restrictions

Connection problems

When servers fail to connect:

1. Check Claude Desktop logs
2. Verify server process is running
3. Test standalone with **Inspector**
4. Verify protocol compatibility

Implementing logging

Server-side logging

When building a server that uses the local stdio **transport**, all messages logged to stderr (standard error) will be captured by the host application (e.g., Claude Desktop) automatically.

Local MCP servers should not log messages to stdout (standard out), as this will interfere with protocol operation.

For all **transports**, you can also provide logging to the client by sending a log message notification:

Python TypeScript

```
server.request_context.session.send_log_message(
  level="info",
  data="Server started successfully",
)
```

Important events to log:

- Initialization steps
- Resource access
- Tool execution
- Error conditions
- Performance metrics

Client-side logging

In client applications:

1. Enable debug logging
2. Monitor network traffic
3. Track message exchanges
4. Record error states

Debugging workflow

Development cycle

1. Initial Development
   - Use **Inspector** for basic testing
   - Implement core functionality
   - Add logging points
2. Integration Testing
   - Test in Claude Desktop
   - Monitor logs
   - Check error handling

Testing changes

To test changes efficiently:

- **Configuration changes**: Restart Claude Desktop
- **Server code changes**: Use Command-R to reload
- **Quick iteration**: Use **Inspector** during development

Best practices

Logging strategy

1. **Structured Logging**
   - Use consistent formats
   - Include context
   - Add timestamps
   - Track request IDs
2. **Error Handling**
   - Log stack traces
   - Include error context
   - Track error patterns
   - Monitor recovery
3. **Performance Tracking**
   - Log operation timing
   - Monitor resource usage
   - Track message sizes
   - Measure latency

Security considerations

When debugging:

1. **Sensitive Data**
   - Sanitize logs
   - Protect credentials
   - Mask personal information
2. **Access Control**
   - Verify permissions
   - Check authentication
   - Monitor access patterns

Getting help

When encountering issues:

1. **First Steps**
   - Check server logs
   - Test with **Inspector**
   - Review configuration
   - Verify environment
2. **Support Channels**
   - GitHub issues
   - GitHub discussions
3. **Providing Information**
   - Log excerpts
   - Configuration files
   - Steps to reproduce
   - Environment details

Next steps

MCP Inspector
Learn to use the MCP Inspector

On this page
Debugging tools overview
Debugging in Claude Desktop
Checking server status
Viewing logs
Using Chrome DevTools
Common issues
Environment variables
Server initialization
Connection problems
Implementing logging
Server-side logging
Client-side logging
Debugging workflow
Development cycle
Testing changes
Best practices
Logging strategy
Security considerations
Getting help
Next steps

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

开发工具

Inspector

In-depth guide to using the MCP Inspector for testing and debugging Model Context Protocol servers

On this page
Getting started
Installation and basic usage
Inspecting servers from NPM or PyPi
Inspecting locally developed servers
Feature overview
Server connection pane
Resources tab
Prompts tab
Tools tab
Notifications pane
Best practices
Development workflow
Next steps

The **MCP Inspector** is an interactive developer tool for testing and debugging MCP servers. While the **Debugging Guide** covers the Inspector as part of the overall debugging toolkit, this document provides a detailed exploration of the Inspector's features and capabilities.

Getting started

Installation and basic usage

The Inspector runs directly through `npx` without requiring installation:

```
npx @modelcontextprotocol/inspector <command>
```

```
npx @modelcontextprotocol/inspector <command> <arg1> <arg2>
```

Inspecting servers from NPM or PyPi

A common way to start server packages from **NPM** or **PyPi**.

NPM package | PyPi package

```
npx -y @modelcontextprotocol/inspector npx <package-name> <args>
# For example
npx -y @modelcontextprotocol/inspector npx server-postgres postgres://127.0.0.1/test
```

Inspecting locally developed servers

To inspect servers locally developed or downloaded as a repository, the most common way is:

TypeScript | Python

```
npx @modelcontextprotocol/inspector node path/to/server/index.js args...
```

Please carefully read any attached README for the most accurate instructions.

Feature overview

The MCP Inspector interface

The Inspector provides several features for interacting with your MCP server:

Server connection pane

- Allows selecting the **transport** for connecting to the server
- For local servers, supports customizing the command-line arguments and environment

Resources tab

- Lists all available resources
- Shows resource metadata (MIME types, descriptions)
- Allows resource content inspection
- Supports subscription testing

Prompts tab

- Displays available prompt templates
- Shows prompt arguments and descriptions
- Enables prompt testing with custom arguments
- Previews generated messages

Tools tab

- Lists available tools
- Shows tool schemas and descriptions
- Enables tool testing with custom inputs
- Displays tool execution results

Notifications pane

- Presents all logs recorded from the server
- Shows notifications received from the server

Best practices

Development workflow

1. Start Development
   - Launch Inspector with your server
   - Verify basic connectivity
   - Check capability negotiation
2. Iterative testing
   - Make server changes
   - Rebuild the server
   - Reconnect the Inspector
   - Test affected features
   - Monitor messages
3. Test edge cases
   - Invalid inputs
   - Missing prompt arguments
   - Concurrent operations
   - Verify error handling and error responses

Next steps

Inspector Repository
Check out the MCP Inspector source code

Debugging Guide
Learn about broader debugging strategies

Was this page helpful? Yes No

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

核心架构

了解 MCP 如何连接 clients、servers 和 LLMs

Model Context Protocol (MCP) 构建在一个灵活且可扩展的架构上。使 LLM 应用和集成之间的无缝通信成为可能。本文档涵盖了核心架构组件和概念。

概述

MCP 遵循一个 client-server 架构，其中：

- **Hosts** 是 LLM 应用（如 Claude Desktop 或 IDEs），它们发起连接
- **Clients** 在 host 应用中为 servers 保持 1:1 的连接
- **Servers** 为 clients 提供上下文、tools 和 prompts

核心组件

协议层

协议层处理高质量框架，请求/响应链接和高级通信模式。

TypeScript Python

```
class Protocol<Request, Notification, Result> {
  // 处理传入请求
  setRequestHandler<T:{schema: T, handler: (request: T, extra: RequestHandlerExtra
  setRequestHandler<T:{schema: T, handler: (notification: T) => Promise<void
  // 发送请求并等待响应
  request<T:{request: Request, schema: T, options? RequestOptions}) Promise<T>
  // 发送单向通知
  notification(notification: Notification): Promise<void>
}
```

关键的 classes 包括：

- `Protocol`
- `Client`
- `Server`

传输层

传输层处理 clients 和 servers 之间的实际通信。MCP 支持多种传输机制：

1. **Stdio**
   - 使用标准输入输出进行通信
   - 适用于本地进程
2. **通过 HTTP 的 SSE**
   - 使用服务器发送事件进行服务器到客户端的消息传递
   - 使用 HTTP POST 进行客户端到服务器的消息传递

所有传输都使用 *JSON-RPC 2.0* 进行消息交换。有关 Model Context Protocol 消息格式的详细信息，请参阅规范。

消息类型

MCP 具有以下主要类型的消息：

1. **Requests** 期望来自另一端的响应：

```
interface Request {
  method: string;
  params?: { ... };
}
```

2. **Notifications** 基于期望响应的单向消息：

```
interface Notification {
  method: string;
  params?: { ... };
}
```

3. **Results** 是对请求的成功响应：

```
interface Result {
  [key: string]: unknown;
}
```

4. **Errors** 表示请求失败：

```
interface Error {
  code: number;
  message: string;
  data?: unknown;
}
```

连接生命周期

1. 初始化

1. Client 发送包含协议版本和能力的 `initialize` 请求
2. Server 以其协议版本和能力响应
3. Client 发送 `initialized` 通知作为确认
4. 开始正常消息交换

2. 消息交换

初始化后，支持以下模式：

- **请求-响应**：客户端或服务器发送请求，另一方响应
- **通知**：任一方发送单向消息

3. 终止

任一方可以终止连接：

- 通过 `close()` 进行干净关闭
- 传输断开
- 错误条件

错误处理

MCP 定义了以下标准错误代码：

```
enum ErrorCode {
  // JSON-RPC 标准代码
  ParseError = -32700,
  InvalidRequest = -32600,
  MethodNotFound = -32601,
  InvalidParams = -32602,
  InternalError = -32603
}
```

SDK 和应用可以定义其自己的错误代码，范围在 -32000 以上。

错误通过以下方式传播：

- 对请求的错误响应
- 传输上的错误事件
- 协议级别的错误处理程序

实现示例

以下是实现 MCP 服务器的基本示例：

TypeScript Python

```
import { Server } from "@modelcontextprotocol/sdk/server/index.js";
import { StdioServerTransport } from "@modelcontextprotocol/sdk/server/stdio.js";

const server = new Server({
  name: "example-server",
  version: "1.0.0"
},{
  capabilities: {
    resources: {}
  }
});

// 处理请求
server.setRequestHandler(ListResourcesRequestSchema, async () => {
  return {
    resources: [
      {
        uri: "example://resource",
        name: "Example Resource"
      }
    ]
  };
});

// 连接传输
const transport = new StdioServerTransport();
await server.connect(transport);
```

最佳实践

传输选择

1. **本地通信**
   - 对于本地进程使用 stdio 传输
   - 对于同一机器的通信效率高
   - 简单的进程管理
2. **远程通信**
   - 在需要 HTTP 兼容性的场景中使用 SSE
   - 考虑安全隐患，包括身份验证和授权

消息处理

1. **请求处理**
   - 彻底验证输入
   - 使用类型安全的模式
   - 优雅地处理错误
   - 实现超时
2. **进度报告**
   - 对于长时间操作使用进度令牌
   - 增量报告进度
   - 在已知时包括总进度
3. **错误管理**
   - 使用适当的错误代码
   - 包含有帮助的错误信息
   - 在错误时清理资源

安全考虑

1. **传输安全**
   - 对于远程连接使用 TLS
   - 验证连接来源
   - 在需要时实现身份验证
2. **消息验证**
   - 验证所有传入消息
   - 清理输入
   - 检查消息大小限制
   - 验证 JSON-RPC 格式
3. **资源保护**
   - 实现访问控制
   - 验证资源路径
   - 监控资源使用
   - 限制请求速率
4. **错误处理**
   - 不泄露敏感信息
   - 记录安全相关错误
   - 实现适当的清理
   - 处理 DoS 场景

调试和监控

1. **日志记录**
   - 记录协议事件
   - 跟踪消息流
   - 监控性能
   - 记录错误
2. **诊断**
   - 实现健康检查
   - 监控连接状态
   - 跟踪资源使用
   - 性能分析
3. **测试**
   - 测试不同的传输
   - 验证错误处理
   - 检查边缘案例
   - 负载测试服务器

On this page
概述
核心组件
协议层
传输层
消息类型
连接生命周期
1. 初始化
2. 消息交换
3. 终止
错误处理
实现示例
最佳实践
传输选择
消息处理
安全考虑
调试和监控

Was this page helpful? Yes No

Powered by Mintlify

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

Resources

Expose data and content from your servers to LLMs

Resources are a core primitive in the Model Context Protocol (MCP) that allow servers to expose data and content that can be read by clients and used as context for LLM interactions.

> Resources are designed to be **application-controlled**, meaning that the client application can decide how and when they should be used. Different MCP clients may handle resources differently. For example:
>
> - Claude Desktop currently requires users to explicitly select resources before they can be used
> - Other clients might automatically select resources based on heuristics
> - Some implementations may even allow the AI model itself to determine which resources to use
>
> Server authors should be prepared to handle any of these interaction patterns when implementing resource support. In order to expose data to models automatically, server authors should use a **model-controlled** primitive such as **Tools**.

Overview

Resources represent any kind of data that an MCP server wants to make available to clients. This can include:

- File contents
- Database records
- API responses
- Live system data
- Screenshots and images
- Log files
- And more

Each resource is identified by a unique URI and can contain either text or binary data.

Resource URIs

Resources are identified using URIs that follow this format:

```
[protocol]://[host]/[path]
```

For example:

- `file:///home/user/documents/report.pdf`
- `postgres://database/customers/schema`
- `screen://localhost/display1`

The protocol and path structure is defined by the MCP server implementation. Servers can define their own custom URI schemes.

Resource types

Resources can contain two types of content:

Text resources

Text resources contain UTF-8 encoded text data. These are suitable for:

- Source code
- Configuration files
- Log files
- JSON/XML data
- Plain text

Binary resources

Binary resources contain raw binary data encoded in base64. These are suitable for:

- Images
- PDFs
- Audio files
- Video files
- Other non-text formats

Resource discovery

Clients can discover available resources through two main methods:

Direct resources

Servers expose a list of concrete resources via the `resources/list` endpoint. Each resource includes:

```
{
  uri: string;          // Unique identifier for the resource
  name: string;         // Human-readable name
  description?: string; // Optional description
  mimeType?: string;    // Optional MIME type
}
```

Resource templates

For dynamic resources, servers can expose **URI templates** that clients can use to construct valid resource URIs:

```
{
  uriTemplate: string;  // URI template following RFC 6570
  name: string;         // Human-readable name for this type
  description?: string; // Optional description
  mimeType?: string;    // Optional MIME type for all matching resources
}
```

Reading resources

To read a resource, clients make a `resources/read` request with the resource URI.

The server responds with a list of resource contents:

```
{
  contents: [
    {
      uri: string;        // The URI of the resource
      mimeType?: string;  // Optional MIME type

      // One of:
      text?: string;      // For text resources
      blob?: string;      // For binary resources (base64 encoded)
    }
  ]
}
```

> Servers may return multiple resources in response to one `resources/read` request. This could be used, for example, to return a list of files inside a directory when the directory is read.

Resource updates

MCP supports real-time updates for resources through two mechanisms:

List changes

Servers can notify clients when their list of available resources changes via the `notifications/resources/list_changed` notification.

Content changes

Clients can subscribe to updates for specific resources:

1. Client sends `resources/subscribe` with resource URI
2. Server sends `notifications/resources/updated` when the resource changes
3. Client can fetch latest content with `resources/read`
4. Client can unsubscribe with `resources/unsubscribe`

Example implementation

Here's a simple example of implementing resource support in an MCP server:

TypeScript Python

```
const server = new Server({
  name: "example-server",
  version: "1.0.0"
}, {
  capabilities: {
    resources: {}
  }
});

// List available resources
server.setRequestHandler(ListResourcesRequestSchema, async () => {
  return {
    resources: [
      {
        uri: "file:///logs/app.log",
        name: "Application Logs",
        mimeType: "text/plain"
      }
    ]
  };
});

// Read resource contents
server.setRequestHandler(ReadResourceRequestSchema, async (request) => {
  const uri = request.params.uri;

  if (uri === "file:///logs/app.log") {
    const logContents = await readLogFile();
    return {
      contents: [
        {
          uri,
          mimeType: "text/plain",
          text: logContents
        }
      ]
    };
  }

  throw new Error("Resource not found");
});
```

Best practices

When implementing resource support:

1. Use clear, descriptive resource names and URIs
2. Include helpful descriptions to guide LLM understanding
3. Set appropriate MIME types when known
4. Implement resource templates for dynamic content
5. Use subscriptions for frequently changing resources
6. Handle errors gracefully with clear error messages
7. Consider pagination for large resource lists
8. Cache resource contents when appropriate
9. Validate URIs before processing
10. Document your custom URI schemes

Security considerations

When exposing resources:

- Validate all resource URIs
- Implement appropriate access controls
- Sanitize file paths to prevent directory traversal
- Be cautious with binary data handling
- Consider rate limiting for resource reads
- Audit resource access
- Encrypt sensitive data in transit
- Validate MIME types
- Implement timeouts for long-running reads
- Handle resource cleanup appropriately

On this page
Overview
Resource URIs
Resource types
Text resources
Binary resources
Resource discovery
Direct resources
Resource templates
Reading resources
Resource updates
List changes
Content changes
Example implementation
Best practices
Security considerations

Was this page helpful? Yes No

Documentation
Python SDK
TypeScript SDK
概览

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

Prompts

Create reusable prompt templates and workflows

Prompts enable servers to define reusable prompt templates and workflows that clients can easily surface to users and LLMs. They provide a powerful way to standardize and share common LLM interactions.

> Prompts are designed to be **user-controlled**, meaning they are exposed from servers to clients with the intention of the user being able to explicitly select them for use.

Overview

Prompts in MCP are predefined templates that can:

- Accept dynamic arguments
- Include context from resources
- Chain multiple interactions
- Guide specific workflows
- Surface as UI elements (like slash commands)

Prompt structure

Each prompt is defined with:

```
{
  name: string;              // Unique identifier for the prompt
  description?: string;      // Human-readable description
  arguments?: [              // Optional list of arguments
    {
      name: string;          // Argument identifier
      description?: string   // Argument description
      required?: boolean;    // Whether argument is required
    }
  ]
}
```

Discovering prompts

Clients can discover available prompts through the `prompts/list` endpoint:

```
// Request
{
  method: "prompts/list"
}

// Response
{
  prompts: [
    {
      name: "analyze-code",
      description: "Analyze code for potential improvements",
      arguments: [
        {
          name: "language",
          description: "Programming language",
          required: true
        }
      ]
    }
  ]
}
```

Using prompts

To use a prompt, clients make a `prompts/get` request:

```
// Request
{
  method: "prompts/get",
  params: {
    name: "analyze-code",
    arguments: {
      "language": "python"
    }
  }
}

// Response
{
  description: "Analyze Python code for potential improvements",
  messages: [
    {
      role: "user",
      content: {
        type: "text",
        text: "Please analyze the following Python code for potential improvements:
      }
    }
  ]
}
```

Dynamic prompts

Prompts can be dynamic and include:

Embedded resource context

```
{
  "name": "analyze-project",
  "description": "Analyze project logs and code",
  "arguments": [
    {
      "name": "timeframe",
      "description": "Time period to analyze logs",
      "required": true
    },
    {
      "name": "files[]",
      "description": "URI of code file to review",
      "required": true
    }
  ]
}
```

When handling the `prompts/get` request:

```
{
  "messages": [
    {
      "role": "user",
      "content": {
        "type": "text",
        "text": "Analyze these system logs and the code file for any issues:"
      }
    },
    {
      "role": "user",
      "content": {
        "type": "resource",
        "resource": {
          "uri": "logs://recent?timeframe=1h",
          "text": "[2024-10-14 10:32:15] ERROR: Connection timeout in network.py:12
          "mimeType": "text/plain"
        }
      }
    },
    {
      "role": "user",
      "content": {
        "type": "resource",
        "resource": {
          "uri": "file:///path/to/code.py",
          "text": "def connect_to_service(timeout=30):\n    retries = 3\n    for at
          "mimeType": "text/x-python"
        }
      }
    }
  ]
}
```

Multi-step workflows

```
const debugWorkflow = {
  name: "debug-error",
  async getMessages(error: string) {
    return [
      {
        role: "user",
        content: {
          type: "text",
          text: `Here's an error I'm seeing: ${error}`
        }
      },
      {
        role: "assistant",
        content: {
          type: "text",
          text: "I'll help analyze this error. What have you tried so far?"
        }
      },
      {
        role: "user",
        content: {
          type: "text",
          text: "I've tried restarting the service, but the error persists."
        }
      }
    ];
  }
};
```

Example implementation

Here's a complete example of implementing prompts in an MCP server:

TypeScript Python

```
import { Server } from "@modelcontextprotocol/sdk/server";
import {
  ListPromptsRequestSchema,
  GetPromptRequestSchema
} from "@modelcontextprotocol/sdk/types";

const PROMPTS = {
  "git-commit": {
    name: "git-commit",
    description: "Generate a Git commit message",
    arguments: [
      {
        name: "changes",
        description: "Git diff or description of changes",
        required: true
      }
    ]
  },
  "explain-code": {
    name: "explain-code",
    description: "Explain how code works",
    arguments: [
      {
        name: "code",
        description: "Code to explain",
        required: true
      },
      {
        name: "language",
        description: "Programming language",
        required: false
      }
    ]
  }
};

const server = new Server({
  name: "example-prompts-server",
  version: "1.0.0"
},
  capabilities: {
    prompts: {}
  }
);

// List available prompts
server.setRequestHandler(ListPromptsRequestSchema, async () => {
  return {
    prompts: Object.values(PROMPTS)
  };
});

// Get specific prompt
server.setRequestHandler(GetPromptRequestSchema, async (request) => {
  const prompt = PROMPTS[request.params.name];
  if (!prompt) {
    throw new Error(`Prompt not found: ${request.params.name}`);
  }

  if (request.params.name === "git-commit") {
    return {
      messages: [
        {
          role: "user",
          content: {
            type: "text",
            text: `Generate a concise but descriptive commit message for these chan
          }
        }
      ]
    };
  }

  if (request.params.name === "explain-code") {
    const language = request.params.arguments?.language || "Unknown";
    return {
      messages: [
        {
          role: "user",
          content: {
            type: "text",
            text: `Explain how this ${language} code works:\n${request.params.arg
          }
        }
      ]
    };
  }

  throw new Error("Prompt implementation not found");
});
```

Best practices

When implementing prompts:

1. Use clear, descriptive prompt names
2. Provide detailed descriptions for prompts and arguments
3. Validate all required arguments
4. Handle missing arguments gracefully
5. Consider versioning for prompt templates
6. Cache dynamic content when appropriate
7. Implement error handling
8. Document expected argument formats
9. Consider prompt composability
10. Test prompts with various inputs

UI integration

Prompts can be surfaced in client UIs as:

- Slash commands
- Quick actions
- Context menu items
- Command palette entries
- Guided workflows
- Interactive forms

Updates and changes

Servers can notify clients about prompt changes:

1. Server capability: `prompts.listChanged`
2. Notification: `notifications/prompts/list_changed`
3. Client re-fetches prompt list

Security considerations

When implementing prompts:

- Validate all arguments
- Sanitize user input
- Consider rate limiting
- Implement access controls
- Audit prompt usage
- Handle sensitive data appropriately
- Validate generated content
- Implement timeouts
- Consider prompt injection risks
- Document security requirements

Was this page helpful? Yes No

C On this page
Overview
Prompt structure
Discovering prompts
Using prompts
Dynamic prompts
Embedded resource context
Multi-step workflows
Example implementation
Best practices
UI integration
Updates and changes
Security considerations

Powered by Mintlify

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

Tools

Enable LLMs to perform actions through your server

Tools are a powerful primitive in the Model Context Protocol (MCP) that enable servers to expose executable functionality to clients. Through tools, LLMs can interact with external systems, perform computations, and take actions in the real world.

> Tools are designed to be **model-controlled**, meaning that tools are exposed from servers to clients with the intention of the AI model being able to automatically invoke them (with a human in the loop to grant approval).

Overview

Tools in MCP allow servers to expose executable functions that can be invoked by clients and used by LLMs to perform actions. Key aspects of tools include:

- **Discovery**: Clients can list available tools through the `tools/list` endpoint
- **Invocation**: Tools are called using the `tools/call` endpoint, where servers perform the requested operation and return results
- **Flexibility**: Tools can range from simple calculations to complex API interactions

Like **resources**, tools are identified by unique names and can include descriptions to guide their usage. However, unlike resources, tools represent dynamic operations that can modify state or interact with external systems.

Tool definition structure

Each tool is defined with the following structure:

```
{
  name: string;          // Unique identifier for the tool
  description?: string;  // Human-readable description
  inputSchema: {         // JSON Schema for the tool's parameters
    type: "object",
    properties: { ... }  // Tool-specific parameters
  }
}
```

Implementing tools

Here's an example of implementing a basic tool in an MCP server:

TypeScript Python

```
const server = new Server({
  name: "example-server",
  version: "1.0.0"
},{
  capabilities: {
    tools: {}
  }
});

// Define available tools
server.setRequestHandler(ListToolsRequestSchema, async () => {
  return {
    tools: [{
      name: "calculate_sum",
      description: "Add two numbers together",
      inputSchema: {
        type: "object",
        properties: {
          a: { type: "number" },
          b: { type: "number" }
        },
        required: ["a", "b"]
      }
    }]
  }
});

// Handle tool execution
server.setRequestHandler(CallToolRequestSchema, async (request) => {
  if (request.params.name === "calculate_sum") {
    const { a, b } = request.params.arguments;
    return {
      toolResult: a + b
    };
  }
  throw new Error("Tool not found");
});
```

Example tool patterns

Here are some examples of types of tools that a server could provide:

System operations

Tools that interact with the local system:

```
{
  name: "execute_command",
  description: "Run a shell command",
  inputSchema: {
    type: "object",
    properties: {
      command: { type: "string" },
      args: { type: "array", items: { type: "string" } }
    }
  }
}
```

API integrations

Tools that wrap external APIs:

```
{
  name: "github_create_issue",
  description: "Create a GitHub issue",
  inputSchema: {
    type: "object",
    properties: {
      title: { type: "string" },
      body: { type: "string" },
      labels: { type: "array", items: { type: "string" } }
    }
  }
}
```

Data processing

Tools that transform or analyze data:

```
{
  name: "analyze_csv",
  description: "Analyze a CSV file",
  inputSchema: {
    type: "object",
    properties: {
      filepath: { type: "string" },
      operations: {
        type: "array",
        items: {
          enum: ["sum", "average", "count"]
        }
      }
    }
  }
}
```

Best practices

When implementing tools:

1. Provide clear, descriptive names and descriptions
2. Use detailed JSON Schema definitions for parameters
3. Include examples in tool descriptions to demonstrate how the model should use them
4. Implement proper error handling and validation
5. Use progress reporting for long operations
6. Keep tool operations focused and atomic
7. Document expected return value structures
8. Implement proper timeouts
9. Consider rate limiting for resource-intensive operations
10. Log tool usage for debugging and monitoring

Security considerations

When exposing tools:

Input validation

- Validate all parameters against the schema
- Sanitize file paths and system commands
- Validate URLs and external identifiers
- Check parameter sizes and ranges
- Prevent command injection

Access control

- Implement authentication where needed
- Use appropriate authorization checks
- Audit tool usage
- Rate limit requests
- Monitor for abuse

Error handling

- Don't expose internal errors to clients
- Log security-relevant errors
- Handle timeouts appropriately
- Clean up resources after errors
- Validate return values

Tool discovery and updates

MCP supports dynamic tool discovery:

1. Clients can list available tools at any time
2. Servers can notify clients when tools change using `notifications/tools/list_changed`
3. Tools can be added or removed during runtime
4. Tool definitions can be updated (though this should be done carefully)

Error handling

Tool errors should be reported within the result object, not as MCP protocol-level errors. This allows the LLM to see and potentially handle the error. When a tool encounters an error:

1. Set `isError` to `true` in the result
2. Include error details in the `content` array

Here's an example of proper error handling for tools:

TypeScript Python

```
try {
  // Tool operation
  const result = performOperation();
  return {
    content: [
      {
        type: "text",
        text: `Operation successful: ${result}`
      }
    ]
  };
} catch (error) {
  return {
    isError: true,
    content: [
      {
        type: "text",
        text: `Error: ${error.message}`
      }
    ]
  };
}
```

This approach allows the LLM to see that an error occurred and potentially take corrective action or request human intervention.

Testing tools

A comprehensive testing strategy for MCP tools should cover:

- **Functional testing**: Verify tools execute correctly with valid inputs and handle invalid inputs appropriately
- **Integration testing**: Test tool interaction with external systems using both real and mocked dependencies
- **Security testing**: Validate authentication, authorization, input sanitization, and rate limiting
- **Performance testing**: Check behavior under load, timeout handling, and resource cleanup
- **Error handling**: Ensure tools properly report errors through the MCP protocol and clean up resources

On this page
Overview
Tool definition structure
Implementing tools
Example tool patterns
System operations
API integrations
Data processing
Best practices
Security considerations
Input validation
Access control
Error handling
Tool discovery and updates
Error handling
Testing tools

Was this page helpful? Yes No

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

Sampling

Let your servers request completions from LLMs

Sampling is a powerful MCP feature that allows servers to request LLM completions through the client, enabling sophisticated agentic behaviors while maintaining security and privacy.

> This feature of MCP is not yet supported in the Claude Desktop client.

How sampling works

The sampling flow follows these steps:

1. Server sends a `sampling/createMessage` request to the client
2. Client reviews the request and can modify it
3. Client samples from an LLM
4. Client reviews the completion
5. Client returns the result to the server

This human-in-the-loop design ensures users maintain control over what the LLM sees and generates.

Message format

Sampling requests use a standardized message format:

```
{
  messages: [
    {
      role: "user" | "assistant",
      content: {
        type: "text" | "image",

        // For text:
        text?: string,

        // For images:
        data?: string,             // base64 encoded
        mimeType?: string
      }
    }
  ],
  modelPreferences?: {
    hints?: [{
      name?: string                // Suggested model name/family
    }],
    costPriority?: number,         // 0-1, importance of minimizing cost
    speedPriority?: number,        // 0-1, importance of low latency
    intelligencePriority?: number  // 0-1, importance of capabilities
  },
  systemPrompt?: string,
  includeContext?: "none" | "thisServer" | "allServers",
  temperature?: number,
  maxTokens: number,
  stopSequences?: string[],
  metadata?: Record<string, unknown>
}
```

Request parameters

Messages

The `messages` array contains the conversation history to send to the LLM. Each message has:

- `role`: Either "user" or "assistant"
- `content`: The message content, which can be:
  - Text content with a `text` field
  - Image content with `data` (base64) and `mimeType` fields

Model preferences

The `modelPreferences` object allows servers to specify their model selection preferences:

- `hints`: Array of model name suggestions that clients can use to select an appropriate model:
  - `name`: String that can match full or partial model names (e.g. "claude-3", "sonnet")
  - Clients may map hints to equivalent models from different providers
  - Multiple hints are evaluated in preference order
- Priority values (0-1 normalized):
  - `costPriority`: Importance of minimizing costs
  - `speedPriority`: Importance of low latency response
  - `intelligencePriority`: Importance of advanced model capabilities

Clients make the final model selection based on these preferences and their available models.

System prompt

An optional `systemPrompt` field allows servers to request a specific system prompt. The client may modify or ignore this.

Context inclusion

The `includeContext` parameter specifies what MCP context to include:

- `"none"`: No additional context
- `"thisServer"`: Include context from the requesting server
- `"allServers"`: Include context from all connected MCP servers

The client controls what context is actually included.

Sampling parameters

Fine-tune the LLM sampling with:

- `temperature`: Controls randomness (0.0 to 1.0)
- `maxTokens`: Maximum tokens to generate
- `stopSequences`: Array of sequences that stop generation
- `metadata`: Additional provider-specific parameters

Response format

The client returns a completion result:

```
{
  model: string,  // Name of the model used
  stopReason?: "endTurn" | "stopSequence" | "maxTokens" | string,
  role: "user" | "assistant",
  content: {
    type: "text" | "image",
    text?: string,
    data?: string,
    mimeType?: string
  }
}
```

Example request

Here's an example of requesting sampling from a client:

```
{
  "method": "sampling/createMessage",
  "params": {
    "messages": [
      {
        "role": "user",
        "content": {
          "type": "text",
          "text": "What files are in the current directory?"
        }
      }
    ],
    "systemPrompt": "You are a helpful file system assistant.",
    "includeContext": "thisServer",
    "maxTokens": 100
  }
}
```

Best practices

When implementing sampling:

1. Always provide clear, well-structured prompts
2. Handle both text and image content appropriately
3. Set reasonable token limits
4. Include relevant context through `includeContext`
5. Validate responses before using them
6. Handle errors gracefully
7. Consider rate limiting sampling requests
8. Document expected sampling behavior
9. Test with various model parameters
10. Monitor sampling costs

Human in the loop controls

Sampling is designed with human oversight in mind:

For prompts

- Clients should show users the proposed prompt
- Users should be able to modify or reject prompts
- System prompts can be filtered or modified
- Context inclusion is controlled by the client

For completions

- Clients should show users the completion
- Users should be able to modify or reject completions
- Clients can filter or modify completions
- Users control which model is used

Security considerations

When implementing sampling:

- Validate all message content
- Sanitize sensitive information
- Implement appropriate rate limits
- Monitor sampling usage
- Encrypt data in transit
- Handle user data privacy
- Audit sampling requests
- Control cost exposure
- Implement timeouts
- Handle model errors gracefully

Common patterns

Agentic workflows

Sampling enables agentic patterns like:

- Reading and analyzing resources
- Making decisions based on context
- Generating structured data
- Handling multi-step tasks
- Providing interactive assistance

Context management

Best practices for context:

- Request minimal necessary context
- Structure context clearly
- Handle context size limits
- Update context as needed
- Clean up stale context

Error handling

Robust error handling should:

- Catch sampling failures
- Handle timeout errors
- Manage rate limits
- Validate responses
- Provide fallback behaviors
- Log errors appropriately

Limitations

Be aware of these limitations:

- Sampling depends on client capabilities
- Users control sampling behavior
- Context size has limits
- Rate limits may apply
- Costs should be considered
- Model availability varies
- Response times vary
- Not all content types supported

On this page
How sampling works
Message format
Request parameters
Messages
Model preferences
System prompt
Context inclusion
Sampling parameters
Response format
Example request
Best practices
Human in the loop controls
For prompts
For completions
Security considerations
Common patterns
Agentic workflows
Context management
Error handling
Limitations

Was this page helpful? Yes No

Documentation
Python SDK
TypeScript SDK
规范

Get Started
MCP 简介
快速入门
MCP 客户端

Your First MCP Server
Python
TypeScript

开发工具
Debugging
Inspector

概念
核心架构
Resources
Prompts
Tools
Sampling
传输层

概念

传输层

了解 MCP 的通信机制

Model Context Protocol (MCP) 中的传输层为 clients 和 servers 之间的通信提供基础。传输层处理消息发送和接收的底层机制。

消息格式

MCP 使用 **JSON-RPC** 2.0 作为其传输格式。传输层负责将 MCP 协议消息转换为 JSON-RPC 格式进行传输，并将接收到的 JSON-RPC 消息转换回 MCP 协议消息。

使用的 JSON-RPC 消息有三种类型：

请求

```
{
  jsonrpc: "2.0",
  id: number | string,
  method: string,
  params?: object
}
```

响应

```
{
  jsonrpc: "2.0",
  id: number | string,
  result?: object,
  error?: {
    code: number,
    message: string,
    data?: unknown
  }
}
```

通知

```
{
  jsonrpc: "2.0",
  method: string,
  params?: object
}
```

内置传输层种类

MCP 包含两个标准传输实现：

标准输入输出 (stdio)

stdio 传输通过标准输入和输出流进行通信。这对于本地集成和命令行工具特别有用。

在以下情况下使用 stdio：

- 构建命令行工具
- 实现本地集成
- 需要简单的进程通信
- 使用 shell 脚本

TypeScript (Server) TypeScript (Client) Python (Server) Python (Client)

```
const server = new Server({
  name: "example-server",
  version: "1.0.0"
}, {
  capabilities: {}
});

const transport = new StdioServerTransport();
await server.connect(transport);
```

服务器发送事件 (SSE)

SSE 传输通过 HTTP POST 请求实现服务器到客户端的流式通信。

在以下情况下使用 SSE：

- 仅需要 server-to-client 的流式通信
- 在受限网络中工作
- 实现简单更新

TypeScript (Server) TypeScript (Client) Python (Server) Python (Client)

```
const server = new Server({
  name: "example-server",
  version: "1.0.0"
}, {
  capabilities: {}
});

const transport = new SSEServerTransport('/message', response);
await server.connect(transport);
```

自定义传输层

MCP 使得为特定需求实现自定义传输层变得简单。任何传输层实现只需符合 Transport 接口：

你可以为以下情况实现自定义传输：

- 自定义网络协议
- 专用通信通道
- 与现有系统集成
- 性能优化

TypeScript Python

```
interface Transport {
  // 开始处理消息
  start(): Promise<void>;

  // 发送 JSON-RPC 消息
  send(message: JSONRPCMessage): Promise<void>;

  // 关闭连接
  close(): Promise<void>;

  // 回调函数
  onclose?: () => void;
  onerror?: (error: Error) => void;
  onmessage?: (message: JSONRPCMessage) => void;
}
```

错误处理

传输实现应处理各种错误场景：

1. 连接错误
2. 消息解析错误
3. 协议错误
4. 网络超时
5. 资源清理

错误处理示例：

TypeScript Python

```
class ExampleTransport implements Transport {
  async start() {
    try {
      // 连接逻辑
    } catch (error) {
      this.onerror?.(new Error(`连接失败: ${error}`));
      throw error;
    }
  }

  async send(message: JSONRPCMessage) {
    try {
      // 发送逻辑
    } catch (error) {
      this.onerror?.(new Error(`消息发送失败: ${error}`));
      throw error;
    }
  }
}
```

最佳实践

在实现或使用 MCP 传输时：

1. 正确处理连接生命周期
2. 实现适当的错误处理
3. 在连接关闭时清理资源
4. 使用适当的超时
5. 在发送前验证消息
6. 记录传输事件以便调试
7. 在适当时实现重连逻辑
8. 处理消息队列中的背压
9. 监控连接健康状况
10. 实现适当的安全措施

安全考虑

在实现传输时：

认证和授权

- 实现适当的认证机制
- 验证客户端凭据
- 使用安全的令牌处理
- 实现授权检查

数据安全

- 使用 TLS 进行网络传输
- 加密敏感数据
- 验证消息完整性
- 实现消息大小限制
- 对输入数据 sanitize

网络安全

- 实现速率限制
- 使用适当的超时
- 处理拒绝服务场景
- 监控异常模式
- 实施适当的防火墙规则

调试传输

一些 Debug 传输层问题的提示：

1. 启用调试日志
2. 监控消息流
3. 检查连接状态
4. 验证消息格式
5. 测试错误场景
6. 使用网络分析工具
7. 实现健康检查
8. 监控资源使用
9. 测试边界情况
10. 使用适当的错误跟踪

Was this page helpful? Yes No

On this page
消息格式
请求
响应
通知
内置传输层种类
标准输入输出 (stdio)
服务器发送事件 (SSE)
自定义传输层
错误处理
最佳实践
安全考虑
认证和授权
数据安全
网络安全
调试传输