MCP 服务器开发指南

本指南将帮助您使用 Python 和 FastMCP 框架开发自己的 MCP 服务器，并将其集成到 Nexent 平台中。

🌐 语言支持

MCP 协议支持多种编程语言，包括：

- **Python** ⭐（推荐）
- **TypeScript**
- **Java**
- **Go**
- **Rust**
- 以及其他支持 MCP 协议的语言

为什么推荐 Python?

本指南使用 **Python** 作为示例语言，原因如下：

- ✅ **简单易学**：语法简洁，上手快速
- ✅ **丰富的框架**：FastMCP 等框架让开发变得非常简单
- ✅ **快速开发**：几行代码即可创建一个可用的 MCP 服务器
- ✅ **生态完善**：丰富的第三方库支持

如果您熟悉其他语言，也可以使用相应的 MCP SDK 进行开发。但如果您是第一次开发 MCP 服务器，我们强烈推荐从 Python 开始。

📋 前置要求

在开始之前，请确保您已安装以下依赖：

```
pip install fastmcp
```

🚀 快速开始

基础示例

以下是一个简单的 MCP 服务器示例，展示了如何使用 FastMCP 创建一个提供字符串处理功能的服务器：

```python
from fastmcp import FastMCP

# 创建MCP服务器实例
mcp = FastMCP(name="String MCP Server")

@mcp.tool(
    name="calculate_string_length",
    description="计算输入字符串的长度"
)
def calculate_string_length(text: str) -> int:
    return len(text)

@mcp.tool(
    name="to_uppercase",
    description="将字符串转换为大写"
)
def to_uppercase(text: str) -> str:
    return text.upper()

@mcp.tool(
    name="to_lowercase",
    description="将字符串转换为小写"
)
def to_lowercase(text: str) -> str:
    return text.lower()

if __name__ == "__main__":
    # 使用SSE协议启动服务
    mcp.run(transport="sse", port=8000)
```

运行服务器

保存上述代码为 `mcp_server.py` ，然后运行：

```
python mcp_server.py
```

您将看到 MCP server 成功启动，且 Server URL 为 `http://127.0.0.1:8000/sse` 。

🔌 在 Nexent 中集成 MCP 服务

开发并启动 MCP 服务后，您需要将其添加到 Nexent 平台中进行使用：

步骤 1：启动 MCP 服务器

确保您的 MCP 服务器正在运行，并记录其访问地址（例如： `http://127.0.0.1:8000/sse` ）。

步骤 2：在 Nexent 中添加 MCP 服务

1. 进入 **智能体开发** 页面
2. 在"选择Agent的工具"页签右侧，点击"**MCP配置**"
3. 在弹出的配置窗口中,输入服务器名称和服务器URL
   - ⚠️ **注意**：
   i. 服务器名称只能包含英文字母和数字，不能包含空格、下划线等其他字符;
   ii. 如果您使用 Docker 容器部署 Nexent，并且 MCP 服务器运行在宿主机上，需要将 `127.0.0.1` 替换为 `host.docker.internal` ，即 `http://host.docker.internal:8000` 才可成功访问宿主机上运行的 MCP 服务器。
4. 点击"**添加**"按钮完成配置

步骤 3：使用 MCP 工具

配置完成后，在创建或编辑智能体时，您可以在工具列表中找到并选择您添加的 MCP 工具。

🔧 包装现有业务

如果您已有现成的业务代码，想要将其包装成 MCP 服务，只需要在工具函数中进行调用即可。这种方式可以快速将现有服务集成到 MCP 生态系统中。

示例：包装 REST API

如果您的业务逻辑已有现成Restful API：

```python
from fastmcp import FastMCP
import requests

# 创建MCP服务器实例
mcp = FastMCP("Course Statistics Server")

@mcp.tool(
    name="get_course_statistics",
    description="根据课程号获取某门课程的成绩统计信息（包含平均分、最高分、最低分等）"
)
def get_course_statistics(course_id: str) -> str:
    # 调用现有的业务API
    api_url = "https://your-school-api.com/api/courses/statistics"
    response = requests.get(api_url, params={"course_id": course_id})

    # 处理响应并返回结果
    if response.status_code == 200:
        data = response.json()
        stats = data.get("statistics", {})
        return f"课程 {course_id} 成绩统计：\n平均分：{stats.get('average', 'N/A')}\n最高分：{sta
    else:
        return f"API调用失败：{response.status_code}"

if __name__ == "__main__":
    # 使用SSE协议启动服务
    mcp.run(transport="sse", port=8000)
```

示例：包装内部服务

如果您的业务逻辑在本地服务中：

```python
from fastmcp import FastMCP
from your_school_module import query_course_statistics

# 创建MCP服务器实例
mcp = FastMCP("Course Statistics Server")

@mcp.tool(
    name="get_course_statistics",
    description="根据课程号获取某门课程的成绩统计信息（包含平均分、最高分、最低分等）"
)
def get_course_statistics(course_id: str) -> str:
    # 直接调用内部业务函数
    try:
        stats = query_course_statistics(course_id)
        return f"课程 {course_id} 成绩统计：\n平均分：{stats.get('average', 'N/A')}\n最高分：{sta
    except Exception as e:
        return f"查询成绩统计时出错：{str(e)}"

if __name__ == "__main__":
    # 使用SSE协议启动服务
    mcp.run(transport="sse", port=8000)
```

更多资源

Python

- FastMCP 文档（本指南使用的框架）

其他语言

- MCP TypeScript SDK
- MCP Java SDK
- MCP Go SDK
- MCP Rust SDK