

使用markmap读取markdown格式文件生成思维导图

0.环境准备

- conda：已经安装好Anaconda或miniconda。
- node：安装好node，配置好node环境，使用最新版markmap需要使用node 18+。

1. 项目环境准备

1.1 安装markmap工具

```
npm install -g markmap-cli
```

1.2 创建项目环境

```
conda create -n mindmap python==3.10
```

```
conda activate mindmap
```

1.3 安装fastAPI

通过fastAPI对其他程序提供HTTP服务，也可以使用flask。

```
pip install fastapi uvicorn
```

1.4 验证markmap功能

#找一个目录新建mindmap文件夹，参考附录的markdown数据生成markdown文件，将markdown文件存放在新建文件夹的markdown文件夹下

#然后执行markmap指令是否能正常执行，并生成html文件

```
markmap markdown/1741852665.md --no-open
```

生成的html文件如下：



markmap + - ↺ ↻

2. 代码逻辑实现

在第一章节新建的根目录下新建markmap.py文件，编写主程序实现逻辑。

```
# 引入依赖包
from fastapi import FastAPI, Request
from fastapi.responses import FileResponse
import time
import subprocess
import os
import sys

# 使用 FastAPI 初始化应用
app = FastAPI()

# 测试根路径
@app.get('/')
def index():
    return "FastAPI server is running!"

# 上传并处理Markdown文件的路径
@app.post('/upload')
async def upload_markdown(request: Request):
    python_path = sys.executable
    print(f"python_path={python_path}")
    content = await request.body()
    content = content.decode('utf-8')
    time_name = str(int(time.time())) # 生成时间戳作为文件名
    md_file_name = time_name + ".md" # Markdown文件名
```

```

html_file_name = time_name + ".html" # HTML文件名

# 创建markdown和html文件夹，如果它们不存在的话
os.makedirs('markdown', exist_ok=True)
os.makedirs('static/html', exist_ok=True)

# 将Markdown内容写入文件
with open(f'markdown/{md_file_name}', 'w', encoding='utf-8') as f:
    f.write(content)

print(f"Markdown file created: markdown/{md_file_name}")

current_dir = os.getcwd()
print(f"Current dir: {current_dir}")
os.chdir(current_dir)
# 使用subprocess调用markmap-cli将Markdown转换为HTML，并移动到static/html目录
try:

    print(f"开始markmap")

    import shutil
    markmap_path = shutil.which('markmap')
    # 判断markmap是否存在
    if markmap_path is None:
        return "Error: markmap command not found. Please make sure it is installed and
added to the system PATH."

    # 构建markdown生产html文件命令
    markdown_cmd = f"markmap markdown/{md_file_name} --output markdown/{html_file_name}
--no-open"

    # 注意在windows环境下一定要使用powershell执行。默认使用的cmd，会出现生成不了html文件的情况。完
    全无输出，也不报错。
    if os.name == 'nt':
        # 在windows上使用PowerShell执行命令
        markdown_cmd = f"powershell -Command markmap markdown/{md_file_name} --output
markdown/{html_file_name} --no-open"

    print(f"即将执行的命令: {markdown_cmd}")
    # 主要shell=True必须加，不然会把markdown_cmd里面的内容整个字符串当作一个命令，而不是markmap命
    令和参数，会报错windows文件不存在
    result = subprocess.run(markdown_cmd,
                             check=True, text=True, shell=True,
                             stdout=subprocess.PIPE, stderr=subprocess.PIPE,
universal_newlines=True)
    print(f"命令返回码: {result.returncode}")
    print(f"命令输出: {result.stdout}")
    print(f"命令错误信息: {result.stderr}")
    if result.returncode != 0:
        raise subprocess.CalledProcessError(result.returncode, result.args,
output=result.stdout, stderr=result.stderr)

    # 尝试将生成的HTML文件移动到static/html文件夹
    os.replace(f'markdown/{html_file_name}', f'static/html/{html_file_name}')

```

```

print(f"HTML file moved to: static/html/{html_file_name}")

# 返回转换后的HTML文件链接
base_url = str(request.base_url)
preview_url = f"{base_url}html/{html_file_name}"
return f'Markdown文件已保存。 点击预览: {preview_url}'
except subprocess.CalledProcessError as e:
    # 如果转换过程中出现错误，返回错误信息
    return f"Error generating HTML file: {e.output}\n{e.stderr}"
except Exception as e:
    return f"Unexpected error: {str(e)}"

# 提供HTML文件的路径
@app.get('/html/{filename}')
def get_html(filename: str):
    return FileResponse(f'static/html/{filename}')

# 启动http服务监听
if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host='0.0.0.0', port=5001)

```

3. 启动服务

在第一章节创建的python虚拟环境mindmap环境下执行编写好的python脚本。

```
python markmap.py
```

```

D:\lark-projects\mindmap>conda activate mindmap

(mindmap) D:\lark-projects\mindmap>python markmap.py
INFO: Started server process [33192]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://0.0.0.0:5001 (Press CTRL+C to quit)

```

附录

大模型微调指南

引言

大模型微调是自然语言处理领域中的一种常见技术，旨在通过调整预训练模型的参数，使其更好地适应特定任务或数据集。

微调步骤

1. 选择预训练模型

- 模型类型：根据任务需求选择合适的预训练模型，如文本生成、文本分类、命名实体识别等。
- 模型规模：根据计算资源选择适当规模的模型，大型模型通常性能更好，但也需要更多的计算资源。

2. 准备数据集

- 数据收集：收集与任务相关的数据，确保数据的质量和多样性。
- 数据预处理：对收集到的数据进行清洗、分词、标注等预处理操作，使其符合模型输入的要求。
- 数据划分：将数据集划分为训练集、验证集和测试集，用于模型训练、验证和测试。

3. 设置训练参数

- 学习率：设置合适的学习率，通常使用较小的学习率进行微调。
- 批量大小（Batch Size）：根据计算资源设置合适的批量大小。
- 训练轮数（Epochs）：根据验证集性能确定训练轮数，避免过拟合。

4. 微调模型

- 加载预训练模型：使用深度学习框架加载预训练模型。
- 修改模型输出层：根据任务需求修改模型的输出层，使其适应特定任务的输出格式。
- 训练模型：在训练集上训练模型，同时使用验证集进行性能监控，调整学习率等参数。

5. 模型评估与测试

- 评估指标：根据任务需求选择合适的评估指标，如准确率、F1分数、BLEU分数等。
- 模型测试：在测试集上评估模型的性能，确保模型具有良好的泛化能力。

6. 模型部署与监控

- 模型导出：将训练好的模型导出为可部署的格式，如ONNX、TensorFlow SavedModel等。
- 模型部署：将模型部署到生产环境中，确保模型能够实时处理输入数据。
- 性能监控：持续监控模型在生产环境中的性能，及时发现并解决问题。

注意事项

- 数据质量：确保数据的质量和多样性，避免数据偏差导致模型性能下降。
- 计算资源：根据计算资源选择合适的模型规模和批量大小，避免资源不足导致训练失败。
- 过拟合：使用正则化、dropout等技术防止模型过拟合，同时关注验证集性能。
- 模型解释性：对于某些任务，可能需要关注模型的解释性，以便更好地理解模型的决策过程。