# 分析 Spring Boot 專案

要分析一個 Spring Boot 專案（可能包含多個模組）並提取所有使用 `@GetMapping` 或 `@PostMapping` 注釋定義的 API，我們需要遍歷專案目錄，識別控制器類，提取 API 路徑，並以指定格式打印它們。以下是一個完成這項任務的完整 Python 腳本，靈感來自提供的代碼結構，但根據新需求進行了調整。輸出將以“XXController: get 路徑 post 路徑”格式列出每個控制器及其相關的 GET 和 POST 路徑。

### 方法

1. **遍歷目錄**：遞歸查找專案目錄及其子目錄中的所有 `.java` 文件。
2. **識別控制器**：檢查每個 Java 文件是否包含 `@Controller` 或 `@RestController` 注釋，以確認它是控制器類。
3. **提取類級路徑**：查找類級別的 `@RequestMapping` 注釋以確定基礎路徑。
4. **提取方法級 API**：查找使用 `@GetMapping` 或 `@PostMapping` 注釋的方法，提取它們的路徑，並與基礎路徑結合（如果有）。
5. **組織和打印**：收集每個控制器的 GET 和 POST 路徑，並以指定格式打印它們。

### Python 腳本

```
import os
import sys
import re
from collections import defaultdict

def find_java_files(root_dir):
    """
    遞歸查找給定目錄及其子目錄中的所有 .java 文件。

    Args:
        root_dir (str): 開始搜索的根目錄。

    Yields:
        str: 每個 .java 文件的完整路徑。
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_paths(line, annotation_type):
    """
```

```
    從 Spring 注釋（@GetMapping、@PostMapping 或 @RequestMapping）中提取路徑值。

    Args:
        line (str): 包含註釋的行。
        annotation_type (str): 注釋類型（'GetMapping'、'PostMapping' 或 'RequestMapping'）。

    Returns:
        list: 從註釋中提取的路徑字符串列表。
    """
    if annotation_type in ['GetMapping', 'PostMapping']:
        match = re.search(rf'@{annotation_type}\((.*)\)', line)
        if match:
            content = match.group(1)
            # 提取引號內的所有字符串字面量
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
    elif annotation_type == 'RequestMapping':
        match = re.search(r'@RequestMapping\((.*)\)', line)
        if match:
            content = match.group(1)
            # 查找 'value' 或 'path' 屬性
            value_match = re.search(r'(value|path)\s*=\s*({[^}]*}|"[^"]*")', content)
            if value_match:
                value = value_match.group(2)
                if value.startswith('{'):
                    paths = re.findall(r'"([^"]*)"', value)
                else:
                    paths = [value.strip('"')]
                return paths
            # 如果沒有 'value' 或 'path'，則假設直接路徑規範
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []

if __name__ == '__main__':
    # 解析命令行參數
    if len(sys.argv) != 2:
        print(" 使用方法: python script.py <root_directory>")
        sys.exit(1)
```

```
root_dir = sys.argv[1]
if not os.path.isdir(root_dir):
    print(f"[ERROR] 指定的路徑不是目錄: {root_dir}")
    sys.exit(1)

print(f"[INFO] 開始分析目錄: {root_dir}")

# 初始化一個字典來存儲控制器映射
controllers = defaultdict(lambda: {'GET': [], 'POST': []})
total_files = 0
error_files = 0

# 處理所有 Java 文件
for java_file in find_java_files(root_dir):
    try:
        with open(java_file, 'r', encoding='utf-8') as f:
            lines = f.readlines()

        # 檢查文件是否是控制器
        if any('@Controller' in line or '@RestController' in line for line in lines):
            controller_name = os.path.basename(java_file).replace('.java', '')

            # 查找類聲明行以分隔類級別和方法級別的註釋
            class_line_index = None
            for i, line in enumerate(lines):
                if re.search(r'public\s+(class|abstract\s+class|interface)\s+\w+', line):
                    class_line_index = i
                    break
            if class_line_index is None:
                continue

            # 提取類級別 @RequestMapping 作為基礎路徑
            base_paths = []
            for line in lines[:class_line_index]:
                if re.search(r'\s*@RequestMapping', line):
                    base_paths = extract_paths(line, 'RequestMapping')
                    break
            if not base_paths:
                base_paths = ['']
```

```
            # 提取方法級別 @GetMapping 和 @PostMapping
            get_paths = []
            post_paths = []
            for line in lines[class_line_index:]:
                if re.search(r'\s*@GetMapping', line):
                    paths = extract_paths(line, 'GetMapping')
                    for base in base_paths:
                        for path in paths:
                            full_path = base + path
                            get_paths.append(full_path)
                elif re.search(r'\s*@PostMapping', line):
                    paths = extract_paths(line, 'PostMapping')
                    for base in base_paths:
                        for path in paths:
                            full_path = base + path
                            post_paths.append(full_path)

            # 存儲唯一路徑
            get_paths = sorted(list(set(get_paths)))
            post_paths = sorted(list(set(post_paths)))

            if get_paths or post_paths:
                controllers[controller_name]['GET'] = get_paths
                controllers[controller_name]['POST'] = post_paths

        total_files += 1
    except Exception as e:
        print(f"[ERROR] 無法讀取文件 {java_file}: {e}")
        error_files += 1

# 打印摘要
print(f"[INFO] 總共嘗試的 Java 文件: {total_files + error_files}")
print(f"[INFO] 成功處理: {total_files}")
print(f"[INFO] 文件錯誤: {error_files}")
print(f"[INFO] 總控制器數: {len(controllers)}")

# 以指定格式打印結果
for controller, mappings in sorted(controllers.items()):
    print(f"{controller}:")
```

```
    for path in mappings['GET']:
        print(f"get {path}")
    for path in mappings['POST']:
        print(f"post {path}")
```

### 說明

- **導入**：我們使用 `os` 進行目錄遍歷，`sys` 進行命令行參數處理，`re` 進行正則表達式匹配，並使用 `defaultdict` 來組織控制器數據。
- `find_java_files`：該函數遞歸地產生專案目錄及其子目錄中的所有 `.java` 文件，通過 `os.walk` 自然處理多個模組。
- `extract_paths`：該函數解析 `@GetMapping`、`@PostMapping` 或 `@RequestMapping` 注釋以提取路徑值。它處理：
  - 單一路徑（例如 `@GetMapping("/path")`）。
  - 多個路徑（例如 `@GetMapping({"/path1", "/path2"})`）。
  - 命名屬性（例如 `@RequestMapping(value = "/path")`）。
- **主邏輯**：
  - **命令行處理**：以根目錄作為輸入，類似於提供的腳本。
  - **控制器檢測**：檢查文件內容中是否包含 `@Controller` 或 `@RestController`。
  - **基礎路徑**：在類聲明之前提取類級別 `@RequestMapping` 路徑。
  - **API 提取**：處理類聲明後的行以查找 `@GetMapping` 和 `@PostMapping`，將路徑與基礎路徑結合。
  - **輸出**：以每行一個控制器及其 GET 和 POST 路徑的格式打印，前綴為“get”或“post”。
- **錯誤處理**：在文件讀取過程中捕捉異常並報告它們，模仿提供代碼的穩健性。

### 範例輸出

對於一個包含 `UserController.java` 文件的 Spring Boot 專案：

```
@RestController
@RequestMapping("/users")
public class UserController {
    @GetMapping("/all")
    public ResponseEntity<?> getAllUsers() { ... }

    @PostMapping("/create")
    public ResponseEntity<?> createUser() { ... }
}
```

運行 `python script.py /path/to/project` 可能會產生：

```
[INFO] 開始分析目錄: /path/to/project
[INFO] 總共嘗試的 Java 文件: 10
[INFO] 成功處理: 10
[INFO] 文件錯誤: 0
[INFO] 總控制器數: 1
UserController:
get /users/all
post /users/create
```

### 注意事項

- **簡化**：該腳本假設註釋在單行上，路徑是字符串字面量，沒有複雜表達式。對於生產使用，Java 解析器可能更加穩健。
- **僅 GET 和 POST**：如要求，它專注於 `@GetMapping` 和 `@PostMapping`，忽略其他方法如 PUT 或 DELETE（儘管思考過程考慮了 `@RequestMapping` 方法）。
- **路徑連接**：直接將類級別和方法級別路徑結合（例如 `/users` + `/all` = `/users/all`）。Spring 會標準化多個斜杠，因此這對於顯示目的已足夠。

該腳本滿足查詢的需求，同時保持與提供代碼相似的結構，確保熟悉性和可靠性。