Analizar Proyectos Spring Boot

Para analizar un proyecto Spring Boot (que puede consistir en múltiples módulos) y extraer todas las APIs definidas con las anotaciones `@GetMapping` o `@PostMapping`, necesitamos recorrer el directorio del proyecto, identificar las clases de controlador, extraer las rutas de la API y mostrarlas en un formato especificado. A continuación se muestra un script completo de Python que realiza esta tarea, inspirado en la estructura del código proporcionado pero adaptado a los nuevos requisitos. La salida listará cada controlador con sus rutas GET y POST asociadas en el formato "XXController: get path post path".

Enfoque

1. **Recorrer el Directorio**: Encontrar recursivamente todos los archivos `.java` en el directorio del proyecto y sus subdirectorios.
2. **Identificar Controladores**: Verificar cada archivo Java para las anotaciones `@Controller` o `@RestController` para confirmar que es una clase de controlador.
3. **Extraer Rutas a Nivel de Clase**: Buscar anotaciones `@RequestMapping` a nivel de clase para determinar las rutas base.
4. **Extraer APIs a Nivel de Método**: Encontrar métodos anotados con `@GetMapping` o `@PostMapping`, extraer sus rutas y combinarlas con las rutas base si están presentes.
5. **Organizar e Imprimir**: Recopilar las rutas GET y POST por controlador e imprimirlas en el formato especificado.

Script de Python

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

def find_java_files(root_dir):
    """
    Encontrar recursivamente todos los archivos .java en el directorio dado y sus subdirectorios.

    Args:
        root_dir (str): El directorio raíz desde donde comenzar la búsqueda.

    Yields:
        str: La ruta completa a cada archivo .java.
    """
    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):
    """
    Extraer valores de ruta de una anotación de Spring (@GetMapping, @PostMapping o @RequestMapping).

    Args:
        line (str): La línea que contiene la anotación.
        annotation_type (str): El tipo de anotación ('GetMapping', 'PostMapping' o 'RequestMapping').

    Returns:
        list: Una lista de cadenas de ruta extraídas de la anotación.
    """
    if annotation_type in ['GetMapping', 'PostMapping']:
        match = re.search(rf'@{annotation_type}\((.*)\)', line)
        if match:
            content = match.group(1)
            # Extraer todos los literales de cadena dentro de comillas
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
    elif annotation_type == 'RequestMapping':
        match = re.search(r'@RequestMapping\((.*)\)', line)
        if match:
            content = match.group(1)
            # Buscar el atributo 'value' o 'path'
            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
            # Si no hay 'value' o 'path', asumir especificación de ruta directa
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
```

```python
if __name__ == '__main__':
    # Analizar argumentos de la línea de comandos
    if len(sys.argv) != 2:
        print("Uso: python script.py <directorio_raíz>")
        sys.exit(1)

    root_dir = sys.argv[1]
    if not os.path.isdir(root_dir):
        print(f"[ERROR] La ruta especificada no es un directorio: {root_dir}")
        sys.exit(1)

    print(f"[INFO] Iniciando análisis del directorio: {root_dir}")

    # Inicializar un diccionario para almacenar los mapeos de controladores
    controllers = defaultdict(lambda: {'GET': [], 'POST': []})
    total_files = 0
    error_files = 0

    # Procesar todos los archivos Java
    for java_file in find_java_files(root_dir):
        try:
            with open(java_file, 'r', encoding='utf-8') as f:
                lines = f.readlines()

            # Verificar si el archivo es un controlador
            if any('@Controller' in line or '@RestController' in line for line in lines):
                controller_name = os.path.basename(java_file).replace('.java', '')

                # Encontrar la línea de declaración de clase para separar las anotaciones a nivel de clase y
                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

                # Extraer @RequestMapping a nivel de clase como rutas base
                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 = ['']

            # Extraer @GetMapping y @PostMapping a nivel de método
            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)

            # Almacenar rutas únicas
            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] No se pudo leer el archivo {java_file}: {e}")
        error_files += 1

# Imprimir resumen
print(f"[INFO] Total de archivos Java intentados: {total_files + error_files}")
print(f"[INFO] Procesados con éxito: {total_files}")
```

```
    print(f"[INFO] Archivos con errores: {error_files}")
    print(f"[INFO] Total de controladores encontrados: {len(controllers)}")

    # Imprimir resultados en el formato especificado
    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}")
```

Explicación

- **Importaciones**: Usamos `os` para recorrer directorios, `sys` para argumentos de la línea de comandos, `re` para expresiones regulares y `defaultdict` para organizar los datos del controlador.
- `find_java_files`: Esta función devuelve recursivamente todos los archivos `.java` en el directorio del proyecto, manejando múltiples módulos de manera natural mediante `os.walk`.
- `extract_paths`: Esta función analiza las anotaciones `@GetMapping`, `@PostMapping` o `@RequestMapping` para extraer los valores de las rutas. Maneja:
  - Rutas únicas (por ejemplo, `@GetMapping("/path")`).
  - Múltiples rutas (por ejemplo, `@GetMapping({"/path1", "/path2"})`).
  - Atributos con nombre (por ejemplo, `@RequestMapping(value = "/path")`).
- **Lógica Principal**:
  - **Manejo de la Línea de Comandos**: Toma un directorio raíz como entrada, similar al script proporcionado.
  - **Detección de Controladores**: Verifica `@Controller` o `@RestController` en el contenido del archivo.
  - **Rutas Base**: Extrae rutas `@RequestMapping` a nivel de clase antes de la declaración de la clase.
  - **Extracción de APIs**: Procesa las líneas después de la declaración de la clase para `@GetMapping` y `@PostMapping`, combinando rutas con rutas base.
  - **Salida**: Imprime cada controlador con sus rutas GET y POST, una por línea, precedidas de “get”o “post”.
- **Manejo de Errores**: Captura excepciones durante la lectura del archivo y las informa, reflejando la robustez del código proporcionado.

Salida de Ejemplo

Para un proyecto Spring Boot con un archivo `UserController.java`:

```
@RestController
@RequestMapping("/users")
```

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

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

Ejecutar `python script.py /ruta/al/proyecto` podría producir:

```
[INFO] Iniciando análisis del directorio: /ruta/al/proyecto
[INFO] Total de archivos Java intentados: 10
[INFO] Procesados con éxito: 10
[INFO] Archivos con errores: 0
[INFO] Total de controladores encontrados: 1
UserController:
get /users/all
post /users/create
```

Notas

- **Simplificaciones**: El script asume que las anotaciones están en una sola línea y que las rutas son literales de cadena sin expresiones complejas. Para uso en producción, un analizador de Java podría ser más robusto.
- **Solo GET y POST**: Como se solicitó, se centra en `@GetMapping` y `@PostMapping`, ignorando otros métodos como PUT o DELETE (aunque el rastro de pensamiento consideró `@RequestMapping` con métodos).
- **Concatenación de Rutas**: Combina rutas a nivel de clase y de método directamente (por ejemplo, `/users` + `/all` = `/users/all`). Spring normaliza múltiples barras diagonales, por lo que esto es suficiente para fines de visualización.