Spring Boot Projekte analysieren

Um ein Spring Boot-Projekt (das aus mehreren Modulen bestehen kann) zu analysieren und alle mit `@GetMapping` oder `@PostMapping` definierten APIs zu extrahieren, müssen wir das Projektverzeichnis durchsuchen, Controller-Klassen identifizieren, die API-Pfade extrahieren und sie in einem bestimmten Format ausgeben. Unten ist ein vollständiges Python-Skript, das diese Aufgabe erledigt, inspiriert von der Struktur des bereitgestellten Codes, aber an die neuen Anforderungen angepasst. Die Ausgabe wird jede Steuerung mit ihren zugehörigen GET- und POST-Pfaden im Format "XXController: get Pfad post Pfad"auflisten.

Vorgehensweise

1. **Durchsuchen des Verzeichnisses**: Rekursiv alle `.java`-Dateien im Projektverzeichnis und seinen Unterverzeichnissen finden.
2. **Controller identifizieren**: Jede Java-Datei auf `@Controller` oder `@RestController`-Annotationen überprüfen, um zu bestätigen, dass es sich um eine Controller-Klasse handelt.
3. **Klassenebenen-Pfade extrahieren**: Nach `@RequestMapping`-Annotationen auf Klassenebene suchen, um Basis-Pfade zu bestimmen.
4. **Methodenebenen-APIs extrahieren**: Methoden suchen, die mit `@GetMapping` oder `@PostMapping` annotiert sind, deren Pfade extrahieren und sie mit Basis-Pfaden kombinieren, falls vorhanden.
5. **Organisieren und Drucken**: GET- und POST-Pfade pro Controller sammeln und sie im angegebenen Format ausgeben.

Python-Skript

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

def find_java_files(root_dir):
    """
    Rekursiv alle .java-Dateien im angegebenen Verzeichnis und seinen Unterverzeichnissen finden.

    Args:
        root_dir (str): Das Stammverzeichnis, von dem die Suche beginnen soll.

    Yields:
        str: Der vollständige Pfad zu jeder .java-Datei.
    """
    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):
    """
    Pfadwerte aus einer Spring-Annotation (@GetMapping, @PostMapping oder @RequestMapping) extrahieren.

    Args:
        line (str): Die Zeile, die die Annotation enthält.
        annotation_type (str): Der Typ der Annotation ('GetMapping', 'PostMapping' oder 'RequestMapping').

    Returns:
        list: Eine Liste von Pfadzeichenfolgen, die aus der Annotation extrahiert wurden.
    """
    if annotation_type in ['GetMapping', 'PostMapping']:
        match = re.search(rf'@{annotation_type}\((.*)\)', line)
        if match:
            content = match.group(1)
            # Alle Zeichenfolgenliterale innerhalb von Anführungszeichen extrahieren
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
    elif annotation_type == 'RequestMapping':
        match = re.search(r'@RequestMapping\((.*)\)', line)
        if match:
            content = match.group(1)
            # Nach 'value' oder 'path'-Attribut suchen
            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
            # Wenn kein 'value' oder 'path', direkte Pfadspezifikation annehmen
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
```

```
if __name__ == '__main__':
    # Befehlszeilenargumente analysieren
    if len(sys.argv) != 2:
        print("Verwendung: python script.py <root_directory>")
        sys.exit(1)

    root_dir = sys.argv[1]
    if not os.path.isdir(root_dir):
        print(f"[FEHLER] Der angegebene Pfad ist kein Verzeichnis: {root_dir}")
        sys.exit(1)

    print(f"[INFO] Analyse des Verzeichnisses wird gestartet: {root_dir}")

    # Initialisieren eines Wörterbuchs, um Controller-Zuordnungen zu speichern
    controllers = defaultdict(lambda: {'GET': [], 'POST': []})
    total_files = 0
    error_files = 0

    # Alle Java-Dateien verarbeiten
    for java_file in find_java_files(root_dir):
        try:
            with open(java_file, 'r', encoding='utf-8') as f:
                lines = f.readlines()

            # Überprüfen, ob die Datei ein Controller ist
            if any('@Controller' in line or '@RestController' in line for line in lines):
                controller_name = os.path.basename(java_file).replace('.java', '')

                # Zeile der Klassendeklaration finden, um Klassenebenen- und Methodenebenen-Annotationen zu t
                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

                # Klassenebenen-@RequestMapping als Basis-Pfade extrahieren
                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 = ['']

            # Methodenebenen-@GetMapping und @PostMapping extrahieren
            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)

            # Eindeutige Pfade speichern
            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"[FEHLER] Datei {java_file} konnte nicht gelesen werden: {e}")
        error_files += 1

# Zusammenfassung ausgeben
print(f"[INFO] Gesamtzahl der versuchten Java-Dateien: {total_files + error_files}")
print(f"[INFO] Erfolgreich verarbeitet: {total_files}")
```

```
    print(f"[INFO] Dateien mit Fehlern: {error_files}")
    print(f"[INFO] Gesamtzahl der gefundenen Controller: {len(controllers)}")

    # Ergebnisse im angegebenen Format ausgeben
    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}")
```

Erklärung

- **Imports**: Wir verwenden `os` für die Verzeichnisdurchsuchung, `sys` für Befehlszeilenargumente, `re` für reguläre Ausdrücke und `defaultdict`, um Controller-Daten zu organisieren.
- `find_java_files`: Diese Funktion gibt rekursiv alle `.java`-Dateien im Projektverzeichnis aus, wobei mehrere Module durch `os.walk` natürlich gehandhabt werden.
- `extract_paths`: Diese Funktion analysiert `@GetMapping`, `@PostMapping` oder `@RequestMapping`-Annotationen, um Pfadwerte zu extrahieren. Sie behandelt:
  - Einzige Pfade (z.B. `@GetMapping("/pfad")`).
  - Mehrere Pfade (z.B. `@GetMapping({"/pfad1", "/pfad2"})`).
  - Benannte Attribute (z.B. `@RequestMapping(value = "/pfad")`).
- **Hauptlogik**:
  - **Befehlszeilenverarbeitung**: Nimmt ein Stammverzeichnis als Eingabe, ähnlich wie im bereitgestellten Skript.
  - **Controller-Erkennung**: Überprüft auf `@Controller` oder `@RestController` im Dateiinhalt.
  - **Basis-Pfade**: Extrahiert Klassenebenen-`@RequestMapping`-Pfade vor der Klassendeklaration.
  - **API-Extraktion**: Verarbeitet Zeilen nach der Klassendeklaration für `@GetMapping` und `@PostMapping`, kombiniert Pfade mit Basis-Pfaden.
  - **Ausgabe**: Gibt jeden Controller mit seinen GET- und POST-Pfaden aus, einen pro Zeile, vorangestellt mit “get”oder “post”.
- **Fehlerbehandlung**: Fängt Ausnahmen beim Dateilesen ab und meldet sie, was die Robustheit des bereitgestellten Codes spiegelt.

Beispielausgabe

Für ein Spring Boot-Projekt mit einer Datei `UserController.java`:

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

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

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

Führen Sie `python script.py /pfad/zum/projekt` aus, könnte dies produzieren:

```
[INFO] Analyse des Verzeichnisses wird gestartet: /pfad/zum/projekt
[INFO] Gesamtzahl der versuchten Java-Dateien: 10
[INFO] Erfolgreich verarbeitet: 10
[INFO] Dateien mit Fehlern: 0
[INFO] Gesamtzahl der gefundenen Controller: 1
UserController:
get /users/all
post /users/create
```

Hinweise

- **Vereinfachungen**: Das Skript geht davon aus, dass Annotationen auf einzelnen Zeilen sind und Pfade Zeichenfolgenliterale ohne komplexe Ausdrücke sind. Für den Produktionsgebrauch könnte ein Java-Parser robuster sein.
- **Nur GET und POST**: Wie angefordert, konzentriert es sich auf `@GetMapping` und `@PostMapping`, ignoriert andere Methoden wie PUT oder DELETE (obwohl die Denkspur `@RequestMapping` mit Methoden berücksichtigte).
- **Pfadverkettung**: Kombiniert Klassenebenen- und Methodenebenen-Pfade direkt (z.B. `/users` + `/all` = `/users/all`). Spring normalisiert mehrere Schrägstriche, sodass dies für Anzeigezwecke ausreicht.

Dieses Skript erfüllt die Anforderungen der Abfrage, während es eine Struktur ähnlich dem bereitgestellten Code beibehält, was Vertrautheit und Zuverlässigkeit gewährleistet.