تحليل مشاريع □□□□□□ □□□□

لتحليل مشروع □□□□□ □□□□ □□□□ الذي قد يتكون من عدة وحدات واستخراج جميع واجهات برمجة التطبيقات □□□□□ التي تم تعريفها باستخدام تعليقات @GetMapping أو @PostMapping، علينا أن نمر على دليل المشروع، ونحدد فئات التحكم، ونستخرج مسارات واجهات برمجة التطبيقات، ونطبعها في تنسيق محدد. أدناه هو نص □□□□□□ كامل يقوم بهذه المهمة، مستوحى من بنية الكود المقدم ولكن مخصص للطلبات الجديدة. سيتم أن يكون الناتج قائمة لكل محكم مع مسارات □□□ و □□□□ المرتبطة به في التنسيق □□□□□□□□□□□□□: □□□ □□□□ □□□□ □□□□□.

النهج

1. **مرور على الدليل**: البحث عن جميع ملفات .java في الدليل المشروع و فروعه بشكل متكرر.
2. **تحديد المحكمات**: فحص كل ملف □□□□ للحصول على تعليقات @Controller أو @RestController لتأكيد أنه فئة محكم.
3. **استخراج مسارات مستوى الفئة**: البحث عن تعليقات @RequestMapping على مستوى الفئة لتحديد المسارات الأساسية.
4. **استخراج واجهات برمجة التطبيقات على مستوى الطريقة**: البحث عن الطرق التي تم تعليقاتها @GetMapping أو @PostMapping، واستخراج مساراتها، ودمجها مع المسارات الأساسية إذا كانت موجودة.
5. **تنظيم وطباعة**: جمع مسارات □□□ و □□□□ لكل محكم وطباعتها في التنسيق المحدد.

□□□□□□ النص

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

def find_java_files(root_dir):
    """
                .java                   .

    Args:
        root_dir (str):               .

    Yields:
        str:                 .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):
    """
                    Spring (@GetMapping, @PostMapping,   @RequestMapping).

    Args:
        line (str):           .
        annotation_type (str):        )'GetMapping', 'PostMapping',   'RequestMapping').

    Returns:
        list:                       .
    """
    if annotation_type in ['GetMapping', 'PostMapping']:
        match = re.search(rf'@{annotation_type}\((.*)\)', line)
        if match:
            content = match.group(1)
            #
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []
    elif annotation_type == 'RequestMapping':
        match = re.search(r'@RequestMapping\((.*)\)', line)
        if match:
            content = match.group(1)
            #       'value'   '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
            #           'value'   'path',
            paths = re.findall(r'"([^"]*)"', content)
            return paths
        return []

if __name__ == '__main__':
    #
```

```
if len(sys.argv) != 2:
    print(:    " python script.py <root_directory>")
    sys.exit(1)

root_dir = sys.argv[1]
if not os.path.isdir(root_dir):
    print(f"[ ]            :   }root_dir}")
    sys.exit(1)

print(f"[    ]        :    }root_dir}")

#
controllers = defaultdict(lambda: {'GET': [], 'POST': []})
total_files = 0
error_files = 0

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

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

            #
            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

            #      @RequestMapping
            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 = ['']

            #      @GetMapping   @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)

            #
            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"[ ]              }java_file}: {e}")
        error_files += 1

#
print(f"[    ]         Java :     }total_files + error_files}")
print(f"[    ]         :    }total_files}")
print(f"[    ]       :    }error_files}")
print(f"[    ]           :     }len(controllers)}")
```

```
    #
    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}")
```

شرح

- **الاستيرادات**: نستخدم os لمرور الدليل، sys لمعالجة المعطيات من السطر الأوامر، re للعبارات العادية، و defaultdict لتنظيم بيانات المحكم.
- **find_java_files**: هذه الدالة تبحث عن جميع ملفات .java في الدليل المشروع بشكل متكرر، مع التعامل مع وحدات متعددة بشكل طبيعي عبر os.walk.
- **extract_paths**: هذه الدالة تفكك تعليقات @GetMapping، @PostMapping، أو @RequestMapping لاستخراج قيم المسارات. وهي تتعامل مع:
  - المسارات الفردية مثل□ @GetMapping("/path")□.
  - المسارات المتعددة مثل□ @GetMapping({"/path1", "/path2"})□.
  - المسماة المسارات مثل□ @RequestMapping(value = "/path")□.
- **المنطق الرئيسي**:
  - **معالجة السطر الأوامر**: يأخذ دليل جذر المشروع كمدخل، مشابه للكود المقدم.
  - **تحديد المحكمات**: التحقق من @Controller أو @RestController في محتوى الملف.
  - **مسارات أساسية**: استخراج مسارات @RequestMapping على مستوى الفئة قبل إعلان الفئة.
  - **برمجة واجهات التطبيقات**: معالجة السطور بعد إعلان الفئة للحصول على @GetMapping و @PostMapping، ودمج المسارات مع المسارات الأساسية.
  - **الخروج**: طباعة كل محكم مع مسارات □□□ و □□□□، كل واحدة في سطر مع إضافة □□□□ أو □□□□□□.
- **معالجة الأخطاء**: القبض على الاستثناءات أثناء قراءة الملفات وطباعتها، مما يعكس قوة الكود المقدم.

النموذج الخرج

لمشروع □□□□□□ □□□□ يحتوي على ملف UserController.java:

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

    @PostMapping("/create")
```

```java
    public ResponseEntity<?> createUser() { ... }
}
```

تنفذ python script.py /path/to/project قد تنتج:

```
[INFO]        :     /path/to/project
[INFO]          Java :      10
[INFO]          :     10
[INFO]        :     0
[INFO]            :      1
UserController:
get /users/all
post /users/create
```

الملاحظات

- □ **التحليل**: يفترض النص أن التعليقات على سطور منفصلة وأن قيم المسارات نصية بدون تعابير معقدة. لاستخدام أكثر □□□□ قوة، قد يكون المفسر
- □ **فقط □□□□ و □□□ و □□□**: كما تم طلبه، يركز على @GetMapping و @PostMapping، ويغفل عن طرق أخرى مثل □□□ أو □□□□□□ على□ الرغم من أن التفكير في مسار @RequestMapping مع □الطرق.
- □ **دمج المسارات**: يدمج المسارات على مستوى الفئة و مستوى الطريقة مباشرة مثل□ /users □ /all □ /users/all□. □□□□□□ يطبق المسارات المتعددة بشكل تلقائي، لذلك يكون هذا كافيًا لأغراض العرض.