تحليل ملفات □□□ □□□□□

للتعامل مع استفسارك، سأقدم حلًا لتوليد مخطط التبعية في صيغة □□□ عن طريق العثور على جميع ملفات pom.xml في دليل بشكل تكراري وتحليل تبعياتها، دون افتراض بنية مشروع □□□□□ متعدد الأجزاء الثابتة مثل تلك في المثال □project/module1/pom.xml، .□.إلخ سيركز النص على جمع جميع ملفات pom.xml أولاً، ثم تحليلها لتحديد تبعيات الوحدات وإخراج المخطط.

نظرة عامة على الحل

انه ما سيقوم به النص:

1. **جمع ملفات pom.xml بشكل تكراري**: استعرض الدليل الجذر المحدد للعثور على جميع ملفات pom.xml، بغض النظر عن هيكلها أو موقعها.
2. **تحليل معلومات الوحدات**: استخراج groupId و artifactId و التبعيات من كل pom.xml.
3. **معالجة وراثة groupId**: إذا لم يحدد pom.xml groupId، فاستيراده من إعلان <parent>، إذا كان موجودًا.
4. **تحديد التبعيات داخل المشروع**: تطابق التبعيات مع الوحدات داخل المشروع بناءً على groupId و artifactId.
5. **إنشاء إخراج □□□**: إنشاء مخطط موجه للتبعيات بين الوحدات.

يستخدم هذا النهج ال يعتمد على هيكل دليل مسبق أو افتراض أن ملفات pom.xml في الدلائل الأبوة.

النص البرمجي □□□□□□

انه النص البرمجي المخصص لطلبك:

```python
import os
import sys
import xml.etree.ElementTree as ET

#             Maven POM
NS = "{http://maven.apache.org/POM/4.0.0}"

#         groupId
group_id_cache = {}

def get_group_id(pom_path, pom_map):
    """
```

```
    groupId      pom.xml                    .

:
    pom_path (str):       pom.xml.
    pom_map (dict):         pom.xml          .

:
    str: groupId .
"""
if pom_path in group_id_cache:
    return group_id_cache[pom_path]

tree = ET.parse(pom_path)
root = tree.getroot()
group_id_elem = root.find(NS + 'groupId')

if group_id_elem is not None:
    group_id = group_id_elem.text.strip()
else:
    #
    parent = root.find(NS + 'parent')
    if parent is not None:
        parent_group_id = parent.find(NS + 'groupId').text.strip()
        parent_artifact_id = parent.find(NS + 'artifactId').text.strip()
        parent_relative_path = parent.find(NS + 'relativePath')
        if parent_relative_path is not None and parent_relative_path.text:
            parent_pom_path = os.path.normpath(
                os.path.join(os.path.dirname(pom_path), parent_relative_path.text)
            )
        else:
            #                          relativePath
            parent_pom_path = os.path.join(os.path.dirname(pom_path), '..', 'pom.xml')
            parent_pom_path = os.path.normpath(parent_pom_path)

        if parent_pom_path in pom_map:
            group_id = get_group_id(parent_pom_path, pom_map)
        else:
            raise ValueError(f"            POM       }pom_path}: {parent_pom_path}")
    else:
        raise ValueError(f"         groupId         }pom_path}")
```

```
    group_id_cache[pom_path] = group_id
    return group_id

def get_artifact_id(pom_path):
    """
        artifactId      pom.xml.

    :
        pom_path (str):       pom.xml.

    :
        str: artifactId .
    """
    tree = ET.parse(pom_path)
    root = tree.getroot()
    artifact_id_elem = root.find(NS + 'artifactId')

    if artifact_id_elem is None:
        raise ValueError(f"     pom.xml     artifactId: {pom_path}")

    return artifact_id_elem.text.strip()

def get_dependencies(pom_path):
    """
                    pom.xml.

    :
        pom_path (str):       pom.xml.

    :
        list:           (groupId, artifactId)    .
    """
    tree = ET.parse(pom_path)
    root = tree.getroot()
    dependencies = []

    for dep in root.findall(NS + 'dependencies/' + NS + 'dependency'):
        dep_group_id_elem = dep.find(NS + 'groupId')
        dep_artifact_id_elem = dep.find(NS + 'artifactId')
```

```
        if dep_group_id_elem is not None and dep_artifact_id_elem is not None:
            dep_group_id = dep_group_id_elem.text.strip()
            dep_artifact_id = dep_artifact_id_elem.text.strip()
            dependencies.append((dep_group_id, dep_artifact_id))

    return dependencies

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)

    #     :1               pom.xml
    pom_files = [
        os.path.join(root, file)
        for root, _, files in os.walk(root_dir)
        for file in files if file == 'pom.xml'
    ]

    if not pom_files:
        print(f"               pom.xml    }root_dir}")
        sys.exit(1)

    #     :2           POMs
    pom_map = {pom_file: None for pom_file in pom_files}

    #     :3
    modules = {}  # (groupId, artifactId) -> pom_path
    for pom_file in pom_files:
        try:
            group_id = get_group_id(pom_file, pom_map)
            artifact_id = get_artifact_id(pom_file)
            modules[(group_id, artifact_id)] = pom_file
        except ValueError as e:
```

```
        print(f"   :    }pom_file}    :    }e}")
        continue

# 	:4
dependencies = set()
for pom_file in pom_files:
    try:
        importer_group_id = get_group_id(pom_file, pom_map)
        importer_artifact_id = get_artifact_id(pom_file)
        importer_key = (importer_group_id, importer_artifact_id)
        deps = get_dependencies(pom_file)
        for dep_group_id, dep_artifact_id in deps:
            dep_key = (dep_group_id, dep_artifact_id)
            if dep_key in modules and dep_key != importer_key:
                #            (importer, imported)      artifactId
                dependencies.add((importer_artifact_id, dep_artifact_id))
    except ValueError as e:
        print(f"   :                  }pom_file}: {e}")
        continue

#     :5          DOT
print('digraph G {')
for from_module, to_module in sorted(dependencies):
    print(f'  "{from_module}" -> "{to_module}";')
print('}')
```

كيفية عمله

1. المدخل السطرية

- يأخذ معلمة واحدة: <root_directory>, نقطة البدء للبحث التكراري.
- يثبت أن هذا دليل.

2. البحث عن ملفات pom.xml

- يستخدم os.walk لاستعراض شجرة الدليل بشكل تكراري وجمع جميع ملفات pom.xml في قائمة.

3. تحليل معلومات الوحدات

- □ □□□□□ □□ □**groupId**□:
  - □ يستخرج من كل pom.xml.
  - □ إذا لم يكن موجودًا، يبحث عن قسم <parent> ويحل groupId من □□□ الأب المرجع باستخدام relativePath أو□ إغفاله□. تم إذا الأب الدليل على الافتراض
  - □ يحفظ النتائج لتجنب التحليل المتكرر.
- □ □□□□□□□□ □□ □**artifactId**□: يجب أن يكون موجودًا في كل pom.xml.
- □ **التبعيات**: يستخرج أزواج (groupId, artifactId) من القسم <dependencies>.

4. تحليل التبعيات

- □ يبني خريطة (groupId, artifactId) إلى pom_path لجميع الوحدات.
- □ لكل pom.xml، يحدد التبعيات ضد خريطة الوحدات للعثور على التبعيات داخل المشروع.
- □ يبعد التبعيات الذاتية □حيث تعتمد الوحدة على نفسها□.
- □ يسجل التبعيات كزوجات (importer_artifactId, imported_artifactId).

5. إخراج □□□

- □ يخرج مخططًا موجهًا في صيغة □□□، باستخدام artifactId كعلامات العقد للسهولة.

مثال على الاستخدام

فرض أن هيكل الدليل غير منتظم:

```
myproject/
   app/pom.xml (groupId="com.myapp", artifactId="app",        "core")
   libs/core/pom.xml (groupId="com.myapp", artifactId="core")
   tools/util/pom.xml (groupId="com.myapp", artifactId="util",        "core")
```

أشغل النص البرمجي:

```
python script.py myproject
```

الخروج:

```
digraph G {
  "app" -> "core";
  "util" -> "core";
}
```

من كل أن يظهر app و util تعتمد على core.

وقيود ملاحظات

- **الدليل حسب تجميع لا**: تطلب لم لأنه الدليل، هيكل حسب الوحدات تجميع لا التفكير، مسار في الأصلي النص عكس المخطط. في مباشرة artifactId يستخدم ذلك.
- **الأب حل**: إغفاله. تم إذا الأب الدليل في الأب [illegible] أن على الافتراض أو <parent> القسم في relativePath على يعتمد خطأ. يرفع المحلولة، الملفات مجموعة في الأب [illegible] يكن لم إذا
- **بسيطة مطابقة**: الإصدارات. تجاهل فقط، artifactId و groupId بواسطة التبعيات تطابق
- **الخصائص استبدال لا**: مثل [illegible] خصائص يعالج لا ${project.groupId}.
- **الأخطاء مع الجة**: بالكامل. الفشل من بدلاً تحذير مع فيها المشكوك pom.xml ملفات يخطي

ثابتة. ملفات بنية افتراض دون[illegible] التحليل ثم أولاً، [illegible] جميع على الحصول تكراري، بشكل التردد[illegible] طلب النص هذا يحقق تعديلات! إلى تحتاج كنت إذا أخبرني