

# 分析 Java 依賴

要回答你的查詢，我會修改提供的 Python 腳本，以生成 Java 類別的依賴圖，其中依賴按指定的「層級」的包層次結構分組。例如，如果 `level=1`，則包按頂層分組（例如「org」），如果 `level=2`，則按第二層分組（例如「org.springframework」）。這意味著在分析依賴之前，我們根據類別包名的前 `level` 部分將類別累積到組中，圖表將顯示這些組之間的依賴，而不是個別類別。

以下是修改後的腳本，後面是對更改的說明及其如何滿足你的需求。

## 修改後的腳本

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

def get_package(file_path):
    """
    從 .java 文件中提取包名。

    Args:
        file_path (str): .java 文件的路徑。

    Returns:
        str: 包名，如果未找到則為 None。
    """
    try:
        with open(file_path, 'r', encoding='utf-8') as f:
            for line in f:
                match = re.search(r'^\s*package\s+([\w.]+);', line)
                if match:
                    return match.group(1)
    except Exception as e:
        print(f"警告：無法讀取 {file_path}: {e}")
    return None

def get_specific_imports(file_path):
    """
    從 .java 文件中提取特定類別的導入，排除通配符導入。

    Args:
```

*file\_path (str): .java 文件的路徑。*

*Returns:*

*list: 完全限定導入類別名的列表。*

"""

```
imports = []
try:
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            match = re.search(r'^\s*import\s+([\w.]+);', line)
            if match:
                imp = match.group(1)
                # 排除通配符導入 (例如, import java.util.*;)
                if not imp.endswith('.*'):
                    imports.append(imp)
except Exception as e:
    print(f"警告：無法讀取 {file_path}: {e}")
return imports
```

```
def get_package_group(full_class_name, level):
```

"""

根據包名的前 *'level'* 部分獲取包組。

*Args:*

*full\_class\_name (str): 完全限定類別名 (例如 "org.springframework.boot.App")。*

*level (int): 要包含的包層次數 (例如, 1 表示 "org", 2 表示 "org.springframework")。*

*Returns:*

*str: 包組 (例如 "org" 或 "org.springframework")。*

"""

```
package = '.'.join(full_class_name.split('.')[:-1]) # 提取包, 排除類別名
parts = package.split('.')
if len(parts) <= level:
    return package # 如果包的部分數少於或等於 level, 則使用完整包
else:
    return '.'.join(parts[:level]) # 使用前 'level' 部分
```

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

# 檢查命令行參數: root\_directory 和 level

```
if len(sys.argv) != 3:
```

```

print(" 用法：python script.py <root_directory> <level>")
sys.exit(1)

root_dir = sys.argv[1]
try:
    level = int(sys.argv[2])
    if level < 1:
        raise ValueError
except ValueError:
    print(" 錯誤：level 必須是正整數")
    sys.exit(1)

all_classes = set()

# 第一次遍歷：收集項目中的所有完全限定類別名
for root, dirs, files in os.walk(root_dir):
    for file in files:
        if file.endswith('.java'):
            file_path = os.path.join(root, file)
            package = get_package(file_path)
            if package:
                class_name = file.replace('.java', '')
                full_class_name = f"{package}.{class_name}"
                all_classes.add(full_class_name)

# 存儲包組之間的依賴
group_dependencies = set()

# 第二次遍歷：根據包組分析依賴
for root, dirs, files in os.walk(root_dir):
    for file in files:
        if file.endswith('.java'):
            file_path = os.path.join(root, file)
            package = get_package(file_path)
            if package:
                class_name = file.replace('.java', '')
                full_class_name = f"{package}.{class_name}"
                importer_group = get_package_group(full_class_name, level)
                imports = get_specific_imports(file_path)
                for imp in imports:

```

```

# 只包括項目內的依賴
# 排除自依賴
if imp in all_classes and imp != full_class_name:
    imported_group = get_package_group(imp, level)
    if imported_group != importer_group:
        group_dependencies.add((importer_group, imported_group))

# 以 DOT 格式輸出依賴圖
print('digraph G {')
for from_group, to_group in sorted(group_dependencies):
    print(f'    "{from_group}" -> "{to_group}";')
print('}')

```

## 更改說明

### 1. 命令行參數：

- **原始：**python script.py <root\_directory>
- **修改後：**python script.py <root\_directory> <level>
- 添加對第二個參數 level 的支持，指定包層次結構層級。腳本檢查是否提供了兩個參數，並檢查 level 是否為正整數。

### 2. 新函數：get\_package\_group：

- 添加一個函數，根據指定的 level 計算類別的包組。
- 例如，對於完全限定類別名(例如“org.springframework.boot.App”)，它提取包(“org.springframework.boot”)，將其分割成部分(“org”、“springframework”、“boot”)，並取前 level 部分：
  - 如果 level=1：返回 “org”。
  - 如果 level=2：返回 “org.springframework”。
  - 如果包的部分數少於 level (例如 “com.example” 且 level=3)，則返回完整包 (“com.example”)。

### 3. 依賴分組：

- **原始：**使用 defaultdict(set) 存儲個別類別之間的依賴。
- **修改後：**使用 set(group\_dependencies) 存儲包組之間的有向邊作為元組 (from\_group, to\_group)。
- 對於每個類別：
  - 使用 get\_package\_group 計算其包組 (importer\_group)。
  - 對於每個特定導入 (在項目內的 imp in all\_classes 且不是類別本身 imp != full\_class\_name)：
    - \* 計算導入類別的包組 (imported\_group)。
    - \* 如果組不同 (imported\_group != importer\_group)，則將邊添加到 group\_dependencies。
- set 確保唯一性，因此相同組之間的多個依賴結果為單個邊。

### 4. DOT 輸出：

- **原始**：打印個別類別之間的邊(例如“org.springframework.boot.App”->“org.apache.commons.IOUtils”)。
- **修改後**：打印包組之間的邊（例如 “org.springframework” -> “org.apache” 當 level=2）。
- 邊按字母順序排序以保持一致的輸出。

## 滿足你的需求

- **支持層級**：腳本現在接受 level 參數，以在分析依賴之前按包層次結構分組。
- **層級 = 1**：按頂層包分組所有類別（例如 “org”）。例如，“org.springframework.boot.App”和“org.apache.commons.IOUtils”都屬於“org”組，因此“org”內的導入之間不顯示為邊。
- **層級 = 2**：按前兩個包層次分組類別(例如“org.springframework”)。例如，從“org.springframework.boot.App”到“org.apache.commons.IOUtils”的導入創建一個從“org.springframework”到“org.apache”的邊。
- **在依賴分析之前累積包**：腳本根據 level 確定每個類別的包組，然後分析其導入，確保依賴在組之間，而不是個別類別。
- **示例遵從**：對於你的示例邊：
  - 原始：“org.springframework.boot.web.servlet.server.Session” -> “org.springframework.boot.convert.Duration”
  - 當 level=2：兩個類別都在 “org.springframework”，因此不添加邊（同一組）。
  - 當 level=3：“org.springframework.boot.web” -> “org.springframework.boot.convert”，添加這些不同組之間的邊。

## 使用示例

- **命令**：python script.py /path/to/project 1
  - 組：“org”、“com”等。
  - 輸出可能包括：“org” -> “com”如果“org”包中的類別導入“com”包中的類別。
- **命令**：python script.py /path/to/project 2
  - 組：“org.springframework”、“org.apache”等。
  - 輸出可能包括：“org.springframework” -> “org.apache”。

## 注意事項

- 腳本保留了僅包括項目內依賴（all\_classes 中的導入）並排除通配符導入和自依賴的原始行為。
- 沒有包聲明的類別被跳過，與原始腳本一致。
- 輸出是 DOT 格式的有向圖，可以使用 Graphviz 等工具查看。

這個修改後的腳本應該完全滿足你按包層次結構層級分組依賴的請求。