Java 依赖分析

要解决你的问题，我将修改提供的 Python 脚本，以生成 Java 类的依赖图，其中依赖项按指定的“级别”分组。例如，如果 `level=1`，则包按顶级（例如，“org”）分组，如果 `level=2`，则按第二级（例如，“org.springframework”）分组。这意味着在分析依赖项之前，我们根据类的包名的前 `level` 部分将类累积到组中，图表将显示这些组之间的依赖关系，而不是单个类。

以下是修改后的脚本，后面是对更改的解释以及它如何满足你的需求。

修改后的脚本

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

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

    参数:
        file_path (str): .java 文件的路径。

    返回:
        str: 包名，如果未找到则为 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：无法读取 {file_path}: {e}")
    return None

def get_specific_imports(file_path):
    """
    从 .java 文件中提取特定的类导入，排除通配符导入。

    参数:
```

```
        file_path (str): .java 文件的路径。

    返回:
        list: 完全限定的导入类名列表。
    """
    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)
                    # 排除通配符导入（例如，import java.util.*;）
                    if not imp.endswith('.*'):
                        imports.append(imp)
    except Exception as e:
        print(f" 警告：无法读取 {file_path}: {e}")
    return imports

def get_package_group(full_class_name, level):
    """
    根据包名的前 'level' 部分获取包组。

    参数:
        full_class_name (str): 完全限定的类名（例如，“org.springframework.boot.App”）。
        level (int): 要包含的包级别数（例如，1 表示 “org”，2 表示 “org.springframework”）。

    返回:
        str: 包组（例如，“org” 或 “org.springframework”）。
    """
    package = '.'.join(full_class_name.split('.')[:-1])  # 提取包，排除类名
    parts = package.split('.')
    if len(parts) <= level:
        return package  # 如果包的部分数少于或等于 level，则使用完整包
    else:
        return '.'.join(parts[:level])  # 使用前 'level' 部分

if __name__ == '__main__':
    # 检查命令行参数：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(" 错误: level 必须是正整数")
    sys.exit(1)

all_classes = set()

# 第一次遍历：收集项目中的所有完全限定的类名
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)

# 存储包组之间的依赖关系
group_dependencies = set()

# 第二次遍历：根据包组分析依赖关系
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:
```

```
                # 仅包括项目内的依赖关系
                # 排除自依赖
                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 格式输出依赖图
print('digraph G {')
for from_group, to_group in sorted(group_dependencies):
    print(f'  "{from_group}" -> "{to_group}";')
print('}')
```

更改说明

1. **命令行参数**：
   - **原始**：`python script.py <root_directory>`
   - **修改后**：`python script.py <root_directory> <level>`
   - 添加对第二个参数 `level` 的支持，指定包层次级别。脚本检查是否提供了两个参数，并且 `level` 是正整数。
2. **新函数：`get_package_group`**：
   - 添加一个函数，根据指定的 `level` 计算类的包组。
   - 对于完全限定的类名（例如，“org.springframework.boot.App”），它提取包（“org.springframework.boot”），将其分成部分（“org”，“springframework”，“boot”），并取前 `level` 部分：
     - 如果 `level=1`：返回“org”。
     - 如果 `level=2`：返回“org.springframework”。
     - 如果包的部分数少于 `level`（例如，“com.example”与 `level=3`），则返回完整包（“com.example”）。
3. **依赖分组**：
   - **原始**：使用 `defaultdict(set)` 存储单个类之间的依赖关系。
   - **修改后**：使用 `set`（`group_dependencies`）存储包组之间的有向边作为元组 `(from_group, to_group)`。
   - 对于每个类：
     - 使用 `get_package_group` 计算其包组（`importer_group`）。
     - 对于每个特定导入(在项目内的导入，`imp in all_classes`)且不是类本身(`imp != full_class_name`)：
       - 计算导入类的包组（`imported_group`）。
       - 如果组不同（`imported_group != importer_group`），则将边添加到 `group_dependencies`。
   - `set` 确保唯一性，因此相同组之间的多个依赖关系将导致单个边。
4. **DOT 输出**：

- **原始**：打印单个类之间的边（例如，“org.springframework.boot.App” -> “org.apache.commons.IOUtils”）。
- **修改后**：打印包组之间的边（例如，“org.springframework” -> “org.apache” 当 `level=2`）。
- 边按字母顺序排序以确保输出一致。

如何满足你的需求

- **支持级别**：脚本现在接受 `level` 参数，以在分析依赖关系之前按包层次级别分组。
- **级别 = 1**：按顶级包（例如，“org”）分组所有类。例如，“org.springframework.boot.App” 和“org.apache.commons.IOUtils”都属于“org”组，因此“org”组内的导入不会显示为边。
- **级别 = 2**：按前两个包级别（例如，“org.springframework”）分组类。例如，从“org.springframework.boot.App”到“org.apache.commons.IOUtils”的导入将创建从“org.springframework”到“org.apache”的边。
- **在依赖分析之前累积包**：脚本根据 `level` 确定每个类的包组，然后分析其导入，确保依赖关系在组之间，而不是单个类之间。
- **示例符合性**：对于你的示例边：
  - 原始：`"org.springframework.boot.web.servlet.server.Session" -> "org.springframework.boot.convert.Dura`
  - 当 `level=2` 时：两个类都在“org.springframework”组中，因此不会添加边（同一组）。
  - 当 `level=3` 时：`"org.springframework.boot.web" -> "org.springframework.boot.convert"`，添加这两个不同组之间的边。

使用示例

- **命令**：`python script.py /path/to/project 1`
  - 组：`"org"`, `"com"` 等。
  - 输出可能包括：`"org" -> "com"` 如果“org”包中的类导入“com”包中的类。
- **命令**：`python script.py /path/to/project 2`
  - 组：`"org.springframework"`, `"org.apache"` 等。
  - 输出可能包括：`"org.springframework" -> "org.apache"` 如果“org.springframework”包中的类导入“org.apache”包中的类。

注意事项

- 脚本保留了仅包括项目内依赖关系（在 `all_classes` 内的导入）并排除通配符导入和自依赖的原始行为。
- 没有包声明的类被跳过，与原始脚本一致。
- 输出是 DOT 格式的有向图，可以使用 Graphviz 等工具查看。

这个修改后的脚本应该完全满足你按包层次级别分组依赖关系的请求。