Java 依存関係の分析

あなたの質問に応えるために、提供された Python スクリプトを修正して、Java クラスの依存関係グラフを生成するようにしました。依存関係は、指定された「レベル」のパッケージ階層でグループ化されます。例えば、`level=1` の場合、パッケージは最上位レベル（例：“org”）でグループ化され、`level=2` の場合は、2 番目のレベル（例：“org.springframework”）でグループ化されます。これは、依存関係を分析する前に、クラスをパッケージ名の最初の `level` 部分に基づいてグループにまとめることを意味し、グラフはこれらのグループ間の依存関係を示すことになります。

以下に、修正されたスクリプトとその変更点、およびあなたの要件を満たす方法についての説明を示します。

修正されたスクリプト

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

def get_package(file_path):
    """
    .java ファイルからパッケージ名を抽出します。

    引数:
        file_path (str): .java ファイルへのパス。

    戻り値:
        str: パッケージ名、または見つからない場合は 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: {file_path}を読み取れませんでした: {e}")
    return None

def get_specific_imports(file_path):
    """
    .java ファイルから特定のクラスのインポートを抽出します。ワイルドカードインポートは除外します。
```

```
    引数:
        file_path (str): .java ファイルへのパス。

    戻り値:
        list: 完全修飾されたインポートされたクラス名のリスト。
    """
    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)
                    # ワイルドカードインポート（例：import java.util.*;）を除外
                    if not imp.endswith('.*'):
                        imports.append(imp)
    except Exception as e:
        print(f" 警告: {file_path}を読み取れませんでした: {e}")
    return imports

def get_package_group(full_class_name, level):
    """
    パッケージ名の最初の'level'部分に基づいてパッケージグループを取得します。

    引数:
        full_class_name (str): 完全修飾されたクラス名（例："org.springframework.boot.App"）。
        level (int): 含めるパッケージレベルの数（例：1 は"org"、2 は"org.springframework"）。

    戻り値:
        str: パッケージグループ（例："org" または"org.springframework"）。
    """
    package = '.'.join(full_class_name.split('.')[:-1])  # クラス名を除くパッケージを抽出
    parts = package.split('.')
    if len(parts) <= level:
        return package  # パッケージが level より少ないまたは等しい部分を持つ場合は、全パッケージを使用
    else:
        return '.'.join(parts[:level])  # 最初の'level'部分を使用

if __name__ == '__main__':
```

```
# コマンドライン引数の確認：root_directory と level
if len(sys.argv) != 3:
    print(" 使用法: 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(" エラー: level は正の整数でなければなりません")
    sys.exit(1)

all_classes = set()

# 第一パス：プロジェクト内のすべての完全修飾クラス名を収集
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)

# パッケージグループ間の依存関係を保存
group_dependencies = set()

# 第二パス：パッケージグループに基づいて依存関係を分析
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:
                    # プロジェクト内のクラスに対する依存関係のみを含める
                    # 自依存関係を除外
                    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))

# DOT 形式で依存関係グラフを出力
print('digraph G {')
for from_group, to_group in sorted(group_dependencies):
    print(f'  "{from_group}" -> "{to_group}";')
print('}')
```

変更点の説明

1. **コマンドライン引数**:
   - **元**: `python script.py <root_directory>`
   - **修正**: `python script.py <root_directory> <level>`
   - `level` という 2 つ目の引数をサポートするようにしました。この引数はパッケージ階層レベルを指定します。スクリプトは正確に 2 つの引数が提供されていることを確認し、`level` が正の整数であることを確認します。
2. **新しい関数:** `get_package_group`:
   - 指定された `level` に基づいてクラスのパッケージグループを計算するための関数を追加しました。
   - 完全修飾されたクラス名（例：“org.springframework.boot.App”）を受け取り、パッケージ（“org.springframework.boot”）を抽出し、部分（“org”, “springframework”, “boot”）に分割し、最初の `level` 部分を取得します：
     - `level=1` の場合: “org”を返します。
     - `level=2` の場合: “org.springframework”を返します。
     - パッケージが `level` より少ない部分を持つ場合（例：“com.example”で `level=3`）、全パッケージ（“com.example”）を返します。
3. **依存関係のグループ化**:
   - **元**: `defaultdict(set)` を使用して個々のクラス間の依存関係を保存しました。
   - **修正**: `set`（`group_dependencies`）を使用して、パッケージグループ間の有向エッジをタプル（`(from_group, to_group)`）として保存します。
   - 各クラスについて:
     - `get_package_group` を使用してそのパッケージグループ（`importer_group`）を計算します。

- プロジェクト内の特定のインポート（`imp in all_classes`）であり、クラス自身でない（`imp != full_class_name`）場合:
  * インポートされたクラスのパッケージグループ（`imported_group`）を計算します。
  * グループが異なる場合（`imported_group != importer_group`）、`group_dependencies` にエッジを追加します。
- `set` は一意性を保証するため、同じグループ間の複数の依存関係は 1 つのエッジにまとめられます。

4. **DOT 出力**:
   - **元**: 個々のクラス間のエッジ（例：“org.springframework.boot.App”-> “org.apache.commons.IOUtils”）を出力しました。
   - **修正**: パッケージグループ間のエッジ（例：“org.springframework”-> “org.apache”で `level=2`）を出力します。
   - エッジは一貫した出力のためにソートされます。

要件を満たす方法

- **レベルのサポート**: スクリプトは現在、`level` パラメータを受け入れて、依存関係を分析する前にパッケージをグループ化するようになりました。
- **レベル = 1**: クラスを最上位レベルのパッケージ（例：“org”）でグループ化します。例えば、“org.springframework.boot.App”と”org.apache.commons.IOUtils”はともに”org”グループに属するため、“org”内のインポート間のエッジは表示されません。
- **レベル = 2**: クラスを最初の 2 つのパッケージレベル（例：“org.springframework”）でグループ化します。例えば、“org.springframework.boot.App”から”org.apache.commons.IOUtils”へのインポートは、“org.springframework”から”org.apache”へのエッジを作成します。
- **依存関係分析前にパッケージを集計**: スクリプトは `level` に基づいて各クラスのパッケージグループを決定し、インポートを分析する前に依存関係をグループ間にすることを保証します。
- **例の準拠**: あなたの例のエッジについて:
  - 元: `"org.springframework.boot.web.servlet.server.Session" -> "org.springframework.boot.convert.Duratio`
  - `level=2` の場合: 両方のクラスは”org.springframework”に属するため、エッジは追加されません（同じグループ）。
  - `level=3` の場合: “org.springframework.boot.web”-> “org.springframework.boot.convert”、これらの異なるグループ間にエッジを追加します。

使用例

- **コマンド**: `python script.py /path/to/project 1`
  - グループ: “org”, “com”など。
  - 出力には、“org”パッケージのクラスが”com”パッケージのクラスをインポートする場合、“org”-> “com”が含まれるかもしれません。

- **コマンド**: `python script.py /path/to/project 2`
  - グループ: “org.springframework”, “org.apache”など。
  - 出力には、“org.springframework”パッケージのクラスが”org.apache”パッケージのクラスをインポートする場合、“org.springframework”-> “org.apache”が含まれるかもしれません。

注意点

- スクリプトは、プロジェクト内の依存関係（`all_classes` 内のインポート）のみを含め、ワイルドカードインポートと自依存関係を除外する、元の動作を維持します。
- パッケージ宣言がないクラスはスキップされ、元のスクリプトと一貫しています。
- 出力は、Graphviz などのツールで表示できる有向グラフの DOT 形式です。

この修正されたスクリプトは、パッケージ階層レベルで依存関係をグループ化することで、あなたの要件を完全に満たすはずです。