分析 Maven POM 以生成依賴圖

要解決您的查詢，我將提供一個生成依賴圖的解決方案，該圖以 DOT 格式生成，並通過遞歸查找目錄中的所有 `pom.xml` 文件，分析其依賴項，而不假設固定的 Maven 多模塊項目結構（例如 `project/module1/pom.xml` 等）。該腳本將專注於首先收集所有 `pom.xml` 文件，然後分析它們以確定模塊依賴項並輸出圖形。

解決方案概述

以下是腳本將執行的操作：1. **遞歸收集 `pom.xml` 文件**：遍歷指定的根目錄以查找所有 `pom.xml` 文件，而不考慮其位置或結構。2. **解析模塊信息**：從每個 `pom.xml` 中提取 `groupId`、`artifactId` 和依賴項。3. **處理 `groupId` 繼承**：如果 `pom.xml` 沒有指定 `groupId`，則從 `<parent>` 宣告中繼承它（如果存在）。4. **識別項目內部依賴項**：根據 `groupId` 和 `artifactId` 將依賴項與項目中的模塊匹配。5. **生成 DOT 輸出**：創建模塊之間依賴關係的有向圖。

這種方法不依賴於預定義的目錄層次結構，也不假設父 `pom.xml` 文件在父目錄中。

Python 腳本

以下是針對您需求量身定做的 Python 腳本：

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

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

# 用於避免冗餘解析的 groupId 緩存
group_id_cache = {}

def get_group_id(pom_path, pom_map):
    """
    從 pom.xml 文件中提取 groupId，考慮從父項繼承。

    Args:
        pom_path (str): pom.xml 文件的路徑。
        pom_map (dict): pom.xml 路徑到其解析數據的映射。
```

```
    Returns:
        str: 模塊的 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:
        # 檢查父宣告
        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:
                # 如果省略相對路徑，則默認為父目錄
                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" 未找到 {pom_path} 的父 POM: {parent_pom_path}")
        else:
            raise ValueError(f"{pom_path} 未指定 groupId 或父項")

    group_id_cache[pom_path] = group_id
    return group_id

def get_artifact_id(pom_path):
    """
```

```
    從 pom.xml 文件中提取 artifactId。

    Args:
        pom_path (str): pom.xml 文件的路徑。

    Returns:
        str: 模塊的 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 必須指定 artifactId: {pom_path}")

    return artifact_id_elem.text.strip()

def get_dependencies(pom_path):
    """
    從 pom.xml 文件中提取依賴項列表。

    Args:
        pom_path (str): pom.xml 文件的路徑。

    Returns:
        list: 每個依賴項的 (groupId, artifactId) 元組列表。
    """
    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__':
    # 檢查命令行參數
    if len(sys.argv) != 2:
        print(" 用法: python script.py <root_directory>")
        sys.exit(1)

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

    # 第 1 步：遞歸查找所有 pom.xml 文件
    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" 在 {root_dir} 中未找到 pom.xml 文件")
        sys.exit(1)

    # 第 2 步：構建所有 POM 的字典以進行父查找
    pom_map = {pom_file: None for pom_file in pom_files}

    # 第 3 步：提取模塊信息
    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" 警告: 由於錯誤 {e} 而跳過 {pom_file}")
            continue

    # 第 4 步：分析依賴項
    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:
                    # 使用 artifactId 簡化，將依賴項添加為 (importer, imported) 元組
                    dependencies.add((importer_artifact_id, dep_artifact_id))
        except ValueError as e:
            print(f" 警告：處理 {pom_file} 的依賴項時出錯：{e}")
            continue

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

運作方式

1. 命令行輸入

- 接受一個參數：`<root_directory>`，遞歸搜索的起點。
- 驗證它是否為目錄。

2. 查找 `pom.xml` 文件

- 使用 `os.walk` 遞歸遍歷目錄樹並將所有 `pom.xml` 文件收集到列表中。

3. 解析模塊信息

- **組 ID (`groupId`)**：
  - 從每個 `pom.xml` 中提取。
  - 如果不存在，則查找 `<parent>` 部分並從引用的父 POM 解析 `groupId`，使用 `relativePath`（如果省略則默認為父目錄）。
  - 緩存結果以避免重新解析。

- **工件 ID (`artifactId`)**：每個 `pom.xml` 必須存在。
- **依賴項**：從 `<dependencies>` 部分提取 `(groupId, artifactId)` 對。

4. 依賴項分析

- 為所有模塊構建 `(groupId, artifactId)` 到 `pom_path` 的映射。
- 為每個 `pom.xml`，檢查其依賴項以查找項目內部依賴項。
- 排除自依賴（模塊依賴於自身）。
- 將依賴項記錄為 `(importer_artifactId, imported_artifactId)` 對。

5. DOT 輸出

- 以 DOT 格式輸出有向圖，使用 `artifactId` 作為節點標籤以簡化。

示例用法

假設您的目錄結構不規則：

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

運行腳本：

```
python script.py myproject
```

輸出：

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

這顯示 `app` 和 `util` 都依賴於 `core`。

注意事項和限制

- **不按目錄分組**：與原始腳本不同，這個版本不按目錄結構分組模塊，因為您沒有要求它。它直接在圖中使用 `artifactId`。
- **父解析**：依賴於 `<parent>` 部分中的 `relativePath`，或者如果省略則假設父 POM 在父目錄中。如果父 POM 不在收集的集合中，則引發錯誤。
- **簡單匹配**：僅根據 `groupId` 和 `artifactId` 匹配依賴項，忽略版本。
- **無屬性替換**：不處理 Maven 屬性，例如 `${project.groupId}`。
- **錯誤處理**：跳過有問題的 `pom.xml` 文件並發出警告，而不是完全失敗。

這個腳本滿足您的要求，即“遞歸迭代，首先獲取所有 POM，然後分析”，而不假設固定的文件結構。如果需要調整，請告訴我！