分析 Java 包

根据你的查询，你需要一个脚本，如果 `level` 参数为 0 或未提供，则使用整个包名（所有部分），实际上将 `level = 0` 视为“无限”或“使用所有部分”，而不是限制为特定数量的部分（例如，当 `level = 1` 时，只使用第一部分）。下面，我将解释如何在分析 Java 文件以提取包名的 Python 脚本中实现这一点，并提供一个完全符合你要求的解决方案。

理解需求

- **当 `level = 0` 或未提供时**：使用从 Java `import` 语句中提取的完整包名(例如 `java.util` 或 `com.google.common.eventbus`)而不截断它。
- **当 `level > 0` 时**：将包名截断为前 `level` 部分（例如，对于 `com.google.common.eventbus`，`level = 2` 将得到 `com.google`）。
- 脚本仍应处理 Java 文件，从 `import` 语句中提取包名，并适当处理边缘情况。

解决方案方法

1. **参数解析**：
   - 如果只提供根目录（例如 `python script.py /path/to/dir`），则设置 `level = 0`，这意味着使用完整的包名。
   - 如果提供了 `level`（例如 `python script.py /path/to/dir 2`），则使用它来截断包名，确保它是非负整数。
   - 如果参数无效，则退出并报错。
2. **包提取**：
   - 从 `import` 语句中提取包名（例如，`import java.util.ArrayList;` 产生 `java.util`）。
   - 使用 Java 命名约定：包名通常是小写的，类名以大写字母开头。
3. **包截断逻辑**：
   - 如果 `level = 0`，将完整的包名添加到结果集中。
   - 如果 `level > 0`，按点（`.`）分割包名并取前 `level` 部分。
4. **日志和输出**：
   - 明确指出是否使用完整包名或截断后的包名。
   - 列出所有找到的唯一包。

实现

以下是完全符合你要求的 Python 脚本：

```
import os
import sys
from collections import Counter

def find_java_files(root_dir):
    """
    递归查找给定目录及其子目录中的所有 .java 文件。

    参数:
        root_dir (str): 开始搜索的根目录。

    返回:
        str: 每个 .java 文件的完整路径。
    """
    for dirpath, dirnames, filenames in os.walk(root_dir):
        for filename in filenames:
            if filename.endswith('.java'):
                yield os.path.join(dirpath, filename)

def extract_package(import_statement):
    """
    从导入语句中提取包名。

    使用约定：包名通常是小写的，类名以大写字母开头。处理通配符导入 (*)。

    参数:
        import_statement (str): Java 文件中的导入语句行。

    返回:
        str: 包名，如果未确定则为空字符串。
    """
    parts = import_statement.split()
    if parts[0] == 'import':
        parts = parts[1:]
    if parts[0] == 'static':
        parts = parts[1:]
    import_path = ' '.join(parts).strip(';').strip()
    identifiers = import_path.split('.')
    for i, ident in enumerate(identifiers):
        if ident == '*' or (ident and ident[0].isupper()):
```

```
            package_parts = identifiers[:i]
            break
    else:
        package_parts = []
    package = '.'.join(package_parts)
    return package

if __name__ == '__main__':
    # 解析命令行参数
    if len(sys.argv) < 2 or len(sys.argv) > 4:
        print(" 用法: python script.py <root_directory> [level] [--count]")
        sys.exit(1)

    root_dir = sys.argv[1]
    level = 0
    count = False

    if len(sys.argv) == 3:
        if sys.argv[2] == "--count":
            count = True
        elif sys.argv[2].isdigit():
            level = int(sys.argv[2])
        else:
            print(f" 无效参数: {sys.argv[2]}")
            sys.exit(1)
    elif len(sys.argv) == 4:
        if sys.argv[3] == "--count" and sys.argv[2].isdigit():
            level = int(sys.argv[2])
            count = True
        else:
            print(f" 无效参数: {sys.argv[2]} {sys.argv[3]}")
            sys.exit(1)

    # 验证目录是否存在
    if not os.path.isdir(root_dir):
        print(f"[ERROR] 指定的路径不是目录: {root_dir}")
        sys.exit(1)

    # 日志分析开始
    level_str = " 使用完整包名" if level == 0 else f" 在级别 {level}"
```

```
count_str = " 带出现次数" if count else ""
print(f"[INFO] 开始分析目录: {root_dir} {level_str} {count_str}")

# 初始化变量
package_counter = Counter()
total_files = 0
error_files = 0

# 处理 Java 文件
for java_file in find_java_files(root_dir):
    try:
        with open(java_file, 'r', encoding='utf-8') as f:
            file_packages = set()
            for line in f:
                line = line.strip()
                if line.startswith('import'):
                    package = extract_package(line)
                    if package:
                        if level > 0:
                            parts = package.split('.')
                            truncated_package = '.'.join(parts[:level])
                        else:
                            truncated_package = package
                        file_packages.add(truncated_package)
        for pkg in file_packages:
            package_counter[pkg] += 1
        total_files += 1
    except Exception as e:
        print(f"[ERROR] 无法读取文件 {java_file}: {e}")
        error_files += 1
        continue

# 打印摘要
print(f"[INFO] 尝试的总 Java 文件: {total_files + error_files}")
print(f"[INFO] 成功处理: {total_files}")
print(f"[INFO] 文件出错: {error_files}")
if count:
    print(f"[INFO] 带出现次数的唯一包总数: {len(package_counter)}")
else:
    print(f"[INFO] 唯一包总数: {len(package_counter)}")
```

```
    # 打印结果并排序
    if package_counter:
        if count:
            print("[INFO] 分析完成。打印带出现次数的唯一包（按出现次数降序排序）：")
            # 按出现次数降序，然后按包名升序排序
            for pkg, cnt in sorted(package_counter.items(), key=lambda x: (-x[1], x[0])):
                print(f"{pkg}: {cnt}")
        else:
            print("[INFO] 分析完成。打印唯一包（按包名升序排序）：")
            # 按包名升序排序
            for pkg in sorted(package_counter):
                print(pkg)
    else:
        print("[INFO] 未找到包。")
```

运行方式

- `python script.py /path/to/java/project`：分析目录中的所有 `.java` 文件，并使用完整包名（`level = 0`）。
- `python script.py /path/to/java/project 2`：将包名截断为前 2 部分（例如，`com.google.common.eventbus` 变为 `com.google`）。

示例输出

假设你有一个包含以下内容的 Java 文件：

```
import java.util.ArrayList;
import com.google.common.eventbus.EventBus;
```

- **当 `level = 0`（或未提供级别）时：**

```
[INFO] 开始分析目录：/path/to/java/project 使用完整包名。
[INFO] 进入目录：/path/to/java/project
[INFO] 处理文件：/path/to/java/project/MyFile.java
[INFO] 尝试的总 Java 文件：1
[INFO] 成功处理：1
[INFO] 文件出错：0
[INFO] 唯一包总数：2
[INFO] 分析完成。打印唯一包：
com.google.common.eventbus
java.util
```

- **当 level = 2 时**：

```
[INFO] 开始分析目录：/path/to/java/project 级别：2
[INFO] 进入目录：/path/to/java/project
[INFO] 处理文件：/path/to/java/project/MyFile.java
[INFO] 尝试的总 Java 文件：1
[INFO] 成功处理：1
[INFO] 文件出错：0
[INFO] 级别 2 的唯一包总数：2
[INFO] 分析完成。打印级别 2 的唯一包：
com.google
java.util
```

- **关键特性**：
  - **完整包名**：当 level = 0 或未传递时，使用从中提取的完整包名（例如 java.util，com.google.common.eventbus）。
  - **截断**：当 level > 0 时，取前 level 部分。
  - **错误处理**：检查有效目录和非负 level。
  - **唯一性**：将包存储在 set 中以避免重复。

处理的边缘情况

- **缺少级别**：默认为 level = 0，使用完整包名。
- **无效级别**：如果 level 为负或不是整数，则退出并报错。
- **短包**：如果包的部分数少于 level（例如，java 与 level = 2），则使用完整包（java）。
- **空导入**：跳过无法提取包的行。