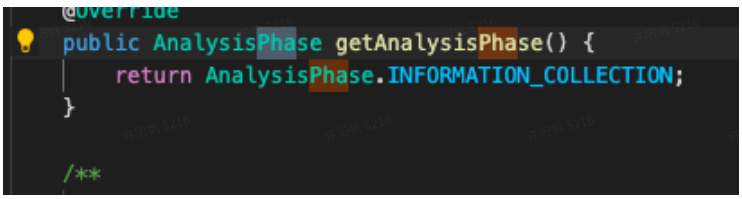
# DependencyCheck

## 原理概览

主入口 app.java app() -> runscan() -> engine.analyzeDependencies() -> writeReports()，其中包含了一系列初始化操作，比如数据库检查，命令行操作，扫描目录等。

依赖分析按顺序分为13个phase，包括了初始化，信息收集，标识符收集，标识符分析，获取结果等内容，具体可以查看 `AnalysisPhase.java` 这个文件。根据map的数据结构以及一些其他信息，应该是不同的phase调用不同的分析器，如pythonPackage的分析器就是信息收集的phase。




```
for (AnalysisPhase phase : mode.getPhases()) {
    final List<Analyzer> analyzerList = analyzers.get(phase);

    for (final Analyzer analyzer : analyzerList) {
        final long analyzerStart = System.currentTimeMillis();
        try {
            initializeAnalyzer(analyzer);
        } catch (InitializationException ex) {
            exceptions.add(ex);
            if (ex.isFatal()) {
                continue;
            }
        }

        if (analyzer.isEnabled()) {
            executeAnalysisTasks(analyzer, exceptions);

            final long analyzerDurationMillis = System.currentTimeMillis() - analyzerStart;
            final long analyzerDurationSeconds = TimeUnit.MILLISECONDS.toSeconds(analyzerDurationMillis);
            LOGGER.info("Finished {} ({} seconds)", analyzer.getName(), analyzerDurationSeconds);
        } else {
            LOGGER.debug("Skipping {} (not enabled)", analyzer.getName());
        }
    }
}
```

分析的时候是一个双层循环，按照每个phase，寻找这个phase包含的analyzer，然后执行对应的task，最后关闭这些分析器，分析结束。

## 获取dependency

这里的引用比较复杂，但整体就是一个包含dependency的list，然后在scan的过程中收集相应的文件，如果扩展名符合要求，就将其添加入这个list

## 执行task

通过JDK的ExecutorService.invokeAll异步进行，简单来说就是为其分配一个线程池和相关api来完成一组任务

这个AnalysisTask也是一个实现了callable接口的公共类，Callable是**返回线程执行的结果并且可能抛出异常的线程类，包含参数和执行函数如下：**

```
    public AnalysisTask(Analyzer analyzer, Dependency dependency, Engine engine, List<Throwal
        this.analyzer = analyzer;
        this.dependency = dependency;
        this.engine = engine;
        this.exceptions = exceptions;
     }
    public Void call() {
        if (shouldAnalyze()) {
            LOGGER.debug("Begin Analysis of '{}' ({})", dependency.getActualFilePath(), anal
            try {
                analyzer.analyze(dependency, engine);
            } catch (AnalysisException ex) {
                LOGGER.warn("An error occurred while analyzing '{}' ({}).", dependency.getAc
                LOGGER.debug("", ex);
                exceptions.add(ex);
            } catch (Throwable ex) {
                LOGGER.warn("An unexpected error occurred during analysis of '{}' ({}): {}",
                        dependency.getActualFilePath(), analyzer.getName(), ex.getMessage())
                LOGGER.error("", ex);
                exceptions.add(ex);
            }
        }
        return null;
     }
```

# Analyzer

各类分析器的接口 core/src/main/java/org/owasp/dependencycheck/analyzer/Analyzer.java

主体是内部的正则表达式和analyze函数，在分析的过程中会给dependency实体添加信息即evidence

```
    private void analyzeFileContents(Dependency dependency, File file)
            throws AnalysisException {
        final String contents;
        try {
            contents = FileUtils.readFileToString(file, Charset.defaultCharset()).trim();
        } catch (IOException e) {
            throw new AnalysisException("Problem occurred while reading dependency file.", e);
        }
        if (!contents.isEmpty()) {
            final String source = file.getName();
            gatherEvidence(dependency, EvidenceType.VERSION, VERSION_PATTERN, contents,
                    source, "SourceVersion", Confidence.MEDIUM);
            addSummaryInfo(dependency, SUMMARY_PATTERN, 4, contents,
                    source, "summary");
            if (INIT_PY_FILTER.accept(file)) {
                addSummaryInfo(dependency, MODULE_DOCSTRING, 2,
                        contents, source, "docstring");
            }
            gatherEvidence(dependency, EvidenceType.PRODUCT, TITLE_PATTERN, contents,
                    source, "SourceTitle", Confidence.LOW);

            gatherEvidence(dependency, EvidenceType.VENDOR, AUTHOR_PATTERN, contents,
                    source, "SourceAuthor", Confidence.MEDIUM);
            gatherHomePageEvidence(dependency, EvidenceType.VENDOR, URI_PATTERN,
                    source, "URL", contents);
            gatherHomePageEvidence(dependency, EvidenceType.VENDOR, HOMEPAGE_PATTERN,
                    source, "HomePage", contents);
```

如对于pythonPackage，其核心逻辑即检查 `__init__.py` ,根据其python官方网站给出的标准。

```
protected void analyzeDependency(Dependency dependency, Engine engine)
        throws AnalysisException {
    dependency.setEcosystem(DEPENDENCY_ECOSYSTEM);
    final File file = dependency.getActualFile();
    final File parent = file.getParentFile();
    final String parentName = parent.getName();
    if (INIT_PY_FILTER.accept(file)) {
        //by definition, the containing folder of __init__.py is considered the package, even the file is empty
        //"The __init__.py files are required to make Python treat the directories as containing packages"
        //see section "6.4 Packages" from https://docs.python.org/2/tutorial/modules.html;
        dependency.addEvidence(EvidenceType.PRODUCT, file.getName(), "PackageName", parentName, Confidence.HIGH
        dependency.setName(parentName);

        final File[] fileList = parent.listFiles(PY_FILTER);
        if (fileList != null) {
            for (final File sourceFile : fileList) {
                analyzeFileContents(dependency, sourceFile);
            }
        }
    } else {
        engine.removeDependency(dependency);
    }
```

有一些是包装过的，不过大同小异。

```java
    public final void analyze(Dependency dependency, Engine engine) throws AnalysisException {
        if (this.isEnabled()) {
            analyzeDependency(dependency, engine);
        }
    }

    /**
     * Analyzes a given dependency. If the dependency is an archive, such as a
     * WAR or EAR, the contents are extracted, scanned, and added to the list of
     * dependencies within the engine.
     *
     * @param dependency the dependency to analyze
     * @param engine     the engine scanning
     * @throws AnalysisException thrown if there is an analysis exception
     */
    protected abstract void analyzeDependency(Dependency dependency, Engine engine) throws AnalysisException;

    /**
     * The close method does nothing for this Analyzer.
     *
     * @throws Exception thrown if there is an exception
     */
    @Override
    public final void close() throws Exception {
        if (isEnabled()) {
            closeAnalyzer();
        }
    }
```

# 生成报告

通过writeReports函数调用ReportGenerator，再创建一个context，写入对应信息。

```java
private VelocityContext createContext(String applicationName, List<Dependency> dependencies,
        List<Analyzer> analyzers, DatabaseProperties properties, String groupID,
        String artifactID, String version, ExceptionCollection exceptions) {

    final ZonedDateTime dt = ZonedDateTime.now();
    final String scanDate = DateTimeFormatter.RFC_1123_DATE_TIME.format(dt);
    final String scanDateXML = DateTimeFormatter.ISO_INSTANT.format(dt);
    final String scanDateJunit = DateTimeFormatter.ISO_LOCAL_DATE_TIME.format(dt);

    final VelocityContext ctxt = new VelocityContext();
    ctxt.put("applicationName", applicationName);
    Collections.sort(dependencies, Dependency.NAME_COMPARATOR);
    ctxt.put("dependencies", dependencies);
    ctxt.put("analyzers", analyzers);
    ctxt.put("properties", properties);
    ctxt.put("scanDate", scanDate);
    ctxt.put("scanDateXML", scanDateXML);
    ctxt.put("scanDateJunit", scanDateJunit);
    ctxt.put("enc", new EscapeTool());
    ctxt.put("rpt", new ReportTool());
    ctxt.put("WordUtils", new WordUtils());
    ctxt.put("VENDOR", EvidenceType.VENDOR);
    ctxt.put("PRODUCT", EvidenceType.PRODUCT);
    ctxt.put("VERSION", EvidenceType.VERSION);
    ctxt.put("version", settings.getString(Settings.KEYS.APPLICATION_VERSION, "Unknown"));
    ctxt.put("settings", settings);
    if (version != null) {
        ctxt.put("applicationVersion", version);
    }
    if (artifactID != null) {
        ctxt.put("artifactID", artifactID);
    }
```

通过cve数据库实例传入的properties与sql数据库连接，并查询到对应依赖的漏洞信息。但还没有找到是如何根据依赖进行搜索的。

```
public Properties getProperties() {
    final Properties prop = new Properties();
    try (Connection conn = databaseManager.getConnection();
            PreparedStatement ps = getPreparedStatement(conn, SELECT_PROPERTIES);
            ResultSet rs = ps.executeQuery()) {
        while (rs.next()) {
            prop.setProperty(rs.getString(1), rs.getString(2));
        }
    } catch (SQLException ex) {
        LOGGER.error("An unexpected SQL Exception occurred; please see the verbose log for more details.");
        LOGGER.debug("", ex);
    }
    return prop;
}
```