调研汇报

npm 简介

npm 是 Node.js 官方默认的包管理器，核心作用是管理项目所需的代码包（Package）。
npm可以自动生成 / 维护 package.json 文件（项目依赖清单），记录包的名称、版本、依赖关系等。

npm 生态规模

npm 受到全球超过 1700 万开发人员的依赖，拥有超过 200 万个软件包，是世界上最大的软件注册表。

npm 包管理机制

所有 npm 包都存储在官方的云端数据库，开发者可以将自己的包上传，也可以从数据库下载。

package.json

每个 npm 包都包含 `package.json` 文件，核心字段包括：

- `name`：包的唯一名称。
- `version`：包的版本号。
- `peerDependencies`:声明对宿主包的版本要求。
- `main`：包的入口文件。

各种依赖：

1. **dependencies**：生产环境依赖
    - 特性：项目运行时必需，会被打包工具（如Webpack）纳入最终产物
    - SCA关注：核心检测对象，直接影响生产环境安全
2. **devDependencies**：开发环境依赖
    - 特性：仅用于开发阶段（如测试、构建工具），默认不包含在生产部署中
    - 特殊规则：`npm install --production`会跳过此类依赖
    - SCA关注：仍需检测，开发环境漏洞可能影响供应链安全
3. **peerDependencies**：同伴依赖
    - 特性：声明对宿主包的版本要求（如React插件依赖特定React版本）
    - 特殊规则：不会自动安装，需由项目手动安装兼容版本
    - SCA关注：版本不匹配可能导致运行时错误或间接引入漏洞
4. **optionalDependencies**：可选依赖
    - 特性：安装失败不会导致整个安装过程失败
    - 特殊规则：常用于提供跨平台支持（如不同OS需要不同实现）
    - SCA关注：需检测存在时的版本风险
5. **bundledDependencies**：捆绑依赖

- 特性：发布包时会将这些依赖一同打包，安装时无需再从registry下载
- 特殊规则：依赖代码会被包含在发布包中，可能隐藏真实依赖关系
- SCA关注：需解压检测捆绑的依赖版本，易被传统SCA工具遗漏

`package.json` 文件用于描述项目的基本信息、依赖关系、可执行脚本等，例如：

```
{
    "name": "",
    "version": "",
    "description": "",
    "scripts": {},
    "dependencies": {
        "axios": "^0.19.2",
    },
    "devDependencies": {
        "eslint": "^5.16.0",
    },
    "peerDependencies": {},
    "engines": {}
}
```

package-lock.json

`package.json` 中声明的版本是 “范围”（如 `axios: ^1.4.0`），可能安装 `1.4.x` 的最新版；而 `package-lock.json` 会精确锁定每个依赖的版本（如axios: `1.4.2`）、下载地址和哈希值。
它的作用是“冻结”项目依赖的完整结构和精确版本，确保所有开发者 / 环境安装的依赖完全一致，例如：

```
{
    "name": "",
    "version": "",
    "lockfileVersion": 2,
    "requires": true,
    "packages": {
        "": {
            "name": "",
            "version": "",
            "dependencies": {
                "axios": "^0.19.2",
            },
            "devDependencies": {
                "eslint": "^5.16.0",
            }
        },
        "node_modules/@achrinza/node-ipc": {
            "version": "9.2.2",
            "resolved": "https://registry.npmmirror.com/@achrinza/node-ipc/-/node-ipc-9.2.2.tgz",
            "integrity": "sha512-b90U39dx0cU6emsOvy5hxU4ApNXnE3+Tuo8XQZfiKTGelDwpMwBVgBP7QX6dGTcJgu/miyJuNJ/2naFBliNWEw==",
```

```
            "dev": true,
            "dependencies": {
                "@node-ipc/js-queue": "2.0.3",
                "event-pubsub": "4.3.0",
                "js-message": "1.0.7"
            },
            "engines": {
                "node": "8 || 10 || 12 || 14 || 16 || 17"
            }
        }
    }
}
```

package-lock.json 的格式并非一成不变，它会随着 npm 版本迭代升级。 `lockfileVersion: 2` 是 npm 7 及以上版本引入的新格式（替代了 npm 5-6 的 `lockfileVersion: 1` ），核心变化就是用 `packages` 字段统一管理所有依赖信息，包括项目本身和所有子依赖。
其中 `packages:{ "": { ... } }` 对应项目本身即package.json中的直接依赖。
`packages` 下的其他键时项目的间接依赖。
在 npm 6 及之前（ `lockfileVersion: 1` ）， `package-lock.json` 的顶层有 `dependencies` 字段，直接包含所有依赖（包括直接和间接），结构相对扁平。

版本管理（SemVer 语义化版本）

npm 的版本管理基于 SemVer 规范，格式为 `主版本号.次版本号.修订号` （如 `1.4.2` ），不同号段代表不同的更新类型，避免版本混乱。

`SemVer` 的 3 个号段含义

主版本号（Major）：不兼容的 API 更新（如 `1.0.0 → 2.0.0` ），升级后可能需要修改代码。 次版本号（Minor）：向后兼容的功能新增（如 `1.4.0 → 1.5.0` ），不影响现有功能。 修订号（Patch）：向后兼容的 bug 修复（如 `1.4.1 → 1.4.2` ），仅修复问题，无功能变化。

版本范围符号

在 `package.json` 中，版本号前的符号代表 “允许安装的版本范围”，常见符号如下： `^` ：允许次版本和修订号更新（如 `^1.4.0` 可安装`1.4.x` 、 `1.5.x` ，但不超过 `2.0.0` ）。 `~` ：仅允许修订号更新（如 `~1.4.0` 可安装 `1.4.1` 、 `1.4.2` ，但不超过 `1.5.0` ）。 *：允许任意版本（不推荐，风险高）。 无符号：精确版本（如 `1.4.0` ，仅安装该版本）。

dependencies 和 devDependencies 中记录的依赖版本通常是范围版本，遵循 npm 的版本规则，格式如下：
^1.2.3：允许更新 minor 或 patch 版本（如 1.2.3 → 1.3.0 或 1.2.4，但不允许 2.0.0）。
~1.2.3：允许更新 patch 版本（如 1.2.3 → 1.2.4，但不允许 1.3.0）。
1.2.3：精确版本（仅安装 1.2.3）。
*：任意版本。

完整SCA工具框架

1. **Snyk CLI（部分开源）**
    - 功能：完整的开源安全扫描工具，支持NPM依赖检测

- 特点：提供漏洞修复建议和许可证合规检查
- 开源地址：https://github.com/snyk/cli

2. **Dependency-Check**

- 功能：OWASP开发的依赖检查工具，支持JavaScript生态
- 特点：集成多个漏洞数据库，可生成详细报告
- 开源地址：https://github.com/jeremylong/Dependency-Check

3. **Trivy**

- 功能：容器和代码漏洞扫描工具，支持NPM项目
- 特点：扫描速度快，支持离线模式
- 开源地址：https://github.com/aquasecurity/trivy

END