https://github.com/Advanced-Frontend/Daily-Interview-Question/issues/22 #1. npm 模块安装机制：

- 发出npm install命令
- 查询node_modules目录之中是否已经存在指定模块
  - 若存在，不再重新安装
  - 若不存在
    - npm 向 registry 查询模块压缩包的网址
    - 下载压缩包，存放在根目录下的.npm目录里
    - 解压压缩包到当前项目的node_modules目录 #2. npm 实现原理 输入 npm install 命令并敲下回车后，会经历如下几个阶段（以 npm 5.5.1 为例）：

####1. 执行工程自身 preinstall 当前 npm 工程如果定义了 preinstall 钩子此时会被执行。

####2. 确定首层依赖模块 首先需要做的是确定工程中的首层依赖，也就是 dependencies 和 devDependencies 属性中直接指定的模块（假设此时没有添加 npm install 参数）。

工程本身是整棵依赖树的根节点，每个首层依赖模块都是根节点下面的一棵子树，npm 会开启多进程从每个首层依赖模块开始逐步寻找更深层级的节点。

####3. 获取模块 #####获取模块是一个递归的过程，分为以下几步：

- 获取模块信息

  在下载一个模块之前，首先要确定其版本，这是因为 package.json 中往往是 semantic version（semver，语义化版本）。此时如果版本描述文件（npm-shrinkwrap.json 或 package-lock.json）中有该模块信息直接拿即可，如果没有则从仓库获取。如 packaeg.json 中某个包的版本是 ^1.1.0，npm 就会去仓库中获取符合 1.x.x 形式的最新版本。

- 获取模块实体 上一步会获取到模块的压缩包地址（resolved 字段），npm 会用此地址检查本地缓存，缓存中有就直接拿，如果没有则从仓库下载。 查找该模块依赖，如果有依赖则回到第1步，如果没有则停止。 ####4 模块扁平化（dedupe） 上一步获取到的是一棵完整的依赖树，其中可能包含大量重复模块。比如 A 模块依赖于 loadsh，B 模块同样依赖于 lodash。在 npm3 以前会严格按照依赖树的结构进行安装，因此会造成模块冗余。

  从 npm3 开始，默认加入了一个 dedupe 的过程。它会遍历所有节点，逐个将模块放在根节点下面，也就是 node-modules 的第一层。当发现有重复模块时，则将其丢弃。

这里需要对重复模块进行一个定义，它指的是模块名相同且 semver 兼容。每个 semver 都对应一段版本允许范围，如果两个模块的版本允许范围存在交集，那么就可以得到一个兼容版本，而不必版本号完全一致，这可以使更多冗余模块在 dedupe 过程中被去掉。

比如 node-modules 下 foo 模块依赖 lodash@^1.0.0，bar 模块依赖 lodash@^1.1.0，则 ^1.1.0 为兼容版本。

而当 foo 依赖 lodash@^2.0.0，bar 依赖 lodash@^1.1.0，则依据 semver 的规则，二者不存在兼容版本。会将一个版本放在 node_modules 中，另一个仍保留在依赖树里。

举个例子，假设一个依赖树原本是这样：

node_modules -- foo ---- lodash@version1

-- bar ---- lodash@version2

假设 version1 和 version2 是兼容版本，则经过 dedupe 会成为下面的形式：

node_modules -- foo

-- bar

-- lodash（保留的版本为兼容版本）

假设 version1 和 version2 为非兼容版本，则后面的版本保留在依赖树中：

node_modules -- foo -- lodash@version1

-- bar ---- lodash@version2

####5.安装模块 这一步将会更新工程中的 node_modules，并执行模块中的生命周期函数（按照 preinstall、install、postinstall 的顺序）。

####6.执行工程自身生命周期 当前 npm 工程如果定义了钩子此时会被执行（按照 install、postinstall、prepublish、prepare 的顺序）。

最后一步是生成或更新版本描述文件，npm install 过程完成。