Lerna入门与实战

一、Lerna简介

1.1 lerna背景

维护过多个package项目的同学可能都会遇到一个问题：package是放在一个仓库里维护还是放在多个仓库里单独维护。当package数量较少的时候，多不会有太大问题，但package数量逐渐增多时，一些问题逐渐暴露出来：

- package之间相互依赖，开发人员需要在本地手动执行npm link，维护版本号的更替；
- issue难以统一追踪，管理，因为其分散在独立的repo里；
- 每一个package都包含独立的node_modules，而且大部分都包含babel,webpack等开发时依赖，安装耗时冗余并且占用过多空间。

正是在这一需求背景下， babel 团队推出的多包管理工具lerna，旨在优化基于Git+npm的多package项目的包管理方式。 像现在流行的vue-cli, create脚手架工具都有用到lerna。lerna是架构优化的产物，而架构优化的主要目标是以提高ROI为核心的多package管理。

作为一种多包依赖解决方案，lerna具体如下特点：

- 可以管理公共依赖和单独依赖；
- 多package相互依赖直接内部 link，不必发版；
- 支持项目的单独发布和全体发布；
- 多包放一个git仓库，利于代码管理，如配置统一的代码规范；

1.2 两种模式

lerna有两种工作模式，Independent模式和Fixed/Locked模式。这两种模式有什么区别呢？lerna的默认模式是Fixed/Locked mode，在这种模式下，ler当作一个整体来对待，每次发布packges，都是全量发布，无论修改与否。但是在Independent mode下，lerna会配合Git，检查文件变动，只发布有改

二、快速上手

2.1 安装lerna

使用lerna之前，需要全局安装lerna，安装的命令如下：

```
yarn global add lerna
#or
npm install lerna -g
```

如果是在已经存在的项目中安装lerna，使用下面的方式：

```
lerna bootstap
```

2.2 初始化项目

使用lerna 初始化项目的方式和使用npm方式类似。首先，我们在一个空目录中执行如下初始化命令。

```
lerna init
```

默认使用的是固定模式，packages下的所有包共用一个版本号，如果使用独立模式，需要在init后面加一个参数。

```
lerna init --independent
```

执行上面的命令后，lerna会创建一个lerna.json配置文件和packages文件夹，此时项目的目录结构如下。

```
lerna-repo/
  packages/
  package.json
  lerna.json
```

lerna.json中的packages配置是一个通配符列表，用于匹配包含了package.json的的目录。

2.2.1 lerna.json

其中，lerna.json的文件配置内容如下：

```
{
  "version": "1.1.3",
  "npmClient": "npm",
  "command": {
    "publish": {
      "ignoreChanges": ["ignored-file", "*.md"],
      "message": "chore(release): publish",
      "registry": "https://npm.pkg.github.com"
    },
    "bootstrap": {
      "ignore": "component-*",
      "npmClientArgs": ["--no-package-lock"]
    }
  },
  "packages": ["packages/*"]
}
```

下面是部分属性的说明：

- version：当前版本
- npmClient：指定运行命令的客户端，设定为"yarn"则使用yarn运行，默认值是"npm"。
- command.publish.ignoreChanges：通配符的数组，其中的值不会被 lerna 监测更改和发布，使用它可以防止因更改发布不必要的新版本。
- command.publish.message：执行发布版本更新时的自定义提交消息。
- command.publish.registry：使用它来设置要发布的自定义注册 url，而非 npmjs.org。
- command.bootstrap.ignore：运行lerna bootstrap指令时会忽视该字符串数组中的通配符匹配的文件。
- command.bootstrap.npmClientArgs：该字符串数组中的参数将在lerna bootstrap命令期间直接传递给npm install。
- command.bootstrap.scope：该通配符的数组会在lerna bootstrap命令运行时限制影响的范围。
- packages ： 表示包位置的全局变量数组。

2.2.2 其他命令

除了上面的init命令外，项目使用过程中还会用到很多其他有用的命令。

- lerna create：此命令的作用是用来创建一个子包名为xx的项目。
- lerna add：此命令用于安装依赖，格式为lerna add [@version] [–dev]。
- lerna list：查看当前包名列表。
- lerna link：将所有相互依赖的包符号链接在一起。
- lerna exec：在每个包中执行任意命令。
- lerna run：在每个包中运行npm脚本如果该包中存在该脚本。

2.3 新建模块

接下来，新建moduleA和moduleB两个模块， 并且moduleA需要依赖moduleB。

```
lerna create module-a
lerna create module-b
```

在module-a中引入module-b，module-a代码如下。

```
// module-a
'use strict';
const moduleB = require('module-b');
console.log('moduleB:', moduleB());


module.exports = moduleA;

```

```
9
10 function moduleA() {
11     return 'it's module a';
12 }
```

module-b的代码如下。

```
1  // module-b
2  'use strict';
3
4
5  module.exports = moduleB;
6
7
8  function moduleB() {
9      return 'it's module b';
10 }
```

此时，我们运行node packages/module-a/lib/module-a.js 可能会报错，提示找不到module-b模块，这是因为我们还没在moduleA中安装依赖，需要使用
安装依赖。

```
1  lerna add module-b --scope=module-a
```

可以看到，module-a node_modules目录下安装了module-b包，重新运行上面的module-a代码会输出如下内容。

```
1  $ node packages/module-a/lib/module-a.js
2  moduleB: it's module b
```

2.4 发布

执行发布的时候，需要Git工具的配合，请确认此时该lerna工程已经连接到Git的远程仓库。如果是第一次发布，可能需要先执行如下命令。

```
1  npm login --registry=https://registry.npmjs.org
```

然后，我们将module-a、module-b进行统一改个名称，比如统一加上@m_alfred前缀。

```
1  @m_alfred/module-a
2  @m_alfred/module-b
```

然后，执行在输入用户名及密码之后执行发布命令。

```
1  lerna publish
```

发布完成之后，即可在npm仓库看到对应的提交信息。

```
1  $ lerna publish
2  lerna notice cli v3.22.1
3  lerna info current version 0.0.2
4  lerna info Looking for changed packages since v0.0.2
5  ? Select a new version (currently 0.0.2) Patch (0.0.3)
6
7
8  Changes:
9   - @m_alfred/module-a: 0.0.2 => 0.0.3
10  - @m_alfred/module-b: 0.0.2 => 0.0.3
11
```

发布成功后，单独修改moduleB代码，然后再执行发布命令。

```
1  Changes:
2   - @m_alfred/module-a: 0.0.3 => 0.0.4
3   - @m_alfred/module-b: 0.0.3 => 0.0.4
```

可以看到，module-a版本也升级了，这是由于我们在初始化erna init的时候使用的是默认的Fixed mode模式，所以packages下所有包都会使用同一版本

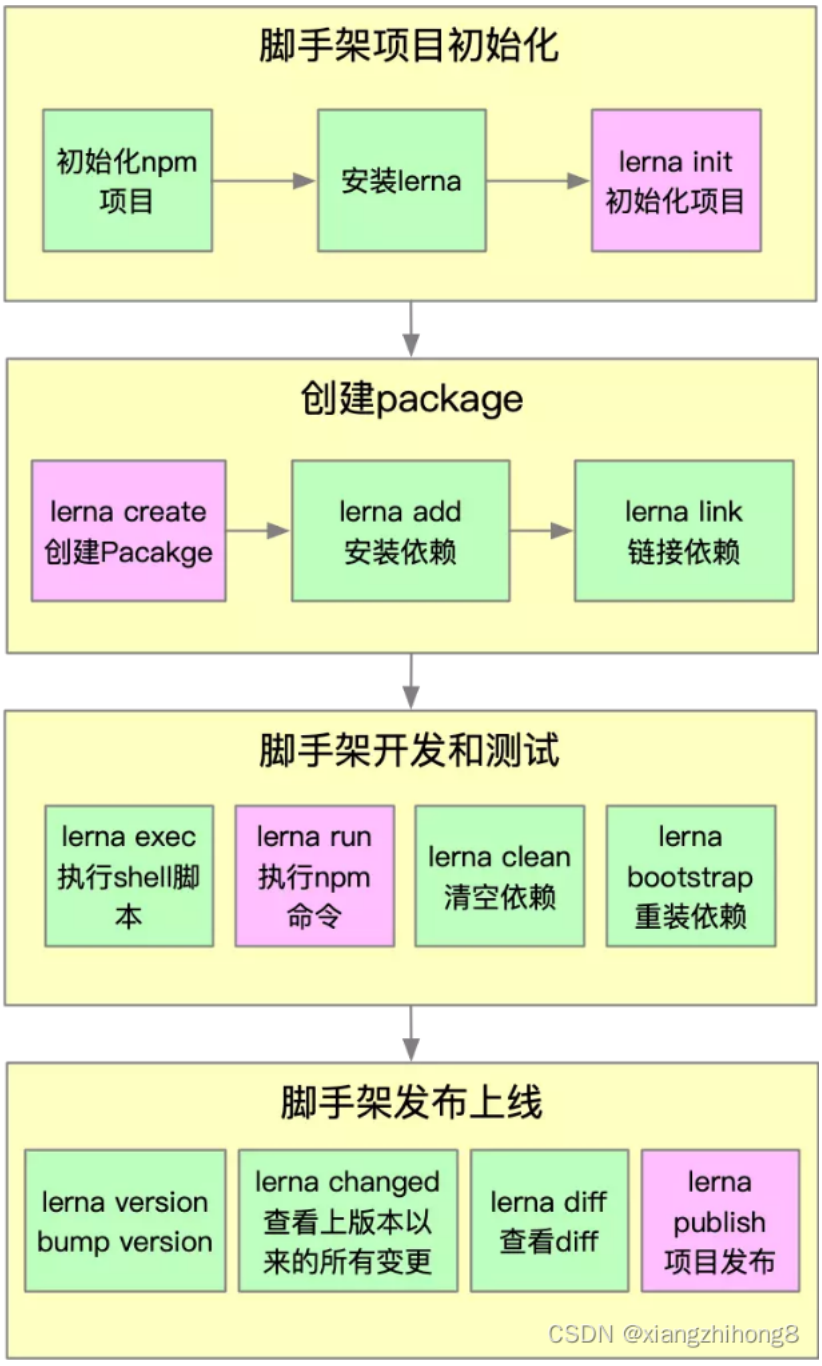
2.5 共用devDependencies

在开发过程中，很多模块都会依赖babel、eslint等模块，这些大多都是可以共用的。此时，我们可以通过lerna link convert命令将各个包package.json中devDependencies移动到根目录的package.json中，将它们提升到项目根目录中，这样做的好处有：

- 所有包使用相同版本的依赖，统一管理；
- 可使用自动化工具让根目录下的依赖保持更新；
- 减少依赖的安装时间，一次安装，多处使用；
- 节省存储空间，安装在根目录的node_module下。

三、工作过程总结

下面是使用lerna的完整工作流程图。



参考：http://www.febeacon.com/lerna-docs-zh-cn/routes/basic/about.html