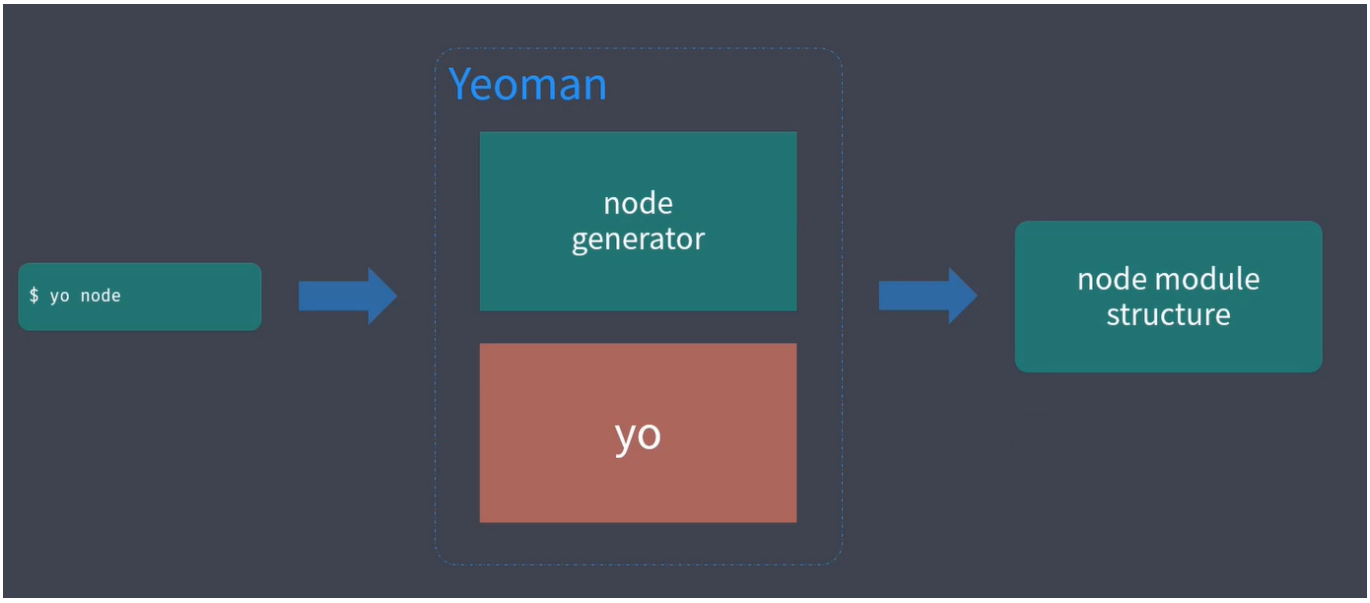
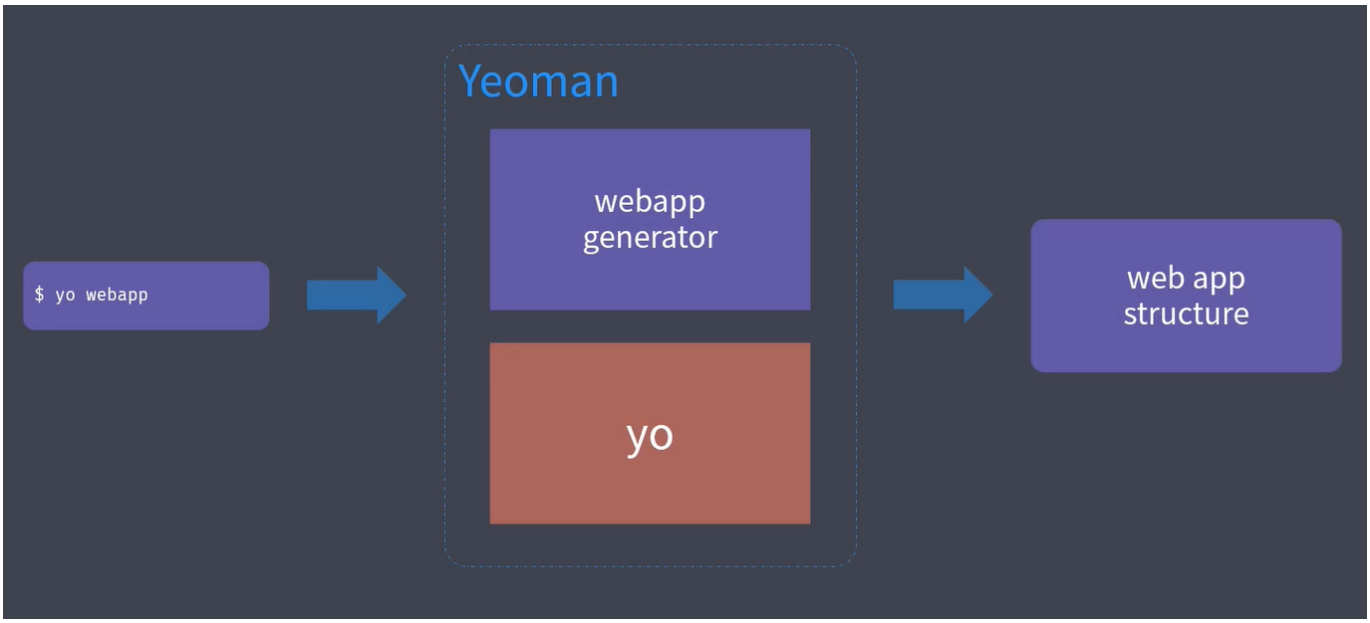
第二节 Yeoman

1.定义

Yeoman 是一个通用的脚手架系统，允许创建任何类型的应用程序（Web，Java，Python，C＃等）。用 yeoman 写脚手架非常简单， yeoman 提供了 yeoman-generator 让我们快速生成一个脚手架模板，我们的主要工作就是把模板文件写好。现在我们来用 yeoman 写一个生成 javascript 插件的脚手架吧。
脚手架功能：

- 自动构建编译和打包
- 支持 es6 语法
- 支持单元测试
- 支持 jsdoc 生成文档
- 支持 eslint 语法检查
- 自动生成 changelog





2.准备工作

1. 明确你的需求
2. 找到合适的Generator
3. 全局范围安装扎到Generator
4. 通过Yo运行对应的Generator
5. 通过命令行交互填写选项
6. 生成你所需要的的项目结构

2.1 使用generator

安装yeoman和generator-generator

```
yarn global install yo -g
npm install generator-generator -g
```

安装对应的generator
更多的generator[https://yeoman.io/generators/]

```
npm install generator-node --global # or yarn global add generator-node
```

通过yo 运行generator

```
$ cd my-module
$ yo node
```

2.2 Sub Generator

- 在原有项目的基础上场景特定类型的文件(例如以下配置文件等)

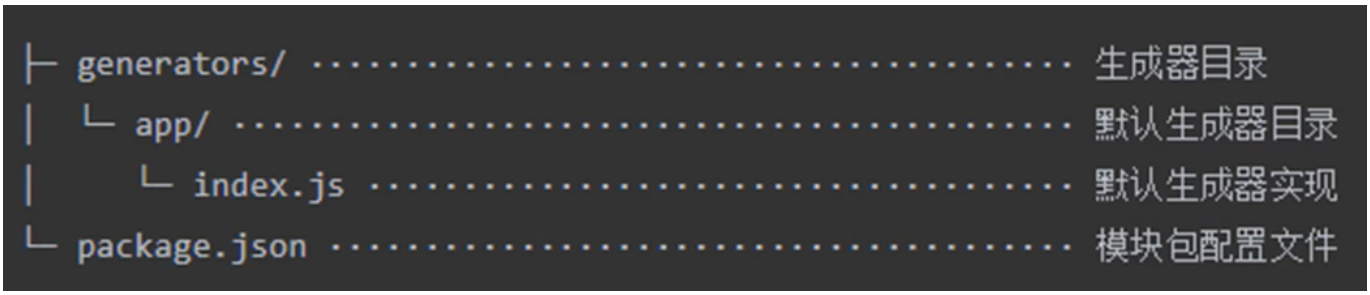
例如创建一个node cli应用

```
yo node:cli
```

并不是每一个generator 都会有子集生成器,需要官网查看

3.自定义Generator

创建Generator本质上就是创建一个npm模块
而且命名必需是generator-<name>的形式



yeoman 提供了一个基本生成器，你可以扩展它以实现自己的行为。这个基础生成器将帮你减轻大部分工作量。在生成器的 index.js 文件中，以下是扩展基本生成器的方法：

```
yarn add yeoman-generator
```

```
var Generator = require("yeoman-generator");

module.exports = class extends Generator {};
```

yeoman 生命周期函数执行顺序如下：

- initializing – 初始化函数
- prompting – 接收用户输入阶段
- configuring – 保存配置信息和文件
- default – 执行自定义函数
- writing – 生成项目目录结构阶段
- conflicts – 统一处理冲突，如要生成的文件已经存在是否覆盖等处理
- install – 安装依赖阶段
- end – 生成器结束阶段

我们常用的就是 initializing、prompting、default、writing、install 这四种生命周期函数。看下例子：

```
"use strict";
const Generator = require("yeoman-generator");
const chalk = require("chalk"); // 让console.log带颜色输出
const yosay = require("yosay");
const mkdirp = require("mkdirp"); // 创建目录

module.exports = class extends Generator {
  initializing() {
    this.props = {};
  }

  // 接受用户输入
  prompting() {
    // Have Yeoman greet the user.
    this.log(
      yosay(
```

```
        `Welcome to the grand ${chalk.red(
          "generator-javascript-plugin"
        )} generator!`
      )
    );

    const prompts = [
      {
        type: "confirm",
        name: "someAnswer",
        message: "Would you like to enable this option?",
        default: true
      }
    ];

    return this.prompt(prompts).then(props => {
      // To access props later use this.props.someAnswer;
      this.props = props;
    });
  }

  // 创建项目目录
  default() {
    if (path.basename(this.destinationPath()) !== this.props.name) {
      this.log(`\nYour generator must be inside a folder named
        ${this.props.name}\n
        I will automatically create this folder.\n`);

      mkdirp(this.props.name);
      this.destinationRoot(this.destinationPath(this.props.name));
    }
  }
  // 写文件
  writing() {
    // 将templates目录的代码拷贝到目标目录
    // templates目录默认路径是generators/app/templates
    this.fs.copy(
      this.templatePath("dummyfile.txt"),
      this.destinationPath("dummyfile.txt")
    );
```

```
    this._writingPackageJSON();
  }

  // 以下划线_开头的是私有方法
  _writingPackageJSON() {
    // this.fs.copyTpl(from, to, context)
    this.fs.copyTpl(
      this.templatePath("_package.json"),
      this.destinationPath("package.json"),
      {
        name: this.props.name,
        description: this.props.description,
        keywords: this.props.keywords.split(","),
        author: this.props.author,
        email: this.props.email,
        repository: this.props.repository,
        homepage: this.props.homepage,
        license: this.props.license
      }
    );
  }

  // 安装依赖
  install() {
    this.installDependencies();
  }
};
```

在writting中使用fs模块写入文件

```
// 此文件作为Generator 的核心入口
// 需要继承自yeoman Generator的类型
// Yeoman Generator在工作时会自动调用我们在此类型中定义的一些生命周期方法
const Generator = require('yeoman-generator')

module.exports = class extends Generator {
    writing() {
        this.fs.write(this.destinationPath('temp.txt'),Math.random()
```

```
    .toString())
    }
}
```

4.编写模板文件

可以使用`<%= name %>`这样的模板语法使用脚手架中的`context`上下文，无论是用户输入的数据，还是程序自己的变量(类似ejs的语法规则)

```
这是一个模板文件
内部可以使用 EJS 模板标记输出数据
例如：<%= title %>

    <%= success %>
其他的 EJS 语法也支持

<% if (success) { %>
哈哈哈
<% }%>
```

```
// 此文件作为Generator 的核心入口
// 需要继承自yeoman Generator的类型
// Yeoman Generator在工作时会自动调用我们在此类型中定义的一些生命周期方法
const Generator = require('yeoman-generator')

module.exports = class extends Generator {
    writing() {
        // this.fs.write(this.destinationPath('temp.txt'),Math.rando
m().toString())
        // 模板文件路径
        const tmpl = this.templatePath('foo.txt')
        // 输出目标路径
        const output = this.destinationPath('foo.txt')
        // 模板文件上下文
        const context = {title: 'Hello world',success: true}
        // 自动把数据映射到模板文件上
        this.fs.copyTpl(tmpl,output,context)
    }
```

```
}
```

5.接收用户输入

重写Generator的中的promting方法

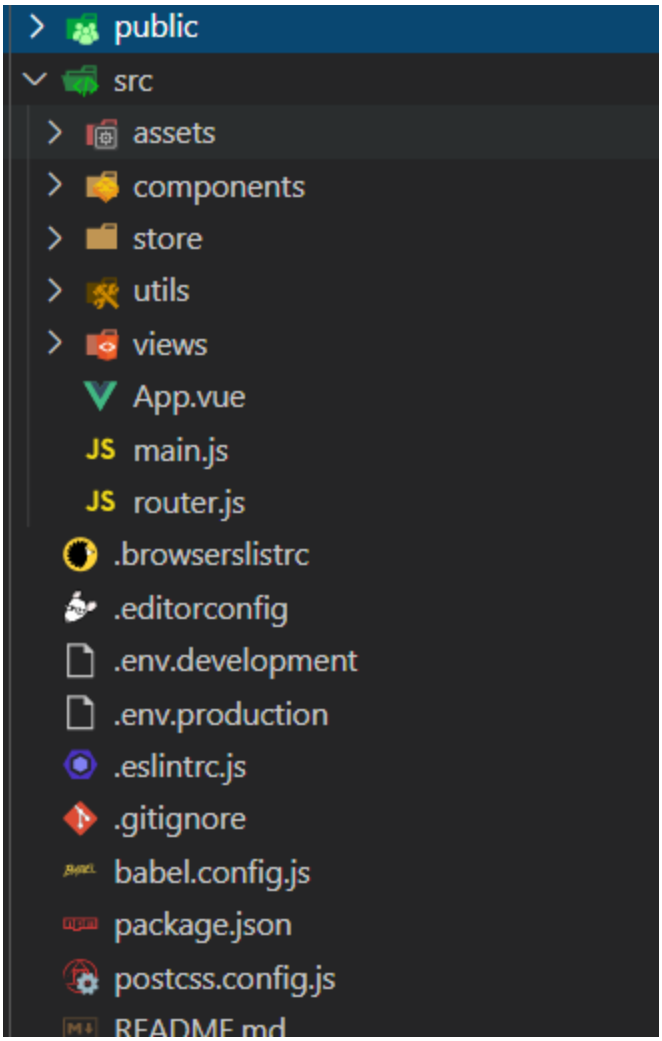
```
// 此文件作为Generator 的核心入口
// 需要继承自yeoman Generator的类型
// Yeoman Generator在工作时会自动调用我们在此类型中定义的一些生命周期方法
const Generator = require('yeoman-generator')

module.exports = class extends Generator {
    prompting() {
        // Yeoman在询问用户环节会自动调用此方法
        // 在此方法中可以调用父类的prompt()方法发出对用户的命令行询问
        return this.prompt([
            {
                type: 'input', // 类型为用户输入
                name: 'name', //用户输入接收的key
                message: 'Your project name', // 提示消息
                default: this.appname // appname为项目生成目录名称,也可以自定义
            }
        ]).then(answers => {
            // answers 是一个对象 {name: 'input value'}
            this.answers = answers
        })
    }
    writing() {
        // this.fs.write(this.destinationPath('temp.txt'),Math.random().toString())
        // 模板文件路径
        const tmpl = this.templatePath('foo.txt')
        // 输出目标路径
        const output = this.destinationPath('foo.txt')
        // 模板文件上下文
        const context = {title: 'Hello world',success: true}
        // 自动把数据映射到模板文件上
        this.fs.copyTpl(tmpl,output,context)
```

```
32
33          const tmpl2 = this.templatePath('bar.html')
34          const output2 = this.destinationPath('bar.html');
35          const context2 = this.answers
36          this.fs.copyTpl(tmpl2,output2,context2)
37      }
38  }
```

6.vue 脚手架综合案例

6.1 复制最基本的vue项目文件



6.2 创建generator-vue

安装yeoman-generator依赖

```
1 yarn add yeoman-generator
```

```
const Generator = require('yeoman-generator')

module.exports = class extends Generator {
    prompting() {
        return this.prompt([
            {
                type: 'input',
                name: 'name',
                message: 'your project name',
                default: this.appname
            }
        ]).then(answers => {
            this.answers = answers
        })
    }
    writing() {
            // 把每一个文件都通过模板转换到目标路径

    const templates = [
        '.browserslistrc',
        '.editorconfig',
        '.env.development',
        '.env.production',
        '.eslintrc.js',
        '.gitignore',
        'babel.config.js',
        'package.json',
        'postcss.config.js',
        'README.md',
        'public/favicon.ico',
        'public/index.html',
        'src/App.vue',
        'src/main.js',
        'src/router.js',
        'src/assets/logo.png',
        'src/components/HelloWorld.vue',
        'src/store/actions.js',
```

```
        'src/store/getters.js',
        'src/store/index.js',
        'src/store/mutations.js',
        'src/store/state.js',
        'src/utils/request.js',
        'src/views/About.vue',
        'src/views/Home.vue'
        ]
        templates.forEach(item => {
            this.fs.copyTpl(
                this.templatePath(item),
                this.destinationPath(item),
                this.answers
            )
        })
    }
}
```

7.发布Generator

实际上是发布一个npm模块

- 创建一个git本地仓库,并发布到远程
- 设置npm 远程仓库的账号邮箱
- 使用npm / yarn publish ,记住要关掉淘宝镜像