Element3.0工程化实践

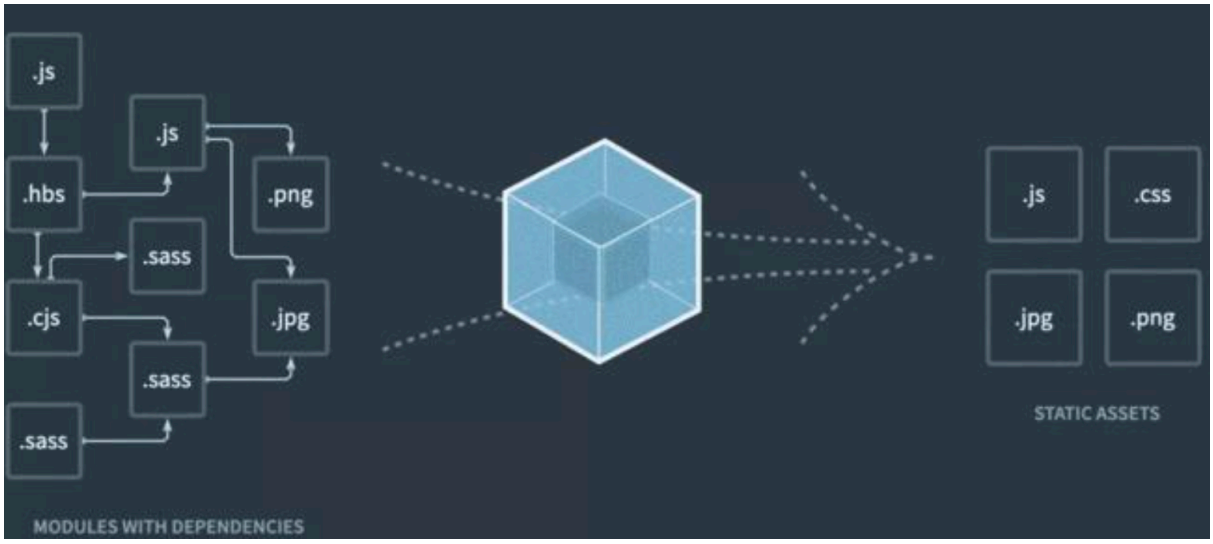
一、前端工程化是什么

> 前端工程化概述 https://juejin.im/post/6844904073817227277

随着对前端功能和性能的不断提高，前端早就不是一段内嵌于页面的一段JS代码了。已经进化为一个系统复杂的工程了。

前端的功能化

1. 模块化



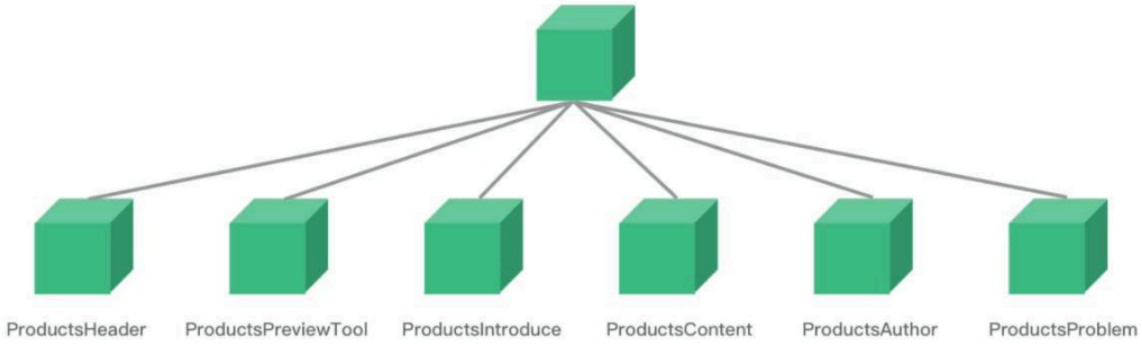
一个文件分拆为多个互相依赖的文件，最后进行统一打包和加载，保证高效多人协作。

- JS模块 CMD AMD CommonJS 及 ES6 Module

- CSS模块 Sass Less Stylus
  - 资源模块化 文件、CSS、图片通过JS进行统一依赖关联

2. 组件化

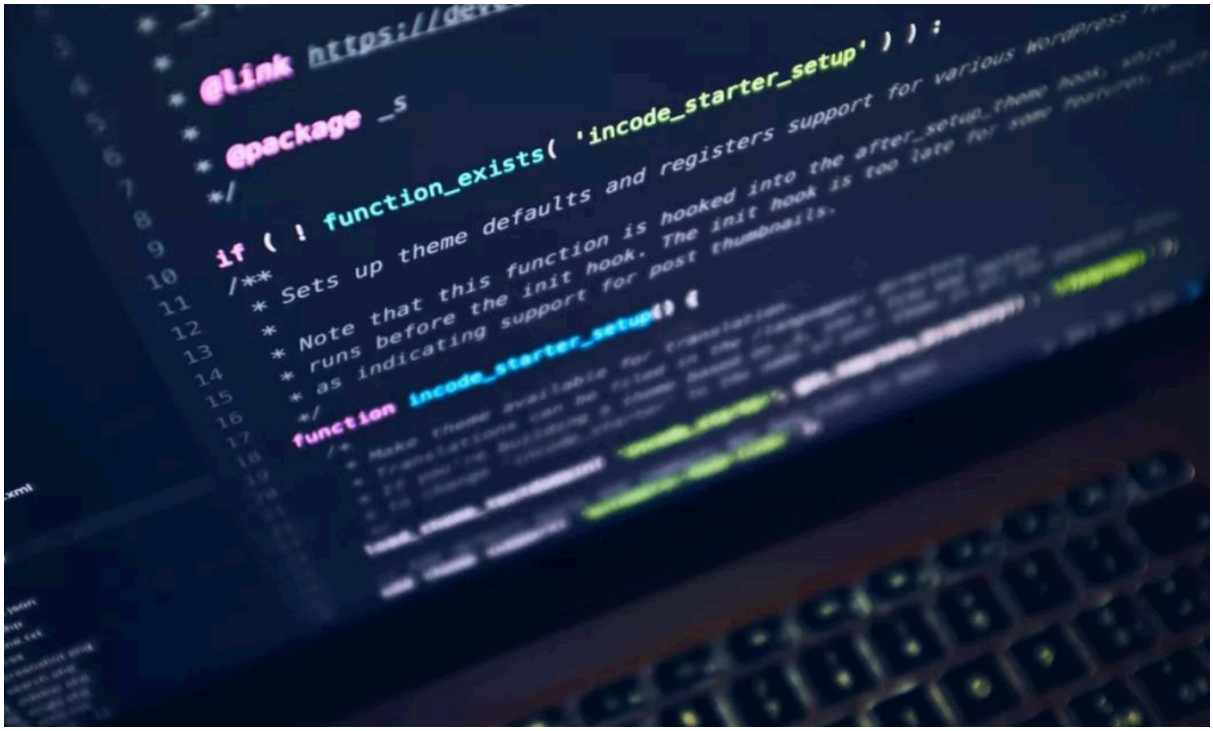
相对于文件的拆分，组件是对于UI层面的拆分，每一个组件需要包括对应的CSS、图片、JS逻辑、视图模板等并且能完成一个独立的功能。



3. 自动化

- 调试
- 编译
- 部署
- 测试
- 文档化

4. 规范性



   - 项目目录结构
   - 语法提示
   - 编码风格规范
   - 联调规范
   - 文件命名规范
   - 代码样式规范
   - git flow

二、实战步骤

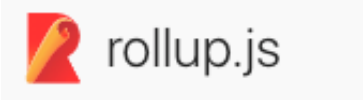
0 可行性研究

0.1 Vue组件与插件

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Document</title>
    <script src="/node_modules/vue/dist/vue.global.js"></script>
    <script src="/dist/element3-ui.global.js"></script>
    <link href="/lib/theme-chalk/index.css" rel="stylesheet" />
    <style></style>
  </head>

  <body>
    <div id="app"></div>
```

```
    <script>
      const { createApp, reactive, computed, watchEffect } = Vue;

      const MyButton = {
        name: "MyButton",
        data: function () {
          return {
            count: 0,
          };
        },
        template:
          '<button v-on:click="count++">You clicked me {{ count }} times.
</button>',
      };

      // 添加插件
      MyButton.install = (app) => app.component("MyButton", MyButton);

      // 组件库
      const Element = {
          MyButton,
          install: app => {
              app.use(MyButton)
          }
      }

      const MyComponent = {
        template: `
                <my-button />
            `,
      };

      createApp(MyComponent)
        // .use(MyButton)
        .use(Element)
        .mount("#app");
    </script>
  </body>
</html>
```

0.2 rollup打包



rollup是一款小巧的javascript模块打包工具，更适合于库应用的构建工具;可以将小块代码编译成大块复杂的代码，基于ES6 modules,它可以让你的 bundle 最小化，有效减少文件请求大小,vue在开发的时候用的是webpack,但是最后将文件打包在一起的时候用的是 rollup.js

> 首次发表在个人博客

- rollup官方文档
- rollupGithub

> https://juejin.im/post/6844903570974703629 Rollup基础

Button

/src/MyButton.js

```
export default {
  name: "MyButton",
  data: function () {
    return {
      count: 0,
    };
  },
  template:
    '<button v-on:click="count++">You clicked me {{ count }} times.</button>',
};
```

入口

/src/entry.js

```
import MyButton from "./MyButton";
import SfcButton from "./SfcButton.vue";
import JsxButton from "./JsxButton.vue";

// 添加插件
MyButton.install = (app) => app.component("MyButton", MyButton);
SfcButton.install = (app) => app.component("SfcButton", SfcButton);
JsxButton.install = (app) => app.component("JsxButton", JsxButton);

// 组件库
const Element = {
```

```
  MyButton,
  SfcButton,
  JsxButton,
  install: (app) => {
    app.use(MyButton);
    app.use(SfcButton);
    app.use(JsxButton);
  },
};

export default Element;
```

格式声明

> https://juejin.im/post/6885542715782594568 AMD CMD UMD区别

- amd – 异步模块定义，用于像 RequireJS 这样的模块加载器
- cjs – CommonJS，适用于 Node 和 Browserify/Webpack
- es – 将软件包保存为 ES 模块文件
- iife – 一个自动执行的功能，适合作为`<script>`标签。（如果要为应用程序创建一个捆绑包，您可能想要使用它，因为它会使文件大小变小。）
- umd – 通用模块定义，以 amd，cjs 和 iife 为一体

```
yarn add vue
yarn add @vue/compiler-sfc # vue单文件组件
yarn add @babel/core   # babel核心
yarn add @rollup/plugin-babel
yarn add rollup-plugin-terser # 压缩代码
yarn add rollup-pluginutils
```

```
// rollup.config.js
// const vuePlugin = require("../../rollup-plugin-vue/index");
import babel from "@rollup/plugin-babel";
import vuePlugin from "rollup-plugin-vue";

const es = {
    input: "src/entry.js",
    output: {
      file: "dist/index.bundle.js",
      name: "Element",
      format: "es",
```

```
      globals: {
        vue: "Vue",
      },
    },
    external: ["vue"],
    plugins: [
      babel(),
      vuePlugin({
        css: true,
      }),
    ],
  };

const iife = {
  input: "src/entry.js",
  output: {
    file: "dist/index.js",
    name: "Element",
    format: "iife",
    globals: {
      vue: "Vue",
    },
  },
  external: ["vue"],
  plugins: [
    babel(),
    vuePlugin({
      css: true,
    }),
  ],
};

import { terser } from "rollup-plugin-terser";
const minEs = {
  input: "src/entry.js",
  external: ["vue"],
  output: {
    file: "dist/index.min.js",
    name: "Element",
    format: "umd",
  },
  plugins: [
    babel(),
    vuePlugin({
      css: true,
    }),
    terser(),
  ],
};
```

```
const cjs = {
  input: "src/entry.js",
  external: ["vue"],
  output: {
    file: "dist/index.cjs.js",
    name: "Element",
    format: "cjs",
  },
  plugins: [
    babel(),
    vuePlugin({
      css: true,
    }),
  ],
};

export default [es,iife, minEs, cjs];
```

测试页面

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Document</title>
    <script src="https://unpkg.com/vue@next"></script>
    <script src="dist/index.js"></script>
    <style></style>
  </head>

  <body>
    <div id="app"></div>
    <script>
      const { createApp, reactive, computed, watchEffect } = Vue;

      const MyComponent = {
        template: `
                <my-button />
                <sfc-button />
                <jsx-button />
            `,
      };

      createApp(MyComponent)
        .use(Element)
```

```
        .mount("#app");
    </script>
  </body>
</html>
```

0.4 单文件组件

SfcButton.vue

```
<template>
  <button>Sfc 666</button>
</template>
<script>
export default {
  name: "SfcButton",
};
</script>
```

```
const vuePlugin = require("../../rollup-plugin-vue/index");
// import vuePlugin from "rollup-plugin-vue";
# plugin
vuePlugin({
css: true,
}),
```

0.5 JSX支持

> https://github.com/vuejs/jsx-next
> jsx的定义

> JSX 是一种类似于 XML 的 JavaScript 语法扩展 JSX 不是由引擎或浏览器实现的。相反，我们将使用像 Babel 这样的转换器将 JSX 转换为常规 JavaScript。基本上，JSX 允许我们在 JavaScript 中使用类似 HTML 的语法。

jsx的优势

1. 可以将 模版分离 这样模版的每个部分更加独立，又可以随机的组合，复用性更高。相比与组件的组合，粒度更细
2. 使用 js 可配置每项要渲染的 dom，更加动态可配置化

```
import babel from "@rollup/plugin-babel";
# plugin
babel(),
```

JsxButton.vue

```
<script>
export default {
  name: "JsxButton",
  render() {
    return <button>JSX 666</button>;
  },
};
</script>
```

###

1. 模块化与组件化

```
npm init -y
```

> https://github.com/cuixiaorui/course-vue3-test/tree/main/chapters/two
>
> 参考资料

1.1 编写Buttun组件

```
yarn add vue@next
```

/packages/button/Button.vue

```
<template>
  <button
    class="el-button"
    @click="handleClick"
    :disabled="buttonDisabled || loading"
    :autofocus="autofocus"
    :type="nativeType"
    :class="[
      type ? 'el-button--' + type : '',
```

```
      buttonSize ? 'el-button--' + buttonSize : '',
      {
        'is-disabled': buttonDisabled,
        'is-loading': loading,
        'is-plain': plain,
        'is-round': round,
        'is-circle': circle,
      },
    ]"
  >
    <i class="el-icon-loading" v-if="loading"></i>
    <i :class="icon" v-if="icon && !loading"></i>
    <span v-if="$slots.default">
      <slot></slot>
    </span>
  </button>
</template>
<script>
import { computed, inject, toRefs, unref, getCurrentInstance } from "vue";

export default {
  name: "ElButton",

  props: {
    type: {
      type: String,
      default: "default",
    },
    size: {
      type: String,
      default: "",
    },
    icon: {
      type: String,
      default: "",
    },
    nativeType: {
      type: String,
      default: "button",
    },
    loading: Boolean,
    disabled: Boolean,
    plain: Boolean,
    autofocus: Boolean,
    round: Boolean,
    circle: Boolean,
  },
  emits: ["click"],
  setup(props, ctx) {
```

```
    const { size, disabled } = toRefs(props);

    const buttonSize = useButtonSize(size);
    const buttonDisabled = useButtonDisabled(disabled);

    const handleClick = (evt) => {
      ctx.emit("click", evt);
    };

    return {
      handleClick,
      buttonSize,
      buttonDisabled,
    };
  },
};

const useButtonSize = (size) => {
  const elFormItem = inject("elFormItem", {});

  const _elFormItemSize = computed(() => {
    return unref(elFormItem.elFormItemSize);
  });

  const buttonSize = computed(() => {
    return (
      size.value ||
      _elFormItemSize.value ||
      (getCurrentInstance().proxy.$ELEMENT || {}).size
    );
  });

  return buttonSize;
};

const useButtonDisabled = (disabled) => {
  const elForm = inject("elForm", {});

  const buttonDisabled = computed(() => {
    return disabled.value || unref(elForm.disabled);
  });

  return buttonDisabled;
};
</script>
```

1.2 集成Babel

```
yarn add babel
yarn add babel-plugin-syntax-dynamic-import
yarn add babel-plugin-syntax-jsx
yarn add babel-preset-env
yarn add @babel/plugin-proposal-optional-chaining
yarn add @babel/preset-env
yarn add @vue/babel-plugin-jsx
```

.babelrc

```
{
  "presets": [["@babel/preset-env", { "targets": { "node": "current" } }]],
  "plugins": [
    "syntax-dynamic-import",
    ["@vue/babel-plugin-jsx"],
    "@babel/plugin-proposal-optional-chaining",
    "@babel/plugin-proposal-nullish-coalescing-operator"
  ],
  "env": {
    "utils": {
      "presets": [
        [
          "env",
          {
            "loose": true,
            "modules": "commonjs",
            "targets": {
              "browsers": ["> 1%", "last 2 versions", "not ie <= 8"]
            }
          }
        ]
      ],
      "plugins": [
        [
          "module-resolver",
          {
            "root": ["element-ui"],
            "alias": {
              "element-ui/src": "element-ui/lib"
            }
          }
        ]
      ]
```

```
    },
    "test": {
      "plugins": ["istanbul"],
      "presets": [["env", { "targets": { "node": "current" } }]]
    },
    "esm": {
      "presets": [["@babel/preset-env", { "modules": false }]]
    }
  }
}
```

1.2 集成VTU

安装依赖

```
yarn add jest
# 此版本这个支持Vue3.0
yarn add vue-jest@5.0.0-alpha.5
yarn add babel-jest
yarn add @vue/compiler-sfc@3.0.2
yarn add @vue/test-utils@next
yarn add typescript
```

jest.config.js

```
module.exports = {
  testEnvironment: 'jsdom', // 默认JSdom
  roots: [
    '<rootDir>/src',
    '<rootDir>/packages',
  ], //
  transform: {
    '^.+\\.vue$': 'vue-jest', // vue单文件
    '^.+\\js$': 'babel-jest' // esm最新语法 import
  },
  moduleFileExtensions: ['vue', 'js', 'json', 'jsx', 'ts', 'tsx', 'node'],
  testMatch: ['**/__tests__/**/*.spec.js'],
  // 别名
  moduleNameMapper: {
    '^element-ui(.*)$': '<rootDir>$1',
    '^main(.*)$': '<rootDir>/src$1'
  }
```

```
}
```

/packages/button/**tests**/Button.spec.js

```
import Button from "../Button.vue";
import { mount } from "@vue/test-utils";

it("content", () => {
  const Comp = {
    template: `<div><Button>默认按钮</Button></div>`,
  };

  const wrapper = mount(Comp, {
    global: {
      components: {
        Button,
      },
    },
  });

  expect(wrapper.findComponent({ name: "ElButton" }).text()).toContain(
    "默认按钮"
  );
});

describe("size", () => {
  it("should have a el-button--mini class when set size prop value equal to
mini", () => {
    const wrapper = mount(Button, {
      props: {
        size: "mini",
      },
    });

    expect(wrapper.classes()).toContain("el-button--mini");
  });

  it("should have a el-button--mini class by elFormItem ", () => {
    const wrapper = mount(Button, {
      global: {
        provide: {
          elFormItem: {
            elFormItemSize: "mini",
          },
```

```
        },
      },
    });

    expect(wrapper.classes()).toContain("el-button--mini");
  });
  it("should have a el-button--mini class by $ELEMENT value ", () => {
    const wrapper = mount(Button, {
      global: {
        config: {
          globalProperties: {
            $ELEMENT: {
              size: "mini",
            },
          },
        },
      },
    });

    expect(wrapper.classes()).toContain("el-button--mini");
  });
});

it("type", () => {
  const wrapper = mount(Button, {
    props: {
      type: "primary",
    },
  });

  expect(wrapper.classes()).toContain("el-button--primary");
});

it("plain", () => {
  const wrapper = mount(Button, {
    props: {
      plain: true,
    },
  });

  expect(wrapper.classes()).toContain("is-plain");
});
it("round", () => {
  const wrapper = mount(Button, {
    props: {
      round: true,
    },
  });
```

```
  expect(wrapper.classes()).toContain("is-round");
});

it("circle", () => {
  const wrapper = mount(Button, {
    props: {
      circle: true,
    },
  });

  expect(wrapper.classes()).toContain("is-circle");
});
it("loading", () => {
  const wrapper = mount(Button, {
    props: {
      loading: true,
    },
  });

  expect(wrapper.find(".el-icon-loading").exists()).toBe(true);
  expect(wrapper.classes()).toContain("is-loading");
});

describe("icon", () => {
  it("should show icon element", () => {
    const wrapper = mount(Button, {
      props: {
        icon: "el-icon-edit",
      },
    });

    expect(wrapper.find(".el-icon-edit").exists()).toBe(true);
  });

  it("should not show icon element when set loading prop equal to true", () =>
{
    const wrapper = mount(Button, {
      props: {
        loading: true,
        icon: "el-icon-edit",
      },
    });

    expect(wrapper.find(".el-icon-edit").exists()).toBe(false);
  });
});

describe("click", () => {
  it("should emit click event ", () => {
```

```
    const wrapper = mount(Button);

    wrapper.trigger("click");

    expect(wrapper.emitted("click")).toBeTruthy();
  });

  it("should not emit click event when disabled equal to true", () => {
    const wrapper = mount(Button, {
      props: {
        disabled: true,
      },
    });

    wrapper.trigger("click");

    expect(wrapper.emitted("click")).toBeFalsy();
  });

  it("should not emit click event when elForm disabled equal to true", () => {
    const wrapper = mount(Button, {
      global: {
        provide: {
          elForm: {
            disabled: true,
          },
        },
      },
    });

    wrapper.trigger("click");

    expect(wrapper.emitted("click")).toBeFalsy();
  });

  it("should not emit click event when loading prop equal to true", () => {
    const wrapper = mount(Button, {
      props: {
        loading: true,
      },
    });

    wrapper.trigger("click");

    expect(wrapper.emitted("click")).toBeFalsy();
  });
});

it("native-type", () => {
```

```
  const wrapper = mount(Button, {
    props: {
      nativeType: "button",
    },
  });

  expect(wrapper.attributes("type")).toBe("button");
});
```

测试

```
"test": "jest --runInBand", # 序列化执行
```

1.4 样式打包

```
yarn config set registry https://registry.npm.taobao.org/

npm i gulp --sass_binary_site=https://npm.taobao.org/mirrors/node-sass/
// 制定镜像安装
npm i node-sass --sass_binary_site=https://npm.taobao.org/mirrors/node-sass/
yarn add gulp
yarn add gulp-autoprefixer --ignore-scripts
yarn add gulp-sass --ignore-scripts
yarn add gulp-cssmin --ignore-scripts
yarn add node-sass

# cp-cli
yarn add cp-cli
yarn add tslib
```

/bin/gen-cssfile

package.json

```
"build:theme": "gulp build --gulpfile packages/theme-chalk/gulpfile.js && cp-
cli packages/theme-chalk/lib lib/theme-chalk",
```

1.5 Rollup打包

> https://www.rollupjs.com/ Rollup中文网
>
> https://juejin.im/post/6844903731343933453 使用 rollup 打包 JS

```
yarn add rollup
yarn add rollup-plugin-peer-deps-external
yarn add rollup-plugin-scss
yarn add rollup-plugin-terser
yarn add rollup-plugin-vue
yarn add @rollup/plugin-node-resolve
yarn add @rollup/plugin-commonjs
yarn add @rollup/plugin-json
yarn add @rollup/plugin-replace
yarn add @rollup/plugin-babel
yarn add rollup-plugin-vue
```

Package.json

```
"build:next": "rollup -c",
```

//

```
import pkg from './package.json'
// 等 rollup-plugin-vue 发版后在切换官方版
// 暂时先用本地的 rollup-plugin-vue
// 修复了 render 函数的编译问题，但是还没发版
// import vuePlugin from 'rollup-plugin-vue'
const vuePlugin = require('./rollup-plugin-vue/index')
import scss from 'rollup-plugin-scss'
import peerDepsExternal from 'rollup-plugin-peer-deps-external'
import resolve from '@rollup/plugin-node-resolve'
import commonjs from '@rollup/plugin-commonjs'
import json from '@rollup/plugin-json'
import replace from '@rollup/plugin-replace'
import babel from '@rollup/plugin-babel'
import { terser } from 'rollup-plugin-terser'

const name = 'Element3'
```

```
const createBanner = () => {
  return `/*!
  * ${pkg.name} v${pkg.version}
  * (c) ${new Date().getFullYear()} kkb
  * @license MIT
  */`
}

const createBaseConfig = () => {
  return {
    input: 'src/entry.js',
    external: ['vue'],
    plugins: [
      peerDepsExternal(),
      babel(),
      resolve({
        extensions: ['.vue', '.jsx']
      }),
      commonjs(),
      json(),
      vuePlugin({
        css: true
      }),
      scss()
    ],
    output: {
      sourcemap: false,
      banner: createBanner(),
      externalLiveBindings: false,
      globals: {
        vue: 'Vue'
      }
    }
  }
}

function mergeConfig(baseConfig, configB) {
  const config = Object.assign({}, baseConfig)
  // plugin
  if (configB.plugins) {
    baseConfig.plugins.push(...configB.plugins)
  }

  // output
  config.output = Object.assign({}, baseConfig.output, configB.output)

  return config
}
```

```
function createFileName(formatName) {
  return `dist/element3-ui.${formatName}.js`
}

// es-bundle
const esBundleConfig = {
  plugins: [
    replace({
      __DEV__: `(process.env.NODE_ENV !== 'production')`
    })
  ],
  output: {
    file: createFileName('esm-bundler'),
    format: 'es'
  }
}

// es-browser
const esBrowserConfig = {
  plugins: [
    replace({
      __DEV__: true
    })
  ],
  output: {
    file: createFileName('esm-browser'),
    format: 'es'
  }
}

// es-browser.prod
const esBrowserProdConfig = {
  plugins: [
    terser(),
    replace({
      __DEV__: false
    })
  ],
  output: {
    file: createFileName('esm-browser.prod'),
    format: 'es'
  }
}

// cjs
const cjsConfig = {
  plugins: [
    replace({
```

```
      __DEV__: true
    })
  ],
  output: {
    file: createFileName('cjs'),
    format: 'cjs'
  }
}
// cjs.prod
const cjsProdConfig = {
  plugins: [
    terser(),
    replace({
      __DEV__: false
    })
  ],
  output: {
    file: createFileName('cjs.prod'),
    format: 'cjs'
  }
}

// global
const globalConfig = {
  plugins: [
    replace({
      __DEV__: true,
      'process.env.NODE_ENV': true
    })
  ],
  output: {
    file: createFileName('global'),
    format: 'iife',
    name
  }
}
// global.prod
const globalProdConfig = {
  plugins: [
    terser(),
    replace({
      __DEV__: false
    })
  ],
  output: {
    file: createFileName('global.prod'),
    format: 'iife',
    name
  }
```

```
}

const formatConfigs = [
  esBundleConfig,
  esBrowserProdConfig,
  esBrowserConfig,
  cjsConfig,
  cjsProdConfig,
  globalConfig,
  globalProdConfig
]

function createPackageConfigs() {
  return formatConfigs.map((formatConfig) => {
    return mergeConfig(createBaseConfig(), formatConfig)
  })
}

export default createPackageConfigs()
```

1.6 编写Entry入口

/packages/button/index.js

```
import ElButton from "./Button.vue";

/* istanbul ignore next */
ElButton.install = function (app) {
  app.component(ElButton.name, ElButton);
};

export default ElButton;
```

/src/entry.js

```
// 用于构建时的入口
// Basic
import ElButton from "../packages/button";
import { version } from "../package.json";

const components = [ElButton];
```

```
const install = (app, opts = {}) => {
  app.use(setupGlobalOptions(opts));

  components.forEach((component) => {
    app.use(component);
  });
};

const setupGlobalOptions = (opts = {}) => {
  return (app) => {
    app.config.globalProperties.$ELEMENT = {
      size: opts.size || "",
      zIndex: opts.zIndex || 2000,
    };
  };
};

const elementUI = {
  version,
  install,
};

export { ElButton, install };

export default elementUI;
```

1.7 验证

/example/button.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>Document</title>
    <script src="/node_modules/vue/dist/vue.global.js"></script>
    <script src="/dist/element3-ui.global.js"></script>
    <link href="/lib/theme-chalk/index.css" rel="stylesheet">
    <style ></style>
  </head>

  <body>
    <div id="app"></div>
    <script>
```

```
      const { createApp, reactive, computed, watchEffect } = Vue;
      const { ElButton } = Element3;
      console.log("ElButton", Element3);

      const MyComponent = {
        template: `
                <el-button icon="el-icon-search" @click="click">{{
state.message }}</el-button>
            `,
        setup() {
          const state = reactive({
            message: "Hello Vue 3!!",
          });
          watchEffect(() => {
            console.log("state change ", state.message);
          });
          function click() {
            state.message = state.message.split("").reverse().join("");
          }
          return {
            state,
            click,
          };
        },
      };
      createApp(MyComponent)
        .use(Element3)
        // .use(ElButton)
        .mount("#app")

    </script>
  </body>
</html>
```

2. 开发规范

> - JS代码规范
>   - airbnb-中文版
>   - standard (24.5k star) 中文版
>   - 百度前端编码规范 3.9k
> - CSS代码规范
>   - styleguide 2.3k

    - spec 3.9k

2.1 项目目录结构

```
.
├── build                 # 编译脚本
├── coverage              # 覆盖率报告
├── examples              # 代码范例
├── lib                   # CSS样式 编译后
├── node_modules
├── packages              # 组件代码
├── rollup-plugin-vue
├── scripts               # 脚本 发布、提交信息检查
├── src                   # 通用代码
├── test                  # 测试
└── types                 # TS类型定义
```

2 .2 文件命名规范

```
.
├── button
│   ├── Button.vue              # 组件SFC
│   ├── __tests__
│   │   └── Button.spec.js      # 测试文件
│   └── index.js                # 组件入口
```

2.3 代码样式规范（ESLint）

- JS代码规范
    - airbnb-中文版
    - standard (24.5k star) 中文版
    - 百度前端编码规范 3.9k
- CSS代码规范
    - styleguide 2.3k
    - spec 3.9k

```
# .eslintrc.js
module.exports = {
  root: true,
  env: {
    browser: true,
```

```
    es2020: true,
    node: true,
    jest: true
  },
  globals: {
    ga: true,
    chrome: true,
    __DEV__: true
  },
  extends: [
    'plugin:json/recommended',
    'plugin:vue/vue3-essential',
    'eslint:recommended',
    '@vue/prettier'
  ],
  parserOptions: {
    parser: 'babel-eslint'
  },
  rules: {
    'no-console': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'no-debugger': process.env.NODE_ENV === 'production' ? 'warn' : 'off',
    'prettier/prettier': 'error'
  }
}
```

```
# .eslintignore
src/utils/popper.js
src/utils/date.js
examples/play
*.sh
node_modules
lib
coverage
*.md
*.scss
*.woff
*.ttf
src/index.js
dist
```

```
yarn add eslint
yarn add eslint-formatter-pretty
yarn add eslint-plugin-json
yarn add eslint-plugin-prettier
yarn add eslint-plugin-vue
yarn add @vue/eslint-config-prettier
yarn add babel-eslint
yarn add prettier
```

package.json

```
{
"scripts": {
   "lint": "eslint --no-error-on-unmatched-pattern --ext .vue --ext .js --ext
.jsx packages/**/ src/**/ --fix",
  },
}
```

2.6 Git版本规范

分支管理

一般项目分主分支（master）和其他分支。

当有团队成员要开发新功能或改 BUG 时，就从 master 分支开一个新的分支。例如项目要从客户端渲染改成服务端渲染，就开一个分支叫 ssr，开发完了再合并回 master 分支。

如果改一个 BUG，也可以从 master 分支开一个新分支，并用 BUG 号命名（不过我们小团队嫌麻烦，没这样做，除非有特别大的 BUG）。

Commit规范

- 内容规范

```
<type>(<scope>): <subject>
<BLANK LINE>
<body>
<BLANK LINE>
<footer>
复制代码
```

大致分为三个部分(使用空行分割):

1. 标题行: 必填, 描述主要修改类型和内容
2. 主题内容: 描述为什么修改, 做了什么样的修改, 以及开发的思路等等

3. 页脚注释: 可以写注释，BUG 号链接

- type: commit 的类型
    - feat: 新功能、新特性
    - fix: 修改 bug
    - perf: 更改代码，以提高性能
    - refactor: 代码重构（重构，在不影响代码内部行为、功能下的代码修改）
    - docs: 文档修改
    - style: 代码格式修改, 注意不是 css 修改（例如分号修改）
    - test: 测试用例新增、修改
    - build: 影响项目构建或依赖项修改
    - revert: 恢复上一次提交
    - ci: 持续集成相关文件修改
    - chore: 其他修改（不在上述类型中的修改）
    - release: 发布新版本
    - workflow: 工作流相关文件修改

1. scope: commit 影响的范围, 比如: route, component, utils, build...
2. subject: commit 的概述
3. body: commit 具体修改内容, 可以分为多行.
4. footer: 一些备注, 通常是 BREAKING CHANGE 或修复的 bug 的链接.

示例

fix（修复BUG）

如果修复的这个BUG只影响当前修改的文件，可不加范围。如果影响的范围比较大，要加上范围描述。

例如这次 BUG 修复影响到全局，可以加个 global。如果影响的是某个目录或某个功能，可以加上该目录的路径，或者对应的功能名称。

```
// 示例1
fix(global):修复checkbox不能复选的问题
// 示例2 下面圆括号里的 common 为通用管理的名称
fix(common): 修复字体过小的BUG，将通用管理下所有页面的默认字体大小修改为 14px
// 示例3
fix: value.length -> values.length
复制代码
```

feat（添加新功能或新页面）

```
feat：添加网站主页静态页面

这是一个示例，假设对点检任务静态页面进行了一些描述。

这里是备注，可以是放BUG链接或者一些重要性的东西。
复制代码
```

chore（其他修改）

chore 的中文翻译为日常事务、例行工作，顾名思义，即不在其他 commit 类型中的修改，都可以用 chore 表示。

```
chore：将表格中的查看详情改为详情
复制代码
```

其他类型的 commit 和上面三个示例差不多，就不说了。

自动化提交验证

验证 git commit 规范，主要通过 git 的 `pre-commit` 钩子函数来进行。当然，你还需要下载一个辅助工具来帮助你进行验证。

下载辅助工具

```
npm i -D husky
```

在 `package.json` 加上下面的代码

```
"husky": {
  "hooks": {
    "pre-commit": "npm run lint",
    "commit-msg": "node script/verify-commit.js",
    "pre-push": "npm test"
  }
}
```

然后在你项目根目录下新建一个文件夹 `script`，并在下面新建一个文件 `verify-commit.js`，输入以下代码：

```
const msgPath = process.env.HUSKY_GIT_PARAMS
const msg = require('fs')
.readFileSync(msgPath, 'utf-8')
.trim()
```

```
const commitRE =
/^(feat|fix|docs|style|refactor|perf|test|workflow|build|ci|chore|release|workf
low)(\(.+\))?: .{1,50}/

if (!commitRE.test(msg)) {
    console.log()
    console.error(`
        不合法的 commit 消息格式。
        请查看 git commit 提交规范：https://github.com/woai3c/Front-end-
articles/blob/master/git%20commit%20style.md
    `)

    process.exit(1)
}
```
复制代码

现在来解释下各个钩子的含义：

1. `"pre-commit": "npm run lint"`，在 `git commit` 前执行 `npm run lint` 检查代码格式。
2. `"commit-msg": "node script/verify-commit.js"`，在 `git commit` 时执行脚本 `verify-commit.js` 验证 commit 消息。如果不符合脚本中定义的格式，将会报错。
3. `"pre-push": "npm test"`，在你执行 `git push` 将代码推送到远程仓库前，执行 `npm test` 进行测试。如果测试失败，将不会执行这次推送。

/scripts/verifyCommit.js

```
// Invoked on the commit-msg git hook by yorkie.

const chalk = require('chalk')
const msgPath = process.env.GIT_PARAMS
const msg = require('fs').readFileSync(msgPath, 'utf-8').trim()

const commitRE = /^(revert: )?
(feat|fix|docs|dx|style|refactor|perf|test|workflow|build|ci|chore|types|wip|re
lease)(\(.+\))?(.{1,10})?: .{1,50}/
const mergeRe = /^(Merge pull request|Merge branch)/

if (!commitRE.test(msg)) {
  if (!mergeRe.test(msg)) {
    console.log(msg)
    console.error(
      `  ${chalk.bgRed.white(' ERROR ')} ${chalk.red(
        `invalid commit message format.`
      )}\n\n` +
        chalk.red(
          `  Proper commit message format is required for automated changelog
generation. Examples:\n\n`
        ) +
```

```
        `    ${chalk.green(`feat(compiler): add 'comments' option`)}\n` +
        `    ${chalk.green(
          `fix(v-model): handle events on blur (close #28)`
        )}\n\n` +
        chalk.red(
          `  See https://github.com/vuejs/vue-next/blob/master/.github/commit-
convention.md for more details.\n`
        )
    )
    process.exit(1)
  }
}
```

3. 自动化

3.1 文档自动化

StoryBook

3.2 规范检查

```
yarn add husky@4.3.0
```

.huskyrc

```
{
    "hooks": {
        "pre-commit": "npm run lint",
        "commit-msg": "node scripts/verifyCommit.js",
        "pre-push": "npm run test"
  },
}
```

3.4 回归测试

GitHub Action

```
.github/workflows/main.yml
```

3.3 持续集成CI

Travis CI 提供的是持续集成服务，它仅支持 Github，不支持其他代码托管。它需要绑定 Github 上面的项目，还需要该项目含有构建或者测试脚本。只要有新的代码，就会自动抓取。然后，提供一个虚拟机环境，执行测试，完成构建，还能部署到服务器。只要代码有变更，就自动运行构建和测试，反馈运行结果。确保符合预期以后，再将新代码集成到主干。

这个项目需要Travis在提交后自动进行测试并且向codecov提供测试报告。

- 测试
- 报告分析

登录TravicCI网站

登录https://www.travis-ci.org/网站

使用github账号登录系统

配置.travis.yml

运行自动化测试框架

```
language: node_js                # 项目语言，node 项目就按照这种写法就OK了
node_js:
- 13.2.0       # 项目环境
cache:         # 缓存 node_js 依赖，提升第二次构建的效率
  directories:
  - node_modules
test:
  - npm run test # 运行自动测试框架
```

> 参考教程：Travis CI Tutorial

上传配置到github

启动持续集成

通过github账号登录travis

获取持续集成通过徽标

将上面 URL 中的 {GitHub 用户名} 和 {项目名称} 替换为自己项目的即可，最后可以将集成完成后的 markdown 代码贴在自己的项目上

```
http://img.shields.io/travis/{GitHub 用户名}/{项目名称}.svg
复制代码
```

####

3.5 持续交付CD - 上传Npm库

创建发布脚本

publish.sh

```
#!/usr/bin/env bash
npm config get registry # 检查仓库镜像库
npm config set registry=http://registry.npmjs.org
echo '请进行登录相关操作：'
npm login # 登陆
echo "-------publishing-------"
npm publish # 发布
npm config set registry=https://registry.npm.taobao.org # 设置为淘宝镜像
echo "发布完成"
exit
```

执行发布

```
./publish.sh
复制代码
```

填入github用户名密码后

3.7 覆盖率测试Codecov

Codecov是一个开源的测试结果展示平台，将测试结果可视化。Github上许多开源项目都使用了Codecov来展示单测结果。Codecov跟Travis CI一样都支持Github账号登录，同样会同步Github中的项目。

```
yarn add codecov
```

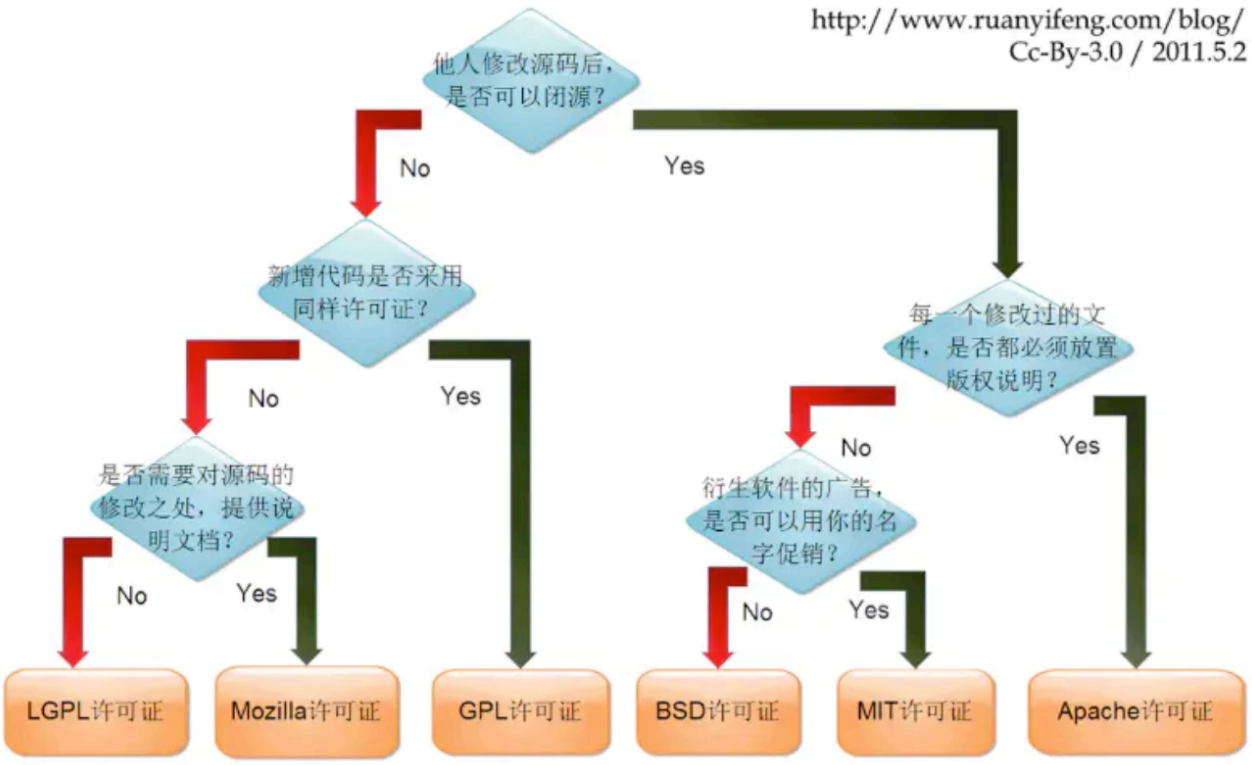
```
"scripts": {
...,
"codecov": "codecov"
}
```

4. 其他

4.1 标准的README文档

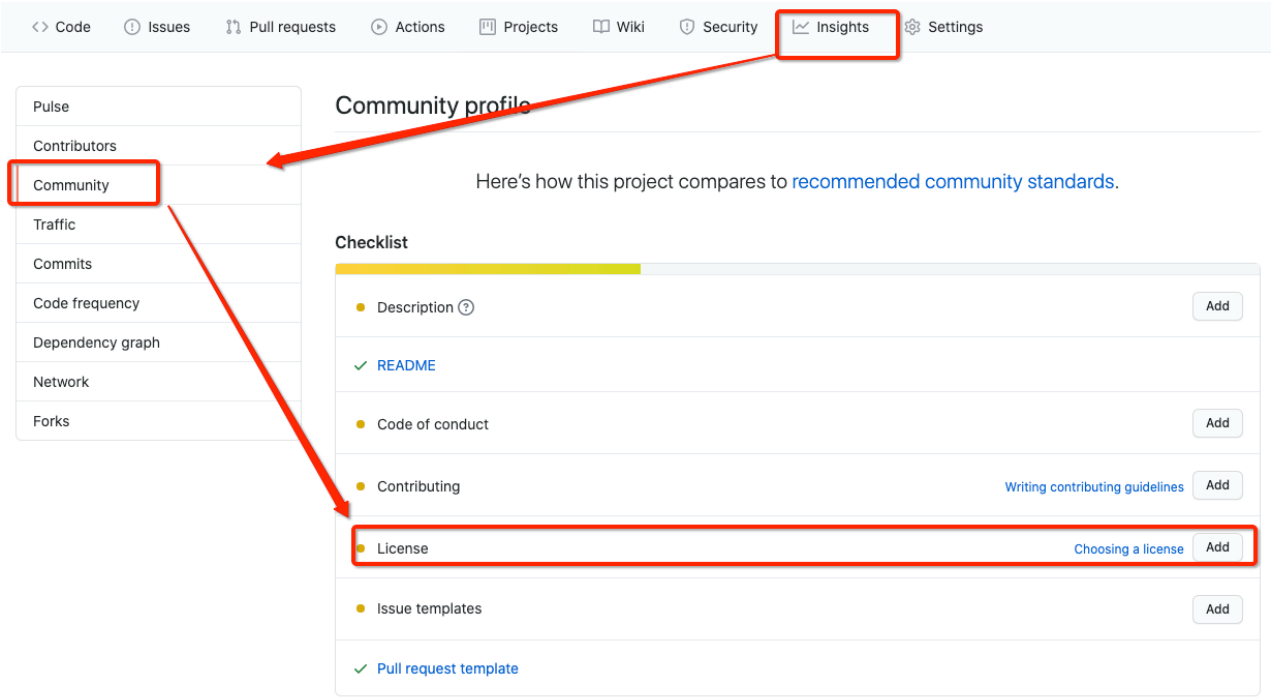
4.2 开源许可证

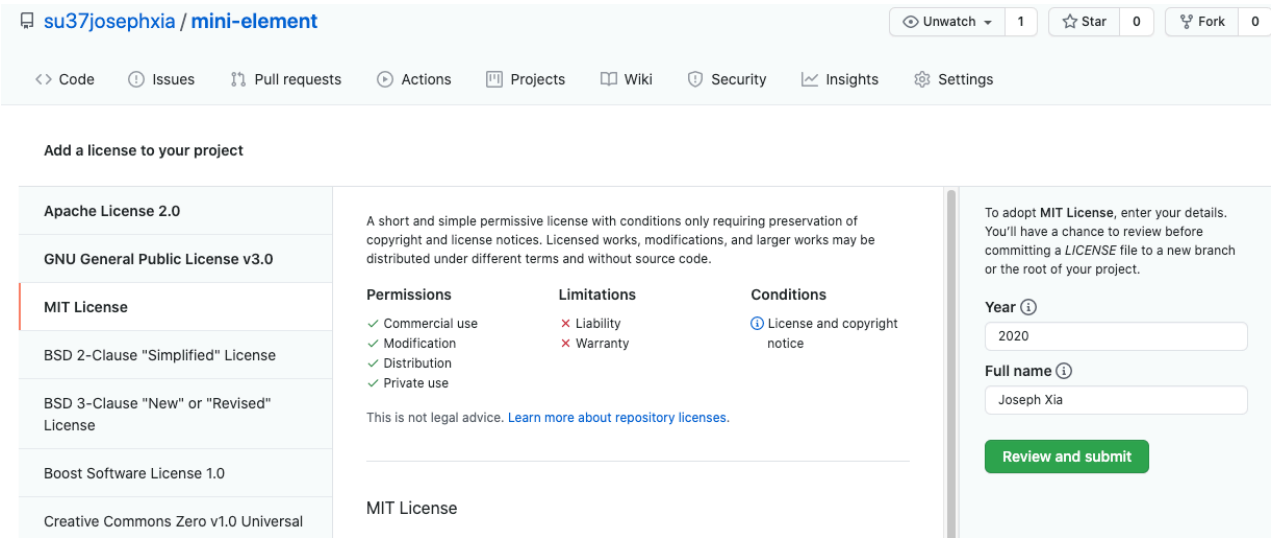
每个开源项目都需要配置一份合适的开源许可证来告知所有浏览过我们的项目的用户他们拥有哪些权限，具体许可证的选取可以参照阮一峰前辈绘制的这张图表：



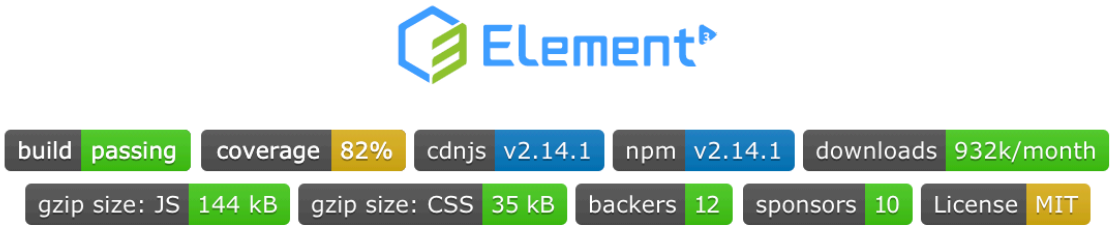
那我们又该怎样为我们的项目添加许可证了？其实 Github 已经为我们提供了非常简便的可视化操作：我们平时在逛 github 网站的时候，发现不少项目都在 README.md 中添加徽标，对项目进行标记和说明，这些小图标给项目增色不少，不仅简单美观，而且还包含清晰易懂的信息。

1. 打开我们的开源项目并切换至 **Insights** 面板
2. 点击 Community 标签
3. 如果您的项目没有添加 License，在 **Checklist** 里会提示您添加许可证，点击 **Add** 按钮就进入可视化操作流程了





4.3 申请开源徽标 (Badge)



Github 徽章 https://docs.github.com/cn/free-pro-team@latest/actions/managing-workflow-runs/adding-a-workflow-status-badge

三、附录

3.3 Vue-cli插件开发

> https://www.cnblogs.com/amysu/p/11539852.html vue-cli插件集合
>
> rollup与webpack对比 https://www.jianshu.com/p/60070a6d7631

3.4 Rollup与Webpack的对比

1. 范例

使用Rollup的开源项目：

- vue
- vuex
- vue-router
- Vue，Ember，Preact，D3，Three.js，Moment

使用webpack的项目：

- 饿了么UI
- mint-ui
- vue脚手架项目

Rollup必选

```
- ES转换 import -> require 无需babel
- 模块解析
- Tree-sharking 静态分析剪裁
```

Webpack

```
- 涉及到静态资源 css、html
- 复杂代码拆分合并
```

3.5 lerna

monorepo

优化托管在 Git/NPM 上的多 package 代码库的工作流的一个管理工具