预习

一. 常见的自动化构建工具

什么是构建，构建的目的和本质是什么?

构建的本质?

- 浏览器从一而终的只支持，以 script 标签的形式引入：

HTML | 复制代码

```html
<!-- 方式1 -->
<script src="./static/a.js" ></script>
<script defer src="./static/b.js" ></script>
<script async src="./static/c.js" ></script>

<!-- 方式2,相当于async -->
<script>
  const script = document.createElement('script');
  script.src = './b.js';
  script.type = 'text/javascript';
  script.onload = handleLoad;
  script.onerror = handleError;
  document.head.append(script);

  const loadScript = (src) => {
    return new Promise((resolve, reject) => {
      const script = document.createElement('script');
      script.src = src;
      script.type = 'text/javascript';
      script.onload = resolve;
      script.onerror = reject;
      document.head.append(script);
    })
  }
</script>

<!-- 方式3 -->
<script>var a = 3; var b = 4;</script>

<!-- 方式4 -->
<script>
  import { bar } from './bar.js'
</script>
```

构建工具的介绍

抛开一些 snowpack 等不聊，聊一些现代化的：

babel

关于 Babel 的一些历史：

babel 最开始叫 6to5，顾名思义是 es6 转 es5，但是后来随着 es 标准的演进，有了 es7、es8 等， 6to5 的名字已经不合适了，所以改名为了 babel。

babel 是巴别塔的意思，来自圣经中的典故：

> 当时人类联合起来兴建希望能通往天堂的高塔，为了阻止人类的计划，上帝让人类说不同的语言，使人类相互之间不能沟通，计划因此失败，人类自此各散东西。此事件，为世上出现不同语言和种族提供解释。这座塔就是巴别塔。

babel 的功能

1. 转译 `esnext` `typescript` `flow` 等到目标环境支持的JS：
    a. 并且还可以把目标环境不支持的 api 进行 polyfill；
    b. 在 babel7 中的一个 preset-env，可以指定targets 进行转换，转换也更加的精准，产物更小
        i. core-js
        ii. preset-env 映射的配置。
2. 一些特定用途的代码转换
    a. React -- jsx -- React.createElement / jsx()
    b. taro 的转译
    c. 自动国际化。。。
3. 代码检测
    a. linter

babel 的实战

JavaScript | 复制代码

```
"use strict";

var _module = require("./module");

console.log("abc" + (0, _module.foo)());
"use strict";

Object.defineProperty(exports, "__esModule", {
  value: true
});
exports.foo = void 0;

var foo = function foo() {
  return 30 * 15;
};

exports.foo = foo;


///// .babelrc ////////
{
    "presets":[
        [
            "@babel/preset-env",
            {
                "useBuiltIns": "entry",
                "corejs": 3,
                "targets": {
                    "ie": "11"
                    // "esModule": true
                }
            }

        ]
    ]
}
```

- 从结果上看，babel 不管你是不是浏览器使用了，他只是针对浏览器做一些降级编译；
- 如果要想最后打包出来的东西，完全支持浏览器，需要特定的其他内容。

browserify

browserify 的功能

- 浏览器端的前端打包工具
- 主要用于在浏览器中使用
- 方便模块细分，每个模块自成，通过 `require` 引用其他模块
- 旧时代的产物，勉强支持一些 css ，但是不友好，且年久失修。

browserify 的实战

JavaScript | 复制代码

```
const browserify = require('browserify');
const babelify = require('babelify');
const fs = require('fs');

browserify('./2_bro/src/index.js', {
    basedir: './',
    transform: [['babelify', {presets: ['@babel/preset-env']}]],
    paths: ['src']
})
.bundle()
.pipe(fs.createWriteStream('./2_bro/dist/bundle2.js'))
```

https://bundlers.tooling.report/

由 chrome core team 核心成员以及业内著名开发者打造的构建工具对比平台。

web-vitals

CLS LCP FID

gulp

gulp 的功能

先来看一下 antd 的构建

https://github.com/ant-design/ant-design/blob/master/package.json#L63

由 `antd` 的 `package.json` 可以看到， `antd` 的构建使用了 `antd-tools` ；

https://github.com/ant-design/antd-tools/blob/master/lib/cli/run.js

在看一下 antd 的产物和 package.json

三证齐全：

bundle.js

gulp 的实战

JavaScript | 复制代码

```
function compile_to_es5() {
    gulp.src([getProjectPath('3_gulp/src/**/*.js')])
    .pipe(babel({
        presets: [
            [
                "@babel/preset-env",
                {
                    "useBuiltIns": "entry",
                    "corejs": 3,
                    "targets": {
                        "ie": "11"
                        // "esModule": true
                    }
                }

            ]
        ]
    }))
    .pipe(concat('all.js'))
    .pipe(gulp.dest('3_gulp/dist'))
}
```

JavaScript | 复制代码

```
const gulp = require('gulp');

function runTask() {
    const taskInstance = gulp.task('compile');
    taskInstance.apply(gulp);
}

require('../gulpfile.js');

runTask();
```

rollup

rollup 的功能

rollup 的实战

vite

vite 的功能

vite 的源码分析

从入口处：`runServe()` 函数开始。