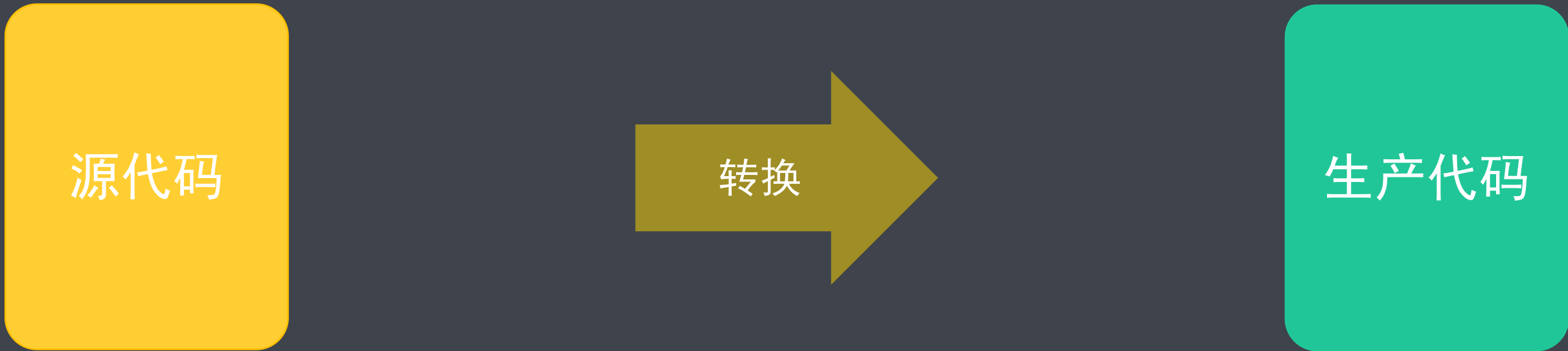


自动化构建

什么是构建

构建过程



为什么构建（构建内容）

- 一些代码需要编译（CSS, JS），保证浏览器的兼容性；
 - 将 Less 或 Sass 转换成 CSS
 - 将 ES6+ 的新语法转成 ES5
- 有些代码需要压缩（CSS, JS, HTML, 图片等）；
 - 压缩之后的代码体积更小，加载更快，节省带宽
- 有些代码需要做格式化校验，统一代码风格；

构建过程



构建初体验

将 less 转成 css



编写

```
@body-bg: #dff;  
@body-color: red;  
  
body {  
  margin: 0 auto;  
  padding: 0 20px;  
  background: @body-bg;  
  color: @body-color;  
}
```

运行

浏览器



编写

```
@body-bg: #dff;  
@body-color: red;  
  
body {  
  margin: 0 auto;  
  padding: 0 20px;  
  background: @body-bg;  
  color: @body-color;  
}
```

构建

```
body {  
  margin: 0 auto;  
  padding: 0 20px;  
  background: #dff;  
  color: red;  
}
```

运行

浏览器

步骤

STEPS

- 安装 less 插件 (npm i less -g)
- 通过 lessc 命令转换 (lessc input.less output.css)

什么是自动化构建

自动化构建是指将手动构建任务，通过命令自动执行的过程。

自动化构建过程



自动化构建过程

一条命令



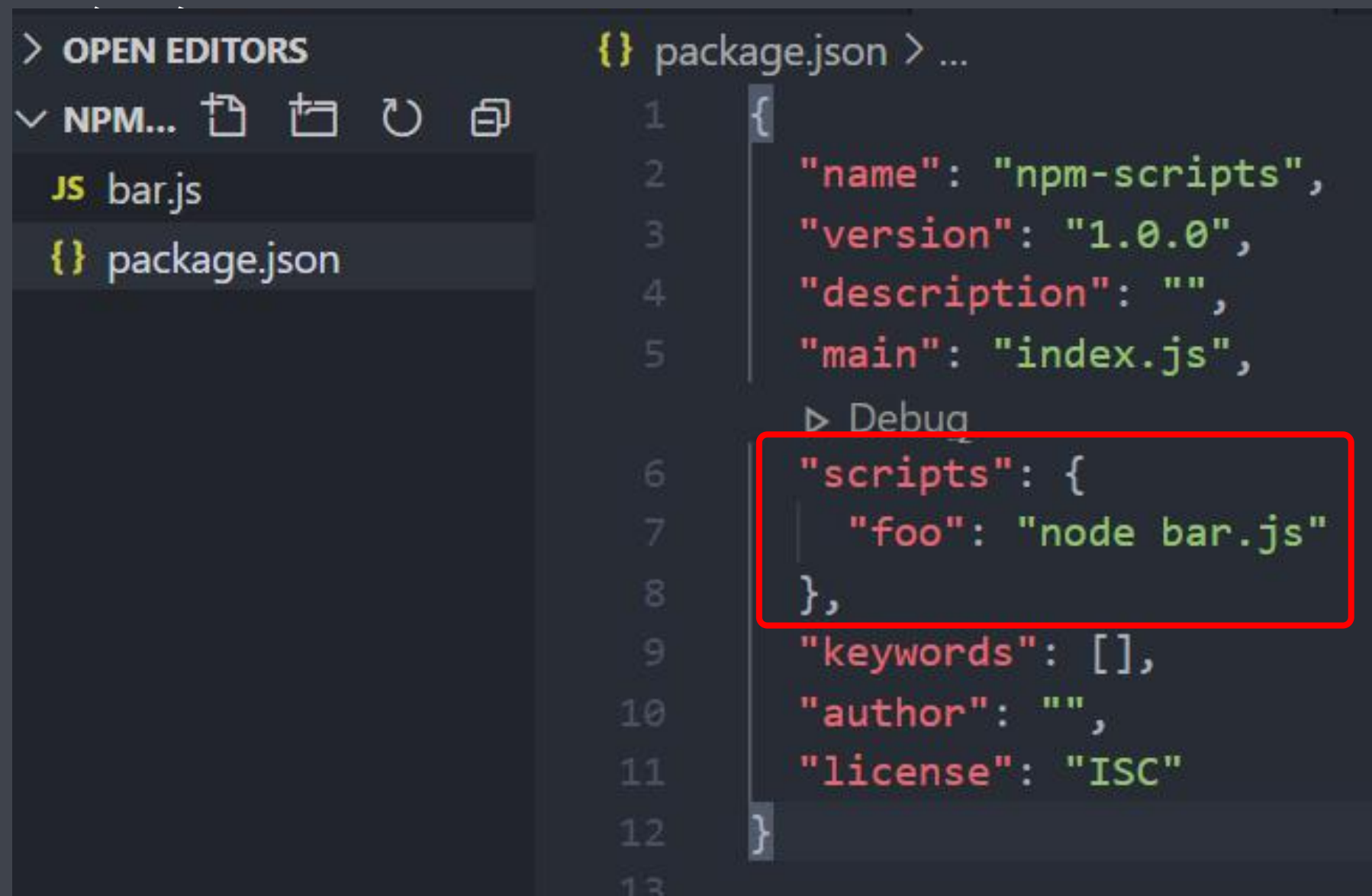
npm scripts

实现自动化构建的最简方式

什么是 npm scripts

什么是 npm scripts

- npm 允许在 package.json 文件中，使用 scripts 字段定义脚本



```
> OPEN EDITORS
NPM... JS bar.js {} package.json
{} package.json > ...
1 {
2   "name": "npm-scripts",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "foo": "node bar.js"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC"
12 }
```



```
$ npm run foo

# 等同于执行
$ node bar.js
```

npm scripts 自定义脚本命令

1. 声明命令
(package.json)

```
“scripts” : {  
    “foo” : “node bar.js”  
}
```

2. 执行命令
(命令行)

```
npm run foo  
等同于  
node bar.js
```


自动化构建样式文件

- 手动:



```
lessc input.less output.css
```

- 自动:



```
# 在 package.json 中定义构建任务  
"scripts": {  
  "style": "lessc input.less output.css"  
}
```

```
# 命令行中执行命令  
npm run style
```

npm scripts 中任务的执行方式

并行 / 串行

npm scripts 命令的执行方式

并行执行 (`parallel`)

任务1 & 任务2

任务之间没有先后顺序，同时执行可以提高执行效率

串行执行 (`series`)

任务1 && 任务2

任务之间有先后顺序，先执行前一个任务，后执行下一个

npm scripts 命令的执行方式

任务1 & 任务2 & 任务3

任务1

任务2

任务3

任务1 && 任务2 && 任务3

任务1

任务2

任务3

自动化构建



& （并行执行）在 Windows 下不起作用

npm-run-all 插件



先在项目中安装

```
npm i npm-run-all -D
```

并行执行: 其中 *p* 是 *parallel* (并行) 的意思

```
npm-run-all -p 脚本1 脚本2 脚本3
```

或简写为

```
run-p 脚本1 脚本2 脚本3
```

串行执行: 其中 *s* 是 *series* (串行) 的意思

```
npm-run-all -s 脚本1 脚本2 脚本3
```

或简写为

```
run-s 脚本1 脚本2 脚本3
```

npm scripts 实践

构建样式文件

构建样式文件

- 将 less 转成 css
 - `npm i less -g`
 - `lessc input.less output.css`
- 压缩 css 文件
 - `npm i minify -g`
 - `minify output.css > output.min.css`

构建样式文件





编写

```
@body-bg: #dff;  
@body-color: red;  
  
body {  
  margin: 0 auto;  
  padding: 0 20px;  
  background: @body-bg;  
  color: @body-color;  
}
```

转换

```
body {  
  margin: 0 auto;  
  padding: 0 20px;  
  background: #dff;  
  color: red;  
}
```

压缩

```
body{margin:0 auto;padding:0 20px;background-color:#dff;color:red}
```

运行

浏览器

构建命令

转换

串行执行

压缩

```
"scripts": {  
  "style": "lessc styles/main.less styles/main.css && minify styles/main.css > styles/main.min.css"  
},
```

步骤

STEPS

- 初始化项目 (`npm init --yes`)
- 添加 scripts 命令 (`less + minify`)
- 执行 scripts 命令 (`npm run 命令`)

npm scripts 实践

构建脚本文件



编写

```
var show_msg = () => {  
  alert('Hello');  
}
```

运行

浏览器



编写

```
var show_msg = () => {  
  alert('Hello');  
}
```

构建

```
function show_msg() {  
  alert('Hello');  
}
```

运行

浏览器

Babel 插件可以将 ES6+ 新语法转成
ES5

ECMAScript 的版本与发布时间

2009

ES5

2015

ES6

2016

ES2016

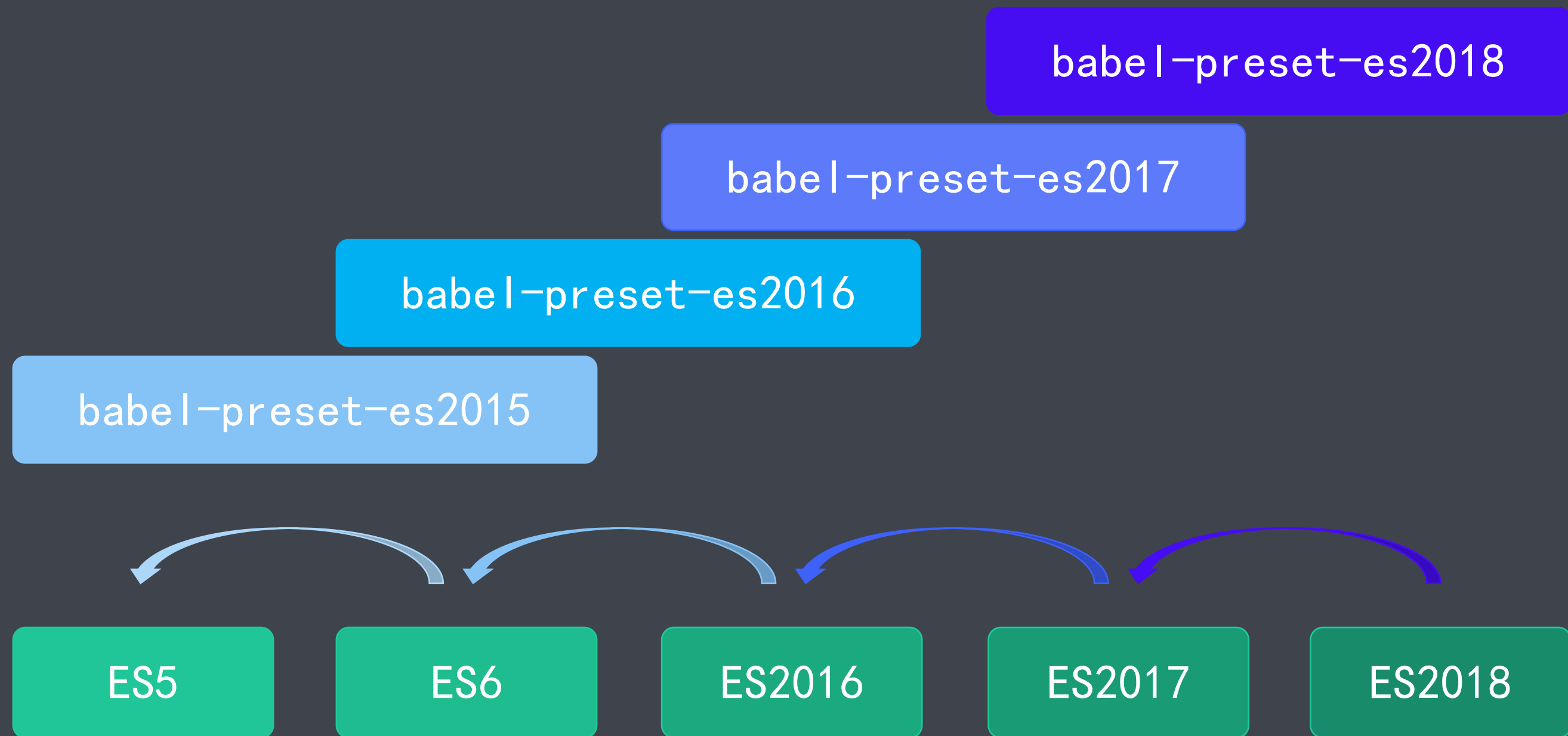
2017

ES2017

2018

ES2018

Babel 转换规则



Babel 转换规则

babel-preset-env



Babel 转换命令

单个文件

```
babel input.js --out-file output.js
```

或:

```
babel input.js -o output.js
```

整个目录

```
babel src --out-dir dist
```

或:

```
babel src -d dist
```

步骤

STEPS

- 初始化项目 (`npm init -yes`)
- 安装 Babel (`npm install -g babel-core babel-cli`)
- 安装转码规则 (`npm install -g babel-preset-env`)
- 配置转换规则 (`.babelrc`)
- 在 `npm scripts` 中添加转换命令 (`babel src -d dist`)
- 执行转换命令

代码格式校验

代码格式校验

- 为什么
 - 不同的工程师，写的代码风格不同
 - 项目代码提交时，需要保持统一的代码格式
- 如何实现（ 通过工具完成代码格式校验 ）
 - 提供编码规范
 - 根据编码规范，自动检查代码

ESLint

对 JavaScript 代码格式进行检查

使用 ESLint

- 初始化项目 (`npm init --yes`)
- 安装 ESLint (`npm i eslint -g`)
- 初始化配置文件 (`eslint --init`)
- 检查 JS 代码格式
 - 单个文件 (`eslint path/filename.js`)
 - 整个目录 (`eslint path/dirname`)

StyleLint

对 CSS 代码格式进行检查

使用 StyleLint

- 初始化项目 (`npm init --yes`)
- 安装 StyleLint (`npm install --global stylelint`)
- 安装检测标准 (`npm install --global stylelint-config-standard`)
- 创建配置文件 (`.stylelintrc.json`)
- 检查 CSS 代码格式
 - 单个文件 (`stylelint path/filename.css`)
 - 整个项目 (`stylelint **/*.cssG`) / O / U

拉勾教育

— 互联网人实战大学 —



下载「拉勾教育App」
获取更多内容