babel 安装与介绍

安装：`npm install -D @babel/cli @babel/core @babel/preset-env`，其中`@babel/preset-env`提供了`es6`转`es5`的功能

`babel` 的工作主要有：语法转换、补齐 API

`babel`默认只做语法转换，但不转换新的 API，如`Promise`等，需要用到`polyfill`

编写配置文件`babel.config.js`（也可以使用`.babelrc`或`.babelrc.js`或写在`package.json`里）：

```
module.exports = {
  presets: ["@babel/env"],
  plugins: [],
};
```

编译执行命令：`npx babel main.js -o compiled.js`

babel 的配置文件

配置文件导出一个对象，包含有4个键值：`presets`、`plugins`、`minified`、`ignore`

第一种写法（推荐）：`babel.config.js`或`.babelrc.js`，可以写`js`从而进行一些逻辑处理

```
module.exports = {
  presets: ["es2015", "react"],
  plugins: ["transform-decorators-legacy"],
};
```

第二种写法：`package.json`

```
{
  "babel": {
    "presets": ["es2015", "react"],
    "plugins": ["transform-decorators-legacy"]
  }
}
```

第三种写法：`.babelrc`

```
{
    "presets": ["es2015", "react"],
```

```
    "plugins": ["transform-decorators-legacy"]
}
```

插件和预设概述

预设其实就是一组插件的集合

插件和预设传参

将插件或预设数组的一项设置为数组，该数组第一项为预设或插件名，第二项为`options`

```
{
  "preset":[["@babel/preset-env",{
    "useBuiltIns":"entry"
  }]]
}
```

最常用的四个预设和一个插件

`@babel/preset-env`

`@babel/preset-flow`

`@babel/preset-react`

`@babel/preset-typescript`

`@babel/plugin-transform-runtime`

插件和预设执行顺序

- 插件比预设先执行
- 插件执行顺序是从前往后执行
- 预设是从后往前执行

预设的发展史

在`babel 6`中，`babel-preset-latest`是`babel-preset-es2016`、`babel-preset-es2016`、`babel-preset-es2017`的集合

在`babel 7`中，`@babel/preset-env`包含了`babel-preset-latest`的功能并进行增强，也就是说一般开发用这一个就够此外

还可使用:

`@babel/preset-flow`: `JavaScript` 代码的静态类型检查器

`@babel/preset-react`：转换`jsx`语法

`@babel/preset-typescript`：支持`typescript`语法

@babel/preset-env

`browserslist`依赖这个插件`

```
module.exports = {
  presets: [
    [
      "@babel/preset-env",
      {
        targets: {
          chrome: "28",
          ie: 11,
        },

        useBuiltIns: "usage", //还有entry、false

        corejs: 3,

        //这样可以使用tree shaking
        modules: false,
      },
    ],
  ],
};
```

targets

`targets`与`package.json`中的`browserslist`功能一样，如果设置了`targets`会覆盖掉`browserslist`，如果都不设置，则会将`es6`语法转换为`es5`语法，一般都配置`browserslist`，这样就可以和`postcss`共用

useBuiltIns

与`polyfill`行为有关 有`3`个值：

- `false` 全部引入（默认值）
- `entry` 根据`targets`或`browserslist`来引入需要的部分
- `usage` 按需求和目标环境加载，相当于是`entry`这个值的优化版，推荐

corejs

只有在`useBuiltIns`不为`false`时生效

- `2` 默认值

- 3 比较新，可以支持flat等数组方法，推荐

modules

是否转换es6模块语法

amd umd systemjs commonjs cjs auto(默认值) false

@babel/plugin-transform-runtime

作用：

- 提供转换时的辅助函数
- 提供 generator的转换
- 提供 API转换，相当于是polyfill（不污染全局变量）

使用@babel/plugin-transform-runtime和使用@babel/preset-env来polyfill的区别在于：前者不会污染全局变量

```
{
  "plugin":[["babel/plugin-transform-runtime",
{
  //引入辅助函数
  "helpers":true,

  /*设置为2或3表示开启`core-js`相关`api`转换功能：这个一般开发`js`库或`npm`包的人使用，使用了这个就不要使用其他polyfill了，设置false为关闭*/
  //还需要安装@babel/runtimeX对应版本
  "corejs":3,

  //转换generator 默认是true
  regenerator:true,

  //使用webpack打包时设置为true => tree shaKing
  useESModules:true,

  //设置@babel/runtime路径规则，保持默认false就好
  absoluteRuntime:false

  //设置@babel/runtime-corejs版本号，可以减少打包代码体积
  version:
}]]
}
```

babel 插件开发

将animal变量转换为dog

```
module.exports = function ({ type: t }) {
  return {
    name: "animalToDog",
    visitor: {
      Identifier(path, state) {
        if (path.node.name === "animal") {
          path.node.name = "dog";
        }
      },
    },
  };
};
```

将let变量转换为var

```
module.exports = function ({ type: t }) {
  return {
    name: "animalToDog",
    visitor: {
      Identifier(path, state) {
        // state.opts是插件的配置项
        if (path.node.kind === "let" && state.opts.ES5 === true) {
          path.node.kind = "var";
        }
      },
    },
  };
};
```