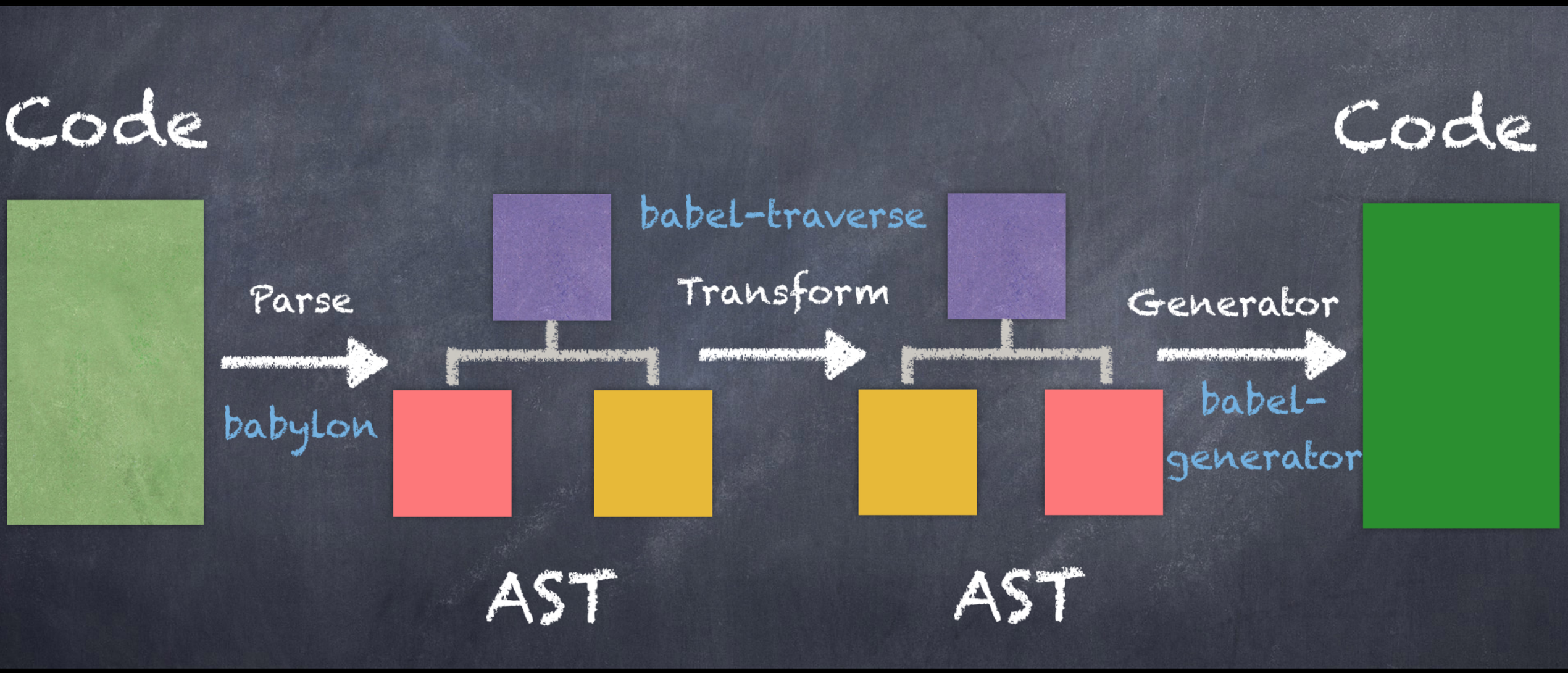
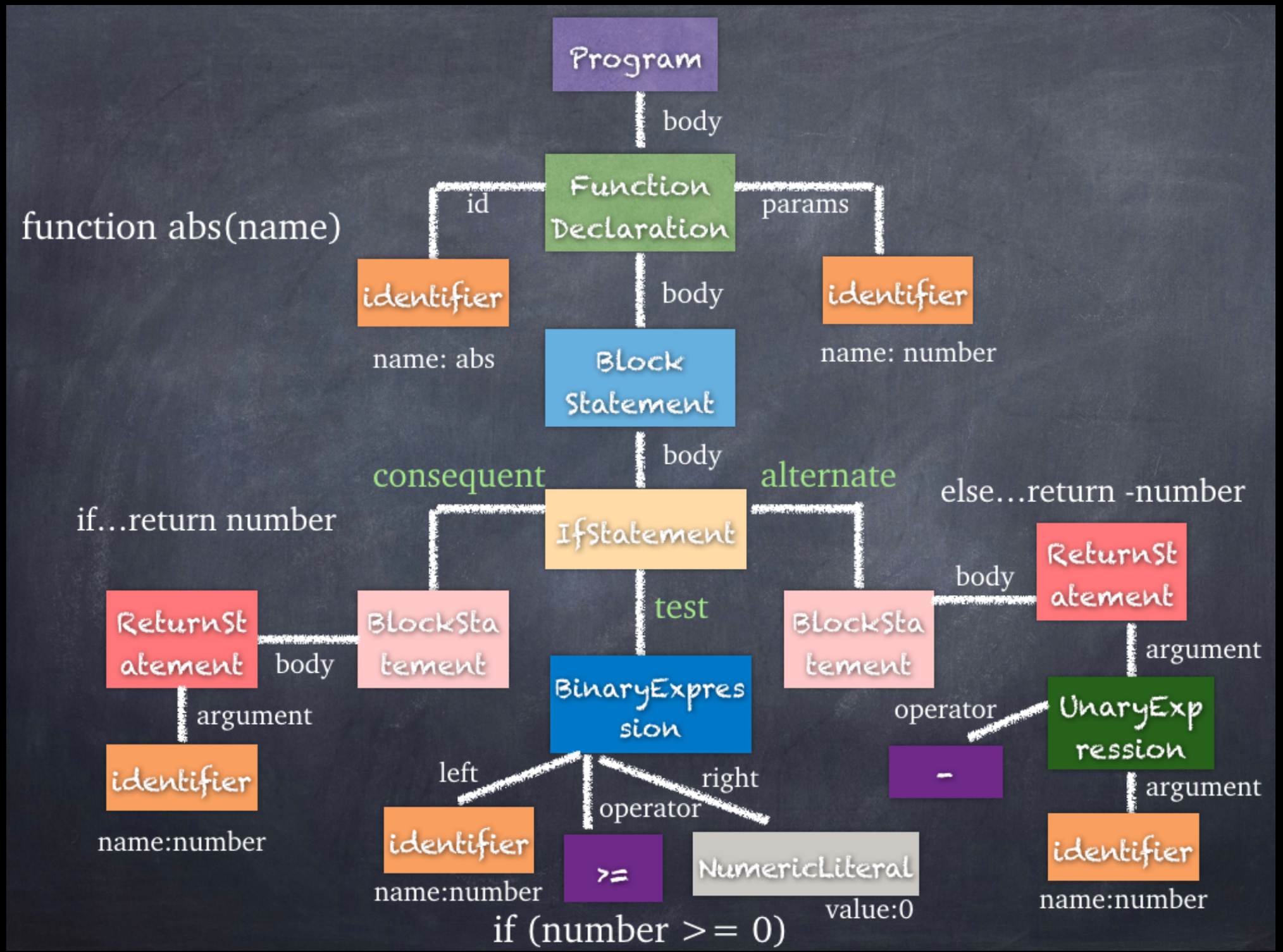
Babel 分享

史拓成


Code
Code
babel-traverse
Parse
Transform
Generator
babylon
babel-
generator
AST
AST

- 解析（Babylon）
    - Babylon 是一个解析器，将 JavaScript 字符串转换为 AST.
- 遍历AST、转换 （babel-traverse）
    - babel-traverse 模块允许你浏览、分析和修改抽象语法树（AST）。
- AST生成代码（babel-generator）
    - 将转换后的抽象语法树（AST）转换为 JavaScript 字符串。

AST抽象语法树


function abs(name)
Program
body
Function Declaration
id
params
identifier
name: abs
body
identifier
name: number
Block Statement
body
consequent
alternate
if…return number
else…return -number
IfStatement
ReturnStatement
body
BlockStatement
test
BlockStatement
ReturnStatement
argument
body
BinaryExpression
operator
UnaryExpression
identifier
name:number
left
operator
right
argument
identifier
>=
NumericLiteral
-
identifier
name:number
value:0
name:number
if (number >= 0)

@babel/parser(Babylon)

- 将代码字符串解析为ast抽象语法树

- babelParser.parse(code, [options])

- babelParser.parseExpression(code, [options])

@babel/traverse

- Babel 使用 babel-traverse 进行树状的遍历，对于 AST 树上的每一个分支我们都会先向下遍历走到尽头，然后向上遍历退出遍历过的节点寻找下一个分支。(Dfs)

- 当Babel处理一个节点时，是以访问者的形式获取节点信息，并进行相关操作，这种方式是通过一个visitor对象来完成的，在visitor对象中定义了对于各种节点的访问函数，这样就可以针对不同的节点做出不同的处理。

- 利用 babel-traverse 这个独立的包对 AST 进行遍历，并解析出整个树的 path，通过挂载的 visitor 读取对应的元信息，这一步叫 set AST 过程

```js
import { declare } from "@babel/helper-plugin-utils";
import syntaxExportDefaultFrom from "@babel/plugin-syntax-export-default-from";
import { types as t } from "@babel/core";

export default declare(api => {
  api.assertVersion(7);

  return {
    name: "proposal-export-default-from",
    inherits: syntaxExportDefaultFrom,

    visitor: {
      ExportNamedDeclaration(path) {      Sebastian McKenzie, 4 years ago • 6.
        const { node, scope } = path;
        const { specifiers } = node;
        if (!t.isExportDefaultSpecifier(specifiers[0])) return;

        const specifier = specifiers.shift();
        const { exported } = specifier;
        const uid = scope.generateUidIdentifier(exported.name);

        const nodes = [
          t.importDeclaration(
            [t.importDefaultSpecifier(uid)],
            t.cloneNode(node.source),
          ),
          t.exportNamedDeclaration(null, [
            t.exportSpecifier(t.cloneNode(uid), exported),
          ]),
        ];

        if (specifiers.length >= 1) {
          nodes.push(node);
        }

        path.replaceWithMultiple(nodes);
      },
    },
  };
});
```

举个🌰

babel-plugin-proposal-export-default-from

babel-generator

- 将ast转换为代码

- const output = generate(ast, { /* options */ }, code);

其他相关模块

- babel-cli
- babel-core
- babel-helpers
- babel-node
- babel-polyfill
- babel-preset-env

Babel-cli

- babel-cli是一个通过命令行对js文件进行换码的工具。

```
[➜  core-js git:(master) babel script.js --out-file build.js
babel:
```

```
[➜  core-js git:(master) babel --plugins transform-es2015-arrow-functions script.
js
babel:
```

@babel/core

- babel.transform(code: string, options?: Object, callback: Function)

- babel.transformFile(filename, options, callback)

- babel.transformFromAst(ast: Object, code?: string, options?: Object, callback: Function): FileNode | null

Babel-loader

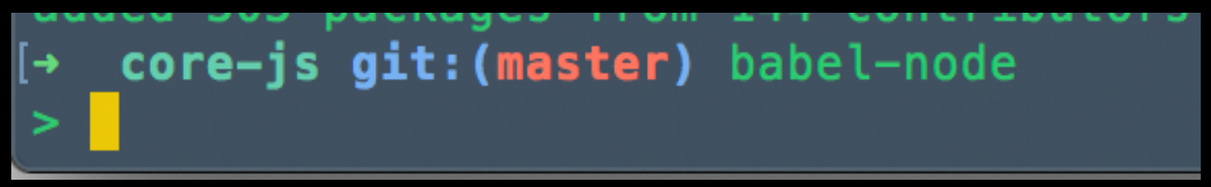
```
var result = void 0;
try {
  result = babel.transform(source, options);
} catch (error) {
  if (forceEnv) restoreBabelEnv(tmpEnv);
  if (error.message && error.codeFrame) {
    var message = error.message;
    var name = void 0;
    var hideStack = void 0;
    if (error instanceof SyntaxError) {
      message = message.replace(STRIP_FILENAME_RE, "");
      name = "SyntaxError";
      hideStack = true;
    } else if (error instanceof TypeError) {
      message = message.replace(STRIP_FILENAME_RE, "");
      hideStack = true;
    }
    throw new BabelLoaderError(name, message, error.codeFrame, hideStack, error);
  } else {
    throw error;
  }
}
```

```
import { buildExternalHelpers, transform } from 'babel-core';

  transform ( code, id ) {
    if ( !filter( id ) ) return null;
    if ( id === HELPERS ) return null;

    const helpers = preflightCheck( options, dirname( id ) );
    const localOpts = assign({ filename: id }, options );

    const transformed = transform( code, localOpts );
    const { usedHelpers } = transformed.metadata;

    if ( usedHelpers.length ) {
      if ( helpers === BUNDLED ) {
        if ( !externalHelpers ) {
          transformed.code += `\n\nimport * as babelHelpers from
        }
      } else if ( helpers === RUNTIME ) {
        if ( !runtimeHelpers ) {
          throw new Error( 'Runtime helpers are not enabled. Eith
        }
      } else {
        usedHelpers.forEach( helper => {
          if ( inlineHelpers[ helper ] ) {
            warnOnce( warn, `The '${helper}' Babel helper is used
          }

          inlineHelpers[ helper ] = true;
        });
      }
    }

    return {
      code: transformed.code,
      map: transformed.map
    };
  }
```

rollup-plugin-babel

babel-node

- Babel 命令行工具



babel-polyfill

- 使用babel-node时，这个polyfill会自动加载。这里要注意的是babel-polyfill是一次性引入你的项目中的，并且同项目代码一起编译到生产环境。而且会污染全局变量。像Map，Array.prototype.find这些就存在于全局空间中。

babel-polyfill

- Core-js
- Regenerator-runtime

babel-runtime

- babel-runtime和babel-plugin-transform-runtime

- babel-runtime不会污染全局空间和内置对象原型。

Babel-runtime

```
require('babel-runtime/core-js/promise');
```

在实际项目开发过程中，我们往往会写很多新的es6 api，每次都要手动引入相应的包比较麻烦，维护起来也不方便，每个文件重复引入也造成代码的臃肿。

babel-plugin-transform-runtime

- 分析我们的 ast 中，是否有引用 babel-rumtime 中的垫片（通过映射关系），如果有，就会在当前模块顶部插入我们需要的垫片

babel-preset-env

- babel-preset-env 能根据当前的运行环境，自动确定你需要的 plugins 和 polyfills。通过各个 es标准 feature 在不同浏览器以及 node 版本的支持情况，再去维护一个 feature 跟 plugins 之间的映射关系，最终确定需要的 plugins。