前端开发项目报告

命令行打包工具

学生姓名：潘越
实习导师：常运涛

2021 年 9 月 2 日

目录

1 项目介绍

本项目实现一个简单的命令行编译工具，能够将 es6 代码转换成 es5 代码，且能够分析文件内部的依赖，例如 a.js 引用了 b.js。最后能够打包成一个 js, 且能够在浏览器跑通。

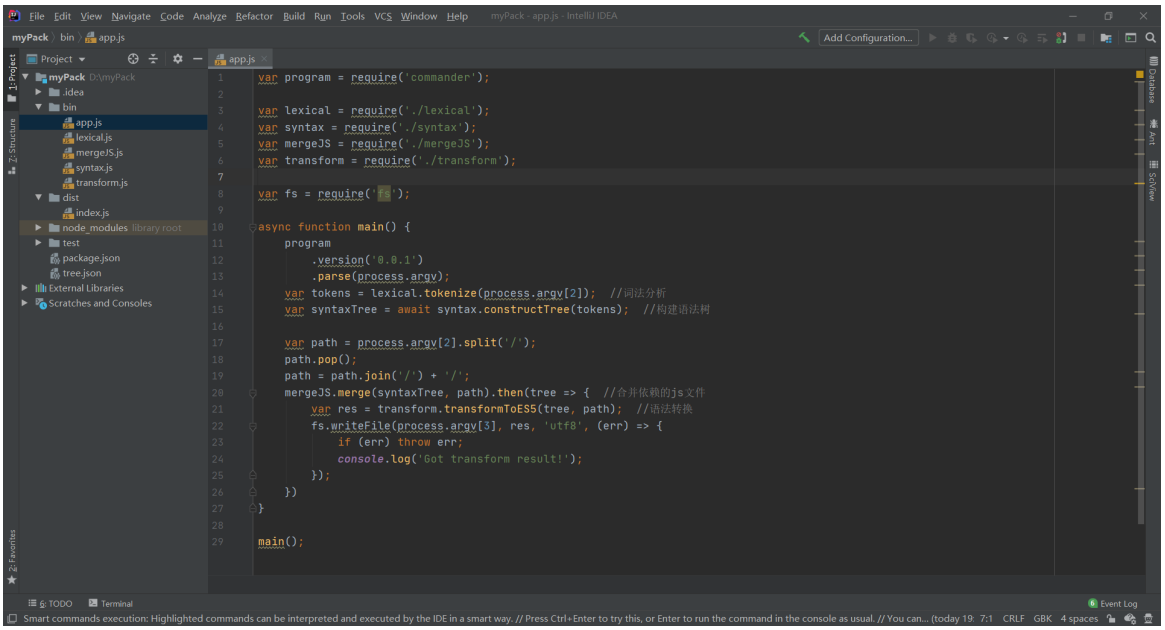
本次开发参考了 commander 命令行工具，node 基本 api 和 babel-core 核心实现的转码，下面对此进行详细的介绍以及使用的一些心得体会。

2 开发过程以及相应的技术介绍

完整项目见 https://github.com/pyyybf/2021-winter-study/tree/main/myPack。

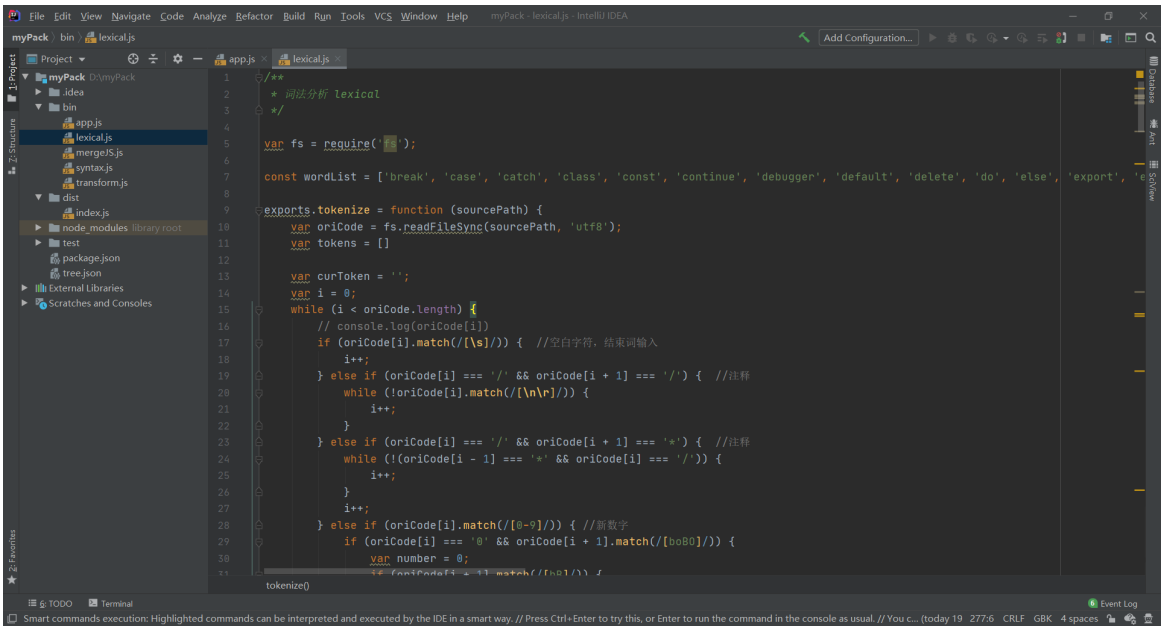
2.1 主函数

完整代码见项目内 bin/app.js。调用 bin 目录下其他 js 文件中的方法，转换代码。



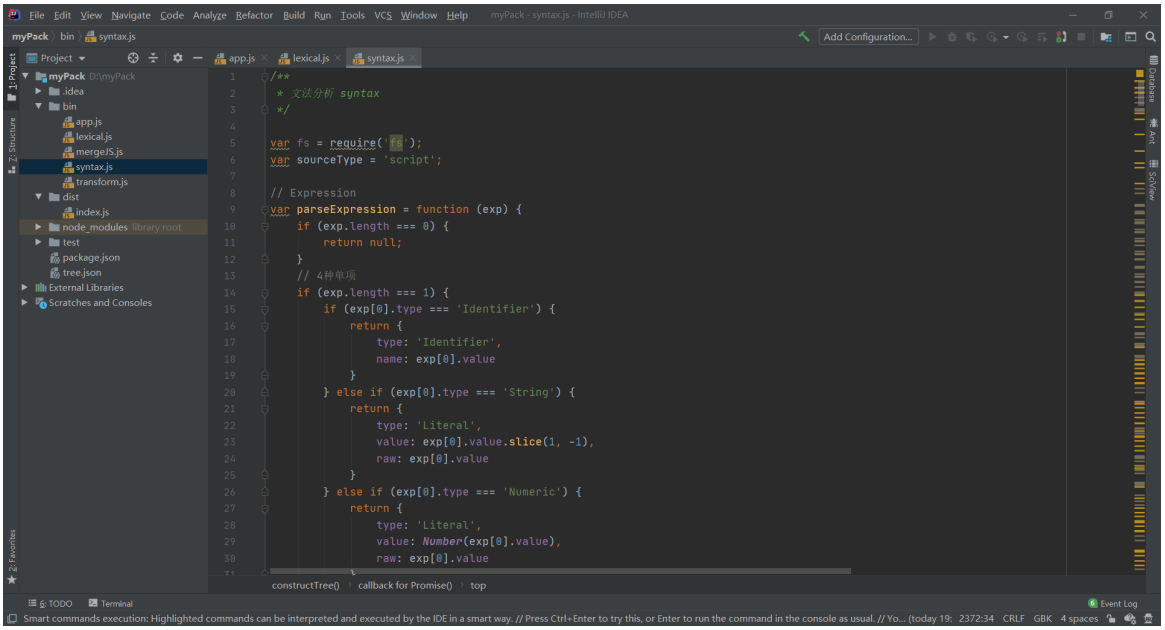
2.2 分词

按顺序读入字符，并按规则分为 token，设置对应 type。完整代码见项目内 bin/lexical.js。



2.3 构建语法树

读取上一步所得 token 数组，构建语法树。完整代码见项目内 bin/syntax.js。



以下为语法树节点说明：

名称	属性	示例
Program	body：Program 中包含的 statement 数组 sourceType：Program 类型，为 script 或 module	
ImportDeclaration	specifiers：import 后的标识符，为 ImportSpecifier 和 ImportDefaultSpecifier 组成的数组 source：import 的的内容，为 Literal	import math from ’./math’; import {mySub as sub} from ’./math’;

ImportSpecifier	local：Identifier，为引入后的标识符 imported：在依赖文件中的标识符	import {myAdd, mySub as sub} from './math';
ImportDefaultSpecifier	local：Identifier，为引入后的标识符	import math from './math';
VariableDeclaration	declarations：声明的变量 VariableDeclarator 数组	var a, b = 1;
VariableDeclarator	id：变量名称，为 Identifier、ArrayPattern 或 ObjectPattern init：初始化值	var [a, b] = [0, 1]; var {c, d} = {c: 0, d: 1};
ArrayPattern	elements：Array 中元素数组，为 Identifier、ArrayPattern 或 ObjectPattern	var [a, b] = [0, 1];
ObjectPattern	properties：包含的属性 Property 数组	var {c, d} = {c: 0, d: 1};

Property	key：属性名称，为 Identifier
	computed：为 boolean
	value：属性值
	kind：init、get 或 set
	method：是否为方法，为 boolean
	shorthand：是否省略初始化，为 boolean
		var obj = {
		  prop: 0,
		  toString(){}
		};

ClassDeclaration	id：类名 superClass：父类名称，为 Identifier body：类定义	class A extends B {}
ClassBody	body：MethodDefinition 数组	class A {}

MethodDefinition	key：方法名
	computed：为 boolean
	value：方法内容
	kind：method 或 constructor
	static：是否静态
		constructor(a) {
		  super(a);
		}
		toString() {
		  return 'string';
		}

FunctionDeclaration	id：函数名
	params：参数数组
	body：函数体
	generator：为 boolean
	expression：是否直接返回表达式
	async：是否异步
		function add(a, b) {
		  return a + b;
		}

Identifier	name：变量名称	i
Literal	value：值 raw：可以直接输出到目标文件的值	true 'test'
ExpressionStatement	expression：包含的表达式	a = 0;
BreakStatement		break;
ContinueStatement		continue;
ReturnStatement	argument：返回的内容	return 0;
BlockStatement	body：Statement 数组	{ a++; }

WhileStatement	test：判断是否进入循环的表达式
	body：循环体
		while (a < 10) {
		  a++;
		}

DoWhileStatement	test：判断是否进入循环的表达式
	body：循环体
		do {
		a++;
		} while (a < 10)

IfStatement	test：条件表达式
	consequent：test 表达式结果为真时的函数块
	alternate：test 表达式结果为假时的函数块
		if (a > 0) {
		a++;
		}

ForStatement	init：初始化循环中用到的变量
	test：判断是否进入循环的条件表达式
	update：循环结束后的操作
	body：循环体
		for (let i = 0; i < 10; i++) {
		sum = sum + i;
		}

ForOfStatement	left：当前遍历到的数组元素
	right：遍历的数组
	body：循环体
		for (var prop of obj) {
		console.log(prop);
		}

ForInStatement	left：当前遍历到的索引
	right：遍历的对象
	body：循环体
		for (var e in elements) {
		console.log(e);
		}

CallExpression	callee：调用函数 arguments：参数数组	mySub(2, 1);
MemberExpression	computed：通过. 或者 [] 引用 object：对象 property：属性	obj.prop obj['prop']

FunctionExpression	id：函数名
	params：参数
	body：函数体
	generator：为 boolean
	expression：是否直接返回表达式
	async：是否异步
		function (a, b) {
		return a + b;
		}

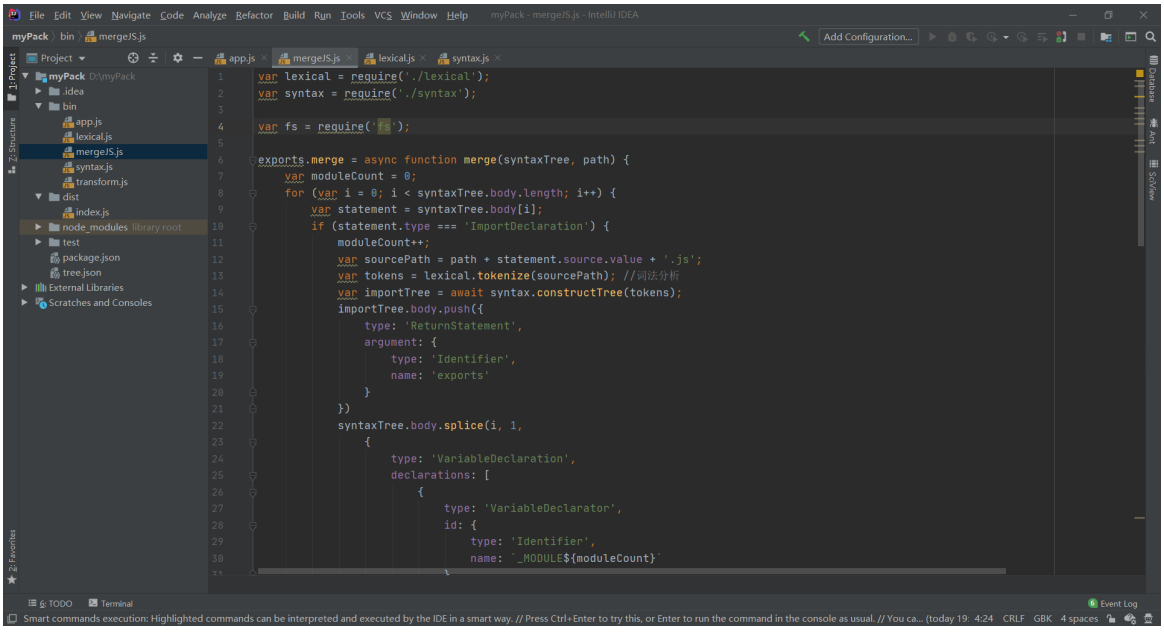
ArrowFunctionExpression	id：函数名
	params：参数
	body：函数体
	generator：为 boolean
	expression：是否直接返回表达式
	async：是否异步
		(a, b) => {
		return a + b;
		}
		(a, b) => a + b

BinaryExpression	operator：运算符号，包含 +、-等 left：左表达式 right：右表达式	a + b
UnaryExpression	operator：运算符号，包含! 等 argument：参与运算的表达式 prefix：符号是否在表达式之前	!flag
UpdateExpression	operator：运算符号，包含! 等 argument：参与运算的表达式 prefix：符号是否在表达式之前	a++

ThisExpression		this
SequenceExpression	expressions：表达式数组	a = 0, b = 1
AssignmentExpression	operator：= left：左表达式 right：右表达式	
ConditionalExpression	test：条件表达式 consequent：test 表达式结果为真时返回的表达式 alternate：test 表达式结果为假时返回的表达式	a > 0 ? 1 : 0
ArrayExpression	elements：元素数组	[1, 0]
ObjectExpression	properties：属性数组	a: 0, b: 1
TemplateLiteral	quasis：模板中的字符串内容 expressions：插入模板的表达式数组	‘Hello, ${name}‘
TemplateElement	value：字符串值 tail：是否为最后一个	‘Hello, ${ }‘
Super		super
AssignmentPattern	left：左表达式，为 Identifier right：右表达式	a = 0
SpreadElement	argument：扩展的表达式	...obj
RestElement	argument：扩展的参数	...arg

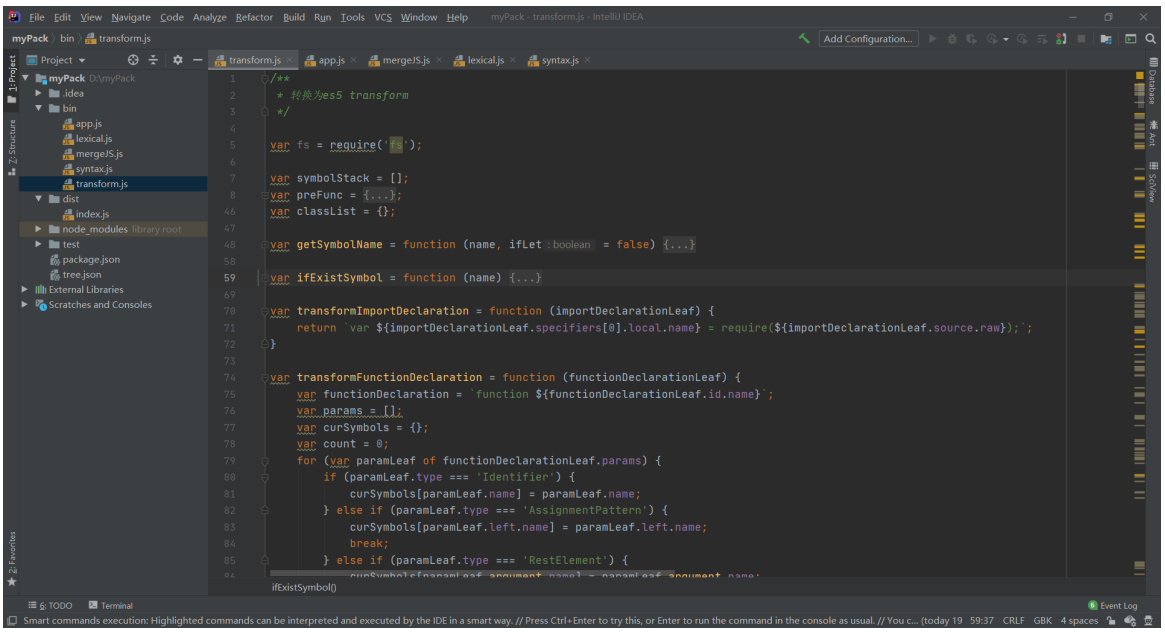
2.4 合并依赖文件

处理上一步所得语法树，读取依赖的文件，并转换为语法树，作为 module 插入到当前语法树中。完整代码见项目内 bin/mergeJS.js。



2.5 转换为 es5 代码

根据上一步所得语法树，按照 es5 规范将代码输出到目标文件。完整代码见项目内 bin/transform.js。

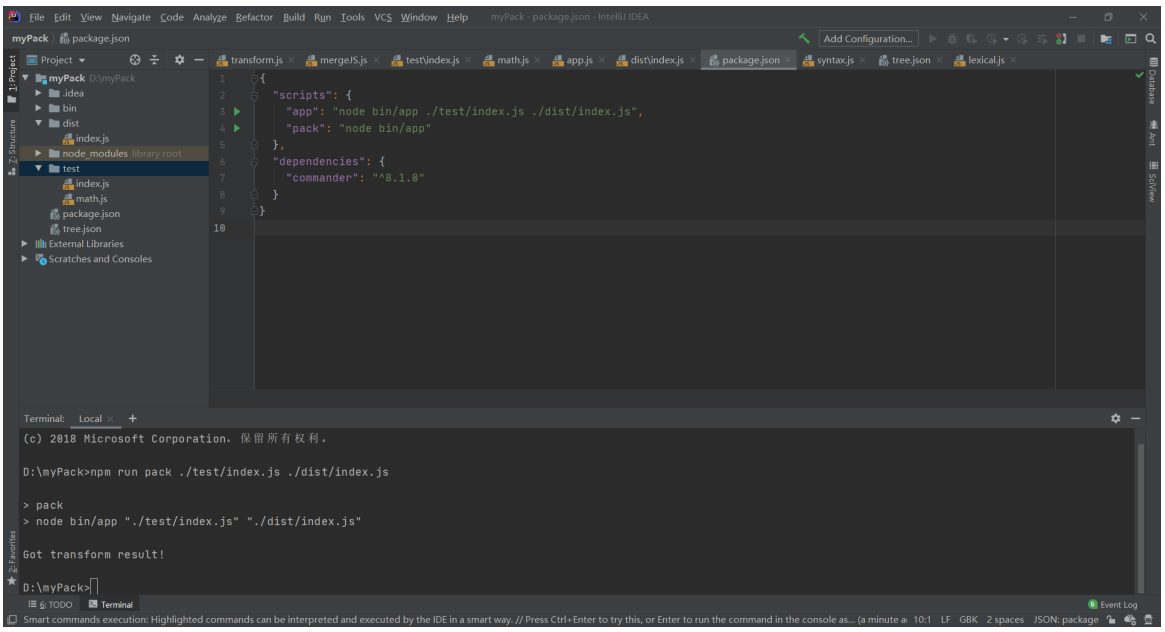


3 项目最终效果

基本实现转码与依赖分析，并且可以在命令行运行。

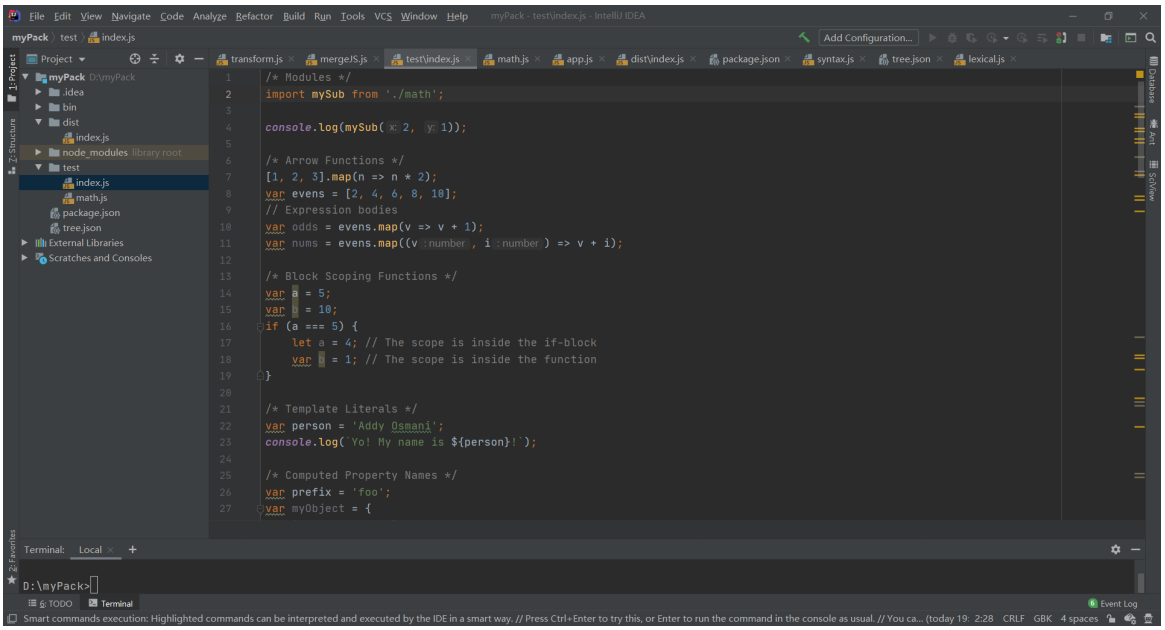
3.1 运行方法

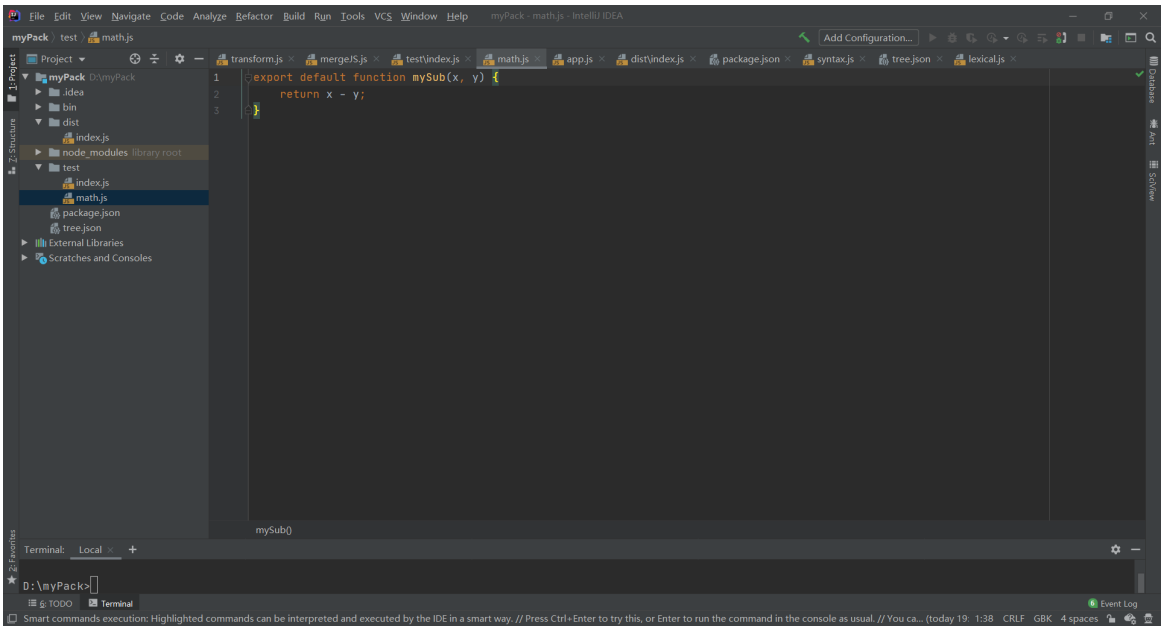
在命令行输入 npm run pack 待转换文件路径目标文件路径，即可进行转换，如下图。



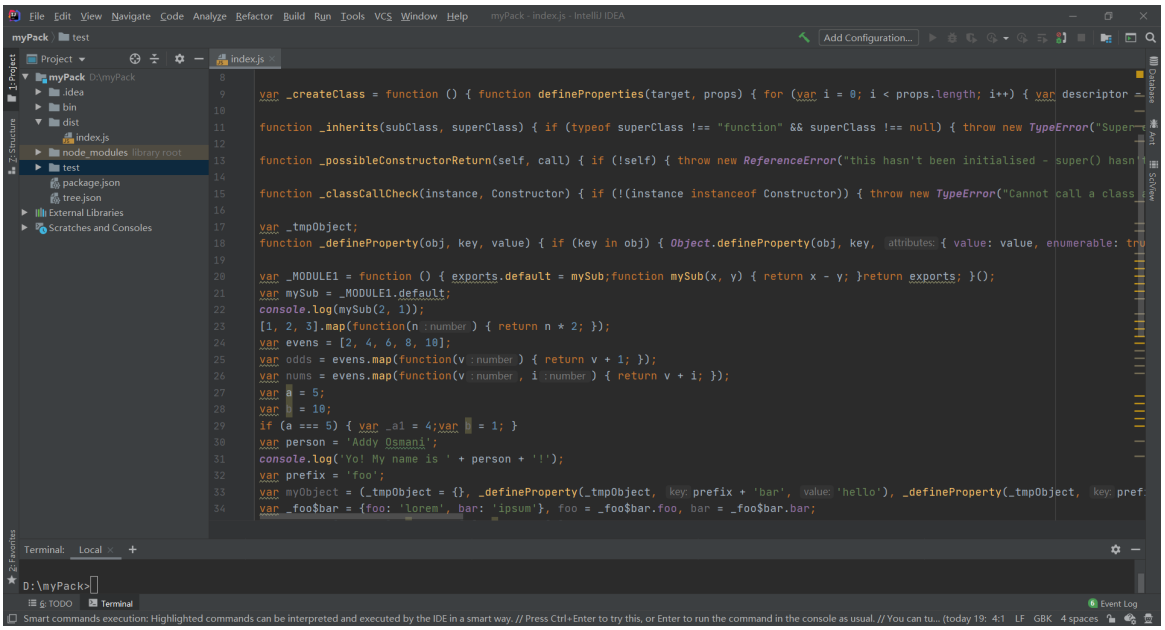
3.2 转换结果

用于测试的 es6 代码（完整测试代码在项目内 test 目录下）：





转换结果（完整转换结果在项目内 dist 目录下）：



4 总结

非常感谢老师给我这次机会进行学习。在这次项目经历中，我不仅巩固了之前的知识，比如 javascript 以及 webpack 等，并将它们运用进项目，更学习了一些我曾经未接触过的新框架，如 React，并在实践中进行熟悉和运用。最后项目中，完成自己的代码转换工具也使我对 babel 和 webpack 的实现原理有了更深的理解，便于以后使用。虽然项目进行的时间不长，但是常老师每一次都认真且耐心地回答我提出的疑问，并给我提供新的解决问题的思路，使我受益匪浅。

感谢常老师在专业知识方面的帮助，我才能够顺利地进行这次科研，并完成最终的项目。