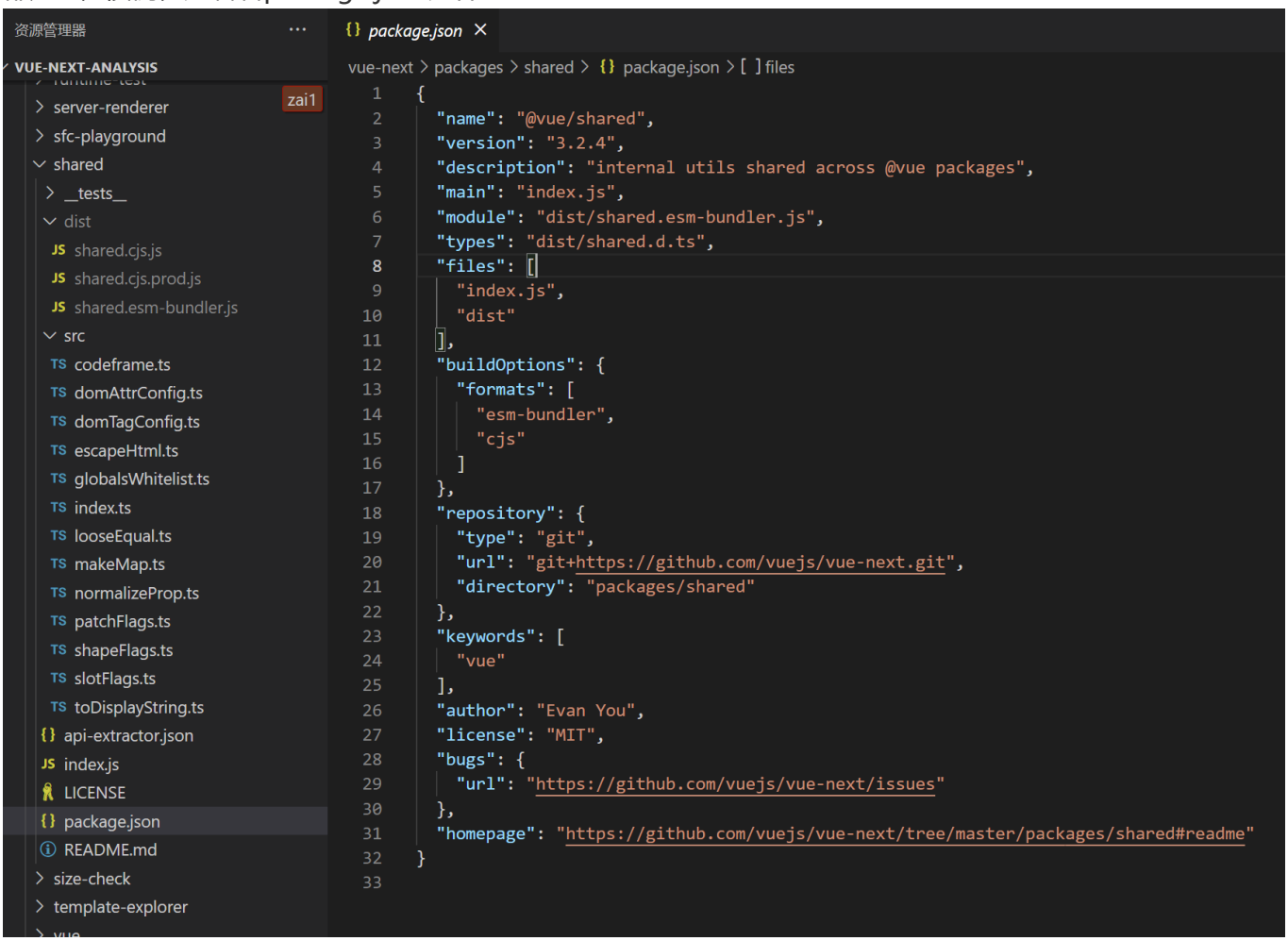
vue3-shared

之前看过vue2-shared，这次就直奔主题吧。vue3-shared的路径在/packages/shared里面。src里面基本上都是ts，惯例，先看看package.json文件



可以看到

```
"main": "index.js",
"module": "dist/shared.esm-bundler.js",
"types": "dist/shared.d.ts",
```

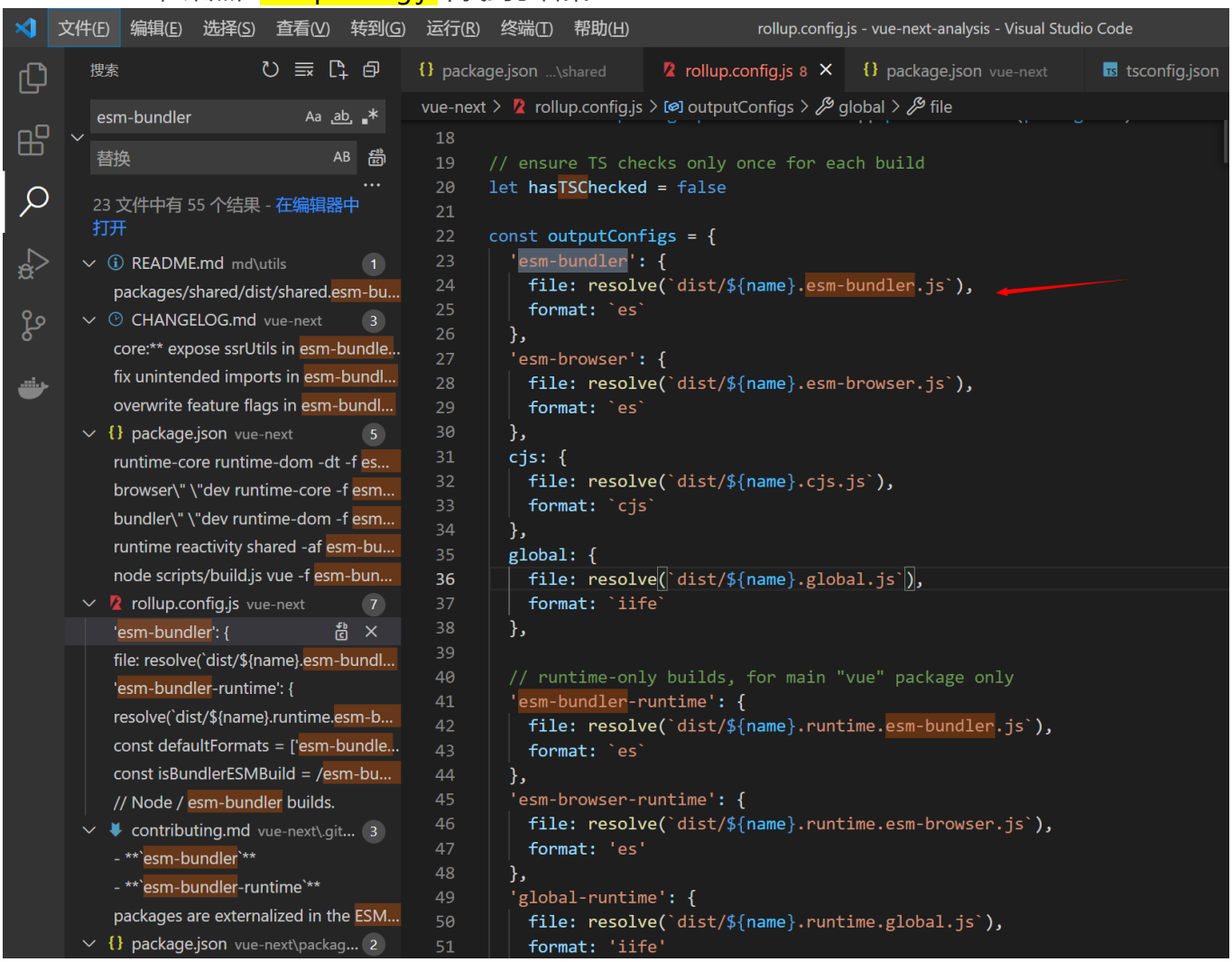
说明入口文件是index.js,打开index.js看看

```
'use strict'

if (process.env.NODE_ENV === 'production') {
  module.exports = require('./dist/shared.cjs.prod.js')
} else {
  module.exports = require('./dist/shared.cjs.js')
}
```

区分了一下生产环境与其他环境。

package.json中的module是dist/shared.esm-bundler.js，bundler打包机。猜测应该是打包生成的，搜索esm-bundler，果然在rollup.config.js中找到了答案



这样shared是如何被使用的我们也就清楚了，大概就是src里的ts经过tsc后再被rollup成shared.esm-bundler，然后再生成shared.cjs.prod与shared.cjs区分不同的环境。

那就废话不多说直接进入src里面探究源码去吧

打开index，导出一堆其他文件

TS index.ts

vue-next > packages > shared > src > TS index.ts > ...

```ts
import { makeMap } from './makeMap'

export { makeMap }
export * from './patchFlags'
export * from './shapeFlags'
export * from './slotFlags'
export * from './globalsWhitelist'
export * from './codeframe'
export * from './normalizeProp'
export * from './domTagConfig'
export * from './domAttrConfig'
export * from './escapeHtml'
export * from './looseEqual'
export * from './toDisplayString'

/**
 * List of @babel/parser plugins that are used for template
 * transforms and SFC script transforms. By default we enab
 * for ES2020. This will need to be updated as the spec mov
 * Full list at https://babeljs.io/docs/en/next/babel-parse
 */
export const babelParserDefaultPlugins = [
  'bigInt',
  'optionalChaining',
  'nullishCoalescingOperator'
] as const

export const EMPTY_OBJ: { readonly [key: string]: any } = _
  ? Object.freeze({})
  : {}
export const EMPTY_ARR = __DEV__ ? Object.freeze([]) : []

export const NOOP = () => {}
```

index

```ts
import { makeMap } from './makeMap'
```

```
export { makeMap }
export * from './patchFlags'
export * from './shapeFlags'
export * from './slotFlags'
export * from './globalsWhitelist'
export * from './codeframe'
export * from './normalizeProp'
export * from './domTagConfig'
export * from './domAttrConfig'
export * from './escapeHtml'
export * from './looseEqual'
export * from './toDisplayString'

/**
 * babel解析器默认插件
 */
export const babelParserDefaultPlugins = [
  'bigInt',
  'optionalChaining',
  'nullishCoalescingOperator'
] as const

/**
 * 创建一个空对象，如果是生产环境冻结该对象
 */
export const EMPTY_OBJ: { readonly [key: string]: any } = __DEV__
  ? Object.freeze({})
  : {}

/**
 * 创建一个空数组，如果是生产环境冻结该数组
 */
export const EMPTY_ARR = __DEV__ ? Object.freeze([]) : []

/**
 * 创建空数组。
 * 当时在看vue2-shared的时候，是为了避免flow使用reset产生无用的转换代码 + 为了避免传入undefined之类的数据导致报错
 * 之后又去segmentfault中查询，找到了令人满意的答案
 * 1.封装库的时候经常会把一些函数初始化为noop
 * Object.defineProperty(foo,key,{
 *    configurable: true,
 *    enumerable: true,
 *    get:()=>{....},
 *    set:NOOP
 * })
 * 2.提高代码可读性，方便
 */
export const NOOP = () => {}

/**
 * 始终返回fasle
 */
export const NO = () => false

/**
 * 判断字符串是不是 on开头，并且on后面是否 不是小写字母
 */
const onRE = /^on[^a-z]/
export const isOn = (key: string) => onRE.test(key)
```

```
/**
 * 判断字符串是否以 onUpdate:开头
 */
export const isModelListener = (key: string) => key.startsWith('onUpdate:')

/**
 * 合并对象，Object.assign，浅拷贝
 */
export const extend = Object.assign

/**
 * 从数组中移除某一项。与vue2相比这次并没有返回移除的结果
 */
export const remove = <T>(arr: T[], el: T) => {
  const i = arr.indexOf(el)
  if (i > -1) {
    arr.splice(i, 1)
  }
}
/**
 * 检查是否拥有自己的属性（不是方法），不查原型链，不查继承
 */
const hasOwnProperty = Object.prototype.hasOwnProperty
export const hasOwn = (
  val: object,
  key: string | symbol
): key is keyof typeof val => hasOwnProperty.call(val, key)

/**
 * 是否是数组
 */
export const isArray = Array.isArray

/**
 * 是否是Map
 */
export const isMap = (val: unknown): val is Map<any, any> =>
  toTypeString(val) === '[object Map]'

/**
 * 是否是Set
 */
export const isSet = (val: unknown): val is Set<any> =>
  toTypeString(val) === '[object Set]'

/**
 * 是否是Date
 */
export const isDate = (val: unknown): val is Date => val instanceof Date

/**
 * 是否是function
 */
export const isFunction = (val: unknown): val is Function =>
  typeof val === 'function'

/**
 * 是否是字符串
 */
export const isString = (val: unknown): val is string => typeof val === 'string'
```

```
/**
 * 是否是symbol
 */
export const isSymbol = (val: unknown): val is symbol => typeof val === 'symbol'

/**
 * 快速检查对象
 * 检查是否符合JSON
 * typeof null   // object
 */
export const isObject = (val: unknown): val is Record<any, any> =>
  val !== null && typeof val === 'object'

/**
 * 是否是Promise
 * 写法与vue2有所不同，之前是isDef(val)现在是isObject，更加严谨
 */
export const isPromise = <T = any>(val: unknown): val is Promise<T> => {
  return isObject(val) && isFunction(val.then) && isFunction(val.catch)
}

/**
 * vue2中是_toString,这次干脆连.call都封装了，笑死了
 */
export const objectToString = Object.prototype.toString
export const toTypeString = (value: unknown): string =>
  objectToString.call(value)

/**
 * 实现类似typeof的功能，但是比typeof精确，可以判断Array，Null
 */
export const toRawType = (value: unknown): string => {
  // extract "RawType" from strings like "[object RawType]"
  return toTypeString(value).slice(8, -1)
}

/**
 * 是否是纯对象，区分array
 */
export const isPlainObject = (val: unknown): val is object =>
  toTypeString(val) === '[object Object]'

/**
 * 是否是数字型字符串
 * ' + parseInt(key, 10) === key 就是去除了0开头的可能
 */
export const isIntegerKey = (key: unknown) =>
  isString(key) &&
  key !== 'NaN' &&
  key[0] !== '-' &&
  '' + parseInt(key, 10) === key

/**
 * 保留属性
 */
export const isReservedProp = /*#__PURE__*/ makeMap(
  // the leading comma is intentional so empty string "" is also included
  ',key,ref,' +
    'onVnodeBeforeMount,onVnodeMounted,' +
    'onVnodeBeforeUpdate,onVnodeUpdated,' +
    'onVnodeBeforeUnmount,onVnodeUnmounted'
```

```
)

/**
 * 缓存数据，fn的返回集的缓存，如果fn没有返回值，则不能起到缓存作用
 * 闭包
 */
const cacheStringFunction = <T extends (str: string) => string>(fn: T): T => {
  const cache: Record<string, string> = Object.create(null)
  return ((str: string) => {
    const hit = cache[str]
    return hit || (cache[str] = fn(str))
  }) as any
}

/**
 * 连字符转小驼峰
 * 小驼峰，第一个字母小写
 * 大驼峰，第一个字母也大写
 * 用了缓存
 */
const camelizeRE = /-(\w)/g
/**
 * @private
 */
export const camelize = cacheStringFunction((str: string): string => {
  return str.replace(camelizeRE, (_, c) => (c ? c.toUpperCase() : ''))
})

/**
 * 小驼峰转连字符
 * 用了缓存
 */
const hyphenateRE = /\B([A-Z])/g
/**
 * @private
 */
export const hyphenate = cacheStringFunction((str: string) =>
  str.replace(hyphenateRE, '-$1').toLowerCase()
)

/**
 * @private
 */
export const capitalize = cacheStringFunction(
  (str: string) => str.charAt(0).toUpperCase() + str.slice(1)
)

/**
 * @private
 */
/**
 * 首字母转大写
 * 用了缓存
 */
export const toHandlerKey = cacheStringFunction((str: string) =>
  str ? `on${capitalize(str)}` : ``
)

// compare whether a value has changed, accounting for NaN.
/**
 * 判断是否是同一个值
```

```
 *
 * 都是 undefined
 * 都是 null
 * 都是 true/false
 * 都是相同长度的字符串且相同字符按相同顺序排列
 * 都是相同对象（意味着每个对象有同一个引用）
 * 都是数字且
 *  +0
 *  -0
 *  NaN
 *
 * Object.is 与 == ===都不同
 */
export const hasChanged = (value: any, oldValue: any): boolean =>
  !Object.is(value, oldValue)

/**
 * 执行数组里面的函数
 * 执行一连串函数？？
 */
export const invokeArrayFns = (fns: Function[], arg?: any) => {
  for (let i = 0; i < fns.length; i++) {
    fns[i](arg)
  }
}

/**
 * def定义对象属性
 */
export const def = (obj: object, key: string | symbol, value: any) => {
  Object.defineProperty(obj, key, {
    configurable: true,
    enumerable: false,
    value
  })
}

/**
 * 转数字，如果失败，返回原val
 */
export const toNumber = (val: any): any => {
  const n = parseFloat(val)
  return isNaN(n) ? val : n
}

/**
 * 全局对象
 * 微信小程序，一般都是空，最后用空对象
 * 单例模式
 */
let _globalThis: any
export const getGlobalThis = (): any => {
  return (
    _globalThis ||
    (_globalThis =
      typeof globalThis !== 'undefined'
        ? globalThis
        : typeof self !== 'undefined'
        ? self
        : typeof window !== 'undefined'
        ? window
```

```
        : typeof global !== 'undefined'
        ? global
        : {})
  )
}
```

关于NOOP,在看vue2-shared的时候就产生了疑问，当时认为是为了避免flow使用reset产生无用的转换代码 + 为了避免传入undefined之类的数据导致报错。但是总是觉得漏了点东西，这次特地在项目中搜了一下NOOP，看一看尤大自己是怎么用它的，果然收获到了满意的答案

```
1.
Object.keys(publicPropertiesMap).forEach(key => {
    Object.defineProperty(target, key, {
      configurable: true,
      enumerable: false,
      get: () => publicPropertiesMap[key](instance),
      // intercepted by the proxy so no need for implementation,
      // but needed to prevent set errors
      set: NOOP
    })
  })

2.
return NOOP

3.
getter = NOOP

4.
if (__DEV__ && get === NOOP) {

5.
instance.render = (Component.render || NOOP) as InternalRenderFunction
```

上面是vue3中的几段代码,大概就明白了尤大弄NOOP的原因
1.方便理解
2.判断时候 代码很简洁
3.避免出错，初始化的时候赋值NOOP，这边1、5都是这个意思
4.川哥有提到，对比直接写function(){},使用noop能被压缩，而直接写匿名函数则无法被压缩

```
export const NOOP = () => {}
```

makeMap

```
export function makeMap(
  str: string,
  expectsLowerCase?: boolean
): (key: string) => boolean {
  const map: Record<string, boolean> = Object.create(null)
  const list: Array<string> = str.split(',')
  for (let i = 0; i < list.length; i++) {
```

```
    map[list[i]] = true
  }
  return expectsLowerCase ? val => !!map[val.toLowerCase()] : val => !!map[val]
}
```

跟vue没啥区别，唯一的区别在于返回值用了!!转成boolean,更加严谨

总结

整体读下来与vue2还是非常相似的，也有细节之处有所不同，比对这些不同之处，还是蛮有意思的一件事情，自己也能学习到更多。