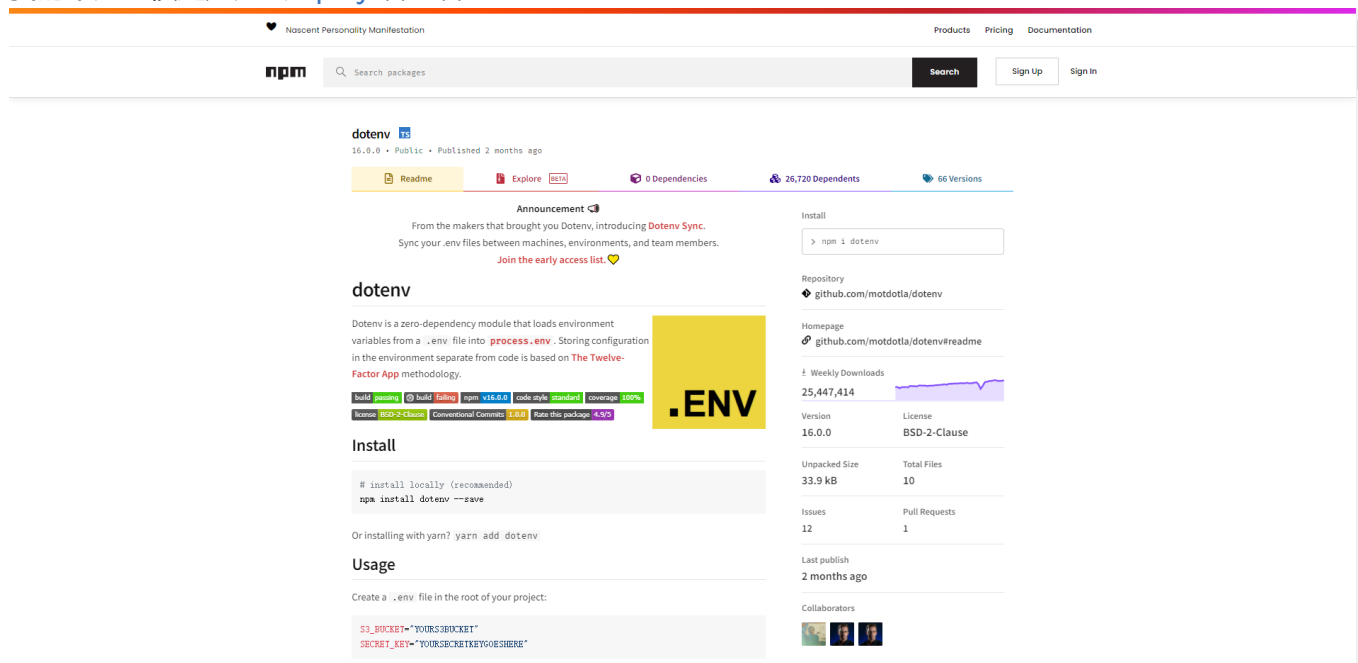


dotenv

我的个人习惯还是先去 [npmjs](#) 看一看



介绍：Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env. Storing configuration in the environment separate from code is based on The Twelve-Factor App methodology.

大概意思就是说Dotenv是个零依赖的模块，将环境变量从.env文件加载到process.env中。基于The Twelve-Factor App methodology将代码与环境配置分开。

The Twelve-Factor App methodology：十二因素应用程序方法论

十二因素应用程序方法论是啥我不关心，以后有时间去补课。

使用方法也很简单：

1. 创建一个.env文件在项目的根目录
2. 尽可能早的在项目中导入和配置dotenv
3. process.env就能拿到.env文件中的配置了

看到介绍的时候我就惊呆了，公司现在的项目就这么配置的！以前就一直认为vue就这样的，具体咋实现的也就没关心。赶紧去项目搜索了一下dotevn，果然在@vue/cli-service中发现了！快乐！

.env

```
S3_BUCKET="YOURS3BUCKET"
SECRET_KEY="YOURSECRETKEYGOESHERE"
```

直接把官方的例子拿过来了，配置很简单直接一个等号就完事了，因为是环境变量所以最好全大写

模拟实现

使用方法与.env文件我们都会写了。还算是比较简单的功能，我习惯对于简单的功能就自己模拟实现再去阅读源码，这样可以收获更多。

咱要做的内容大概就如下：

1. 获取根目录下的.env文件
2. 读取.env文件的内容
3. 逐行读取并按等号切割，split /n, =
4. 然后键值给process.env

```
require('dotenv').config()  
console.log(process.env)
```

上面是官网的例子，说明config是个函数，直接上手

```
import * as fs from "fs"  
import * as Path from 'path';  
  
function ReadFileSync(path, encoding = 'utf-8') {  
  try {  
    return fs.readFileSync(path, encoding);  
  }  
  catch (e) {  
    console.error(e);  
    return null;  
  }  
}  
  
const config = () => {  
  // 找到根目录中的.env文件  
  const dotenvPath = Path.resolve(process.cwd(), '.env')  
  const lines = ReadFileSync(dotenvPath).toString('utf-8').split('\n')  
  for (const line of lines) {  
    const splitArr = line.split('=')  
    process.env[splitArr[0]] = splitArr[1] || ""  
  }  
}
```

初步实现完成，可以去阅读源码了！

config函数肯定是入口，所以从config看起

```
/* @flow */  
/*::  
  
type DotenvParseOptions = {  
  debug?: boolean  
}  
  
// keys and values from src  
type DotenvParseOutput = { [string]: string }
```

```

type DotenvConfigOptions = {
  path?: string, // path to .env file
  encoding?: string, // encoding of .env file
  debug?: string // turn on logging for debugging purposes
}

type DotenvConfigOutput = {
  parsed?: DotenvParseOutput,
  error?: Error
}

*/

const fs = require('fs')
const path = require('path')
const os = require('os')

function log (message /*: string */) {
  console.log(`[dotenv][DEBUG] ${message}`)
}

const NEWLINE = '\n'
const RE_INI_KEY_VAL = /^\\s*([\\w.-]+)\\s*=\\s*(.*)?\\s*$/
const RE_NEWLINES = /\n/g
const NEWLINES_MATCH = /\r\n|\n|\r/

// Parses src into an Object
function parse (src /*: string | Buffer */, options /*: ?DotenvParseOptions */) /*:
DotenvParseOutput */ {
  // 这里是没明白为什么还要做判断，可能是别的文件还有调用这个函数吧
  const debug = Boolean(options && options.debug)
  const obj = {}

  // convert Buffers before splitting into lines and processing
  // 这边比单纯的\n划分更加严谨了
  src.toString().split(NEWLINES_MATCH).forEach(function (line, idx) {
    // matching "KEY" and "VAL" in "KEY=VAL"
    const keyValueArr = line.match(RE_INI_KEY_VAL)
    // matched?
    if (keyValueArr != null) {
      const key = keyValueArr[1]
      // default undefined or missing values to empty string
      let val = (keyValueArr[2] || '')
      const end = val.length - 1
      const isDoubleQuoted = val[0] === '"' && val[end] === '"'
      const isSingleQuoted = val[0] === "'" && val[end] === "'"
      // 多了去引号的步骤
      // if single or double quoted, remove quotes
      if (isSingleQuoted || isDoubleQuoted) {
        val = val.substring(1, end)

        // if double quoted, expand newlines
        if (isDoubleQuoted) {
          val = val.replace(RE_NEWLINES, NEWLINE)
        }
      } else {
        // remove surrounding whitespace
        val = val.trim()
      }
    }
    // 专门弄了个对象用来存键值，返回用。应该是用来单独查看配置文件的时候可以用
    obj[key] = val
  })
}

```

```

    } else if (debug) {
      log(`did not match key and value when parsing line ${idx + 1}: ${line}`)
    }
  })

  return obj
}

function resolveHome (envPath) {
  return envPath[0] === '~' ? path.join(os.homedir(), envPath.slice(1)) : envPath
}

// Populates process.env from .env file
function config (options /*: ?DotenvConfigOptions */) /*: DotenvConfigOutput */ {
  let dotenvPath = path.resolve(process.cwd(), '.env')
  let encoding /*: string */ = 'utf8'
  let debug = false

  // 多了options, 看样子是可以传参path, encoding, debug
  // 有path, .env就不需要固定丢在根目录了
  // debug告知是否是调试
  if (options) {
    if (options.path !== null) {
      dotenvPath = resolveHome(options.path)
    }
    if (options.encoding !== null) {
      encoding = options.encoding
    }
    if (options.debug !== null) {
      debug = true
    }
  }

  // 有IO肯定需要try catch,我是直接把readFileSync封装成函数了
  // 如果是后端我是建议整一个IO,把通用的方法全封装起来,这样就不需要一直try catch了
  try {
    // specifying an encoding returns a string instead of a buffer
    //
    const parsed = parse(fs.readFileSync(dotenvPath, { encoding }), { debug })

    Object.keys(parsed).forEach(function (key) {
      // 这边多了判断, 原属性优先原则
      if (!Object.prototype.hasOwnProperty.call(process.env, key)) {
        process.env[key] = parsed[key]
      } else if (debug) {
        log(`"${key}" is already defined in \`${process.env}\` and will not be overwritten`)
      }
    })

    return { parsed }
  } catch (e) {
    return { error: e }
  }
}

module.exports.config = config
module.exports.parse = parse

```

源码做了更加精细的处理，我自己的活还是太糙了哈哈哈。

总结

还是得多看文章，这其实与源码解析没啥关系。每天都在用.env文件，却不知道这是dotenv的功劳！

Q: 为什么parse中的debug需要Boolean(options && options.debug)操作？按照main.js中来看是完全不需要的

A: options /*: ?DotenvParseOptions 注释声明options可能没传参，我的猜测应该是准确的。

果然在test.ts中发现未传参的例子，问题解决

```
JS main.js TS test.ts 1 X
dotenv > types > TS test.ts > [0] parsed
1  import { config, parse } from "dotenv";
2
3  const env = config();
4  const dbUrl: string | null =
5    env.error || !env.parsed ? null : env.parsed["BASIC"];
6
7  config({
8    path: ".env-example",
9    encoding: "utf8",
10    debug: true
11  });
12
13  const parsed = parse("NODE_ENV=production\nDB_HOST=a.b.c");
14  const dbHost: string = parsed["DB_HOST"];
15
16  const parsedFromBuffer = parse(new Buffer("JUSTICE=league\n"), {
17    debug: true
18  });
19  const justice: string = parsedFromBuffer["JUSTICE"];
20
```

