# 应用集成原理与工具

Principles and Technologies of Application Integration

## 基于 VS Code 的 Node.js API 推荐插件


第七组

151250018 陈统

151250037 冯俊杰

151250038 刚昭

151250172 徐天泽


2018-07-14

# 一、项目背景

我们程序员写代码离不开 API，而且经常要用到自己以前不知道的 API。我们什么时候会想用一个新 API？当我们想要在当前的代码中实现一个功能，又觉得早就有人造过轮子的时候。

在以上情景中，为了找到这个可能存在的 API，我们通常只有两种做法：

- 通过自然语言描述，借助搜索引擎在乱七八糟的页面里找
- 从一个有点相关度的库的文档里找

不管是上述哪种做法，都不能利用我们当前代码的上下文，不能利用形式化的数据，而这些信息又是对于搜索 API 极有价值的：想象只是加入对 API 输出类型的过滤，就足以让纯基于自然语言的搜索结果准确许多。

因此，我们组做了这样一个在代码编辑器端同时利用 __自然语言描述__ 和 __代码上下文__ 推荐 API 的项目。

我们的 API 仓库包含了 Nodejs 的所有库；用户仅需在本地安装 vscode 端的插件，就可以通过远程查询服务器来获得 API 推荐的结果。

# 二、插件使用描述

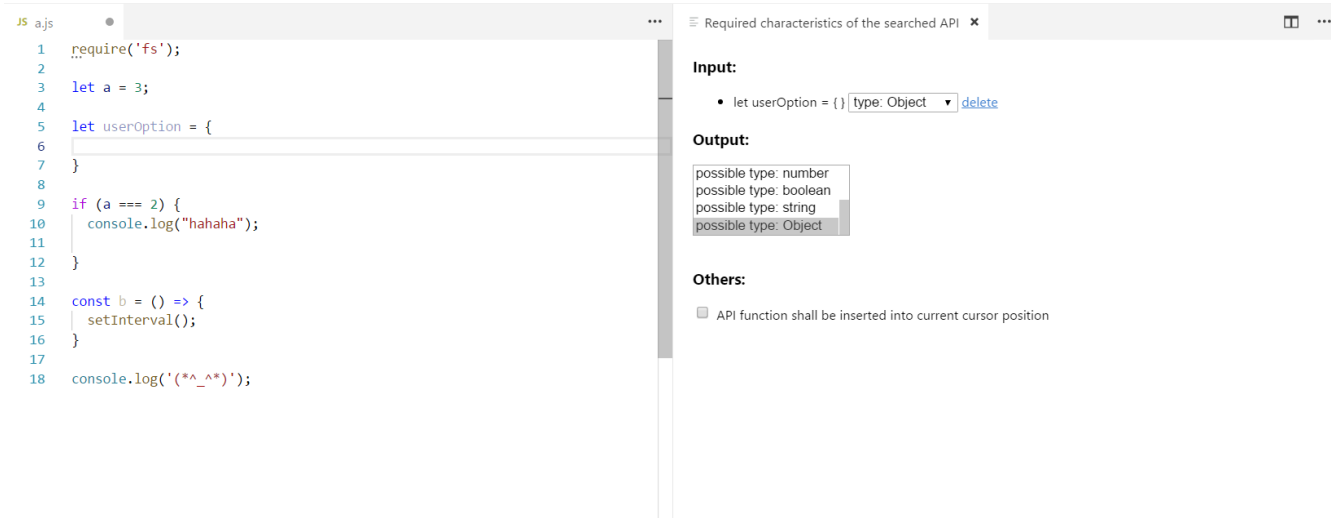
用户在安装插件后，可在编辑器已打开.js文件的条件下使用插件。

为了发送一个搜索请求，用户需要完成以下两个前置操作：

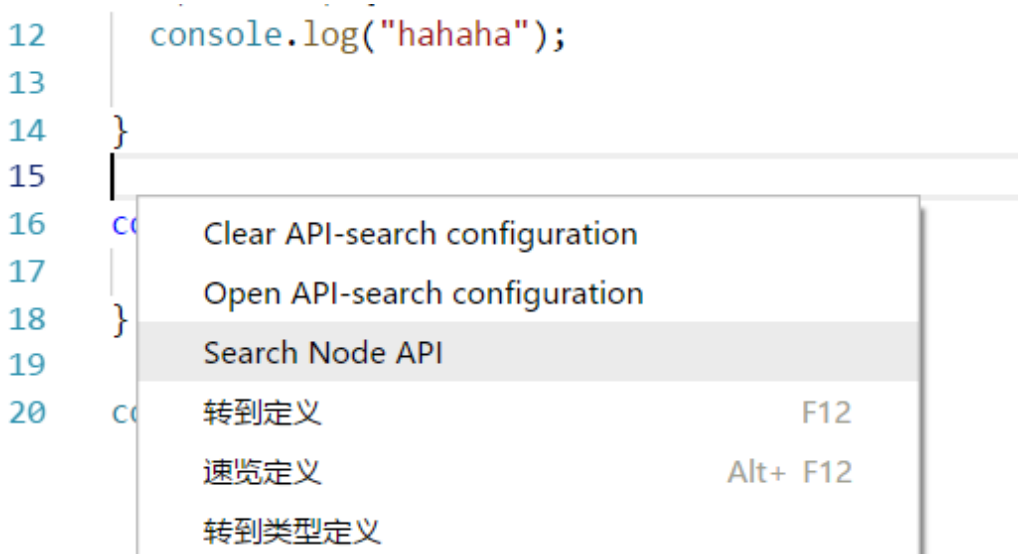
- 指定 API 的输入输出类型，输入的变量
- 使用 search 命令，输入任务的自然语言描述

1. 首先，用户可以在编辑器中选中某变量的赋值语句，在右键菜单中将其标记为输入变量。

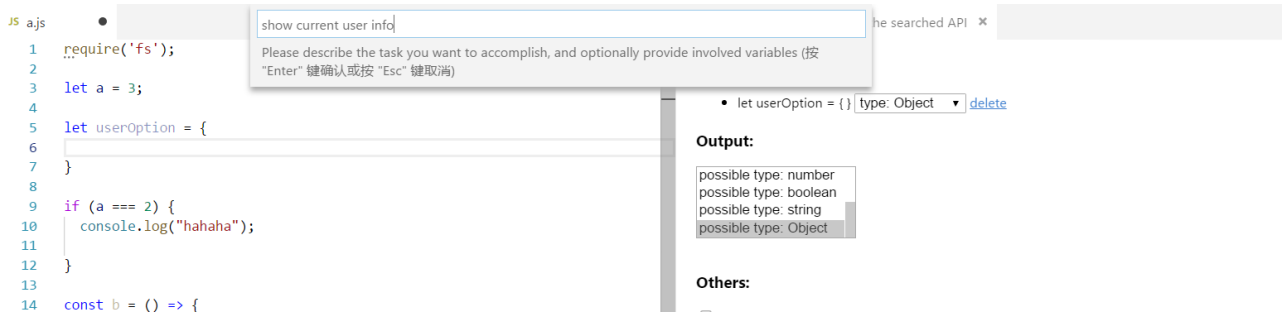
2. 此时，类型配置侧边栏将被打开。用户也可以通过1中显示的打开配置栏选项直接打开。用户可以在这里指定预期的输入和输出变量的类型，以及其它辅助推荐的选项。



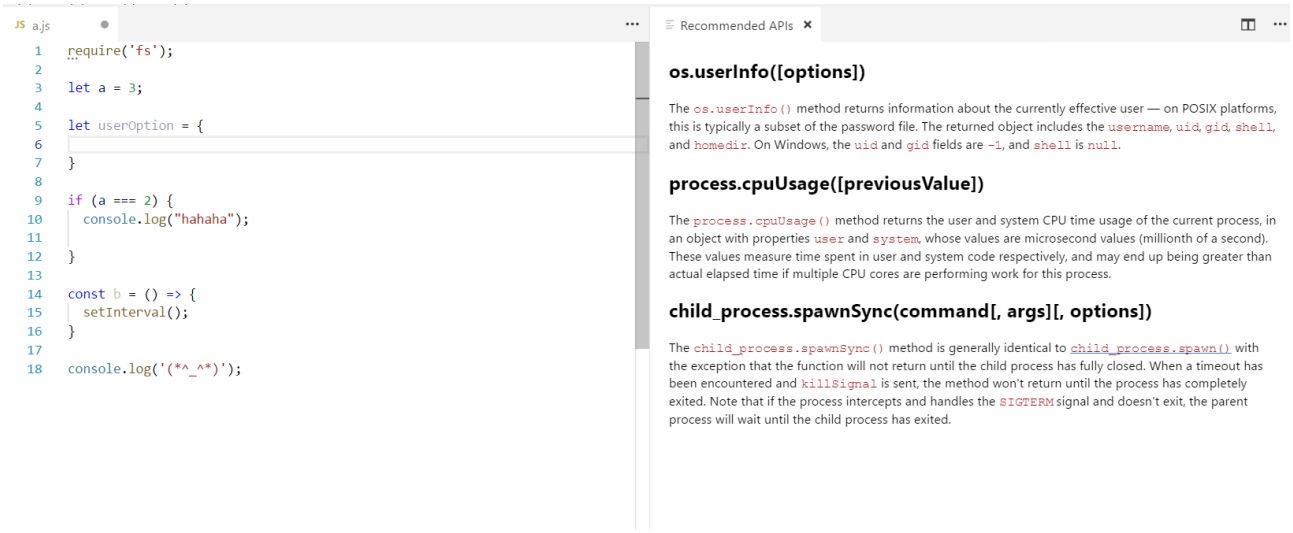

3. 配置完成后，用户可以从空白区域的右键菜单或 vscode 的命令面板中使用 Search Node API 命令。




4. 输入对需要 API 完成的任务的自然语言描述，然后按回车键。



5. 稍等片刻，插件将会把服务器返回的排序过的 API 列表显示在侧边栏。




# 三、后台服务架构

## 1. 架构描述：

本系统后端使用 Node.js express 搭建，对前端提供如下的 RESTful API。

- 方法地址：url/api/search
- 方法类型：POST
- 参数描述
  - desc：用户自己的对需要寻找的方法的描述
  - code：用户的相关代码
  - currentLine：用户当前所在行数，即需要进行搜索的行数
  - inputSelections：JSON 格式的对要搜索的输入参数中的描述列表，每一项需要包含如下信息。
    - content：对参数的代码描述，如 let a = 'bbb'。
    - type：对参数的类型描述，如 string。
  - outputSelections：JSON 格式的对输出类型的可能描述列表，如 ['no return', 'string']
- 返回结果
  - 包含最多十条匹配结果的 HTML。

## 2. 为什么使用 Node.js

- 考虑到项目是对 JavaScript 代码进行分析，可能会基于 esprima 等 js 平台的语法树分析。
- 轻量级服务器。
- nodejs 完善的生态资源。

## 3.为什么使用 RESTful API

1. 前后端分离解耦，前端调用后端资源的 API 是有意义的。
2. API 格式规范，易于前端人员使用。

# 四、API 推荐机制与算法

## 1. 基于方法输入输出的初步筛选机制

项目支持的 API 输入输出类型包括：`Any, Array, Function, number, boolean, string, Object`，其中输出还支持`no return`。

在具体实现中，首先，遍历项目中能够提供帮助的 js 方法，将每一 js 方法的参数列表与用户输入的`inputSelections` 中的每一个项进行比较，若存在相同的项，则将此方法选入当前根据输入筛选匹配的可能结果中。

再遍历根据输入筛选得到的方法列表，若其方法返回类型与用户输入的`outputSelections` 存在交叉项，则选入匹配的结果中。

## 2. 基于用户代码上下文与代码实例上下文匹配的推荐算法

### 1）匹配算法的支持数据

为了获取用于匹配用户代码的 API 实际应用场景的代码实例，我们使用了 StackExchange 提供的 Stack Exchange Data Dump 服务，同时利用 Stack Exchange 提供的Query 查询服务提取到了 StackOverFlow 上所有关于 NodeJS 的回答内容。它们包括了自然语言的回答内容和非常重要和有效的代码回答内容。

在获取到这些基础的问答数据以后，我们利用 Java 程序 （详见代码仓库 StackDataMatch_JAVA）进行了全部 NodeJS API 与 StackOverFlow 回答的代码之间的匹配。我们通过识别代码中方法签名模版的方式识别出每一个方法对应的一些代码实例，并写入数据库。

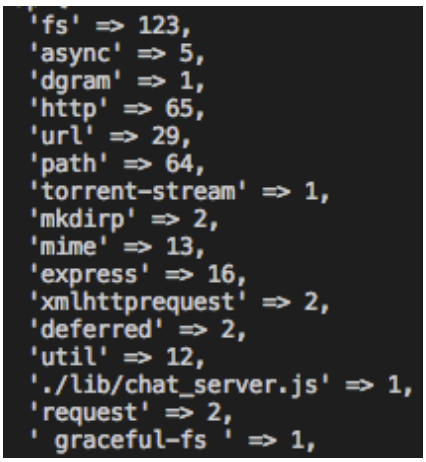
我们最终为 Node.js 的 817 个 API 匹配了 186634 条代码实例。

### 2）代码依赖库的匹配

这是我们使用的第一种基于代码上下文匹配的API推荐算法。我们会为每一个方法预处理出在其代码实例上下文中引用过的外部库（例如使用 require 引入）。然后基于用户上下文引用内容和代码实例的引用内容的匹配结果，计算每一个库函数被当前用户使用的可能性。

举一个例子：对于 fs 文件处理库的 API ： fs.exists()。在它的 288 条代码实例中出现频率较高的分别是 fs（出现123次） http（出现65次） path（出现64次）等。所以当用户代码环境出现同样高频的依赖库时，我们会根据这些依赖库在大量代码实例中出现的频率给该方法出

现的概率加上相应的评分，用户给出的多个依赖库会累积评分。以此我们就可以标示用户有较高可能性调用的 API。




### 3）代码控制结构的匹配

在这里我们所说的控制结构指的是在 NodeJS 中起到代码运行控制作用的结构，例如三种循环，function 结构，if 分支结构, ajax, request, function, ready, post, get。

代码控制结构的匹配指的是我们通过匹配用户代码上下文的控制结构嵌套顺序与代码实例中的控制结构嵌套顺序的相似程度，来为用户标示有较高可能性调用的 API。在这里我们同样通过其控制结构的相似性来量化用户上下文环境中的调用可能性。

### 4）代码上下文方法调用的匹配

我们考虑到，许多 API 的调用之前或之后，都会调用特定的相关联的 API。利用这一点特性，我们预先处理所有 Node.js 的 API 的代码实例中其前后调用的高频特定 API 和其他方法。此后我们通过分析用户当前上下文环境中已经调用过的 API 和其他方法来为用户标示用户在此时有较高可能性调用的 API。在这里我们通过其公共高频调用API相似性来量化用户上下文环境中的调用可能性。

## 2. 基于文档API描述的推荐算法

### 1）算法思路

该过程采用文本相似度匹配的思路，通过用户输入的自然语言形式下的短语或句子，采用文本相似度匹配的做法，与Node.js官方文档中方法的description描述进行匹配并行排序，向用户推荐相关的API。相比于传统API文档的全局搜索方法，对于方法的重要性进行了排序，比较了关键词在不同方法中的相关权重，从而帮助用户有限选择到具价值的方法。在具体实现方法，通过比较了三种短小语段关键词提取算法（TF-IDF、TextRank、Word2Vector），可类比从影评、微博中提取关键词，选择了TF-IDF这一通性较强的方法，原因在于作为相比于传统自然语言形式化较强的方法使用描述，在一些动词，副词的使用上，更加统一和标准，在多次出现后，会由于逆文本频率指数较高缘故被过滤除去，而一些对于识别方法更加重要的名词，则会在一些相关的方法中频率较高，从而实现相关方法的

description上的匹配。作为代码推荐的第三种方式，过关键词进行描述匹配可以在原有的基础上进一步发掘用户的主观意愿，从而提升推荐的准确程度。

### 2）实现过程

实现过程共分为

- 对数据的预处理
- 将全部描述文本作为文件集合，将单一描述作为一个文件，计算其权重
- 统一化，计算词汇在不同方法下的权重，并以此作为推荐的标准

### 3）案例分析

以单词abort为例，该词汇在全部描述中共出现三次，其描述用于分别如下：

| 方法名 | 方法描述 | 相关权重 |
| --- | --- | --- |
| Process.abort() | The process.abort() method causes the Node.js process to exit immediately and generate a core file. | 0.34 |
| AsyncResource.EmitBefore() | Call all before callbacks to notify that a new asynchronous execution context is being entered. If nested calls to emitBefore() are made, the stack of syncIds will be tracked and properly unwound. before and after calls must be unwound in the same order that they are called. Otherwise, an unrecoverable exception will occur and the processwill abort. For this reason, the emitBefore and emitAfter APIs areconsidered deprecated. Please userunInAsyncScope, as it provides a much safer alternative. | 0.17 |
| http.request() | Call all after callbacks. If nested calls to emitBefore() were made, thenmake sure the stack is unwound properly. Otherwise an error will be thrown. If the users callback throws an exception, emitAfter() will automatically becalled for all asyncIds on the stack if the error is handled by a domain oruncaughtException handler.before and after calls must be unwound in the same order that theyare called. Otherwise, an unrecoverable exception will occur and the processwill abort. For this reason, the emitBefore and emitAfter APIs areconsidered deprecated. Please use runInAsyncScope, as it provides a much safer alternative. | 0.11 |

从以上三个方法可以看出，该方法具备较好的解释性，abort一词在以上三个方法中的重要程度依次递减，根据用户输入的词语组合，会依次向用户推荐Process.abort(),AsyncResource.EmitBefore()和http.request()三组方法，有助于发现意向方法。

## 4、三种推荐算法的综合应用

在我们的插件中，我们综合应用了以上三种匹配算法（代码上下文，API描述，方法输入输出）。应用模式如下描述：

我们首先通过方法输入输出的匹配作粗略与正确性完全的匹配，即预先框定一些范围的 Node.js 的 API，且是完全符合用户输入输出匹配要求的。此后，我们分别利用基于代码上下文匹配的推荐算法和基于文档 API 描述的推荐算法来为这些经过初步筛选的 API 打分，其中我们将基于文本描述分析的分数量化在 0 - 1 的区间，基于上下文代码匹配的分数量化在 0 - 2.5 的区间，最终为用户提供排名较靠前的若干 API 供其选择。