# 1. 服务器 node.js

## 1.1. 安装

### 1.1.1. Development

```bash

npm i

npm run dev

open http://localhost:7001/

```

### 1.1.2. Deploy

```bash

npm start

npm stop

```

## 1.2. egg项目结构

> <https://eggjs.org/zh-cn/basics/structure.html>

```txt

/app

│ router.js // Router 主要用来描述请求 URL 和具体承担执行动作的 Controller 的对应关系， 框架约定了 app/router.js 文件用于统一所有路由规则。

│

├─controller // Controller 负责解析用户的输入，处理后返回相应的结果。主要相应路由的调用，并执行service中的方法处理数据

│ │ home.js

│ │

│ └─v1

│ graphs.js

│ topics.js

│

├─middleware // app/middleware/\*\* 用于编写中间件

│ error\_handler.js

│

└─service // Service 就是在复杂业务场景下用于做业务逻辑封装的一个抽象层。将同类处理函数封装成的service类

graphs.js

test.js

topics.js

/config // 框架配置目录

| config.default.js // config.default.js 为默认的配置文件，所有环境都会加载这个配置文件，一般也会作为开发环境的默认配置文件。

| plugin.js // plugin.js 用于配置需要加载的插件

```

## 1.3. RESTful API

目标是使用 RESTful 风格的 API 定义，规范化数据获取和处理的流程。

在egg.js框架中，提供了 app.resources('routerName', 'pathMatch', controller) 快速在一个路径上生成 CRUD 路由结构。具体对应如下：

Method | Path | Route Name | Controller.Action

- | - | - | -

GET | /posts | posts | app.controllers.posts.index

GET | /posts/new | new\_post | app.controllers.posts.new

GET | /posts/:id | post | app.controllers.posts.show

GET | /posts/:id/edit | edit\_post | app.controllers.posts.edit

POST | /posts | posts | app.controllers.posts.create

PUT | /posts/:id | post | app.controllers.posts.update

DELETE | /posts/:id | post | app.controllers.posts.destroy

## 1.4. 获取函数调用图数据的API设计

功能描述：通过此 API 获取函数调用图的节点和边的数据，使用http get请求获取，在请求中须包含所需要的函数调用关系数据的内核版本号、两个相关路径信息。路径数据可一个或全为'/'，表示请求根目录的函数调用图。

api定义： GET /api/v1/graphs

controller.action：app.controllers.v1.graphs.index

必要接收参数：

```json

params: {

version: "4-xx-xx",

source: "/xx",

target: "/xx"

}

```

相应状态码：200

返回数据格式：

```javascript

data：{

Nodes:[],

Edges:[]

}

```

### 1.4.1. 文件结构

```txt

/app

├─router.js

├─controller

│ └─v1

│ └─graphs.js

└─service

└─graphs.js

```

#### 1.4.1.1. 路由文件

app/router.js

路由文件中定义graph路由，根据 RESTful 风格规范，定义一下路由：

```javascript

router.resources('graphs', '/api/v1/graphs', controller.v1.graphs);

```

根据配置处理以下两种访问

GET | /graphs | app.controllers.v1.graphs.index

GET | /graphs/:id | app.controllers.v1.graphs.show

#### 1.4.1.2. 控制器文件

app/controllers/v1/graphs.js

控制器文件中，处理请求的输入数据，调用服务类中的相应函数进行处理

对于路由 /graphs 的访问，由控制器中 index() 函数进行处理

函数使用验证规则，对请求中包含的数据进行初步验证，防止异常输入和恶意输入

验证通过的数据，作为调用服务类中 test() 函数的输入参数

#### 1.4.1.3. 服务文件

app/service/graphs.js

服务文件，包含：处理请求数据，查找数据库，处理读取数据。

处理后的数据提供给页面模块生成函数调用图。

test() 函数，提供生成新函数调用图数据和生成函数调用图中节点展开的新增数据

### 1.4.2. 实现功能

1. 获取函数调用图数据

读取请求中的内核版本和生成函数调用图的路径，生成图表id，查询历史记录

使用内核版本号作为读取数据库表名的依据。

查找对应内核的FDLIST数据表，返回对应的下一级路径或文件内函数列表，同时使用递归方式逐级查找同级和上级路径。将这些路径转换成节点列表。

通过遍历节点列表，使用Promise构建的异步队列查询SOLIST表生成关联边列表。

返回生成的调用图数据，并将数据存入查询历史列表。

2. 获取内部效用图数据

与获取函数调用图数据方式相同，增加per关键字，再查找加点列表过程中不进行递归查找同级和上级路径。

3. 获取展开节点新增数据

通过请求中关键字expand进行判断，

## 1.5. 函数列表API设计

功能描述：通过此 API 获取函数调用列表数据，使用 http get 请求获取，在请求中须包含所需要的函数调用数据的内核版本号、两个相关路径信息。

api定义： GET /api/v1/functions

controller.action：app.controllers.v1.functions.index

必要接收参数：

```json

params: {

version: "X-xx-xx"

source: "/xx"

target: "/xx"

}

```

### 1.5.1. 文件结构

```txt

/app

├─router.js

├─controller

│ └─v1

│ └─functions.js

└─service

└─functions.js

```

#### 1.5.1.1. 服务文件

app/service/functions.js

服务文件，包含：处理请求数据，查找数据库，处理读取数据

test() 函数，提供生成新函数调用图数据和生成函数调用图中节点展开的新增数据

## 1.6. 配置获取API

功能描述：通过此 API 获取配置信息：服务器存在内核版本，文件夹列表等配置信息的数据。

api定义： GET /api/v1/options

controller.action：app.controllers.v1.options.index

## 1.7. API 缓存机制

对于数据获取API，设置缓存机制，能够对相同API、相同参数的请求，处理数据进行缓存，设置过期机制。

# 2. Web页面 Vue.js

## 2.1. 开发

```bash

# 克隆项目

git clone https://github.com/PanJiaChen/vue-element-admin.git

# 进入项目目录

cd vue-element-admin

# 安装依赖

npm install

# 建议不要直接使用 cnpm 安装依赖，会有各种诡异的 bug。可以通过如下操作解决 npm 下载速度慢的问题

npm install --registry=https://registry.npm.taobao.org

# 启动服务

npm run dev

```

浏览器访问 <http://localhost:9527>

## 2.2. 发布

```bash

# 构建测试环境

npm run build:stage

# 构建生产环境

npm run build:prod

```

## 2.3. VUE项目结构

```txt

/src

│ App.vue

│ main.js

│ permission.js

│ settings.js

│

├─components

│ └─Charts

│ │ ContextMenu.vue

│ │ demo.js

│ │ FunList.vue

│ │ Graph.vue

│

├─router

│ │ index.js

│ │

│ └─modules

│ charts.js

│ components.js

│ nested.js

│ table.js

│

└─views

├─charts

graph.vue

```

## 2.4. Graph 函数调用图

基于 Antv G6 实现的函数调用交互图，能够查看内核模块之间的关系。

通过页面上的一组选择器进行内核版本、源路径、目标路径的选择，生成对应的函数调用图

### 2.4.1. 页面文件结构

/src/views/charts/graph.vue为页面文件，文件中包含选择路径版本等选择器组件、函数调用图、函数调用表组件。

/components/Charts/Graph.vue

```txt

/src

├─components

│ └─Charts

│ │ ContextMenu.vue

│ │ FunList.vue

│ └─ Graph.vue

│

└─views

└─charts

└─graph.vue

```

### 2.4.2. 页面实现功能

#### 2.4.2.1. 可搜索下菜单选择路径

使用者选择内核版本后，页面会动态加载路径列表，

#### 2.4.2.2. 函数调用图查看

对于生成的函数调用图，查看效果实现有：

1. 画布缩放

2. 拖动整体

3. 拖动节点

4. 鼠标覆盖高亮

5. 详细信息气泡提示框显示

6. 缩略图

#### 2.4.2.3. 筛选查看

通过开关切换显示不同类型的边，切换类别为

1. 源为非所选路径节点

2. 目标为非所选路径节点

3. 源路径节点 到 目标路径节点

4. 目标路径节点 到 源路径节点

5. 源路径节点 到 源路径节点

6. 目标路径节点 到 目标路径节点

#### 2.4.2.4. 右键菜单

实现通过右键函数调用图中组件，弹出对应菜单

1. 右键节点

菜单内容为：

2. 右键边

菜单内容为：

#### 2.4.2.5. 动态加载部分更新数据

实现动态更新图中节点数据，将所选节点，更新为其下级路径内容

通过对节点右键菜单中的展开节点，对应函数获取节点信息，并向服务器请求当前节点的展开后新增数据，同时图中将此节点与其相连的边进行删除显示，获取到数据后，对图中数据进行加载，重新绘制当前函数调用图

## 2.5. Function List 函数列表

组件用于显示\*\*源路径\*\*中所包含函数调用\*\*目标路径\*\*中函数的调用列表

组件通过调用图边的右键菜单进行调出显示，显示表格内容如下

\# | 源函数 | 所在文件 | 行号 | 调用次数 | 调用行号 | 被调函数 | 所在文件 | 行号

-| - | - | - | - | - | - | - | -

## 2.6. 拖拽式页面布局

基于Vue组件 jbaysolutions/vue-grid-layout 实现拖拽页面组件，进行自定义布局。

使用在函数调用图页面，当前版本使用函数调用图和函数列表两个模块，能够对函数调用图进行尺寸调整（由于函数调用图模块内部需要相应鼠标拖拽，故当前版本删除拖拽函数调用图模块，但将其他可拖动模块放置在周围。），对函数调用列表进行拖拽和尺寸调整。

### 2.6.1. 安装组件

在项目路径下，使用包管理程序进行安装，命令如下

```shell

# 使用 npm

npm install vue-grid-layout --save

# 使用 yarn

yarn add vue-grid-layout

```

在要使用的页面模块中，引入安装好的布局模块

```javascript

import VueGridLayout from 'vue-grid-layout';

```

加入到页面 Vue 组件

```javascript

export default {

components: {

GridLayout: VueGridLayout.GridLayout,

GridItem: VueGridLayout.GridItem

},

// ... data, methods, mounted (), etc.

}

```

### 2.6.2. 使用组件

```html

<grid-layout

:layout.sync="web\_layout" // 布局配置数组

:col-num="12" // 总布局的列数

:row-height="100" // 每行的高度

:is-draggable="true" // 布局是否支持拖拽

:is-resizable="true" // 布vue局是否支持改变尺寸

:margin="[10, 10]" // 定义栅格中的元素边距

>

<grid-item v-for="item in layout" // 布局子项目组件

:x="item.x" // 布局子项目组件配置数据

:y="item.y"

:w="item.w"

:h="item.h"

:i="item.i"

:key="item.i">

{{item.i}}

</grid-item>

</grid-layout>

```

### 2.6.3. 实现功能

在函数调用图页面使用过程中，需要同时查看函数调用图，函数调用表等多个模块，为满足用户使用过程中对于页面布局的需求差异，实现自定义布局功能，具体如下：

#### 2.6.3.1. 函数调用图模块的调整尺寸

函数调用图模块作为查看函数调用的主要组件，内部图模块需要相应鼠标拖拽事件对图进行改变，故取消模块对拖拽布局的功能，能够通过模块右下角一个 “ ┛ ” 型图标对模块进行尺寸改变。同时对布局模块内的函数调用图进行大小调整，调整过程会对内部图进行重新绘制。

#### 2.6.3.2. 函数调用表宽度调整和拖拽布局

函数调用表模高度固定，对模块尺寸体现在宽度上，同样通过模块右下角 “ ┛ ” 型图标对模块进行尺寸改变。鼠标悬浮在调用表上时，显示为方向箭头，按住鼠标左键就可以将模块拖拽改变布局位置。

### 2.6.4. 设计实现

当前页面内容为两个模块，默认布局为上下两部分，当前设计能够通过改变模块尺寸，拖拽模块等操作来实现左右或交换上下位置的布局改变。

通过模块嵌套，将函数调用图和函数调用表模块作为布局模块的子项目，对其子项目进行不同的配置参数，用来实现子项目的不同功能，具体描述如下：

#### 2.6.4.1. 函数调用图子项目

函数调用图子项目主要特点为：不响应拖拽，能够改变尺寸，在模块尺寸改变后，还需要对函数调用图画布尺寸进行修改。

```html

<grid-item

v-if="show\_graph()"

:x="web\_layout[0].x"

:y="web\_layout[0].y"

:w="web\_layout[0].w"

:min-w="6"

:max-w="12"

:h="web\_layout[0].h"

:min-h="5"

:max-h="8"

:i="web\_layout[0].i"

:is-draggable="false"

@resized="resizedGraphEvent"

>

<Graph :layout="G\_layout" :config="config\_graph"/>

</grid-item>

```

在模块标签配置中增加 :is-draggable="false" 属性，关闭拖拽相应，增加 @resized="resizedGraphEvent" 事件，当尺寸改变后会触发 resizedGraphEvent() 函数

```javascript

resizedGraphEvent(i, newH, newW, newHPx, newWPx) {

// console.log('RESIZE i=' + i + ', H=' + newH + ', W=' + newW + ', H(px)=' + newHPx + ', W(px)=' + newWPx)

this.config\_graph.h = Math.floor(newHPx - 40)

this.config\_graph.w = Math.floor(newWPx)

}

```

resizedGraphEvent() 函数将模块尺寸改变后的宽、高数据，赋值给函数调用图模块输入的配置数据。

在函数调用图模块中，会监控 config 中属性的变化，如果时宽、高属性发生了变化，会通过销毁当前画布，以新的画布尺寸进行重新绘图。

```javascript

watch: {

config: {

handler(newValue, oldValue) {

const \_t = this

if (newValue.h !== \_t.graph\_h || newValue.w !== \_t.graph\_w) {

\_t.graph\_h = newValue.h

\_t.graph\_w = newValue.w

\_t.graph.destroy()

\_t.initChart()

\_t.graph.data(\_t.data)

\_t.graph.render()

} else {

this.getdata('new')

}

},

deep: true

}

```

#### 2.6.4.2. 函数调用表子项目

函数调用表子项目能够拖拽，能够改变尺寸，因此配置属性中，只需要对其位置和大小限制进行配置，无需增加相应事件。

```html

<grid-item

v-show="funlist\_show"

:x="web\_layout[1].x"

:y="web\_layout[1].y"

:w="web\_layout[1].w"

:min-w="3"

:max-w="12"

:h="web\_layout[1].h"

:min-h="5"

:max-h="5"

:i="web\_layout[1].i"

>

<FunList :config="config\_funlist" />

</grid-item>

```