途虎养车

养车 就是途虎

# Vue Component实用技巧

马珩

# 个人介绍

组织架构：终端技术中心-商业应用前端组-移动开发组

近期主要工作：参与CRM的小程序转H5部分组件库和api开发

# 大纲总览

- 组件注册
- 动态组件
- 获取组件实例
- 修饰符
- $attrs、$listeners运用
- Q&A

# 组件注册

# 局部注册

```
//引入
import AComponent from './xx/a-component.vue'

components: {
    AComponent
}


//使用，在template中
<a-component></a-component>
```

使用开发好的组件，可能是抽离出来的一些可复用的部分

# 全局注册

```
// 引入
import AComponent from '../xxx/a-component'

// 全局注册
Vue.component('AComponent',AComponent)
```

一些基础组件，全局注册后直接使用，不需要导入

# 基础组件很多，组件库？

```
// 引入
import AComponent from '../xx/a-component'
import BComponent from '../xx/b-component'
import CComponent from '../xx/c-component'
import DComponent from '../xx/d-component'
import EComponent from '../xx/e-component'

// 依次全局注册
Vue.component('a-component',AComponent)
Vue.component('b-component',BComponent)
Vue.component('c-component',CComponent)
Vue.component('d-component',DComponent)
Vue.component('e-component',EComponent)
```

# 注册技巧之——require.context

```
const requireComponent = require.context(
    // 其组件目录的相对路径
    '../components',
    // 是否查询其子目录
    false,
    // 匹配基础组件文件名的正则表达式
    /\w+\.(vue|js)$/
  )
```

自动化导入基础组件\路由

# require.context是什么？

概念：一个webpack的API,获取一个特定的上下文。

使用场景：主要用来实现自动化导入模块,在前端工程中,如果遇到从一个文件夹引入很多模块的情况,可以使用这个API,它会遍历文件夹中的指定文件,然后自动导入,不需要每次显式的调用import导入模块

# 接受三个参数

1.directory {String} -读取文件的路径

2.useSubdirectories {Boolean} -是否遍历文件的子目录

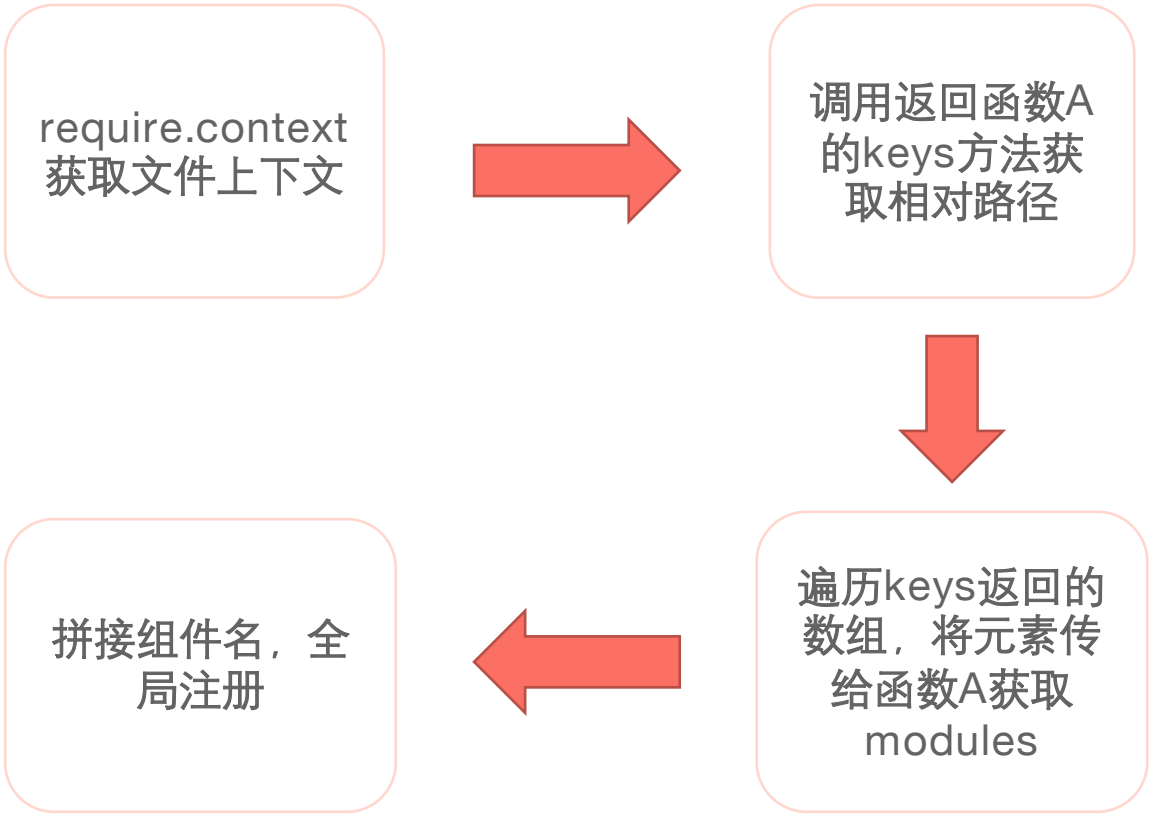
3.regExp {RegExp} -匹配文件的正则

# 返回一个函数，这个函数又接收三个参数

1.resolve {Function} -接受一个参数request,传入相对路径，返回绝对路径）

2.keys {Function} -返回匹配成功模块的名字组成的数组

3.id {String} -执行环境的id,返回的是一个字符串


require.context
获取文件上下文
调用返回函数A
的keys方法获
取相对路径
遍历keys返回的
数组，将元素传
给函数A获取
modules
拼接组件名，全
局注册

# 动态组件

# 场景

小型单页应用，可能并不需要用vue-router控制路由，简单切换组件展示不同的效果。

# 获取组件实例

# ref引用

```
// template
<a-component ref="aaa"></a-component>

//javascript
const vm = this.$refs.aaa
```

通过ref引用获取组件实例

# $parent \ $children

```javascript
//javascript
const parent = this.$parent
const children = this.$children
```

获取父实例和子实例

# provide\inject

```
// 父组件provide出this
provide(){
    return {
      myself: this
    }
  }

// 子组件使用
inject:['myself']
```

跨层级组件通信，非常方便

# 修饰符

# .sync

语法糖，类似v-model，替代v-on事件监听

```
// 不使用sync
<a-component :foo="bar" v-on:update:foo="bar = $event" /> // 父组件
this.$emit('update:foo',this.newVal) // 子组件

// 使用sync
<a-component :foo.sync="bar" />
```

# .native

在组件根元素上直接监听一个原生事件

```
<a-component @click="handleClick" /> // 无效

<a-component @click.native="handleClick" />// 有效
```

# .once

事件只能触发一次，可以用在自定义组件事件上

```
<!-- 点击事件将只会触发一次 -->
<a v-on:click.once="doThis"></a>
```

# .passive

不再拦截默认事件，用于优化浏览器页面滚动的性能，让页面滚动更顺滑

```
<!-- 滚动事件的默认行为（即滚动行为）将会立即触发 -->
<!-- 而不会等待 `onScroll` 完成  -->
<!-- 这其中包含 `event.preventDefault()` 的情况 -->
<div v-on:scroll.passive="onScroll">...</div>
```

# $attrs、$listeners运用

# $attrs

利用$attrs 将原生属性直接传递给子组件,当一个组件没有声明任何 prop 时，这里会包含所有父作用域的绑定 (class 和 style 除外)，并且可以通过 v-bind="$attrs" 传入内部组件

```
<!-- bad -->
<input :name="name" :placeholder="placeholder" :disabled="disabled">

<!-- good -->
<input v-bind="$attrs">
```

# $listeners

包含了父作用域中的 (不含 .native修饰器的) v-on 事件监听器，可以通过 v-on="$listeners" 传入内部组件

```
<!-- bad -->
<a-component @eventOne="methodOne" @eventTwo="methodTwo"
@eventThree="methodThree" />

<!-- good -->
<a-component v-on="$listeners" />
```

# Q&A

# 途虎养车

养 车　就 是 途 虎