**目录**

[vueJS开发注意点](#_vueJS开发注意点)

[vue-cli的使用](#_Vue-cli的使用)

[vue-i18n的使用](#_vue-i18n的使用)

[vue 2 使用Bus.js进行兄弟(非父子)组件通信](#_vue_2_使用Bus.js进行兄弟(非父子)组件通信)

[vue自带组件](#_Vue自带组件)

[router-link](#_router-link)

[router-view](#_router-view)

[vue模板语法.html](#_vue模板语法.html)

[vue的component使用.html](#_vue的component全局使用.html)

[vue的component全局使用.html](#_vue的component全局使用.html_1)

[vue的component全局使用2.html](#_vue的component全局使用2.html)

[vue的component局部使用.html](#_vue的component局部使用.html)

[vue的extend扩展实例构造器.html](#_vue的extend扩展实例构造器.html)

[vue的混入mixins.html](#_vue的混入mixins.html)

[vue的props属性](#_Vue的props属性)

[vue组件中prop自定义属性.html](#_vue组件中prop自定义属性.html)

[vue动态props.html](#_vue动态prop.html)

[父子组件props的使用.html](#_父子组件props的使用.html)

[v-for循环中使用ref.html](#_v-for循环中使用ref.html)

# vueJS开发注意点

1、computed里配的属性，如果依赖了其他属性，给该属性赋值时，注意添加setter方法；

2、this.$nextTrick(function(){});Vue实例里属性都响应over后，会触发其中的回调函数；

# Vue-cli的使用

1、本地安装node.js

2、安装vue

Vue.js提供了两种不同的安装方式：

1）直接引入<script>的方式（此处不做介绍了）

2）Vue-cli脚手架，官方的命令行工具，用于快速构建具有热重载、vuex状态管理、router路由等配置的项目。

安装Vue-cli

npm install --global vue-cli

使用脚手架搭建工程，执行以下命令：

vue init webpack my-project

cd my-project

npm install

若第一步网络无法下载webpack模板，可以先去网站https://github.com/vuejs-templates下载模板，再本地离线安装。

模板下载后，解压到目录C:\Users\yanfangfang/.vue-templates下。

再执行第一步的命令，后面加上--offline

vue init webpack my-project --offline

npm run dev

# vue-i18n的使用

vue-i18n

仓库地址：https://github.com/kazupon/vue-i18n

兼容性：

支持 Vue.js 2.x 以上版本

安装方法：（此处只演示 npm）

npm install vue-i18n

使用方法：

1、在 main.js 中引入 vue-i18n （前提是要先引入 vue）

import VueI18n from 'vue-i18n'

Vue.use(VueI18n)

2、准备本地的翻译信息

const messages = {

zh: {

message: {

hello: '好好学习，天天向上！'

}

},

en: {

message: {

hello: 'good good study, day day up!'

}

}

}

3、创建带有选项的 VueI18n 实例

const i18n = new VueI18n({

locale: 'en', // 语言标识

messages

})

4、把 i18n 挂载到 vue 根实例上

const app = new Vue({

router,

i18n,

...App

}).$mount('#app')

5、在 HTML 模板中使用

<div id="app">

<h1 style="font-size: 16px; text-align: center;">{{ $t("message.hello") }}</h1>

</div>

我们刚才定的语言标识是 “en” 英语，现在改成 “zh” 中文:

const i18n = new VueI18n({

locale: 'zh', // 语言标识

messages

})

这样就可以轻松实现国际化了，实际开发中，页面内容肯定是很多的，我们可以把对应语言的信息保存为不同的 json对象

const i18n = new VueI18n({

locale: 'en', // 语言标识

messages: {

'zh': require('./common/lang/zh'),

'en': require('./common/lang/en')

}

})

zh.js

// 注意：一定是 exports，不是 export，否则会报错，报错信息是下列的中的内容不是 string

module.exports = {

message: {

title: '运动品牌'

},

placeholder: {

enter: '请输入您喜欢的品牌'

},

brands: {

nike: '耐克',

adi: '阿迪达斯',

nb: '新百伦',

ln: '李宁'

}

}

en.js

module.exports = {

message: {

title: 'Sport Brands'

},

placeholder: {

enter: 'Please type in your favorite brand'

},

brands: {

nike: 'Nike',

adi: 'Adidas',

nb: 'New Banlance',

ln: 'LI Ning'

}

}

接下来，在HTML 模板中使用，要特别注意在 js 中的国际化写法

// HTML

<div id="app">

<div style="margin: 20px;">

<h1>{{$t("message.title")}}</h1>

<input style="width: 300px;" class="form-control" :placeholder="$t('placeholder.enter')">

<ul>

<li v-for="brand in brands">{{brand}}</li>

</ul>

</div>

</div>

// JS

data () {

return {

brands: [this.$t('brands.nike'), this.$t('brands.adi'), this.$t('brands.nb'), this.$t('brands.ln')]

}

}

在上面的操作中，我们都是通过手动修改 locale 的属性值来切换语言，实际上，更希望浏览器自动识别，这里可以借助于 cookie

1、新建一个 cookie.js 文件，用于读取cookie

function getCookie(name,defaultValue) {

var arr, reg = new RegExp("(^| )" + name + "=([^;]\*)(;|$)");

if (arr = document.cookie.match(reg))

return unescape(arr[2]);

else

return defaultValue;

}

export {

getCookie

}

2、在 main.js 中引入这个js，并通过 PLAY\_LANG 属性来获取浏览器的语言

const i18n = new VueI18n({

locale: getCookie('PLAY\_LANG','zh'),// 语言标识

messages: {

'zh': require('./common/lang/zh'),

'en': require('./common/lang/en')

}

})

这里需要注意两个极易出错的地方：

（1）、将 $t() 写成了 $()

（2）、json 中在同一个对象里有同名属性

vue-i18n 提供了一个全局配置参数叫 “locale”，通过改变 locale 的值可以实现不同语种的切换

下面的案例借用了 Element UI 的弹窗样式，上面的步骤不再赘述，直接上核心代码

// template

<h2>{{$t('test')}}</h2>

<button type="button" class="btn btn-success" @click="changeLocale">中文/EN</button>

// js方法

changeLocale () {

this.$confirm(this.$t('layer.toggle'), this.$t('layer.tips'), {

confirmButtonText: this.$t('button.ok'),

cancelButtonText: this.$t('button.cancel'),

type: 'warning'

}).then(() => {

let locale = this.$i18n.locale

locale === 'zh' ? this.$i18n.locale = 'en' : this.$i18n.locale = 'zh'

}).catch(() => {

this.$message({

type: 'info',

})

})

}

在配合 Element-UI 一起使用时，会有2个问题：

（1）、页面刷新后，通过按钮切换的语言还原成了最初的语言，无法保存

（2）、框架内部自带的提示文字无法更改，比如像时间选择框内部中的提示文字

关于第一个问题，可以在初始化VueI18n实例时，通过 localStorage 来为 locale 对象赋值

const i18n = new VueI18n({

locale: localStorage.getItem('locale') || 'zh',

messages

})

在切换语言的时候可以缓存不同的语言选项，并且可以长期保存，不会因为刷新网页而改变locale 的属性值

<div class="lang">

<el-dropdown>

<i class="iconfont icon-language4"></i>

<el-dropdown-menu slot="dropdown">

<el-dropdown-item @click.native="toggleLang('zh')" :disabled="$i18n.locale == 'zh'">中文</el-dropdown-item>

<el-dropdown-item @click.native="toggleLang('en')" :disabled="$i18n.locale == 'en'">English</el-dropdown-item>

</el-dropdown-menu>

</el-dropdown>

</div>

toggleLang(lang) {

if(lang == 'zh'){

localStorage.setItem('locale', 'zh')

this.$i18n.locale = localStorage.getItem('locale')

this.$message({

message: '切换为中文！',

type: 'success'

})

} else if (lang == 'en') {

localStorage.setItem('locale', 'en')

this.$i18n.locale = localStorage.getItem('locale')

this.$message({

message: 'Switch to English!',

type: 'success'

})

}

}

关于第二个问题，更改Element 组件内部语言，这里还涉及到 手动处理 vue-i18n@6.x 兼容性问题。官网已经做了详细介绍，这里依葫芦画瓢跟着实现一下

i18n.js

import Vue from 'vue'

import VueI18n from 'vue-i18n'

import locale from 'element-ui/lib/locale';

import zh from './langs/zh'

import en from './langs/en'

import enLocale from 'element-ui/lib/locale/lang/en'

import zhLocale from 'element-ui/lib/locale/lang/zh-CN'

Vue.use(VueI18n)

const messages = {

en: Object.assign(en, enLocale),

zh: Object.assign(zh, zhLocale)

}

console.log(messages.zh)

const i18n = new VueI18n({

locale: localStorage.getItem('locale') || 'zh',

messages

})

locale.i18n((key, value) => i18n.t(key, value)) //为了实现element插件的多语言切换

export default i18n

照如上把国际化文件都整合到一起，避免main.js 中大段引入相关代码。main.js 中与 i18n 相关的就只剩两行代码

import i18n from './i18n/i18n'// 1行

window.app = new Vue({

el: '#app',

router,

store,

i18n, // 2行

components: { App },

template: '<App/>'

})

以上demo地址：<https://github.com/frwupeng517/element-admin/tree/master/src/i18n>

# vue 2 使用Bus.js进行兄弟(非父子)组件通信

简单案例

  vue2中废弃了$dispatch和$broadcast广播和分发事件的方法。父子组件中可以用props和$emit()。如何实现非父子组件间的通信，可以通过实例一个vue实例Bus作为媒介，要相互通信的兄弟组件之中，都引入Bus，之后通过分别调用Bus事件触发和监听来实现组件之间的通信和参数传递。

首先需要在任意地方添加一个bus.js

在bus.js里面 写入下面信息

import Vue from 'vue'

export default new Vue;

在需要通信的组件里都引入Bus.js

如果你的bus.js是自定义一个bus的文件，那from后面就改成你的所放的位置

import Bus from './bus.js'

接下来就是要组件通信了

添加一个 触发 #emit的事件按钮

<template>

<div id="emit"><button @click="bus">按钮</button></div>

</template>

import Bus from './bus.js'

export default {

data() {

return {

message: ''"

}

},

methods: {

bus () {

Bus.$emit('msg', '我要传给兄弟组件们，你收到没有')

}

}}

打开要和$emit通信的另外一个组件

在钩子函数中监听msg事件

<template>

<div id="on">

<p>{{message}}</p>

</div>

</template>

import Bus from './bus.js'

export default {

data() {

return {

message: ''

}

},

mounted() {

let self = this

Bus.$on('msg', (e) => {

self.message = e;

console.log(`传来的数据是：${e}`)

})

}

}

最后<p>会显示来自$emit传来的信息

<router-link>

<router-link> 组件支持用户在具有路由功能的应用中 (点击) 导航。 通过 to 属性指定目标地址，默认渲染成带有正确链接的 <a> 标签，可以通过配置 tag 属性生成别的标签.。另外，当目标路由成功激活时，链接元素自动设置一个表示激活的 CSS 类名。

<router-link> 比起写死的 <a href="..."> 会好一些，理由如下：

1)无论是 HTML5 history 模式还是 hash 模式，它的表现行为一致，所以，当你要切换路由模式，或者在 IE9 降级使用 hash 模式，无须作任何变动。

2)在 HTML5 history 模式下，router-link 会守卫点击事件，让浏览器不再重新加载页面。

3)当你在 HTML5 history 模式下使用 base 选项之后，所有的 to 属性都不需要写 (基路径) 了。

# Vue自带组件

## router-link

<router-link> Props

to

类型: string | Location

required

表示目标路由的链接。当被点击后，内部会立刻把 to 的值传到 router.push()，所以这个值可以是一个字符串或者是描述目标位置的对象。

eg:

<!-- 字符串 -->

<router-link to="home">Home</router-link>

<!-- 渲染结果 -->

<a href="home">Home</a>

<!-- 使用 v-bind 的 JS 表达式 -->

<router-link v-bind:to="'home'">Home</router-link>

<!-- 不写 v-bind 也可以，就像绑定别的属性一样 -->

<router-link :to="'home'">Home</router-link>

<!-- 同上 -->

<router-link :to="{ path: 'home' }">Home</router-link>

<!-- 命名的路由 -->

<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>

<!-- 带查询参数，下面的结果为 /register?plan=private -->

<router-link :to="{ path: 'register', query: { plan: 'private' }}">Register</router-link>

replace

类型: boolean

默认值: false

设置 replace 属性的话，当点击时，会调用 router.replace() 而不是 router.push()，于是导航后不会留下 history 记录。

eg:

<router-link :to="{ path: '/abc'}" replace></router-link>

append

类型: boolean

默认值: false

设置 append 属性后，则在当前 (相对) 路径前添加基路径。例如，我们从 /a 导航到一个相对路径 b，如果没有配置 append，则路径为 /b，如果配了，则为 /a/b

eg:

<router-link :to="{ path: 'relative/path'}" append></router-link>

tag

类型: string

默认值: "a"

有时候想要 <router-link> 渲染成某种标签，例如 <li>。 于是我们使用 tag prop 类指定何种标签，同样它还是会监听点击，触发导航。

eg:

<router-link to="/foo" tag="li">foo</router-link>

<!-- 渲染结果 -->

<li>foo</li>

exact

类型: boolean

默认值: false

"是否激活" 默认类名的依据是 inclusive match (全包含匹配)。 举个例子，如果当前的路径是 /a 开头的，那么 <router-link to="/a"> 也会被设置 CSS 类名。

eg:

按照这个规则，每个路由都会激活<router-link to="/">！想要链接使用 "exact 匹配模式"，则使用 exact 属性：

<!-- 这个链接只会在地址为 / 的时候被激活 -->

<router-link to="/" exact>

## router-view

<router-view> 组件是一个 functional 组件，渲染路径匹配到的视图组件。<router-view> 渲染的组件还可以内嵌自己的 <router-view>，根据嵌套路径，渲染嵌套组件。

其他属性 (非 router-view 使用的属性) 都直接传给渲染的组件， 很多时候，每个路由的数据都是包含在路由参数中。

因为它也是个组件，所以可以配合 <transition> 和 <keep-alive> 使用。如果两个结合一起用，要确保在内层使用 <keep-alive>：

<transition>

<keep-alive>

<router-view></router-view>

</keep-alive>

</transition>

name

类型: string

默认值: "default"

如果 <router-view>设置了名称，则会渲染对应的路由配置中 components 下的相应组件。查看 命名视图 中的例子。

----------------------------------------------------------------------------------------------------------------------------------------------------------------

----------------------------------------------------------------------------------------------------------------------------------------------------------------

Router 构建选项

类型: Array<RouteConfig>

RouteConfig 的类型定义：

declare type RouteConfig = {

path: string;

component?: Component;

name?: string; // 命名路由

components?: { [name: string]: Component }; // 命名视图组件

redirect?: string | Location | Function;

props?: boolean | Object | Function;

alias?: string | Array<string>;

children?: Array<RouteConfig>; // 嵌套路由

beforeEnter?: (to: Route, from: Route, next: Function) => void;

meta?: any;

// 2.6.0+

caseSensitive?: boolean; // 匹配规则是否大小写敏感？(默认值：false)

pathToRegexpOptions?: Object; // 编译正则的选项

}

mode

类型: string

默认值: "hash" (浏览器环境) | "abstract" (Node.js 环境)

可选值: "hash" | "history" | "abstract"

配置路由模式:

hash: 使用 URL hash 值来作路由。支持所有浏览器，包括不支持 HTML5 History Api 的浏览器。

history: 依赖 HTML5 History API 和服务器配置。查看 HTML5 History 模式。

abstract: 支持所有 JavaScript 运行环境，如 Node.js 服务器端。如果发现没有浏览器的 API，路由会自动强制进入这个模式。

base

类型: string

默认值: "/"

应用的基路径。例如，如果整个单页应用服务在 /app/ 下，然后 base 就应该设为 "/app/"。

linkActiveClass

类型: string

默认值: "router-link-active"

全局配置 <router-link> 的默认“激活 class 类名”。参考 router-link。

# vue模板语法.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<style>

**.class1{**

**background: #444;**

**color: #eee;**

**}**

</style>

<body>

<!--

Vue.js 使用了基于 HTML 的模版语法，允许开发者声明式地将 DOM 绑定至底层 Vue 实例的数据。

Vue.js 的核心是一个允许你采用简洁的模板语法来声明式的将数据渲染进 DOM 的系统。

结合响应系统，在应用状态改变时， Vue 能够智能地计算出重新渲染组件的最小代价并应用到 DOM 操作上。

-->

<div id=**"app"**>

<!-- 数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值 -->

<p>**{{ message }}**</p>

</div>

<br>

<br>

<div id=**"app2"**>

<div v-html=**"message"**></div>

</div>

<br>

<br>

<div id=**"app3"**>

<label for=**"r1"**>**修改颜色**</label>

<input type=**"checkbox"** v-model=**"use"** id=**"r1"**>

<br>

<br>

<!-- HTML 属性中的值应使用 v-bind 指令 -->

<div v-bind:class=**"{'class1': use}"**>

**v-bind:class 指令**

</div>

</div>

<br>

<br>

<div id=**"app4"**>

**{{5+5}}**<br>

**{{ ok ? 'YES' : 'NO' }}**<br>

**{{ message.split('').reverse().join('') }}**

<div v-bind:id=**"'list-' + id"**>**菜鸟教程**</div>

</div>

<!--

指令是带有 v- 前缀的特殊属性。

指令用于在表达式的值改变时，将某些行为应用到 DOM 上

-->

<div id=**"app5"**>

<p v-if=**"seen"**>**现在你看到我了**</p>

</div>

<br>

<br>

<div id=**"app6"**>

<!--

参数在指令后以冒号指明

v-bind 指令被用来响应地更新 HTML 属性, href 是参数，告知 v-bind 指令将该元素的 href 属性与表达式 url 的值绑定，缩写:

-->

<pre><a v-bind:href=**"url"**>**菜鸟教程**</a></pre>

<!-- v-on 指令，它用于监听 DOM 事件，缩写@ -->

<a v-on:click=**"doSomething"**>**点击事件**</a>

</div>

<script>

***new*** Vue**({**

el**:** '#app'**,**

data**:** **{**

message**:** '菜鸟教程'

**}**

**});**

***new*** Vue**({**

el**:** '#app2'**,**

data**:** **{**

message**:** '<h1>菜鸟教程</h1>'

**}**

**});**

***new*** Vue**({**

el**:** '#app3'**,**

data**:{**

use**:** ***false***

**}**

**});**

***new*** Vue**({**

el**:** '#app4'**,**

data**:** **{**

ok**:** ***true*,**

message**:** 'RUNOOB'**,**

id **:** 1

**}**

**});**

***new*** Vue**({**

el**:** '#app5'**,**

data**:** **{**

seen**:** ***true***

**}**

**});**

***new*** Vue**({**

el**:** '#app6'**,**

data**:** **{**

url**:** 'http://www.runoob.com'

**},**

methods**:{**

doSomething**:*function*(){**

alert**(**"doSomething"**);**

**}**

**}**

**});**

</script>

</body>

</html>

# vue的component全局使用.html

## vue的component全局使用.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<runoob></runoob>

</div>

<div id=**"app2"**>

<runoob></runoob>

</div>

<!-- <runoob></runoob> 全局组件需使用在Vue实例中-->

<script>

// 注册

Vue.component**(**'runoob'**,** **{**

template**:** '<h1>自定义组件!</h1>'

**})**

// 创建根实例

***new*** Vue**({**

el**:** '#app'

**});**

// 创建根实例

***new*** Vue**({**

el**:** '#app2'

**});**

</script>

</body>

</html>

## vue的component全局使用2.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<runoob></runoob>

</div>

<div id=**"name"**></div>

<div id=**"app2"**>

<runoob></runoob>

</div>

<!-- <runoob></runoob> 全局组件需使用在Vue实例中-->

<script>

// 注册

let runoob **=** Vue.extend**({**

template**:** '<h1>自定义组件!</h1>'

**});**

Vue.component**(**'runoob'**,** runoob**)**

// 创建根实例

***new*** Vue**({**

el**:** '#app'

**});**

// 创建根实例

***new*** Vue**({**

el**:** '#app2'

**});**

</script>

</body>

</html>

## vue的component局部使用.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<runoob></runoob>

</div>

<!-- <runoob></runoob> 局部组件需在Vue实例中定义-->

<script>

***var*** Child **=** **{**

template**:** '<h1>自定义组件!</h1>'

**}**

// 创建根实例

***new*** Vue**({**

el**:** '#app'**,**

components**:** **{**

// <runoob> 将只在父模板可用

'runoob'**:** Child

**}**

**})**

</script>

</body>

</html>

## vue的extend扩展实例构造器.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script type=**"text/javascript"** src=**"js/vue.js"**></script>

</head>

<body>

<author></author>

<div id=**"app"**></div>

<script type=**"text/javascript"**>

// 使用Vue构造器，创建一个“子类”，参数是一个包含组件选项的对象，其中,data选项中必须是函数。

// 返回的是一个“扩展实例构造器”，也就是预设了部分选项的Vue的实例构造器

**var** authorExtend **=** Vue.extend**({**

template**:**"<p><a :href='authorURL'>{{authorName}} & {{name}}</a></p>"**,**

data**:function(){**

**return** **{**

authorName**:**"JSPang"**,**

authorURL**:**"http://www.runoob.com"

**}**

**},**

props **:** **[**'name'**]**

**});**

// 扩展实例构造器还需要挂载

// 对于同一个扩展构造器而言，它的每一个实例其实是可以挂载到不同的标签上的

// 利用propsData传递参数

**var** ae **=** **new** authorExtend**({**propsData**:{**name**:**'all developers'**}}).**$mount**(**'author'**);**

**var** ae2 **=** **new** authorExtend**({**propsData**:{**name**:**'all developers22'**}}).**$mount**(**'#app'**);**

</script>

</body>

</html>

## vue的混入mixins.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<!--

<div id = "databinding"></div>

<div id = "databinding2"></div>

-->

<script type = **"text/javascript"**>

//定义的Vue对象对混入对象没有作用

//var vm = new Vue({

// el: '#databinding',

// data: {},

// methods : {}

//});

//var vm2 = new Vue({

// el: '#databinding2',

// data: {},

// methods : {}

//});

// 定义一个混入对象

***var*** myMixin **=** **{**

created**:** ***function*** **()** **{**

this.startmixin**();** //this指当前的混入对象

**},**

methods**:** **{**

startmixin**:** ***function*** **()** **{**

document.write**(**"欢迎来到混入实例"**);**

**}**

**}**

**};**

***var*** Component **=** Vue.extend**({**

mixins**:** **[**myMixin**]**

**})**

***var*** component **=** ***new*** Component**();**

***var*** component2 **=** ***new*** Component**();**

</script>

</body>

</html>

# Vue的props属性

## vue组件中prop自定义属性.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<!--

prop 是父组件用来传递数据的一个自定义属性。

父组件的数据需要通过 props 把数据传给子组件，子组件需要显式地用 props 选项声明 "prop"

-->

<child message="hello!"></child>

</div>

<script>

// 注册

Vue.component**(**'child'**,** **{**

// 声明 props

props**:** **[**'message'**],**

// 同样也可以在 vm 实例中像 “this.message” 这样使用

template**:** '<span>{{ message }}</span>'

**})**

// 创建根实例

***new*** Vue**({**

el**:** '#app'

**})**

</script>

</body>

</html>

## vue动态prop.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<div>

<input v-model=**"parentMsg"**>

<br>

<!-- 用 v-bind 动态绑定 props 的值到父组件的数据中。每当父组件的数据变化时，该变化也会传导给子组件 -->

<child v-bind:message="parentMsg"></child>

</div>

</div>

<div id=**"app2"**>

<ol>

<!-- v-bind 指令将 todo 传到每一个重复的组件中 -->

<todo-item v-for="item in sites" v-bind:todo="item"></todo-item>

</ol>

</div>

<script>

// 注册

Vue.component**(**'child'**,** **{**

// 声明 props

props**:** **[**'message'**],**

// 同样也可以在 vm 实例中像 “this.message” 这样使用

template**:** '<span>{{ message }}</span>'

**})**

// 创建根实例

***new*** Vue**({**

el**:** '#app'**,**

data**:** **{**

parentMsg**:** '父组件内容'

**}**

**});**

Vue.component**(**'todo-item'**,** **{**

props**:** **[**'todo'**],**

template**:** '<li>{{ todo.text }}</li>'

**});**

***new*** Vue**({**

el**:** '#app2'**,**

data**:** **{**

sites**:** **[**

**{** text**:** 'Runoob' **},**

**{** text**:** 'Google' **},**

**{** text**:** 'Taobao' **}**

**]**

**}**

**});**

</script>

</body>

</html>

## 父子组件props的使用.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"example"**>

<parent></parent>

</div>

<script>

/\*

props为数组时，所有属性都为String默认类型

父组件里使用该属性isFlag时，需要用v-bind:isFlag(:isFlag)，直接用isFlag，报Expected Boolean, got String.

父组件里使用该属性id时，需要用v-bind:id(:id)，直接用id，报Expected Number, got String.

父组件里使用该属性title时，v-bind:title(:title)（可与父组件交互）、title（父传给子固定的一个值）都可以用

\*/

***var*** childNode **=** **{**

template**:** `

**<**div**>{{**id**}},{{**title**}},{{**num**}},{{**isFlag**}}**

**<**ul**>**

**<**li v**-*for*=**"name in names"**>{{**name**}}**</li>

</ul>

</div>**`,**

props:{

'isFlag': Boolean,

'id': Number,

'title':String,

'num': {

'type': Number,

'default': 1001

},

'names': {

'type': Array,

'default': function(){

return [11];

}

}

}

};

var parentNode = {

template: `

<div class=**"parent"**>

<!--

title：直接传值；

names：动态绑定值，父组件改变值子组件自动渲染；

id、isFlag：父组件使用该子组件的属性时需要用:id,:isFlag(不能直接id,isFlag)

-->

<child v-bind:id="20190222" title="测试test" :isFlag="true" :names="pNames"></child>

<button @click=**"change"**>**change**</button>

</div>**`,**

components: {

'child': childNode

},

data(){

return{

msg: 123,

pNames:["yff","cb"]

}

},

methods: {

change: function(){

console.log("click....");

// 子组件names属性在父组件里使用了v-bind动态绑定，父组件改变值子组件也会自动渲染

this.pNames = ["change1","change2"];

}

}

};

// 创建根实例

new Vue({

el: '#example',

components: {

'parent': parentNode

}

})

</script>

</body>

</html>

# v-for循环中使用ref.html

<!DOCTYPE html>

<html>

<head>

<meta charset=**"utf-8"**>

<title>**Vue 测试实例 - 菜鸟教程(runoob.com)**</title>

<script src=**"js/vue.js"**></script>

</head>

<body>

<div id=**"app"**>

<div>

<ul>

<li v-for=**"childMsg in childMsgs"**>

<child :ref="childMsg.ref" v-bind:message="childMsg.msg"></child>

</li>

</ul>

<button name=**""** @click=**"click"**>**get**</button>

</div>

</div>

<script>

// 注册

Vue.component**(**'child'**,** **{**

// 声明 props

props**:** **[**'message'**],**

// 同样也可以在 vm 实例中像 “this.message” 这样使用

template**:** '<span @click="getMessage">{{ message }}</span>'**,**

methods**:{**

getMessage**:*function*(){**

alert**(**this.message**);**

**}**

**}**

**})**

// 创建根实例

***new*** Vue**({**

el**:** '#app'**,**

data**:** **{**

childMsgs**:[**

**{**

ref**:**'childRef1'**,**

msg**:**'Hello ChildRef1!'

**},{**

ref**:**'childRef2'**,**

msg**:**'Hello ChildRef2!'

**},{**

ref**:**'childRef3'**,**

msg**:**'Hello ChildRef3!'

**}**

**]**

**},**

methods**:{**

click**:*function*(){**

***for*(**let index ***in*** this.childMsgs**){**

//当 v-for 用于元素或组件的时候，引用信息将是包含 DOM 节点或组件实例的数组。

this.$refs**[**this.childMsgs**[**index**].**ref**][**0**].**getMessage**();**

**}**

**}**

**}**

**});**

</script>

</body>

</html>