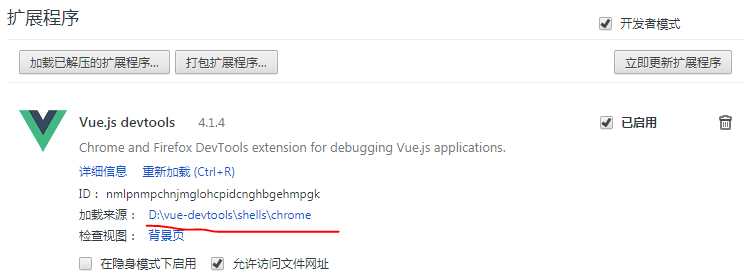
1. **安装Vue Devtools**
2. 找到vue-devtools的github项目，并将其clone到本地某个文件夹下，利用Git命令：git **clone** **[https](https://github.com/vuejs/vue-devtools.git)**[://github.com/vuejs/vue-devtools.git](https://github.com/vuejs/vue-devtools.git)
3. 在本地clone下来的vue-devtools文件夹下，安装项目依赖的npm包，利用npm命令：npm install
4. 然后编译项目文件，利用npm命令：npm run build
5. 最后在浏览器中添加扩展程序。先在游览器输入地址“chrome://extensions/”进入扩展程序页面，点击“加载已解压的扩展程序...”按钮，选择vue-devtools>shells下的chrome文件夹。  
   （5）如果添加成功，则效果图如下，然后在调试vue项目时，就可以在调试面板中看到多了“Vue”这个选项



1. **在html文件中利用<script>标签引入Vue，Vue会被注册为一个全局变量**

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>

注意：https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/链接中包含vue的不同构建版本，可以自主选择版本名称



1. **声明式渲染**
2. **DOM文本插值**：采用简洁的模板语法来声明式地将数据渲染进DOM文档，这样数据和DOM就建立了响应式的关联

首先建立三个文件如下：

***index.html文件***

<html>

<head>

<title>DOM文本插值</title>

*<!-- 引入vue，这样Vue就被声明为全局变量 -->*

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.16/dist/vue.js"></script>

*<!-- 引入外部css文件 -->*

<link href="index.css" rel="stylesheet">

</head>

<body>

*<!-- 模板语法，将message变量值渲染进DOM -->*

<div id="app">

{{ message }}

</div>

*<!-- 引入的外部js文件必须放在页面最后，否则会报错 -->*

<script src="index.js"></script>

</body>

</html>

***index.js文件***

*//* 创建一个Vue实例，其中el属性代表要渲染的元素，data属性代表绑定的数据

*//* el 、data 名字都是固定的，后面还会绑定方法，利用methods属性

var app = new Vue({

el: "#app",

data: {

message:"Hello Vue!"

}

});

***index.css文件***

#app{

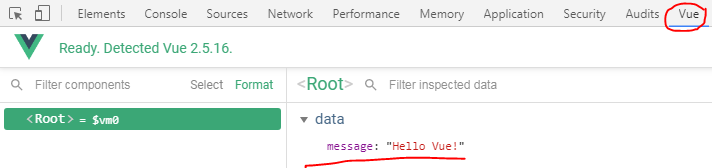
font-size: 18px;

color:green;

}

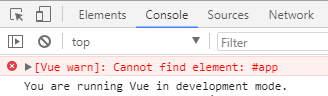
最终演示效果：



-

**注意：**

A.如果将外部js文件index.js放在head元素里面，会出现如下错误：Cannot find element



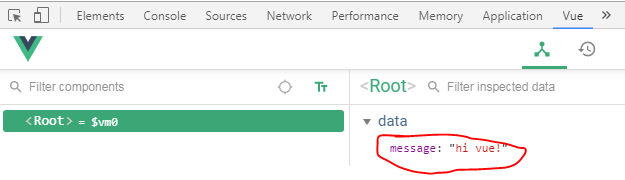
错误原因：把外部js文件放在head里面，导致html页面未加载完成就运行index.js文件

解决办法：**把相关的外部js文件放至页面底部，保证页面全部渲染完成才加载外部js文件**

**B.检测数据和DOM的响应式关联：**

在浏览器的控制台中修改 app.message 的值，就能立马在DOM页面中反映出来，如下：

 ==> 



1. **绑定元素特性：v-bind指令**，绑定数据渲染某个元素对应的某个特性

***index.html部分代码***

*<!-- 将vue实例的message属性绑定span元素的title特性 -->*

<div id="app">

<span v-bind:title="message">鼠标悬停几秒钟查看此处动态绑定的提示信息！</span>

</div>

***index.js文件***

*//* 创建一个Vue实例，其中el属性代表要渲染的元素，data属性代表绑定的数据

*//* el data 名字都是固定的，后面还会绑定方法，利用methods属性

var app = new Vue({

el: "#app",

data: {

message:"页面加载于 " + new Date().toLocaleString()

}

});

最终演示效果：



1. **条件绑定：v-if指令，绑定布尔变量数据来控制一个元素是否显示**

***index.html部分代码***

*<!-- 将vue实例的seen属性绑定p元素的条件判断 -->*

<div id="app">

<p v-if="seen">Now you can see me!</p>

</div>

***index.js***

var app = new Vue({

el: "#app",

data: {

seen: true *//*由seen变量来控制元素是否显示

}

});

当“seen”为true时，p元素才会显示，当设置app.seen=false时，p元素就会隐藏起来



1. **循环绑定：v-for指令，绑定数组元素来渲染一个项目列表**

***index.html部分代码***

*<!-- 用vue实例的todos数组元素来绑定渲染有序列表项 ，相当于 for in循环-->*

<div id="app">

<ol>

<li v-for="todo in todos">

{{ todo.text }}

</li>

</ol>

</div>

***index.js***

var app = new Vue({

el: "#app",

data: {

todos: [

{text: "Study vue"},

{text: "Study javascript"},

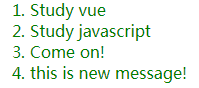
{text: "Come on!"}

]

}

});

最终演示效果：（在todos数组中追加一个元素，列表项中就会新增一项）

app.todos.push({text: "this is new message!"})

1. **处理与用户的交互：v-on指令，绑定数据和方法来添加一个事件监听器**

***index.html部分代码***

*<!-- 用 v-on 绑定一个事件监听器，相当于onclick事件处理程序-->*

<div id="app">

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

<button v-on:click="reverseMessage">逆转信息</button>

</div>

***index.js***

var app = new Vue({

el: "#app",

data: {

message: "Hello Vuejs!"

},

methods:{

reverseMessage:function(){

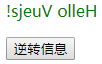
//将字符串分割成单个字符组成的数组，逆转后重新连接成字符串

*this*.message = *this*.message.split("").reverse().join("");}

}

});

最终演示效果：

 单击“逆转信息”后==> 

1. **表单输入绑定：v-model指令，实现表单输入和应用状态之间的双向绑定**

***index.html部分代码***

*<!-- 用 v-model 将表单输入信息和DOM页面信息绑定，都绑定同一个变量即可-->*

<div id="app">

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

<input v-model="message"></input>

</div>

***index.js***

var app = new Vue({

el: "#app",

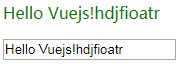
data: {

message: "Hello Vuejs!" *//*message变量一旦变化，应用状态和表单输入都会变化

}

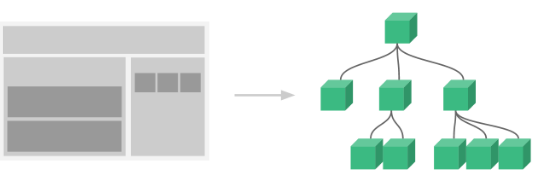
});

最终演示效果：

表单输入时页面也会同步==> 

1. **组件化应用构建**

**组件系统**是 Vue 的一个重要概念，因为它**是一种抽象**，允许我们使用小型、独立和通常可复用的组件构建大型应用，几乎任意类型的应用界面都可以抽象为一个组件树，比如：



***index.html部分代码***

*<!-- 组件化应用构建：将列表项抽象为一个组件复用-->*

<div id="app">

<ol>

*<!--使用自定义组件（自定义元素）todo-item，将自定义todo特性与数组每一项绑定-->*

<todo-item

v-for="item in groceryList"

v-bind:todo="item"

v-bind:key="item.id">

</todo-item>

</ol>

</div>

***index.js***

*//*利用Vue.component定义一个新组件"todo-item"，同时自定义特性“todo"以及模板

*//*注意：自定义组件类似于自定义元素，可以自定义特性，在html页面中当做新标签使用

Vue.component("todo-item",{

props: ["todo"],

template: "<li>{{ todo.text }}</li>"

});

*//* 创建一个Vue实例，其中el属性代表要渲染的元素，data属性代表绑定的数据

var app = new Vue({

el: "#app",

data: {

groceryList:[

{text: "fruit", id: 0},

{text: "vegetable", id: 1},

{text: "animal", id: 2} ]

}

});

最终演示效果：

