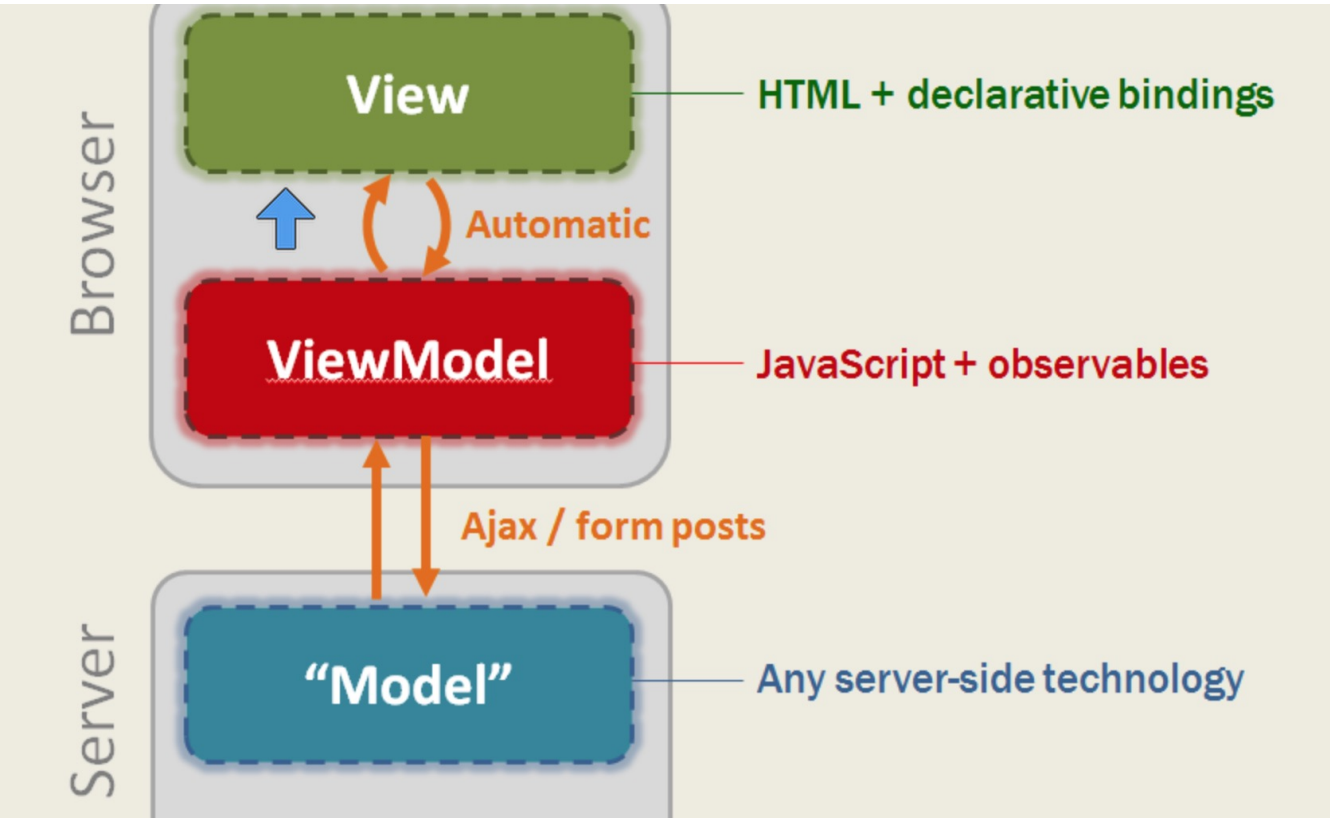
Vue入门文档

第一章 MVVM模式

MVVM 是Model-View-ViewModel 的缩写，它是一种基于前端开发的架构模式，其核心是提供对View 和 ViewModel 的双向数据绑定，这使得ViewModel 的状态改变可以自动传递给 View，即所谓的数据双向绑定。



Vue.js 就是一个提供了 MVVM 风格的双向数据绑定的 Javascript 库，专注于View 层。它的核心是 MVVM 中的 VM，也就是 ViewModel。 ViewModel负责连接 View 和 Model，保证视图和数据的一致性。

第二章什么是Vue

Vue (读音 /vjuː/，类似于 **view**) 是一套用于构建用户界面的**渐进式框架**。与其它大型框架不同的是，Vue 被设计为可以自底向上逐层应用。Vue 的核心库只关注视图层，不仅易于上手，还便于与第三方库或既有项目整合。另一方面，当与现代化的工具链以及各种支持类库结合使用时，Vue 也完全能够为复杂的单页应用提供驱动。

前端框架三巨头：Vue.js、React.js、AngularJS，vue.js以其轻量易用著称，vue.js和React.js发展速度最快。

第三章 Vue可以干什么

1. 解决数据绑定的问题；
2. Vue.js框架生产的主要目的是为了开发大型单页面应用（SPA：Single Page Application）

   Angular.js中对PC端支持的比较良好，但是对移动端支持就一般。

而Vue.js主要支持移动端，也支持PC端。

3. 它还支持组件化。也就是可以将页面封装成若干个组件，采用积木式编程，这样是页面的复用度达到最高（支持组件化）。

第四章 Vue的引入

```
<!-- 开发环境版本，包含了有帮助的命令行警告 -->
<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

或者：

```
<!-- 生产环境版本，优化了尺寸和速度 -->
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
对于生产环境，我们推荐链接到一个明确的版本号和构建文件，以避免新版本造成的不可预期的破坏
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
```

也可以直接下载，本地引入

第五章 Vue的基础语法

5.1 Vue的入门案例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            {{msg}} <!--VUE的数据模板-->
        </div>
        <script>
      //定义Vue的实例
            var vue = new Vue({
                el:"#app",
                data:{
                    msg:'helloworld!'
                }
            });
        </script>
    </body>
</html>
```

5.2 Vue的挂载点和实例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></scr
    </head>
    <body>
        <div id="app">
            {{msg}}
        </div>
        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    msg:'helloworld!'
                }
            });
        </script>
    </body>
</html>
```

挂载点

vue的一个实例

效果:

vue实例中的数据，只在挂载点内的DOM元素中有效。

5.3 Vue实例中的数据区分

我们使用：$区分Vue中属性和用户定义属性

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            {{message}}
```

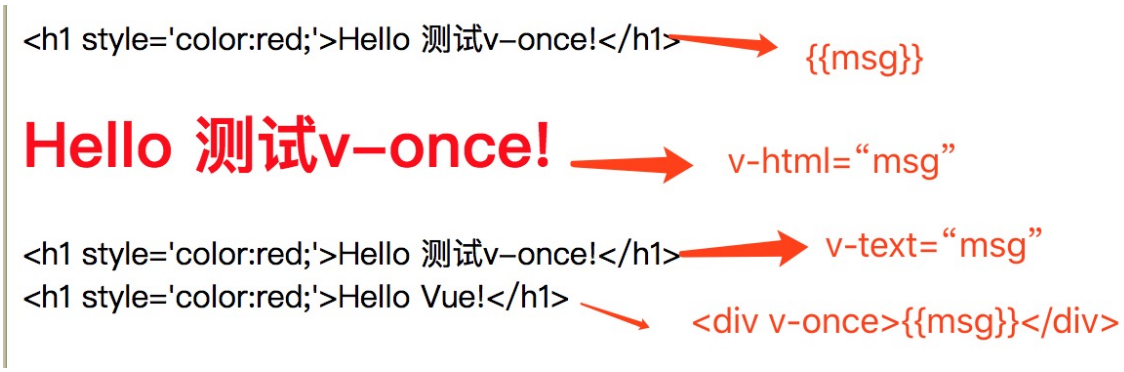
```
            {{title}}
        </div>
        测试 挂载点以外的位置方法vue中的数据无效：{{message}}
        <script>
          var  data = {
                    message:"HelloWorld",
                    title:'千锋集团'
            };
            var app = new Vue({
                el:"#app",//vue作用的dom节点的id，vue作用范围在该节点内部
                data:data
            });
            //vue中data节点的数据可以直接访问
            console.log(app.message);
            //为了区分vue的属性和用户定义的属性，访问vue的属性前$
            //访问vue的属性
            console.log(app.$data);
            console.log(app.$el);
            //访问用户定义的属性
            console.log(data);
        </script>
    </body>
</html>
```

5.4 数据绑定{{}}、v-html、v-text、v-once的使用

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

        <div id="app">
            <!-- 1.数据绑定最常见的形式就是使用 {{...}}（双大括号）的文本插值：  -->
            {{msg}}
            <!--2.使用v-html指定 解析成html代码 -->
            <div v-html="msg">
            </div>
            <!--3.使用v-html指定 解析成文本数据 -->
            <div v-text="msg">
            </div>
      <!-- 4.只渲染元素和组件一次，随后的渲染，使用了此指令的元素/组件及其所有的子节点，都会当作静态内容并跳过，这可以用于优化更新性能。-->
            <div v-once>
                    {{msg}}
            </div>
        </div>
        <script>
            var vue = new Vue({
                el:"#app",
```

```
                data:{
                    msg:"<h1 style='color:red;'>Hello Vue!</h1>"
                }
            });
            vue.msg = "<h1 style='color:red;'>Hello 测试v-once!</h1>";//修改msg的值，v-once修饰的节点中的内容不会改变
        </script>
    </body>
</html>
```

效果：



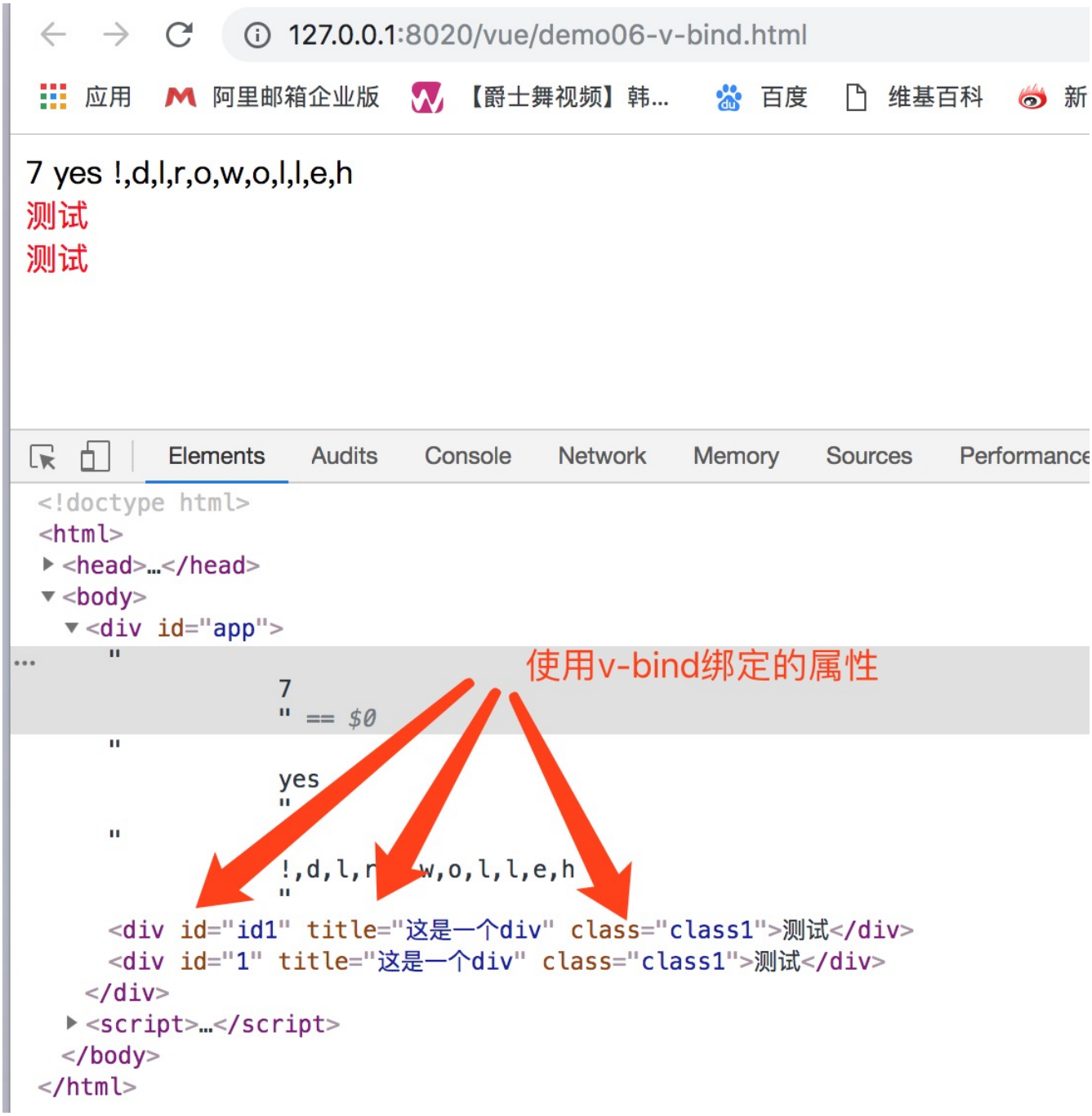
5.5 Vue中的属性绑定v-bind

Vue可以使用v-bind 绑定html标签的属性，属性值可以直接是Vue中定义的数据，如果是普通字符串需要加上''(单引号)

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
        <style>
            .class1{
                color:red;
            }
        </style>
    </head>
    <body>
        <div id="app">
            <!--vue对js的支持-->
            {{5+2}}
            <!--三目运算-->
            {{flag?"yes":"no"}}
            <!--反转字符串-->
            {{msg.split('').reverse().join(',')}}
            <!--v-bind:xxx可缩写为 :xxx
                例如:  v-bind:id缩写为 :id
                注意：不是vue中的对象  在v-bind中的属性中使用需要加上''(单引号)
            v-bind:class="{'class1':flag}":  flag为true，则class="class1",flag为false，则class属性没有值
            -->
```

```
            <div v-bind:id="'id'+id" v-bind:title="title" v-bind:class="{'class1':flag}">测试
</div>
            <!--缩写后-->
            <div :id="id" :title="title" :class="{'class1':flag}">测试</div>
        </div>
        <script>
            var vue = new Vue({
                el:'#app',
                data:{
                    title:'这是一个div',
                    flag:true,
                    id:1,
                    msg:"helloworld!"
                }});
        </script>
    </body>
</html>
```

效果：



5.6 双向数据绑定v-model

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            <!--
                v-mode:使用最多，双向绑定
                msg值绑定给输入框
                输入框的值又能绑定给msg
```

```
            -->
            <input type="text" v-model="msg" />
            {{msg}}
        </div>
        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    msg:"Hello Vue!"
                }
            });
        </script>
    </body>
</html>
```

效果：

大家发现：v-model能把Vue实例中的变量绑定给表单，用户通过表单输入的值也能绑定给Vue的变量。

5.7 Vue中的事件(v-on)和方法(methods)属性

在Vue中通过methods属性定义方法，通过@事件名（如：@click,@focus），绑定函数。

案例一：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

        <div id="app">

            {{count}}

            <!--以前写法：onclick="countNum()"
                Vue写法：  v-on:click:绑定了点击事件
                缩写:@click=""
            -->
            <button  v-on:click="countNum"  >点我</button>
```

```
            <button  @click="countNum"  >缩写版</button>
        </div>

        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    count:1
                },
                methods:{
                    countNum:function(){
                        this.count++;
                    }
                }
            });
        </script>
    </body>
</html>
```

案例二：

案例二：使用@blur校验数据的合法性

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
        <style>
            .msgInfo{
                color:red;
            }
        </style>
    </head>
    <body>
        <div id="app">
            <input type="text" @blur="checkName" v-model="u.uname" />
            <span v-html="umsg" class="msgInfo"></span>
            <br>
            <button  @click="login">登录</button>
            <br>
      <!--扩展：@click.stop.prevent：点击时调用函数，防止标签的默认行为  -->
            <a href="login.html" @click.stop.prevent="login">登录</a>
        </div>
```

```
        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    umsg:"请输入用户名",
                    u:{}
                },
                methods:{
                    checkName:function(){
                        console.log(this.u.uname);
                        var uname = this.u.uname;
                        if((typeof uname) == 'undefined'||uname==null || uname==''){
                            this.umsg = "用户名不能为空";
                            return;
                        }
                        var reg  = /^[a-zA-Z0-9_]{6,}$/;
                        if(!reg.test(uname)){
                            this.umsg = "用户必须大于6位";
                            return;
                        }
                        this.umsg = "✅";
                    },
                    login:function(){
                        alert('去登录');
                    }
                }
            });
        </script>
    </body>
</html>
```

效果:

总结: Vue定义方法需要定义到methods属性中，跟事件绑定用v-on:事件名(如: c-on:click),简写: @click
@blur:失去焦点触发 同js中的 onblur
@click:点击触发 同js中的onclick
@click.stop.prevent:可以防止a标签默认行为

5.8 Vue中的计算属性和侦听器

5.8.1 计算属性(computed)

模板内的表达式非常便利，但是设计它们的初衷是用于简单运算的。在模板中放入太多的逻辑会让模板过重且难以维护。例如：

```
<div id="example">
  {{ message.split('').reverse().join('') }}
</div>
```

当你想要在模板中多次引用此处的翻转字符串时，就会更加难以处理。
所以，对于任何复杂逻辑，你都应当使用**计算属性**。

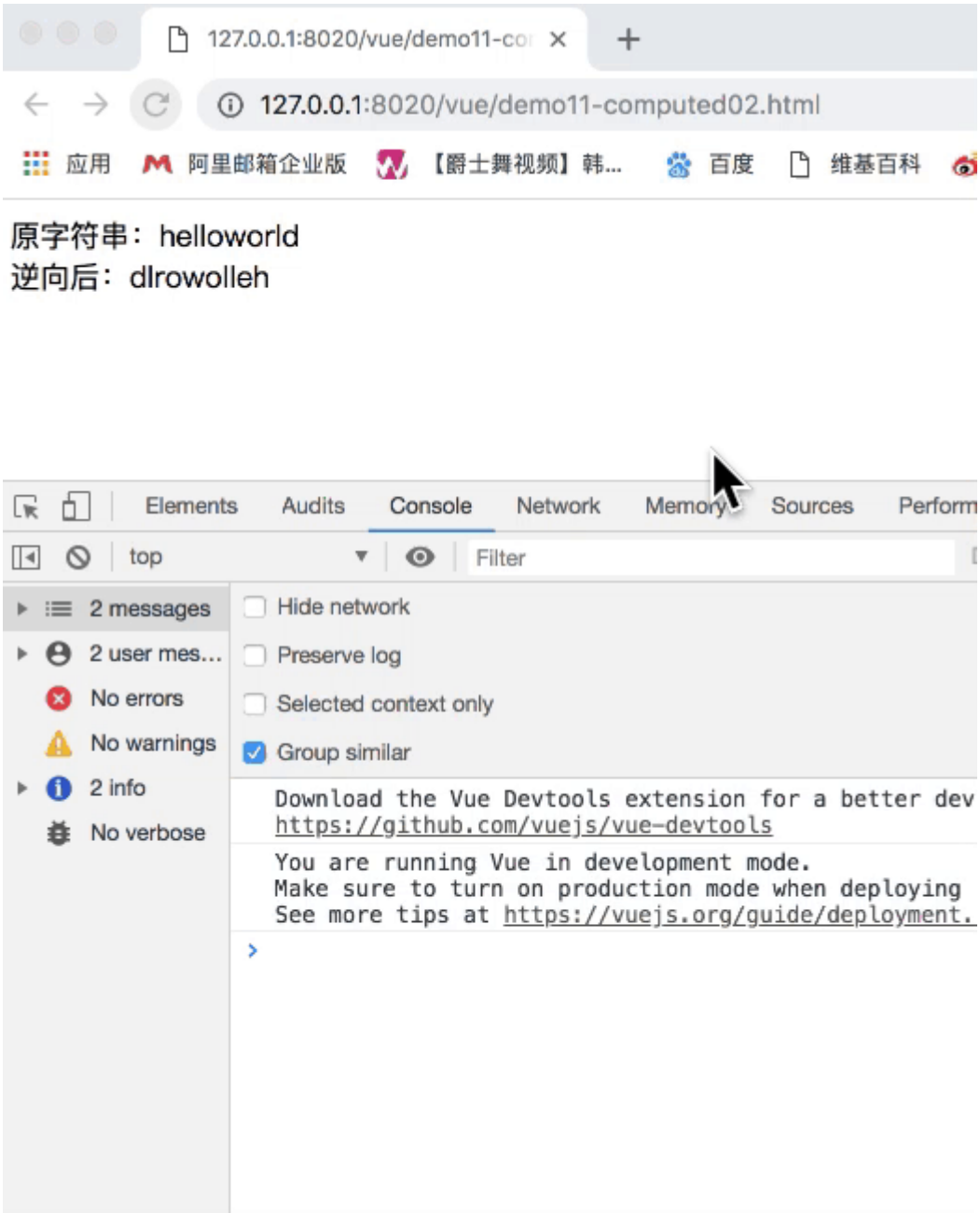
案例：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            原字符串：{{msg}}
            <br>
      <!--计算属性可以直接调用-->
            反转后：{{reversedMessage}}
        </div>
        <script>
            var vm = new Vue({
                el:"#app",
                data:{
                    msg:"helloworld"
                },
                computed:{
                    reversedMessage:function(){
                        return this.msg.split('').reverse().join('');
                    }
                }
            });
        </script>
    </body>
</html>
```

小结:
这里我们声明了一个计算属性 reversedMessage。我们提供的函数将用作属性 vm.reversedMessage 的 getter 函数.
上述代码等价于:

```
computed:{
                    reversedMessage:{
                        //return this.msg.split('').reverse().join('');
                        get:function(){//类似java中的getter方法  获得方法中的返回值
                            return this.msg.split('').reverse().join('');
                        }
                    }
                }
```

Vue 知道 vm.reversedMessage 依赖于 vm.message，因此当 vm.message 发生改变时，所有依赖 vm.reversedMessage 的绑定也会更新。

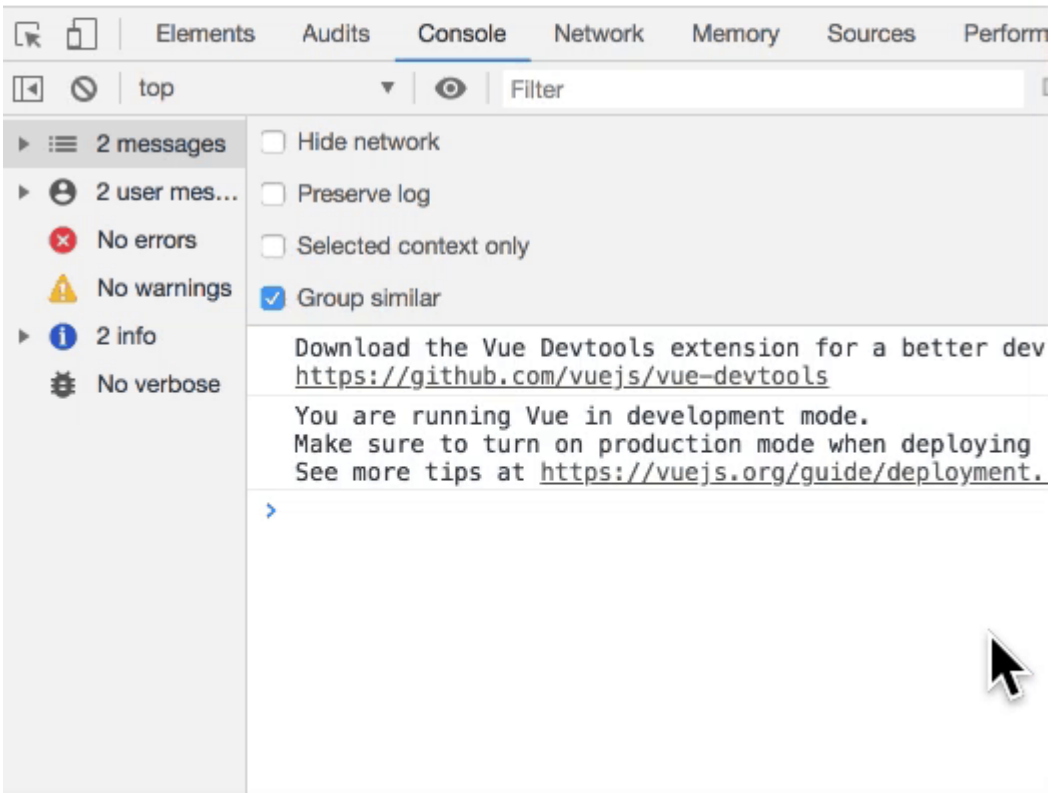


5.8.2 计算属性 VS 方法

案例

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>练习method属性和computed属性区别</title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

        <div id="app">
        methods属性:     {{now1()}}
        <br>
        <!--计算属性基于它们的响应式依赖进行缓存的
            Date.now()不是响应式依赖
        -->
        computed属性:    {{now2}}
        </div>
        <script>
            var vm = new Vue({
                el:"#app",
                data:{
                    message:'HelloWorld!'
                },
                methods:{
                    now1:function(){
                        return Date.now();//系统当前时间   返回毫秒
                    }
                },
                computed:{//get方法
                    now2:function(){
                        return Date.now();//系统当前时间 返回毫秒
                    }
                }
            });
        </script>
    </body>
</html>
```

效果:



通过案例发现：
methods属性会重新调用
computeds属性不会重新调用，使用的是缓存。

使用computed计算属性跟methods属性的异同点：
第一点：
computed属性中定义的方法可以直接使用调用：{{reversedMessage}}
methods属性中定义的方法调用时需要加上小括号(); 如：{{reversedMessage()}}
第二点：
computed计算属性是基于它们的响应式依赖进行缓存的。只在相关响应式依赖发生改变时它们才会重新求值。这就意味着只要 message 还没有发生改变，多次访问 reversedMessage 计算属性会立即返回之前的计算结果，而不必再次执行函数。
methods属性不具有缓存的功能，每次都会调用方法。

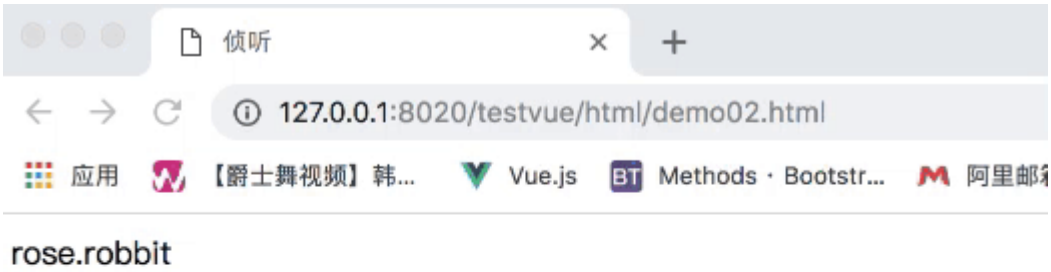
5.8.3 侦听属性(watch)

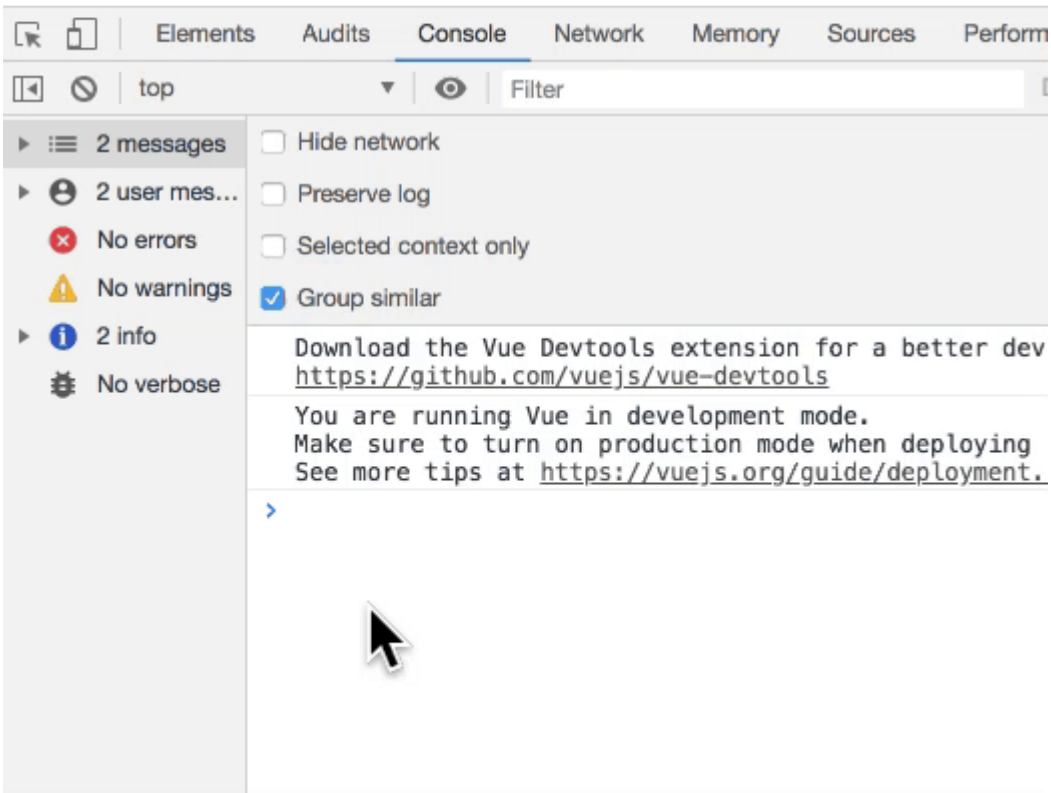
Vue 提供了一种更通用的方式来观察和响应 Vue 实例上的数据变动：**侦听属性。**

案例：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>侦听</title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="root">
                {{fullName()}}
        </div>

    </body>
    <script type="text/javascript">
            var vue = new Vue({
                el:"#root",
                data:{
                    firstName:'rose',
                    lastName:'robbit'
                },
                methods:{
                    fullName:function(){
                        return this.firstName+'.'+this.lastName;
                    }
                },
                //监控
                watch:{
                    //监控firstName的值
                    firstName(newValue){
                        console.log("firstName--->"+newValue);
                        this.fullName();
                    },//监控firstName的值
                    lastName(newValue){
                        console.log("lastName---->"+newValue);
                        this.fullName();
                }
                }
            });
        </script>
</html>
```

效果：





只要侦听的属性发送变化，就会执行侦听属性中对应的方法。

当你有一些数据需要随着其它数据变动而变动时，你很容易滥用 `watch`。然而，通常更好的做法是使用计算属性而不是命令式的 `watch` 回调。

5.8.4 计算属性的setter

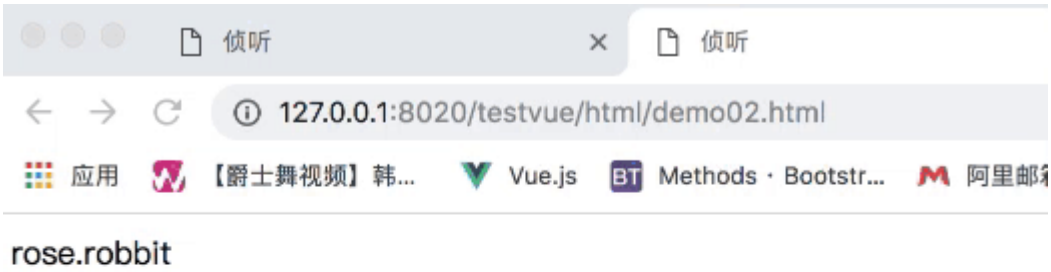
计算属性默认只有 getter，不过在需要时你也可以提供一个 setter：

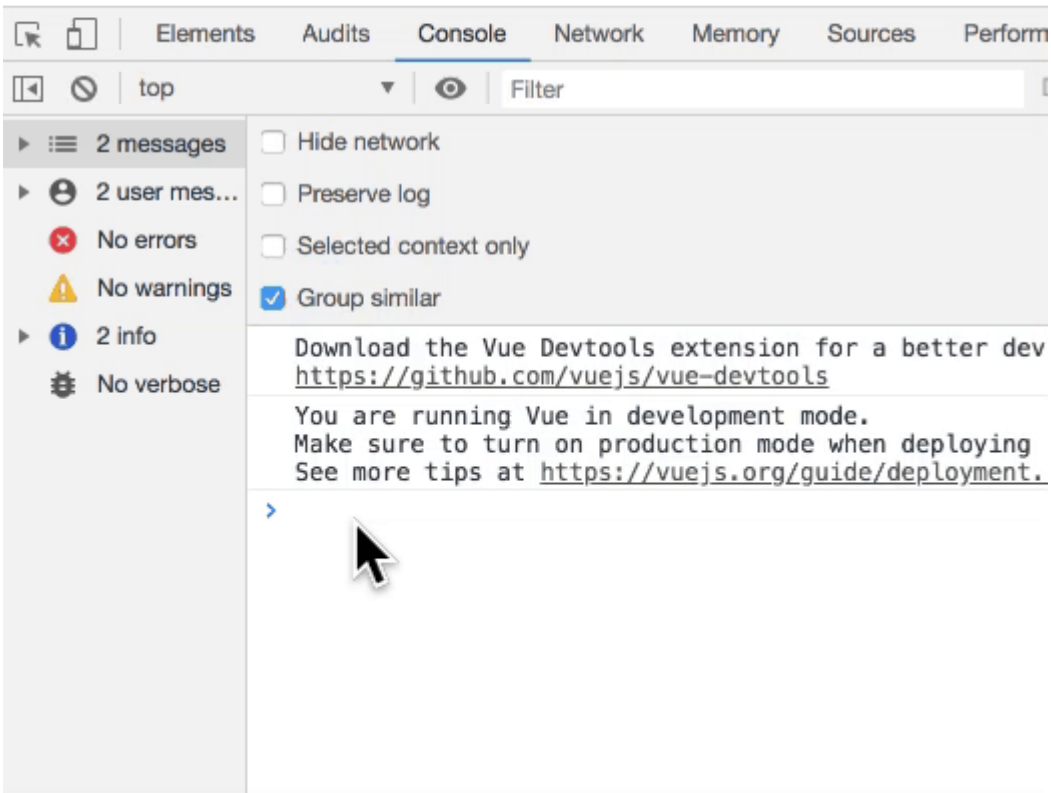
使用setter改造上一个案例：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>侦听</title>
```

```
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="root">
                {{fullName}}
        </div>

    </body>
    <script type="text/javascript">
            var vm = new Vue({
                el:"#root",
                data:{
                    firstName:'rose',
                    lastName:'robbit'
                },
                computed:{
                    fullName:{
                        get:function(){
                            return this.firstName+'.'+this.lastName;
                        },
                        set(newValue){
                            console.log(newValue);
                            var s = newValue.split(".");
                            this.firstName = s[0];
                            this.lastName = s[1];
                        }
                    }
                }
            });
        </script>
</html>
```

效果:





现在再运行 `vm.fullName = 'jack.hello'` 时，setter 会被调用，`vm.firstName` 和 `vm.lastName` 也会相应地被更新。

5.9 v-if,v-show与v-for指令

5.9.1 v-if

`v-if` 指令用于条件性地渲染一块内容。这块内容只会在指令的表达式返回 true 值的时候被渲染。

案例：

```
<!DOCTYPE html>
<html>
    <head>
```

```
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            <span v-if="flag">
                flag 为true显示
            </span>
            <span v-else>
                flag 为false显示
            </span>
        </div>

        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    flag:true
                }
            });
        </script>
    </body>
</html>
```

结果： 渲染" flag 为false显示"。

案例二：v-if在实际工作中可以简化开发，比如列表和新增公用一个页面。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

        <div id="app">

            <button  @click="add">新增</button>

            <div v-if="show">

                <table>
                    <tr>
                        <td>id</td>
                        <td>title</td>
                        <td>author</td>
                        <td>price</td>
                    </tr>
                    <tr>
                        <td>1</td>
```

```
                    <td>mysql从建表到删库</td>
                    <td>老王</td>
                    <td>1</td>
                </tr>
                <tr>
                    <td>2</td>
                    <td>java从入门到放弃</td>
                    <td>李四</td>
                    <td>2</td>
                </tr>
            </table>

        </div>

        <div v-else>
            <form id="" action="xxx">
                id:<input  /><br>
                title:<input  /><br>
                author:<input  /><br>
                price<input  /><br>
                <button >提交</button>
                <button  @click="backFun">返回</button>
            </form>
        </div>


    </div>

    <script>

        var vue = new Vue({
            el:"#app",
            data:{
                show:true
            },
            methods:{
                add:function(){
                    vue.show=false;
                },
                backFun:function(){
                    vue.show = true;
                }
            }
        });
    </script>
</body>
</html>
```

效果:

还有：`v-else-if`，顾名思义，充当 `v-if` 的“else-if 块”，可以连续使用。

5.9.2 v-show

另一个用于根据条件展示元素的选项是 `v-show` 指令。用法大致一样：

```
<h1 v-show="ok">Hello!</h1>
```

不同的是带有 `v-show` 的元素始终会被渲染并保留在 DOM 中。`v-show` 只是简单地切换元素的 CSS 属性 `display`。

注意，`v-show` 不支持 `<template>` 元素，也不支持 `v-else`。

5.9.3 v-if VS v-show

`v-if` 是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。

`v-if` 也是**惰性的**：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。

相比之下，`v-show` 就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于 CSS 进行切换。

一般来说，`v-if` 有更高的切换开销(来回渲染dom)，而 `v-show` 有更高的初始渲染开销。
因此，如果需要非常频繁地切换，则使用 `v-show` 较好；如果在运行时条件很少改变，则使用 `v-if` 较好。

5.9.4 v-for

我们用 `v-for` 指令根据一组数组的选项列表进行渲染。`v-for` 指令需要使用 `item in items` 形式的特殊语法，`items` 是源数据数组并且 `item` 是数组元素迭代的别名。

```
<!DOCTYPE html>
<html>
    <head>
```

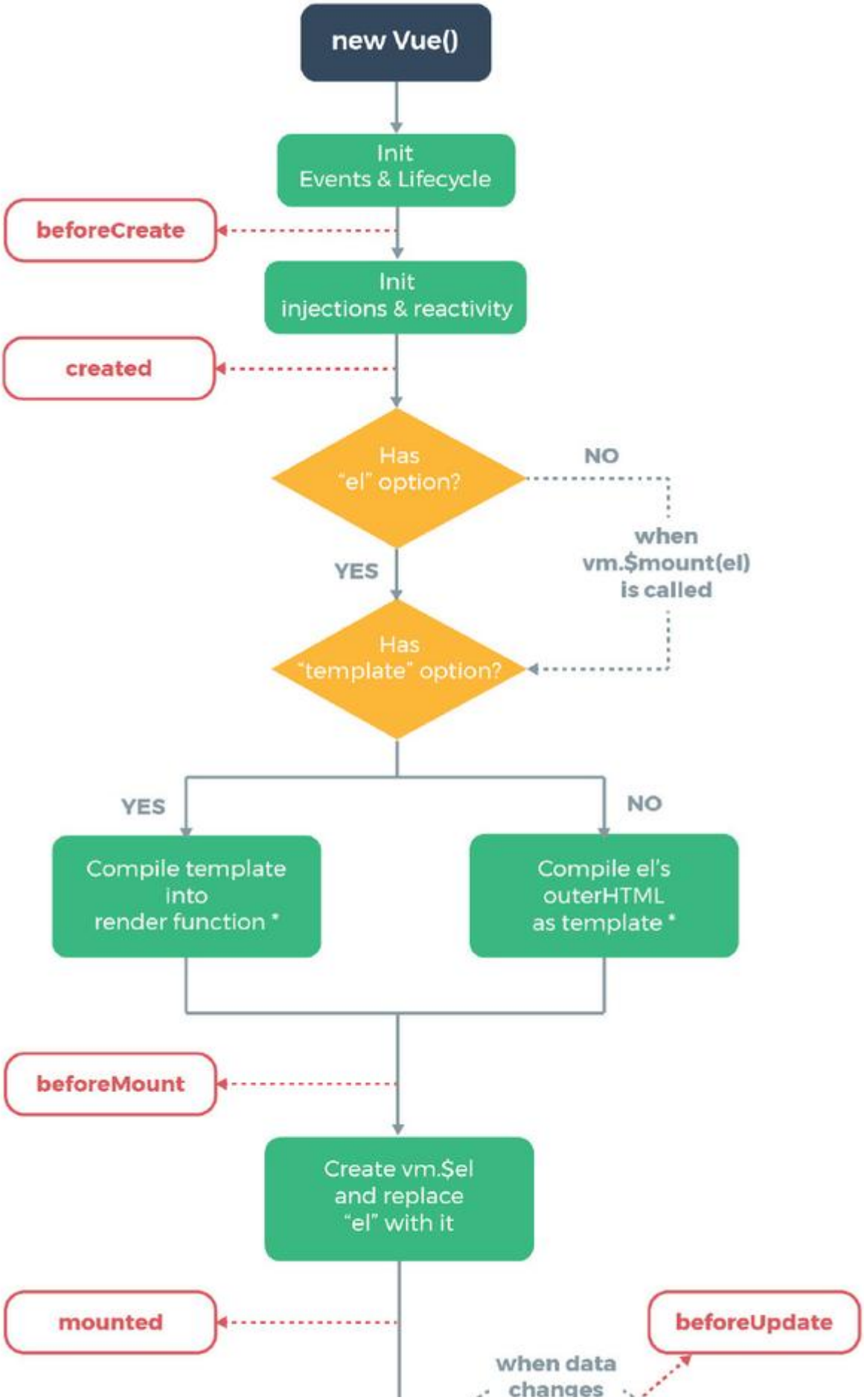
```
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>

        <div id="app">
            <!--遍历数组-->
            <ul v-for="b in bookList">
                <li>{{b.id}} {{b.title}}</li>
            </ul>
            <br />
            <!--处理对象 （用的少）
                其中：key是键 value是值
            -->
            <ul v-for="(value,key) in book">
                <li>{{key}}-{{value}}</li>
            </ul>
    </div>
        <script>
            var vue = new Vue({
                el:"#app",
                data:{
                    bookList:[
                    {id:1,title:'十万个为什么'},
                    {id:2,title:'喜洋洋与灰太狼'}
                    ],
                    book:{id:1,title:'十万个为什么'}
                }
            });
        </script>
    </body>
</html>
```

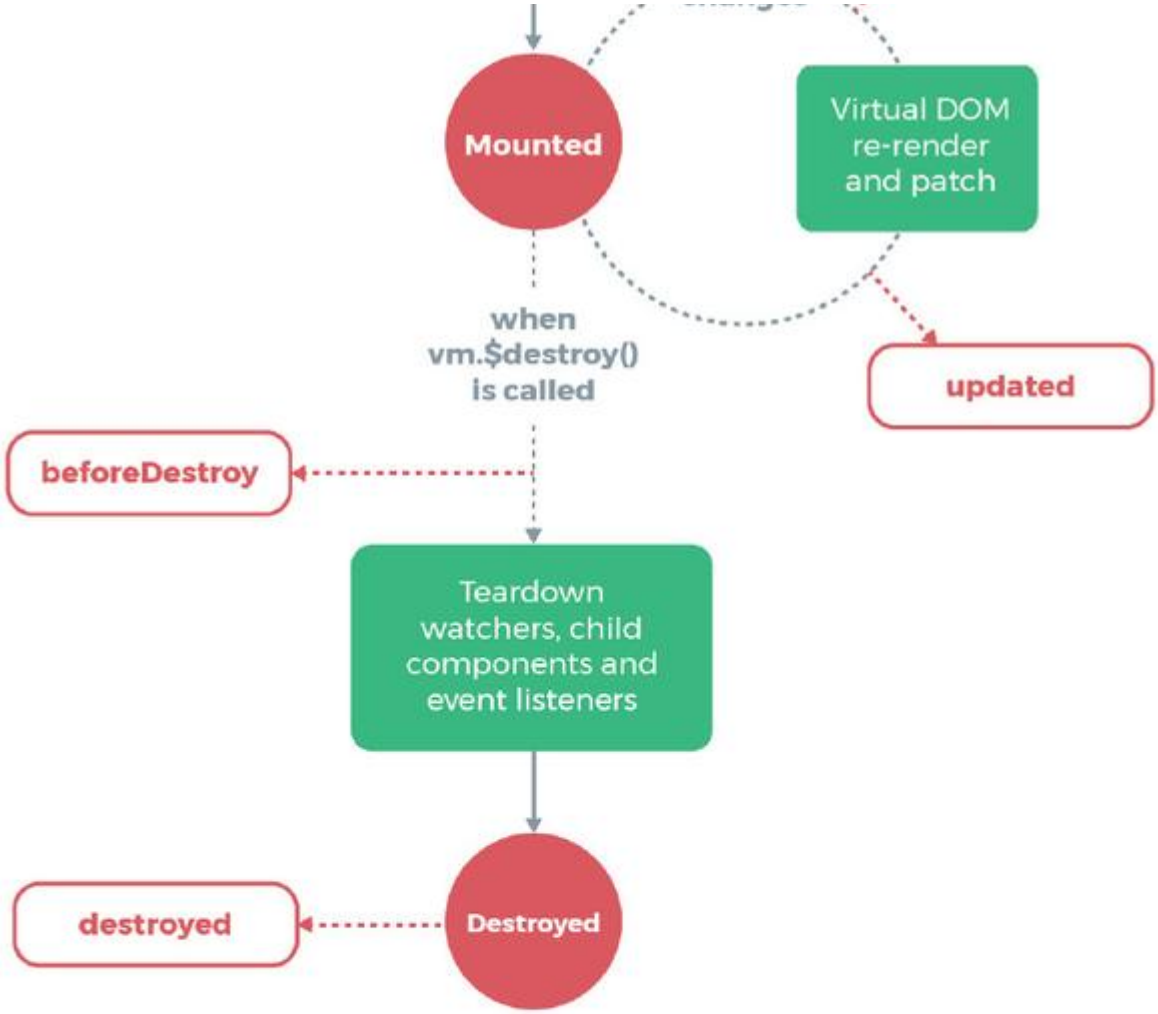
效果：

- 1 十万个为什么
- 2 喜洋洋与灰太狼

- id–1
- title–十万个为什么

5.10 Vue的生命周期





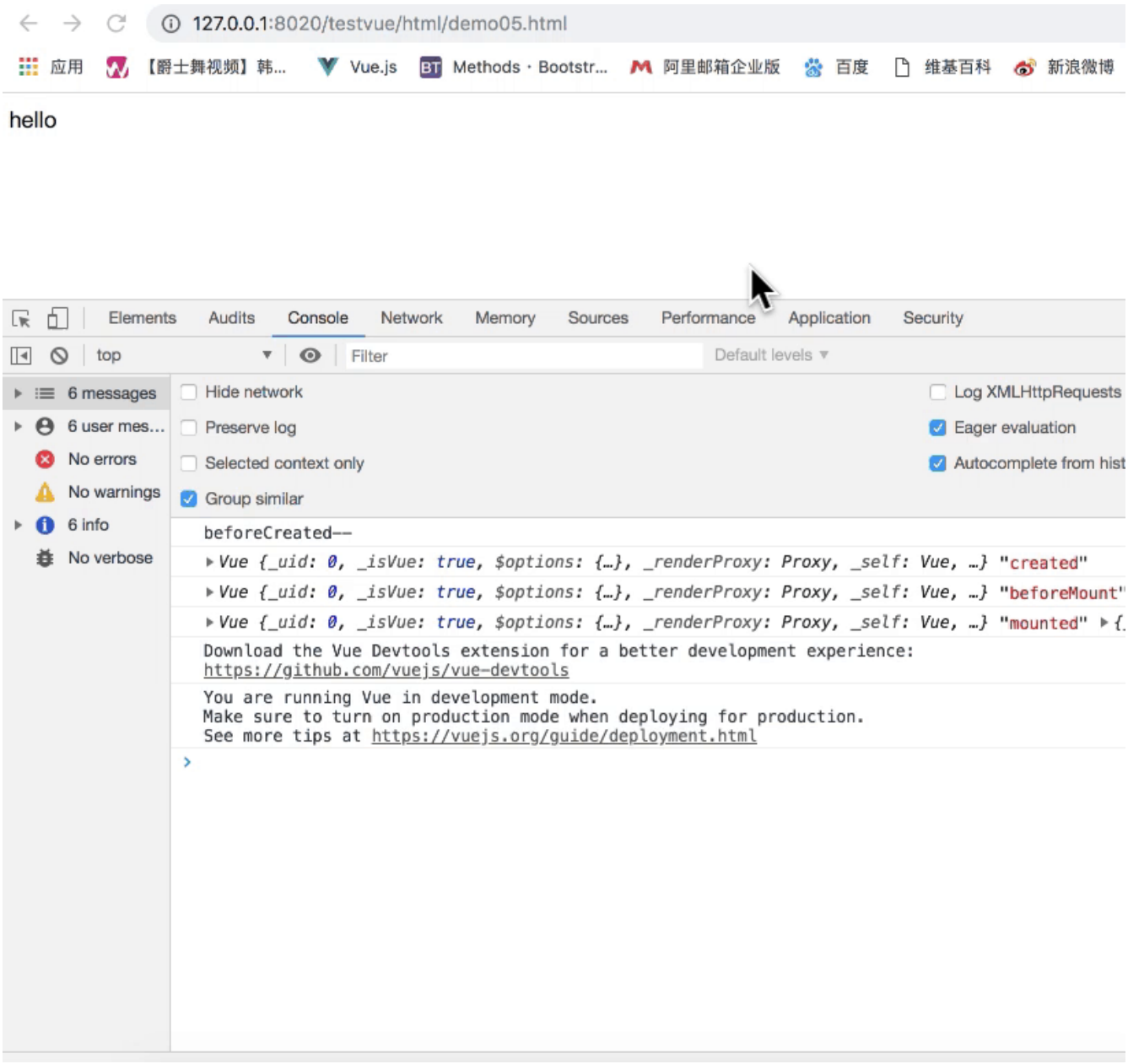
* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

beforeCreate（新对象诞生）
Vue对象用新方法实例化。它创建一个Vue类的对象来处理DOM元素。对象的这个生命阶段可以通过beforeCreated 钩子来访问。我们可以在这个钩子中插入我们的代码，在对象初始化之前执行。
created 创建（具有默认特性的对象）
完成了 data 数据的初始化，el没有初始化
beforeMount：完成了 el 和 data 初始化
mounted ：完成挂载
beforeUpdate：$vm.data更新之后，虚拟DOM重新渲染，可以在这个钩子中进一步地修改$vm.data，这不会触发附加的重渲染过程。
updated：当这个钩子被调用时，组件DOM的data已经更新，所以你现在可以执行依赖于DOM的操作。但是不要在此时修改data，否则会继续触发beforeUpdate、updated这两个生命周期，进入死循环！
beforeDestroy：实例被销毁之前调用。
destroyed：Vue实例销毁后调用。此时，Vue实例指示的所有东西已经解绑定，所有的事件监听器都已经被移除，所有的子实例也已经被销毁。

案例：

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
    </head>
    <body>
        <div id="app">
            {{b}}
        </div>

        <script type="text/javascript">
            var vue = new Vue({
                el:"#app",
                data:{
                    b:'hello'
                },
                beforeCreate:function(){//  可以在这里加加载事件    在加载实例是触发
                    console.log("beforeCreated--");
                },
                created:function(){//初始化完成的事件写在这里，异步请求也在这里调用
                    console.log(this,"created");
                },
                beforeMount:function(){//挂载元素之前
                    console.log(this,"beforeMount");
                },
                mounted:function(){//挂载元素
                    console.log(this,"mounted");
                },
                mounted:function(){//获取页面真实dom  页面已经渲染完毕
                    console.log(this,"mounted",this.$data);
                },
                beforeUpdate:function(){//$vm.data更新之后，虚拟DOM重新渲染
                    console.log(this,"beforeUpdate");
                },
                updated:function(){//组件DOM的data已经更新
                    console.log(this,"updated");
                    //this.b = '22';//不要在update方法中修改数据  容易死循环
                },
                beforeDestroy:function(){//调用Vue的$destory() 方法会触发该方法
                    console.log(this,"beforeDestory");
                },
                destroyed:function(){//调用Vue的$destory() 方法会触发该方法
                    console.log(this,"destory");
                },

            });
            //vue.b = 'test';
            //vue.$destroy();//触发destroye系列钩子函数
        </script>
    </body>
</html
```

效果：




```
beforeDestroy和destroyed当调用了Vue的destory()才会触发。
vue.$destory();
```

第六章 Vue的组件

5.1 全局组件

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
        <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

```
    </head>
    <body>

        <div id="app">
            <!-- 使用组件 -->
            <my-button></my-button>
            <!--可以复用-->
            <my-button></my-button>

        </div>

        <script>
            //全局组件
            Vue.component("my-button",{
                data:function(){
                    return {msg:'全栈工程师'};
                },
                template:"<button style='font-size:30px;'>点我啊 {{msg}} </button>"
            });
            var vue = new Vue({
                el:"#app"
            });

        </script>

    </body>
</html>
```

因为组件是可复用的 Vue 实例，所以它们与 `new Vue` 接收相同的选项，例如 `data`、`computed`、`watch`、`methods` 以及生命周期钩子等。仅有的例外是像 `el`这样根实例特有的选项。

一个组件的 data 选项必须是一个函数，因此每个实例可以维护一份被返回对象的独立的拷贝：

```
data: function () {
  return {
    {msg:'hello'};
  }
}
```

5.2 通过 Prop 向子组件传递数据

```
Vue.component('blog-post', {
  props: ['title'],
  template: '<h3>{{ title }}</h3>'
})
```

一个 prop 被注册之后，你就可以像这样把数据作为一个自定义特性传递进来：

```
<blog-post title="My journey with Vue"></blog-post>
<blog-post title="Blogging with Vue"></blog-post>
<blog-post title="Why Vue is so fun"></blog-post>
```

#####

#####