Vuejs 实战（一）

第一章 过滤器

过滤器是一个通过输入数据，能够及时对数据进行处理并返回一个数据结果的简单的函数。

在实际项目开发中根据实际的需求，可以自己编写所需要的过滤器。

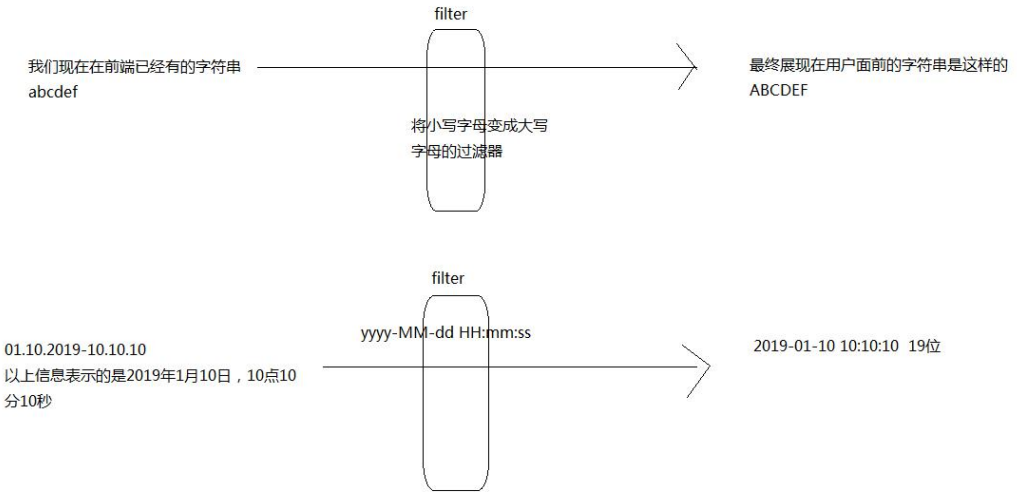
过滤器经常用在数据所需的格式化时使用：

例如

字符串的格式化

以及日期时间的格式化等等

过滤器最大的作用就是体现其复用性，如果我们在前端处理的某些文本信息每一次都需要经过重复的特殊处理，那么我们一定是要编写一个过滤器来使用。



1-1 全局过滤器的使用

全局过滤器指的是所有 vm 对象都能共享使用的过滤器。

过滤器能够使用在两个地方：

（mustache）插值表达式/指令（bind 属性）

过滤器的语法：使用管道符 "|"

过滤器在插值表达式中的使用

案例：将所有的字母变成大写

案例：定义格式化时间的全局过滤器

过滤器在 v-for 中的使用

案例：将所有的商品进行打折（打 8 折，5 折...）

以上的案例，我们都是使用在了插值表达式当中，除了使用在插值表达式中之外，我们的过滤器还可以说使用在 bind 属性指令当中，该形式没有插值表达式使用的广泛。

可以连续使用多个过滤器

1-2 私有过滤器的使用

私有过滤器指的是在指定的 vm 对象中来定义过滤器，该过滤器只在当前的 vm 对象中会发挥作用，其他的 vm 对象不能使用的。

语法：在 vm 对象中指定过滤器相关的属性和属性值

vm

```
filters:{
    filter1
    filter2
    ...
}
```

如果全局过滤器和私有过滤器的命名是一样的，那么默认使用的是私有过滤器。

系统在使用过滤器的时候，根据名称去找相应的过滤器，先找私有过滤器，如果没有私有的，则继续通过该名称寻找全局过滤器。

这种方式也被称之为就近原则（优先使用的是范围窄的）。

全局过滤器和私有过滤器能够搭配起来一起使用的

如果是按钮 123 的顺序

```
<div id="app">
    <p>{{str1 | filter1 | filter2 | filter3}}</p>
</div>
```

执行结果为

```
AAA123bbb
```

如果是按照 321 的顺序

```
<div id="app">
    <p>{{str1 | filter3 | filter2 | filter1}}</p>
</div>
```

执行结果为

```
AAABBB123
```

通过以上测试结果，观察得到以下结论

过滤器的使用：

可以同时使用多个过滤器，这多个过滤器在一起使用的时候，是具有信息传递性的。先处理排在前面的过滤器，得到结果传递到下一个过滤器中继续进行后续处理。

第二章 指令

1-1 自定义全局指令

指令与属性相似，是对指定元素样式或行为的赋予。

我们可以在实际项目开发中自定义一些我们所需的指令来有效的管理元素。

在页面中自定义的全局指令，可以为每一个 vm 对象中的元素提供服

务，只要 vm 中的标签引用了全局指令那么一定会即时生效，一般情况下我们普遍做的都是自定义全局指令来管理元素。

值得一提的是自定义指令需要经常搭配 vuejs 中的钩子函数来进行操作。在我们学习完指令之后，将在下一个章节（对象的生命周期）对钩子函数进行详细的学习。

自定义全局指令的语法：

```
Vue.directive()
```

案例：自定义一个全局指令让文本框自动获取焦点

我们也可以通过参数为指令进行赋值

1-2 自定义私有指令

私有指令是指定义在指定的 vm 对象中，只针对当前 vm 对象描述的元素生效的指令。其他的 vm 对象中的元素，是不能够使用该指令的。

语法：

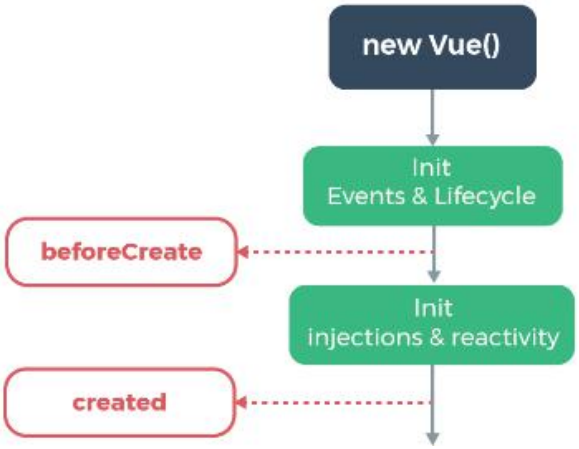
```
vm
    directives:{
        指令 1..
        指令 2...
    }
```

第三章 vuejs 对象的生命周期

3-1 生命周期简介

生命周期是指 Vuejs 对象从创建到销毁的全过程。

生命周期 1：创建阶段



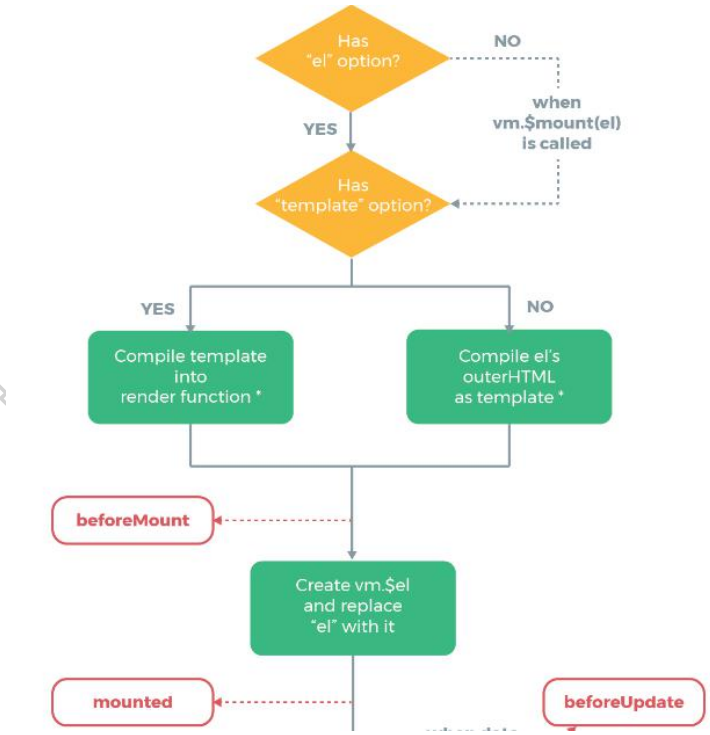
1.new Vue()： 执行代码 var vm = new Vue({}),表示创建一个 Vue 对象（实例）。

2.Init（Events&Lifecycle）:执行完以上代码后步入到对象初始化的前期阶段，表示通过以上代码我们创建了 Vuejs 对象，此时，在新建的对象身上，具备了一些生命周期相关的函数（生命周期的钩子函数）和默认的事件，但是其他的相关的组件还没有被创建（data、methods、filter 等等都没有被创建出来）。

执行 Init（Events&Lifecycle）之后，钩子函数都被创建出来，马上调用生命周期函数 beforeCreate，在该函数执行的时候，我们最常使用的组件 data 和 methods 等，都还没有被创建出来。

3.Init(injections&reactivity):该阶段是对象初始化的后期阶段。执行 Init(injections&reactivity)的方式是调用生命周期函数 created。在该函数中，data 和 methods 都已经被初始化好了。

也就是说，如果要使用 methods 中的方法，或者是操作 data 中的数据，最早可以在 created 方法中来进行操作。



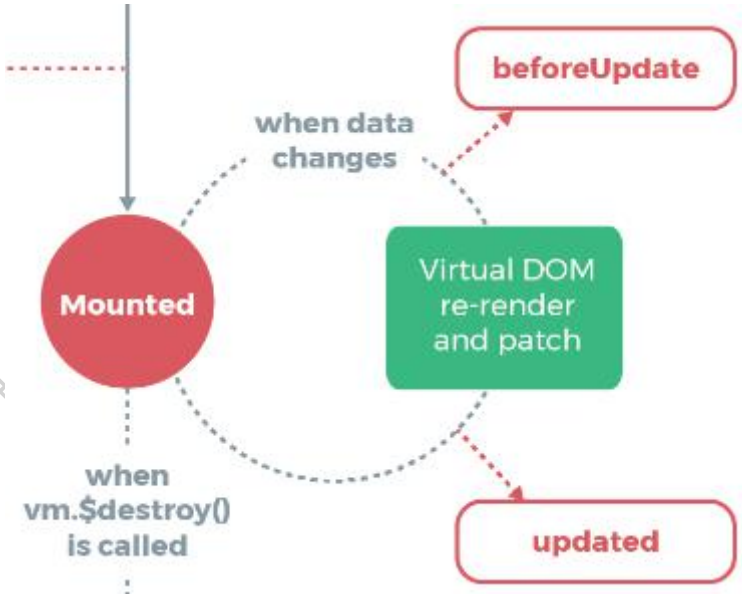
4.Init 对象初始化阶段执行完毕后，通过对元素以及模板进行判断，系统开始编辑模板，将 Vue 代码中的指令进行执行，然后在内存中生成一个编辑好的模板字符串，最终将该模板字符串渲染为内存中的 DOM。

但是此时集锦只是在内存中渲染好了模板，并没有将模板挂载到页面中去。该阶段执行完毕后，执行 beforeMount 方法。

5.Create vm：该阶段是将内存中编译好的模板，替换到浏览器的页面

中。该阶段执行完毕后，执行 mounted 方法。只要执行完了 mounted 方法，就表示整个 Vue 对象已经初始化完毕了，此时正式脱离创建阶段进入到运行阶段。如果要通过某些插件操作页面上的 DOM 节点，最早是要在 mounted 中进行。

生命周期 2：对象运行阶段



6.Virtual DOM:该阶段会根据 data 中的最新数据，重新渲染出一份最新的 DOM 树，当最新的内存 DOM 树被更新了之后，会把最新的 DOM 树重新渲染到页面中去，这时候就完成了使用模型 Model 去渲染视图 View 的过程。

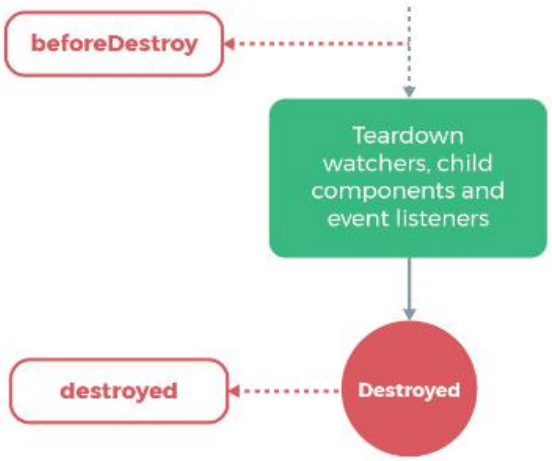
该阶段的执行会使用到两个函数 beforeUpdate 和 updated,这两个函数会根据 data 数据的变化，可重复的执行多次。

当 beforeUpdate 方法执行的时候，页面中的显示还是以前的数据，但是 data 中保存的是更新后的新数据，页面此时还没有和最新的数据

保持同步。

updated 函数执行的时候，页面和 data 数据已经保持同步了，都是最新的数据了。

生命周期 3：对象销毁阶段



7.Teardown（拆卸），该阶段为对象销毁的阶段，当对象实例运行完毕后，如果达到了对象销毁的条件，执行 beforDestroy 函数，该函数的执行正式标志着对象从运行阶段进入到了销毁的阶段。

当 beforDestroy 函数执行的时候，对象身上所有的组件 data、methods、filter、directive 等组件都还处于可用状态（因为对象只是步入到了销毁的阶段，还没有销毁），该函数执行完毕后，对象正式销毁。

8.Destroyed:该阶段为对象销毁后的阶段。该阶段会执行 destroyed 函数，当该函数执行的时候，对象已经被销毁了，里面的 data、methods 等相应的组件也不能使用了。

以上生命周期中使用的所有的函数，就是 Vuejs 生命周期中最重要的钩子函数。

3-2 Vuejs 生命周期中的钩子函数解析

beforeCreate

created

beforeMount

mounted

beforeUpdate

updated

beforeDestroy

destroyed

第四章 vuejs 的 ajax 请求

vuejs 本身不支持发送 ajax 请求的

需要使用插件来实现

vue-resource 插件

axios 插件

4-1 Vuejs 生命周期中的钩子函数解析

使用 vue-resource 插件来实现 get 和 post 请求

vuejs 将请求发送到后台，后台进行请求处理

后台（java、Node.js...）

使用 axios 插件实现 ajax 的 get 和 post 请求

axios 插件的两种使用方式

```
axios({
    method:"get/post"
})
axios.get(
    ""   //url
    {}   //param
)
```

总结：axios 的形式是一个基于 Promise 的 HTTP 请求客户端，用来发出请求。该形式也是 vue2.0 官方推荐的形式，官方在推出了该形式后，同时就不再对之前的 vue-resource 的形式进行更新和维护了。所以更推荐的是使用 axios 的形式来处理 ajax 请求。

4-2 扩展：关于跨域请求的处理

同源：同源是指在一般情况下，浏览器发出请求访问的资源都是在相同的协议、域名、和端口号下的，这样的请求即为默认同源策略的访问。

由于浏览器的同源策略，凡是发送请求 url 的协议、域名、端口号三者之间任意一个（或者多个）与当前页面地址不同即为跨域。

在实际项目开发中，使用的当前页面有可能会发出这样的请求，访问的是非同源的资源，那么我们就需要进行跨域处理。

例如：

http:协议

192.168.1.1ip 地址：域名

8080:端口号

当前页面：

http://192.168.1.1:8080/xxx/xxx.jsp

发出请求：

http://192.168.1.1:8080/xxx/yyy.do

以上我们发出的请求的协议，ip，端口号完全相同，就是发出的是同源请求的操作。

发出请求：

https://192.168.1.1:8080/xxx/yyy.do 协议不同

http://192.168.1.2:8080/xxx/yyy.do 域名（ip）不同

或者 myweb.com 访问 baidu.com 域名不同

http://192.168.1.1:8088/xxx/yyy.do 端口号不同

以上请求协议，ip，端口号三大要素均有不同，需要进行跨域请求的操作。

跨域请求的常用处理方式：

（1）代理方式

代理用于将请求发送给后台服务器，通过服务器来发送请求，然后将请求的结果传递给前端。通过 nginx 代理来实现操作。

优点：跨域服务稳定

缺点：在使用到跨域处理的时候，必须要事先搭建 nginx 服务的代理

环境，比较麻烦

（2）CORS 方式

CORS 是 w3c 标准的方式，通过在 web 服务器端的设置

响应头 Access-Control-Alow-Origin 来指定哪些域可以访问本域的数据。

优点：使用简单，支持基于 HTTP 协议的所有请求方式

缺点：跨域服务响应稍慢

（3）jsonp 方式

通过动态插入一个 script 标签。浏览器对 script 的资源引用没有同源限制，同时资源加载到页面后回立即执行。

优点：使用简单，跨域服务响应快，获取的数据是我们最常见的 json 格式的数据。

缺点：只能发送 get 请求，无法发送 post 请求

由于在开发中发出跨域请求的目的通常是为了取得指定的资源数据，所以一般都是发出 get 请求，由于 jsonp 的形式使用简单，而且关于接收的响应数据，是程序员使用最多的 json 格式的数据，所以该形式在 企业中应用的比较广泛。

对于资源的访问

对于分布式服务的访问