http 协议： http 是超文本传输协议

1.http 有四大特性：

- 基于 TCP/IP 之上作用于应用层。
- 基于请求响应；
- 无状态: 不保存状态，就是不保存记录，可以多次访问，一直访问，(cookie,session,token)；
- 无连接：请求一次响应一次，然后断开(如果使用长连接，使用 websocket(HTTP 协议的大补丁))；

2.数据格式

1. 数据格式之请求格式
   - 请求首行(请求方式，协议版本。。。)
   - 请求头(一大堆 k:v 键值对)
   - \r\n(空行，或者说换行)
   - 请求体:向服务器发送 post 请求的时候，请求体才会有数据，如果是 get 请求是没有请求体的。
2. 数据格式之响应格式
   - 响应首行
   - 响应头
   - \r\n
   - 响应体

3.请求方式

- get 请求：向一方(服务器)要数据。
- post 请求：向一方(服务器)提交数据（eg>用户登陆）

4.响应状态码

```
1xx  服务端已经接收到 你发送的数据，正在处理，你可以继续提交数据，informational(信息状态码)，接收的请求正在处理。
2xx  请求成功。sunccess(成功状态码)请求正常处理完毕
3xx  重定向，redirection(重定向状态码)，需要进行附加操作完成请求
4xx  请求错误(404:请求资源不存在，403:拒绝访问),client Error(客户端错误状态码)，服务器无法处理请求
5xx  服务器内部错误(500) server error(服务器错误状态码)，服务器处理请求出错。
```

get-post 定义的规范

get post 区别 restful 规范

在网站开发的时候通常会对 POST 和 GET 产生混淆，让人混淆的主要原因是基本上 POST 能解决的问题 GET 都能解决，反之亦然。今天就来说说者两者的区别。

GET：字面理解就是获取资源

1、GET 请求标准上是幂等的（用户应该认为请求是安全的-资源不会被修改，这里所以说应该是服务器端并不保证资源不会被修改）；
2、GET 请求可以被浏览器缓存；响应也可以被缓存（根据缓存头信息来处理）；
3、GET 请求可以保存在浏览器历史记录中，也可以作为链接分发或分享，可以收藏为书签；
4、GET 请求的数据都在 URL 中，可以方便都从浏览器中获取数据（因此不能携带诸如密码的明文数据）；
5、GET 请求的长度会有限制（比如 IE 的路径总长度需小于 2048 个字符）；
6、GET 请求的数据只能包含 ASCII 字符；

POST：字面理解就是发布新资源

1、POST 请求标准上不是幂等的（用户应该认为请求是有副作用的-可能会导致资源修改）；
2、POST 请求 URL 可以被浏览器缓存，但是 POST 数据不会被缓存；响应可以被缓存（根据缓存头信息来处理）；
3、POST 请求不便于分发或分享，因为 POST 数据会丢失，不能收藏为书签；
4、POST 请求没有长度限制，可以用来处理“请求数据”很大的场景（只要不超过服务器端的处理能力）；
5、POST 请求的数据不限于 ASCII 字符，可以包含二进制数据；

上面两者区别的解释中幂等可能不太好理解？

幂等（idempotent、idempotence）其实是一个数学或计算机学概念，常见于抽象代数中。幂等具体表现为：

1、对于单目运算，如果一个运算对于在范围内的所有的一个数多次进行该运算所得的结果和进行一次该运算所得的结果是一样的，那么我们就称该运算是幂等的。比如绝对值运算就是一个例子，在实数集中，有 abs(a)=abs(abs(a))。
2、对于双目运算，则要求当参与运算的两个值是等值的情况下，如果满足运算结果与参与运算的两个值相等，则称该运算幂等，如求两个数的最大值的函数，即 max(x,x) = x。
通俗的讲幂等的意味着对同一 URL 的多个请求应该返回同样的结果。但其实也不不是非常的严格，比如新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是和幂等的，因为它总是返回当前的新闻。从根本上说，如果目标是当用户打开一个链接时，他可以确信从自身的角度来看没有改变资源即可。
事实上 Http 定义了与服务器交互的不同方法，最基本的方法有 4 种，分别是 GET，POST，PUT，DELETE。URL 全称是资源描述符，我们可以这样认为：一个 URL 地址，它用于描述一个网络上的资源，而 HTTP 中的 GET，POST，PUT，DELETE 就对应着对这个资源的查，改，增，删 4 个操作。也就是说 GET 一般用于获取/查询资源信息，而 POST 一般用于更新资源信息。所以 GET 在信息修改层面，GET 比 POST 安全。GET 请求一般不应产生副作用。它仅仅是获取资源信息，就像数据库查询一样，不会修改，增加数据，不会影响资源的状态。

在说过了 GET、POST 实际用法后，我们发现很多人都没有按照 HTTP 规范（http://www.ietf.org/rfc/rfc2616.txt）去做。导致这个问题的原因有很多，比如说：

1、很多人贪方便，更新资源时用了 GET，因为用 POST 必须要到 FORM（表单），这样会麻烦一点。
2、3、对资源的增，删，改，查操作，其实都可以通过 GET/POST 完成，不需要用到 PUT 和 DELETE。
早期的 Web MVC 框架设计者们并没有有意识地将 URL 当作抽象的资源来看待和设计，所以导致一个比较严重的问题是传统的 Web MVC 框架基本上都只支持 GET 和 POST 两种 HTTP 方法，而不支持 PUT 和 DELETE 方法。

以上 3 点都是没有严格遵守 HTTP 规范，随着架构的发展，出现了 **REST(Representational State Transfer)，一套支持 HTTP 规范的 RESTful 架构。**
REST 这个词，是 Roy Thomas Fielding 在他 2000 年的博士论文中提出的。REST 即 Representational State Transfer 的缩写。直接翻译是"表现层状态转化"。具体请查看：http://zh.wikipedia.org/wiki/REST
在使用 CXF 开发 Web Service，尤其是在 RESTful 时，时常会遇到如下的格式报错:
No operation matching request path "xxxxxxx" is found, Relative Path: /, HTTP Method: POST, ContentType: application/x-www-form-urlencoded;charset=UTF-8, Accept: */*,. Please enable FINE/TRACE log level for more details.
崩不崩溃，刺不刺激 ;)
总结一下啊!
HTTP-GET 的处理特征如下:

1、将数据添加到 URL
2、利用一个问号（”?”）代表 URL 地址的结尾与数据的开端。
3、每一个数据的元素以 名称/值 (name/value) 的形式出现。
4、利用一个分号(";")来区分多个数据元素。

HTTP-POST 的处理特征如下:

1、将数据包括在 HTTP 主体中。
2、同样的，数据的元素以 名称/值 (name/value) 的形式出现。
3、但是每一个数据元素分别占用主体的一行。

【前端基础系列】理解 GET 与 POST 请求区别

语义区别

- GET 请求用于获取数据
- POST 请求用于提交数据

缓存

- GET 请求能被缓存，以相同的 URL 再去 GET 请求会返回 304
- POST 请求不能缓存

数据长度

HTTP 协议从未规定过 GET/POST 请求长度是多少，所谓的请求长度限制由浏览器和 Web 服务器决定的，各种浏览器和 web 服务器的设定均不一样，这依赖于各个浏览器厂家的规定或者可以根据 web 服务器的处理能力来设定。传统 IE 中 URL 的最大可用长度为 2048 字符，其他浏览器对 URL 长度限制实现上有所不同，POST 请求无长度限制（目前理论上是这样的）。

数据包

多数浏览器对于 POST 采用两阶段发送数据的，先发送请求头，再发送请求体，即使参数再少再短，也会被分成两个步骤来发送（相对于 GET），也就是第一步发送 header 数据，第二步再发送 body 部分。HTTP 是应用层的协议，而在传输层有些情况 TCP 会出现两次连接的过程，HTTP 协议本身不保存状态信息，一次请求一次响应。对于 TCP 而言，通信次数越多反而靠性越低，能在一次连结中传输完需要的消息是最可靠的，尽量使用 GET 请求来减少网络耗时。如果通信时间增加，这段时间客户端与服务器端一直保持连接状态，在服务器侧负载可能会增加，可靠性会下降。

安全

这里的「安全」和通常理解的「安全」意义不同，如果一个方法的语义在本质上是「只读」的，那么这个方法就是安全的。客户端向服务端的资源发起的请求如果使用了是安全的方法，就不应该引起服务端任何的状态变化，因此也是无害的。 此 RFC 定义，GET, HEAD, OPTIONS 和 TRACE 这几个方法是安全的。
但是这个定义只是规范，并不能保证方法的实现也是安全的。

幂等

幂等的概念是指同一个请求方法执行多次和仅执行一次的效果完全相同。按照 RFC 规范，PUT，DELETE 和安全方法都是幂等的。同样，这也仅仅是规范，服务端实现是否幂等是无法确保的。引入幂等主要是为了处理同一个请求重复发送的情况，比如在请求响应前失去连接，如果方法是幂等的，就可以放心地重发一次请求。这也是浏览器在后退/刷新时遇到 POST 会给用户提示的原因：POST 语义不是幂等的，重复请求可能会带来意想不到的后果。

HTTP:Post 和 Get 详解

- 一 原理区别

一般在浏览器中输入网址访问资源都是通过 GET 方式；在 FORM 提交中，可以通过 Method 指定提交方式为 GET 或者 POST，默认为 GET 提交 Http 定义了与服务器交互的不同方法，最基本的方法有 4 种，分别是 GET，POST，PUT，DELETE

URL 全称是资源描述符，我们可以这样认为：一个 URL 地址，它用于描述一个网络上的资源，而 HTTP 中的 GET，POST，PUT，DELETE 就对应着对这个资源的查 ，改 ，增 ，删 4 个操作。GET 一般用于获取/查询 资源信息，而 POST 一般用于更新资源信息。

根据 HTTP 规范，GET 用于信息获取，而且应该是安全的和幂等的 。

1.所谓安全的意味着该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。就是说，它仅仅是获取资源信息，就像数据库查询一样，不会修改，增加数据，不会影响资源的状态。

* 注意：这里安全的含义仅仅是指是非修改信息。

2.幂等的意味着对同一 URL 的多个请求应该返回同样的结果。这里我再解释一下幂等 这个概念：

幂等 （idempotent、idempotence）是一个数学或计算机学概念，常见于抽象代数中。

幂等有以下几种定义：

对于单目运算，如果一个运算对于在范围内的所有的一个数多次进行该运算所得的结果和进行一次该运算所得的结果是一样的，那么我们就称该运算是幂等的。比如绝对值运算就是一个例子，在实数集中，有 abs(a) = abs(abs(a)) 。

对于双目运算，则要求当参与运算的两个值是等值的情况下，如果满足运算结果与参与运算的两个值相等，则称该运算幂等，如求两个数的最大值的函数，有在在实数集中幂等，即 max(x,x) = x 。

看完上述解释后，应该可以理解 GET 幂等的含义了。

但在实际应用中，以上 2 条规定并没有这么严格。引用别人文章的例子：比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。从根本上说，如果目标是当用户打开一个链接时，他可以确信从自身的角度来看没有改变资源即可。

根据 HTTP 规范，POST 表示可能修改变服务器上的资源的请求 。继续引用上面的例子：还是新闻以网站为例，读者对新闻发表自己的评论应该通过 POST 实现，因为在评论提交后站点的资源已经不同了，或者说资源被修改了。

上面大概说了一下 HTTP 规范中，GET 和 POST 的一些原理性的问题。但在实际的做的时候，很多人却没有按照 HTTP 规范去做，导致这个问题的原因有很多，比如说：

1.很多人贪方便，更新资源时用了 GET，因为用 POST 必须要到 FORM（表单），这样会麻烦一点。

2.对资源的增，删，改，查操作，其实都可以通过 GET/POST 完成，不需要用到 PUT 和 DELETE。

3.另外一个是，早期的但是 Web MVC 框架设计者们并没有有意识地将 URL 当作抽象的资源来看待和设计 。还有一个较为严重的问题是传统的 Web MVC 框架基本上都只支持 GET 和 POST 两种 HTTP 方法，而不支持 PUT 和 DELETE 方法。

以上 3 点典型地描述了老一套的风格（没有严格遵守 HTTP 规范），随着架构的发展，现在出现 REST(Representational State Transfer)，一套支持 HTTP 规范的新风格，这里不多说了，可以参考《RESTful Web Services》。

- 二 表现形式区别

搞清了两者的原理区别，我们再来看一下他们实际应用中的区别：

为了理解两者在传输过程中的不同，我们先看一下 HTTP 协议的格式：

HTTP 请求：

```
<request  line>
          <headers>
          <blank  line>
<request-body>
```

在 HTTP 请求中，第一行必须是一个请求行（request line），用来说明请求类型、要访问的资源以及使用的 HTTP 版本。紧接着是一个首部（header）小节，用来说明服务器要使用的附加信息。在首部之后是一个空行，再此之后可以添加任意的其他数据[称之为主体（body）]。

GET 实例：

```
GET /books/?sex=man&name=Professional HTTP/1.1
Host: www.wrox.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
Gecko/20050225 Firefox/1.0.1
Connection: Keep-Alive
```

POST 实例：

```
POST / HTTP/1.1
Host: www.wrox.com
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
Gecko/20050225 Firefox/1.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive <br>    (----此处空一行---
-)     <br>name=Professional%20Ajax&publisher=Wiley
```

有了以上对 HTTP 请求的了解和示例，我们再来看两种提交方式的区别：

（1）GET 提交，请求的数据会附在 **URL 之后（就是把数据放置在 HTTP 协议头中），以?分割 URL 和传输数据，多个参数用&连接；**例如：

```
login.action?name=hyddd&password=idontknow&verify=%E4%BD%A0 %E5%A5%BD。
```

如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用 BASE64 加密，得出如： %E4%BD%A0%E5%A5%BD，其中%XX 中的 XX 为该符号以 16 进制表示的 ASCII。

POST 提交：把提交的数据放置在是 **HTTP 包的包体中**。上文示例中红色字体标明的就是实际的传输数据。因此，GET 提交的数据会在地址栏中显示出来，而 POST 提交，地址栏不会改变。

（2）传输数据的大小：

首先声明：HTTP 协议没有对传输的数据大小进行限制，HTTP 协议规范也没有对 URL 长度进行限制。而在实际开发中存在的限制主要有：

GET:特定浏览器和服务器对 URL 长度有限制，例如 IE 对 URL 长度的限制是 2083 字节(2K+35)。对于其他浏览器，如 Netscape、FireFox 等，理论上没有长度限制，其限制取决于操作系统的支持。因此对于 GET 提交时，传输数据就会受到 URL 长度的限制。

POST:由于不是通过 URL 传值，理论上数据不受限。但实际各个 WEB 服务器会规定对 post 提交数据大小进行限制，Apache、IIS6 都有各自的配置。

（3）安全性：

POST 的安全性要比 GET 的安全性高。注意：这里所说的安全性和上面 GET 提到的"安全"不是同个概念。上面"安全"的含义仅仅是不作数据修改，而这里安全的含义是真正的 Security 的含义，比如：通过 GET 提交数据，用户名和密码将明文出现在 URL 上，因为(1)登录页面有可能被浏览器缓存， (2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用 GET 提交数据还可能会造成 Cross-site request forgery 攻击

（4）Http get,post,soap 协议都是在 http 上运行的

1）get：请求参数是作为一个 key/value 对的序列（查询字符串）附加到 URL 上的查询字符串的长度受到 web 浏览器和 web 服务器的限制（如 IE 最多支持 2048 个字符），不适合传输大型数据集同时，它很不安全。

2）post：请求参数是在 http 标题的一个不同部分（名为 entity body）传输的，这一部分用来传输表单信息，因此必须将 Content-type 设置: application/x-www-form-urlencoded。post 设计用来支持 web 窗体上的用户字段，其参数也是作为 key/value 对传输。

但是：它不支持复杂数据类型，因为 post 没有定义传输数据结构的语义和规则。

3）soap：是 http post 的一个专用版本，遵循一种特殊的 xml 消息格式 Content-type 设置为: text/xml 任何数据都可以 xml 化。

- 三 HTTP 响应

1．HTTP 响应格式：

```
<status  line>
<headers>
<blank  line>
[<response-body>]
```

在响应中唯一真正的区别在于第一行中用状态信息代替了请求信息。状态行（status line）通过提供一个状态码来说明所请求的资源情况。

HTTP 响应实例：

```
HTTP/1.1 200 OK
Date: Sat, 31 Dec 2005 23:59:59 GMT
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 122
<html>
<head>
<title>Wrox Homepage</title>
</head>
<body>
<!-- body goes here -->
</body>
</html>
```

2．最常用的状态码有：

◆200 (OK)：找到了该资源，并且一切正常。

◆304 (NOT MODIFIED)：该资源在上次请求之后没有任何修改。这通常用于浏览器的缓存机制。

◆401 (UNAUTHORIZED)：客户端无权访问该资源。这通常会使得浏览器要求用户输入用户名和密码，以登录到服务器。

◆403 (FORBIDDEN)：客户端未能获得授权。这通常是在 401 之后输入了不正确的用户名或密码。

◆404 (NOT FOUND)：在指定的位置不存在所申请的资源。

- 四 完整示例：

HTTP GET

发送

```
GET
/DEMOWebServices2.8/Service.asmx/CancelOrder?UserID=string&PWD=string&OrderConfirmation=string
HTTP/1.1
Host: api.efxnow.com
```

回复

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml  version="1.0" encoding="utf-8"?>
<objPlaceOrderResponse  xmlns="https://api.efxnow.com/webservices2.3">
<Success>boolean</Success>
<ErrorDescription>string</ErrorDescription>
<ErrorNumber>int</ErrorNumber>
<CustomerOrderReference>long</CustomerOrderReference>
<OrderConfirmation>string</OrderConfirmation>
<CustomerDealRef>string</CustomerDealRef>
</objPlaceOrderResponse>
```

HTTP POST

发送

```
POST /DEMOWebServices2.8/Service.asmx/CancelOrder HTTP/1.1
Host: api.efxnow.com
Content-Type: application/x-www-form-urlencoded
Content-Length: length

UserID=string&PWD=string&OrderConfirmation=string
```

回复

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml  version="1.0" encoding="utf-8"?>
<objPlaceOrderResponse  xmlns="https://api.efxnow.com/webservices2.3">
<Success>boolean</Success>
<ErrorDescription>string</ErrorDescription>
<ErrorNumber>int</ErrorNumber>
<CustomerOrderReference>long</CustomerOrderReference>
<OrderConfirmation>string</OrderConfirmation>
<CustomerDealRef>string</CustomerDealRef>
</objPlaceOrderResponse>
```

SOAP 1.2

发送

```
POST /DEMOWebServices2.8/Service.asmx HTTP/1.1
Host: api.efxnow.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml  version="1.0" encoding="utf-8"?>
<soap12:Envelope  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
<soap12:Body>
    <CancelOrder  xmlns="https://api.efxnow.com/webservices2.3">
    <UserID>string</UserID>
    <PWD>string</PWD>
    <OrderConfirmation>string</OrderConfirmation>
    </CancelOrder>
</soap12:Body>
</soap12:Envelope>
```

回复

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml  version="1.0" encoding="utf-8"?>
<soap12:Envelope  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
<soap12:Body>
    <CancelOrderResponse  xmlns="https://api.efxnow.com/webservices2.3">
    <CancelOrderResult>
    <Success>boolean</Success>
    <ErrorDescription>string</ErrorDescription>
    <ErrorNumber>int</ErrorNumber>
    <CustomerOrderReference>long</CustomerOrderReference>
    <OrderConfirmation>string</OrderConfirmation>
    <CustomerDealRef>string</CustomerDealRef>
    </CancelOrderResult>
    </CancelOrderResponse>
</soap12:Body>
</soap12:Envelope>
```

在写代码过程中，get 与 post 是两种不同的提交方式。下面，列举出两种方式的不同。

方法/步骤

1. get 是从服务器上获取数据，post 是向服务器传送数据。
2. get 是把参数数据队列加到提交表单的 ACTION 属性所指的 URL 中，值和表单内各个字段一一对应，在 URL 中可以看到。post 是通过 HTTP post 机制，将表单内各个字段与其内容放置在 HTML HEADER 内一起传送到 ACTION 属性所指的 URL 地址。用户看不到这个过程。
3. 对于 get 方式，服务器端用 Request.QueryString 获取变量的值，对于 post 方式，服务器端用 Request.Form 获取提交的数据。
4. get 传送的数据量较小，不能大于 2KB。post 传送的数据量较大，一般被默认为不受限制。但理论上，IIS4 中最大量为 80KB，IIS5 中为 100KB。
5. get 安全性非常低，post 安全性较高。

6. HTTP 定义了与服务器交互的不同方法，最基本的方法是 GET 和 POST。事实上 GET 适用于多数请求，而保留 POST 仅用于更新站点。根据 HTTP 规范，GET 用于信息获取，而且应该是 安全的和幂等的。所谓安全的意味着该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。幂等的意味着对同一 URL 的多个请求应该返回同样的结果。完整的定义并不像看起来那样严格。从根本上讲，其目标是当用户打开一个链接时，她可以确信从自身的角度来看没有改变资源。 比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。反之亦然。POST 请求就不那么轻松了。POST 表示可能改变服务器上的资源的请求。仍然以新闻站点为例，读者对文章的注解应该通过 POST 请求实现，因为在注解提交之后站点已经不同了
7. 在 FORM 提交的时候，如果不指定 Method，则默认为 GET 请求，Form 中提交的数据将会附加在 url 之后，以?分开与 url 分开。字母数字字符原 样发送，但空格转换为“+“号，其它符号转换为%XX,其中 XX 为该符号以 16 进制表示的 ASCII（或 ISO Latin-1）值。GET 请求请提交的数据放置在 HTTP 请求协议头中，而 POST 提交的数据则放在实体数据中；GET 方式提交的数据最多只能有 1024 字节，而 POST 则没有此限制。

END

GET、POST 请求的区别分析

1 前言

众所周知，在我们开发项目的过程中，关于 POST 与 GET 请求是我们不得不掌握的知识，那么它们两者之间又有什么区别呢？接下来，我们一起从 HTTP 报文等角度来探讨学习一下关于两者的不同

2 HTTP 请求方法

HTTP 定义了一组请求方法，以表明要对给定资源执行的操作。指示针对给定资源要执行的期望动作. 虽然他们也可以是名词，但这些请求方法有时被称为 HTTP 动词. 每一个请求方法都实现了不同的语义，但一些共同的特征由一组共享：：例如一个请求方法可以是 safe，idempotent，或 cacheable.

方法名称	用法
GET	GET 方法请求一个指定资源的表示形式. 使用 GET 的请求应该只被用于获取数据.
HEAD	HEAD 方法请求一个与 GET 请求的响应相同的响应，但没有响应体.
POST	用于将实体提交到指定的资源，通常导致在服务器上的状态变化或副作用.
PUT	PUT 方法用请求有效载荷替换目标资源的所有当前表示。
DELETE	DELETE 方法删除指定的资源。
CONNECT	CONNECT 方法建立一个到由目标资源标识的服务器的隧道。
OPTIONS	OPTIONS 方法用于描述目标资源的通信选项。
TRACE	TRACE 方法沿着到目标资源的路径执行一个消息环回测试。
PATCH	PATCH 方法用于对资源应用部分修改。

3 标准参考

W3school:http://www.w3school.com.cn/ta...

项目	GET	POST
后退按钮/刷新	无害	数据会被重新提交（浏览器应该告知用户数据会被重新提交）。
书签	可收藏为书签	不可收藏为书签
缓存	能被缓存	不能缓存
编码类型	application/x-www-form-urlencoded	application/x-www-form-urlencoded 或 multipart/form-data。为二进制数据使用多重编码。
历史	参数保留在浏览器历史中。	参数保留在浏览器历史中。
对数据长度的限制	是的。当发送数据时，GET 方法向 URL 添加数据；URL 的长度是受限制的（URL 的最大长度是 2048 个字符）。	无限制
对数据类型的限制	只允许 ASCII 字符。	没有限制。也允许二进制数据。
安全性	与 POST 相比，GET 的安全性较差，因为所发送的数据是 URL 的一部分。在发送密码或其他敏感信息时绝不要使用 GET ！	POST 比 GET 更安全，因为参数不会被保存在浏览器历史或 web 服务器日志中。
可见性	数据在 URL 中对所有人都是可见的。	数据不会显示在 URL 中。

4 副作用和幂等的概念

副作用指对服务器上的资源做改变，搜索是无副作用的，注册是副作用的。

幂等指发送 M 和 N 次请求（两者不相同且都大于 1），服务器上资源的状态一致，比如注册 10 个和 11 个帐号是不幂等的，对文章进行更改 10 次和 11 次是幂等的。因为前者是多了一个账号（资源），后者只是更新同一个资源。

在规范的应用场景上说，Get 多用于无副作用，幂等的场景，例如搜索关键字。Post 多用于副作用，不幂等的场景，例如注册。

5 技术上的不同

- Get 请求能缓存，Post 不能
- Post 相对 Get 安全一点点，因为 Get 请求都包含在 URL 里（当然你想写到 body 里也是可以的），且会被浏览器保存历史纪录。Post 不会，但是在抓包的情况下都是一样的。
- URL 有长度限制，会影响 Get 请求，但是这个长度限制是浏览器规定的，不是 RFC 规定的
- Post 支持更多的编码类型且不对数据类型限制

6 首部

首部分为请求首部和响应首部，并且部分首部两种通用，接下来我们就来学习一部分的常用首部。

1. 通用首部

通用字段	作用
Cache-Control	控制缓存的行为
Connection	浏览器想要优先使用的连接类型，比如 keep-alive
Date	创建报文时间
Pragma	报文指令
Via	代理服务器相关信息
Transfer-Encoding	传输编码方式
Upgrade	要求客户端升级协议
Warning	在内容中可能存在错误

1. 请求首部

请求首部	作用
Accept	能正确接收的媒体类型
Accept-Charset	能正确接收的字符集
Accept-Encoding	能正确接收的编码格式列表
Accept-Language	能正确接收的语言列表
Expect	期待服务端的指定行为
From	请求方邮箱地址
Host	服务器的域名
If-Match	两端资源标记比较
If-Modified-Since	本地资源未修改返回 304（比较时间）
If-None-Match	本地资源未修改返回 304（比较标记）
User-Agent	客户端信息
Max-Forwards	限制可被代理及网关转发的次数
Proxy-Authorization	向代理服务器发送验证信息
Range	请求某个内容的一部分
Referer	表示浏览器所访问的前一个页面
TE	传输编码方式

1. 响应首部

响应首部	作用
Accept-Ranges	是否支持某些种类的范围
Age	资源在代理缓存中存在的时间

响应首部	作用
ETag	资源标识
Location	客户端重定向到某个 URL
Proxy-Authenticate	向代理服务器发送验证信息
Server	服务器名字
WWW-Authenticate	获取资源需要的验证信息

1. 实体首部

实体首部	作用
Allow	资源的正确请求方式
Content-Encoding	内容的编码格式
Content-Language	内容使用的语言
Content-Length	request body 长度
Content-Location	返回数据的备用地址
Content-MD5	Base64 加密格式的内容 MD5 检验值
Content-Range	内容的位置范围
Content-Type	内容的媒体类型
Expires	内容的过期时间
Last_modified	内容的最后修改时间

7 GET 和 POST 报文上的区别

结论：GET 和 POST 方法没有实质区别，只是报文格式不同。

GET 和 POST 只是 HTTP 协议中两种请求方式，而 HTTP 协议是基于 TCP/IP 的应用层协议，无论 GET 还是 POST，用的都是同一个传输层协议，所以在传输上，没有区别。

报文格式上，不带参数时，最大区别就是第一行方法名不同

POST 方法请求报文第一行是这样的 `POST /products/create HTTP/1.1`

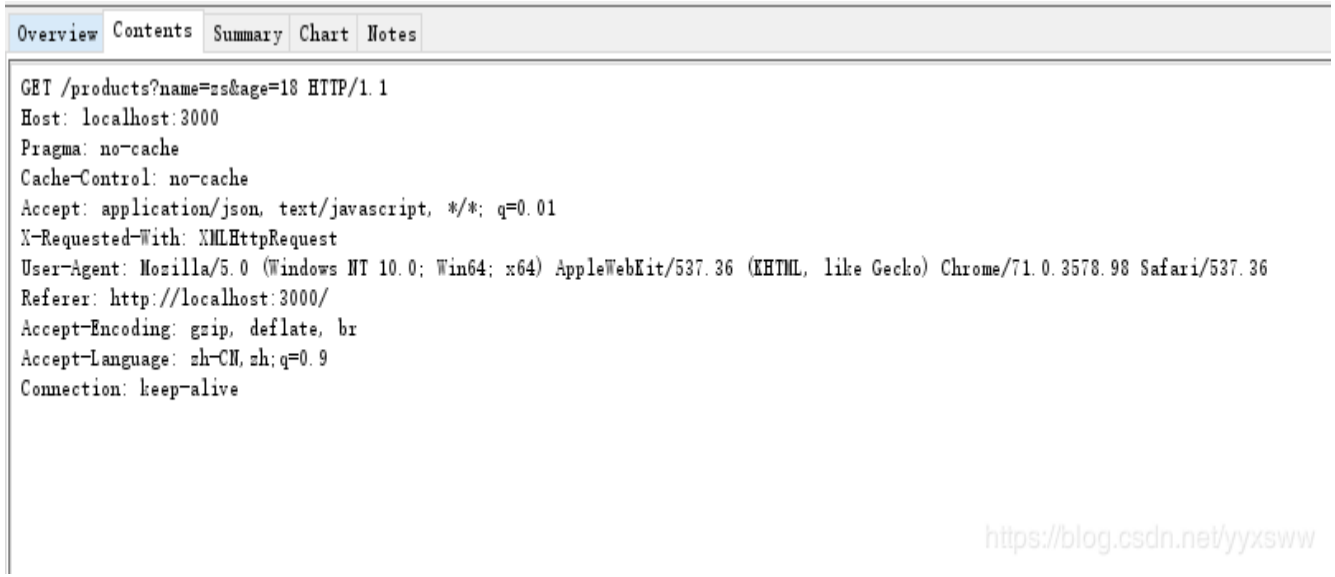
GET 方法请求报文第一行是这样的 `GET /products?name=zs&age=18 HTTP/1.1`

是的，不带参数时他们的区别就仅仅是报文的前几个字符不同而已

带参数时报文的区别呢？ 在约定中，GET 方法的参数应该放在 url 中，POST 方法参数应该放在 body 中

举个例子，如果参数是 `pname='小米 9', pprice=3500`

GET 请求方法的报文如下：



POST 请求方法的报文如下：

现在我们知道了两种方法本质上是 TCP 连接，没有差别，也就是说，如果我不按规范来也是可以的。我们可以在 URL 上写参数，然后方法使用 POST；也可以在 Body 写参数，然后方法使用 GET。当然，这需要服务端支持。

接口评审：明确出需求 然后开会评审 要什么接口 接口参数、返回 json 内容、格式 协定好 再做

第三方接口评审规范

接口测试&Python 实现接口测试

一、接口测试是什么
1、接口测试是软件测试
2、接口测试是集成测试的一部分
3、接口测试不等同于接口自动化测试
4、大家常说的接口测试大部分是指功能性的接口测试，实际还包括性能的接口测试和安全性的接口测试等
5、需要做接口测试的情况大致为：
[1]系统与系统之间的调用。例如：淘宝的订单系统和支付系统；
[2]上层服务对下层服务的调用。例如：服务层对数据层的调用；
[3]服务之间的调用。
二、接口测试的意义
1、稳：接口相对 UI 稳定，当接口自动化创建后，相对稳定的运行
2、低（成本）：因为接口相对稳定，所以一旦建立，不需要大量的维护成本
3、快：执行快、响应快
三、接口测试的流程
流程：
接口文档 — 接口测试计划、方案 — 接口测试用例（评审）— 执行 — 集成到 Jenkins — 接口反馈
1、接口文档
[1]接口文档五要素：接口地址、接口请求的方式、是否有请求参数（参数相关属性）、返回参数说明（参数相关属性）、返回结果样例。

[2]如果没有接口文档，到功能测试阶段，需要自己抓包，抓包工具如 Fiddler 等

2、设计接口测试用例的原则：看测试的目的是什么，如果目的是调通，那么力度可以小点；如果是业务层面，那么需要结合需求文档，用例需要覆盖全面。

3、设计接口测试用例的目的：

[1]检查返回数据类型与接口文档是否一致；

[2]检查返回字段值与数据库值是否一致；

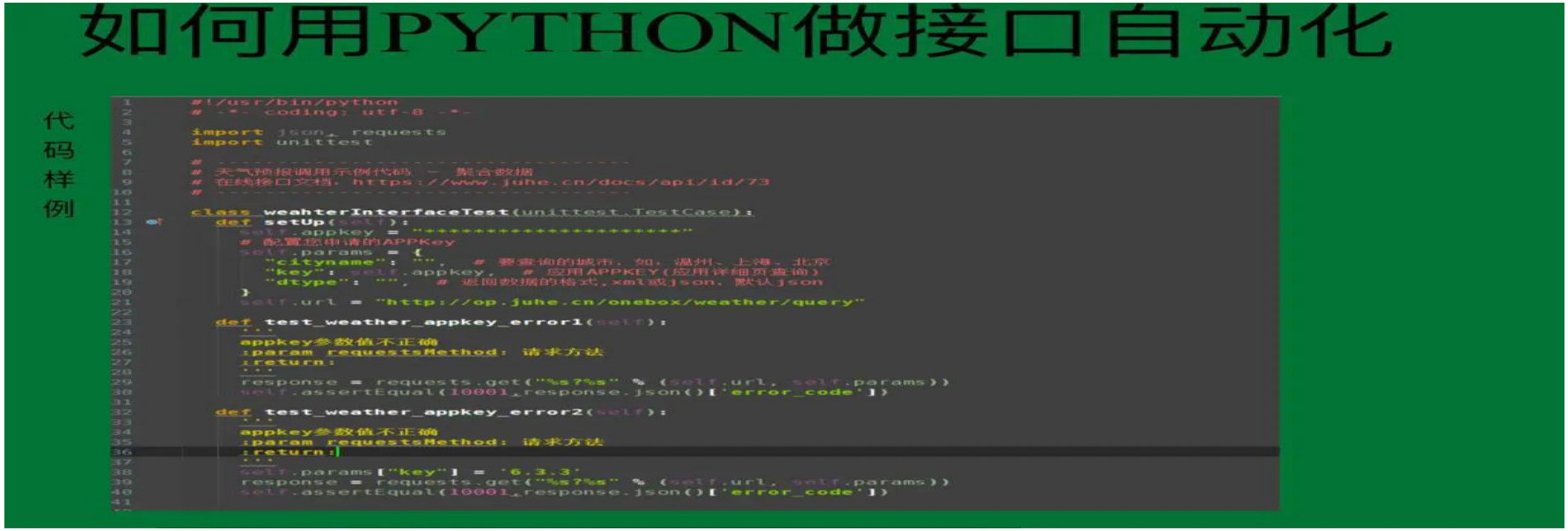
4、测试点

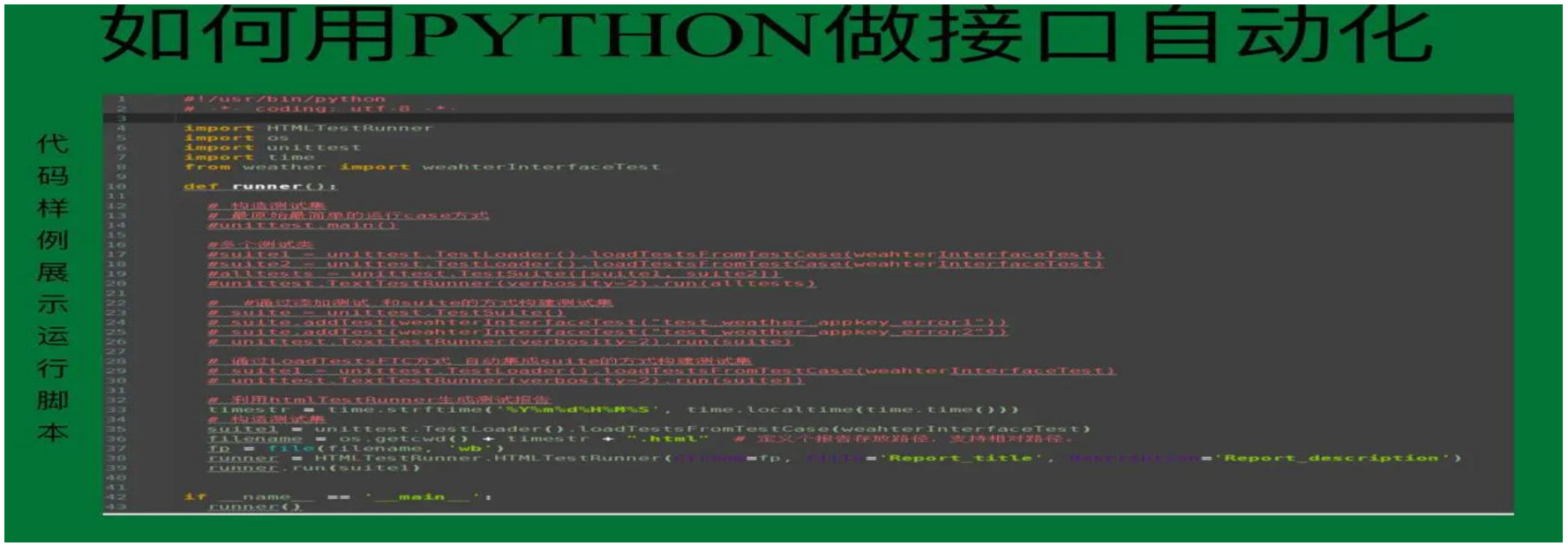
[1]单一接口功能的测试主要测试返回的数据结构是否和接口文档给出的一致

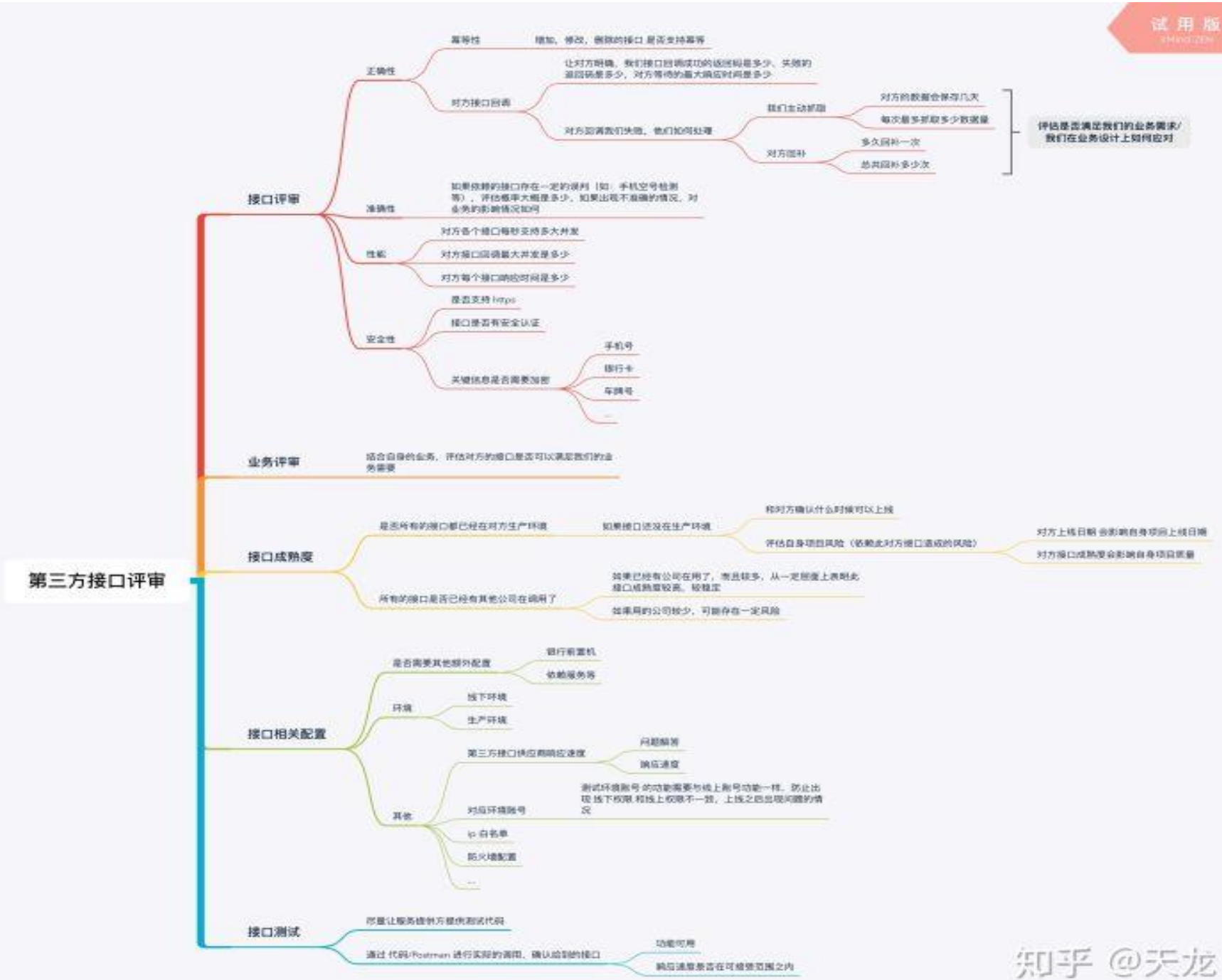
[2]接口的正常功能是否完成

[3]接口的参数检查测试，接口的异常测试

[4]多接口组合测试，实际上是在测试一个业务流。
[5]在测试过程中一次调用多个接口。
四、代码实现






第三方接口评审
接口评审
正确性
幂等性
增加、修改、删除的接口是否支持幂等
对方接口回调
让对方明确，我们接口回调成功的返回码是多少、失败的返回码是多少，对方等待的最大响应时间是多少
对方回调我们失败，他们如何处理
我们主动抓取
对方的数据会保存几次
每次最多抓取多少数据量
对方回补
多久回补一次
总共回补多少次
评估是否满足我们的业务需求/我们在业务设计上如何应对
准确性
如果依赖的接口存在一定的误判（如：手机空号检测等），评估概率大概是多少，如果出现不准确的情况，对业务的影响情况如何
性能
对方各个接口每秒支持多大并发
对方接口回调最大并发是多少
对方每个接口响应时间是多少
安全性
是否支持 https
接口是否有安全认证
关键信息是否需要加密
手机号
银行卡
车牌号
...
业务评审
结合自身的业务，评估对方的接口是否可以满足我们的业务需要
接口成熟度
是否所有的接口都已经在对方生产环境
如果接口还没在生产环境
和对方确认什么时候可以上线
评估自身项目风险（依赖此对方接口造成的风险）
对方上线日期 会影响自身项目上线日期
对方接口成熟度会影响自身项目质量
所有的接口是否已经有其他公司在调用了
如果已经有公司在用了，调用较多，从一定程度上表明此接口成熟度较高，较稳定
如果用的公司较少，可能存在一定风险
接口相关配置
是否需要其他额外配置
银行前置机
依赖服务等
环境
线下环境
生产环境
其他
第三方接口快应商响应速度
问题解答
响应速度
对应环境账号
测试环境账号 的功能需要与线上账号功能一样，防止出现 线下权限 和线上权限不一致，上线之后出现问题的情况
ip 白名单
防火墙配置
...
接口测试
尽量让服务提供方提供测试代码
通过 代码/Postman 进行实际的调用，确认给到的接口
功能可用
响应速度是否在可接受范围之内

第三方接口评审
接口评审
正确性
- 幂等性
- 增加，修改，删除的接口 是否支持幂等
- 对方接口回调
- 让对方明确，我们接口回调成功的返回码是多少，失败的返回码是多少，对方等待的最大响应时间是多少
- 对方回调我们失败，他们如何处理
- 我们主动抓取
- 对方的数据会保存几天
- 每次最多抓取多少数据量
- 对方回补
- 多久回补一次
- 总共回补多少次
准确性
- 如果依赖的接口存在一定的误判（如：手机空号检测等），评估概率大概是多少，如果出现不准确的情况，对业务的影响情况如何
性能
- 对方各个接口每秒支持多大并发
- 对方接口回调最大并发是多少
- 对方每个接口响应时间是多少
安全性
- 是否支持 https
- 接口是否有安全认证
- 关键信息是否需要加密
- 手机号
- 银行卡
- 车牌号
- ...
业务评审
结合自身的业务，评估对方的接口是否可以满足我们的业务需要
接口成熟度
是否所有的接口都已经在对方生产环境
- 如果接口还没在生产环境
- 和对方确认什么时候可以上线
- 评估自身项目风险（依赖此对方接口造成的风险）

- 对方上线日期 会影响自身项目上线日期
- 对方接口成熟度会影响自身项目质量
所有的接口是否已经有其他公司在调用了
- 如果已经有公司在用了，而且较多，从一定层面上表明此接口成熟度较高，较稳定
- 如果用的公司较少，可能存在一定风险
接口相关配置
是否需要其他额外配置
- 银行前置机
- 依赖服务等
环境
- 线下环境
- 生产环境
其他
- 第三方接口供应商响应速度
- 问题解答
- 响应速度
- 对应环境账号
- 测试环境账号 的功能需要与线上账号功能一样，防止出现 线下权限 和线上权限不一致，上线之后出现问题的情况
- ip 白名单
- 防火墙配置
- ...
接口测试
尽量让服务提供方提供测试代码
通过 代码/Postman 进行实际的调用，确认给到的接口
- 功能可用
- 响应速度是否在可接受范围之内

盗亦有道：如何来评审接口文档

评审一项东西，首先是要你自己心理有谱，需要评审那些点，所以评审本质是评审两样东西：第一，该有的点是否都有；第二，这些点是否达到了要求。
在评审接口文档的时候，需要评审：
1.接口参数是否足够，要知道参数本质就是描摹接口，所以参数应该是对一个接口方法的一个很好的说明；场景是否都满足；参数的描述是否足够具体，便于他人了解；
2.返回值是否足够；返回参数的描述是否具体，便于他人了解；
3.错误返回是否全面；
4.示例是否准确，清晰。

接口管理规范：如何写出完美的接口？接口规范定义、接口管理工具推荐?

无规矩不成方圆，为了开发人员间更好的配合，我特意整理了这么一篇文档供大家参考学习，如有意见、见解，请在评论区留言探讨。

接口规范说起来大，其实也就那么几个部分，接口规范、接口管理工具、接口文档编写、开发文档编写。

接口规范定义

一、协议规范

为了确保不同系统/模块间的数据交互，需要事先约定好通讯协议，如：TCP、HTTP、HTTPS 协议。为了确保数据交互安全，建议使用 HTTPS 协议。

二、接口路径规范

作为接口路径，为了方便清晰的区分来自不同的系统，可以采用不同系统/模块名作为接口路径前缀。

格式规范如下：

支付模块 /pay/xx

订单模块 /order/xx

三、版本控制规范

为了便于后期接口的升级和维护，建议在接口路径中加入版本号，便于管理，实现接口多版本的可维护性。如果你细心留意过的话，你会发现好多框架对外提供的 API 接口中(如：Eureka)，都带有版本号的。如：接口路径中添加类似"v1"、"v2"等版本号。

格式规范如下：

/xx/v1/xx

更新版本后可以使用 v2、v3 等、依次递加。

四、接口命名规范

和 Java 命名规范一样，好的、统一的接口命名规范，不仅可以增强其可读性，而且还会减少很多不必要的口头/书面上的解释。

可结合【接口路径规范】、【版本控制规范】，外加具体接口命名(路径中可包含请求数据，如：id 等)，建议具体接口命名也要规范些，可使用"驼峰命名法"按照实现接口的业务类型、业务场景等命名，有必要时可采取多级目录命名，但目录不宜过长，两级目录较为适宜。

格式规范如下：

/user/v1/sys/login 用户服务/模块的系统登录接口

/zoo/v1/zoos/{ID} 动物园服务/模块中，获取 id 为 ID 的动物

具体接口命名，通常有以下两种方式：

接口名称动词前/后缀化

接口名称以接口数据操作的动词为前/后缀，常见动词有：add、delete、update、query、get、send、save、detail、list 等，如：新建用户 addUser、查询订单详情 queryOrderDetail。

接口名称动词+请求方式

> 接口路径中包含具体接口名称的名词，接口数据操作动作以 HTTP 请求方式来区分。常用的 HTTP 请求方式有：
>
> GET：从服务器取出资源（一项或多项）。
>
> POST：在服务器新建一个资源。
>
> PUT：在服务器更新资源（客户端提供改变后的完整资源）。

```
  PATCH：在服务器更新资源（客户端提供改变的属性）。
  DELETE：从服务器删除资源。
如：
   GET /zoo/v1/zoos：列出所有动物园
  POST /zoo/v1/zoos：新建一个动物园
  GET /zoo/v1/zoos/{ID}：获取某个指定动物园的信息
  PUT /zoo/v1/zoos/{ID}：更新某个指定动物园的信息（提供该动物园的全部信息）
  PATCH /zoo/v1/zoos/{ID}：更新某个指定动物园的信息（提供该动物园的部分信息）
  DELETE /zoo/v1/zoos/{ID}：删除某个动物园
  GET /zoo/v1/zoos/{ID}/animals：列出某个指定动物园的所有动物
  DELETE /zoo/v1/zoos/ID/animals/ID：删除某个指定动物园的指定动物
```

五、请求参数规范

请求方式:

按照 GET、POST、PUT 等含义定义，避免出现不一致现象，对人造成误解、歧义。

请求头:

请求头根据项目需求添加配置参数。如：请求数据格式，accept='application/json'等。如有需要，请求头可根据项目需求要求传入用户 token、唯一验签码等加密数据。

请求参数/请求体:

请求参数字段，尽可能与数据库表字段、对象属性名等保持一致，因为保持一致最省事，最舒服的一件事。

六、返回数据规范

统一规范返回数据的格式，对己对彼都有好处，此处以 json 格式为例。返回数据应包含：返回状态码、返回状态信息、具体数据。

格式规范如下:

```
{
  "status":"000000",
  "msg":"success",
  "data": {
    //json 格式的具体数据
  }
}
```

返回数据中的状态码、状态信息，常指具体的业务状态，不建议和 HTTP 状态码混在一起。HTTP 状态，是用来体现 HTTP 链路状态情况，如：404-Not Found。HTTP 状态码和 json 结果中的状态码，并存尚可，用于体现不同维度的状态。

接口管理工具推荐

接口开发完后，最终的目的是提供给其他系统/模块来使用的，因此，接口的管理是必不可少的。

接口管理的痛点

接口的管理常常面临很多的痛苦，这里就列举几个常见的，看看你是否也遇到过。

系统/模块太多、接口太多，没有系统统一管理所有接口。

代码修改后，接口文档没有及时更新，造成接口文档和实际接口不一致的现象。

接口管理系统自主研开发成本高。

接口管理缺少接口 mock 功能。

接口管理工具推荐

在日常工作过程中用过、接触过的接口管理工具也是不尽其数，下面介绍你可能使用过、没有使用过的接口管理工具，同时也介绍这些接口管理工具的优缺点。

word

相信大家之前用来管理接口比较多的应该是 word 吧，开发人员将系统的接口维护在 word 文档里，不管是组内沟通还是和其他团队的接口沟通都离不开这些接口文档，每次修改文档和代码都要同步修改。相信使用 word 的缺点大家应该也很清楚，就是维护和管理很麻烦，我们经常会遇到文档和代码不一致的情况，大部分不一致都是因为接口因为种种原因修改了，开发人员大部分都是只改了代码里的接口实现，而没有去修改接口文档。而且 word 文档搜索接口也很麻烦，没办法建全局索引，只能一个个文档点开查看，想想就很痛苦。但不可否认的是，word 对于一些小团队用起来还是挺方便的，不用搭建系统，给谁一看就明白。

自建接口管理系统

对于一些有一定规模的企业，在各项工程管理活动上都非常正规，各种 ISO 标准要遵守，自然对接口管理的要求也非常高，之前在国有银行，我们就是自建了接口管理系统，自建还是很消耗人力成本的，从开发到后续运维，都要消耗人力，但是自建的好处就是，可以根据公司的要求进行各种花样的定制，我们之前在接口管理系统中加入了很多好用的定制功能，例如接口被哪些系统调用、接口是在哪个批次投产又在哪个批次做过变更等等，这对于架构师来说非常好用，用于分析接口影响范围非常方便。目前开源的接口管理系统还没有能做到这些定制化功能的。

wiki

之前在小团队的时候还用过一段时间的私有 wiki，wiki 特别适合于小团队高速线性迭代开发，在 wiki 上看到的就是最新的接口，团队内所有成员看到的都是一样的，如果接口有变化，相关开发人员修改后立即生效，保证了顺畅的接口沟通。但是 wiki 的缺点也很多，接口文档只是静态页面，无法实现一些动态效果，无法实现追溯等等缺点。

RAP

相信很多互联网公司都在使用 RAP，RAP 是阿里开源的一套接口管理系统，RAP 可以比较方便的管理公司所有系统的接口，同时还有比较完善的权限管理，还可以做接口 mock，方便开发人员在接口功能还没有完成的时候能够及时发布出去，给调用方去使用。但是 RAP 的缺点就是每个接口都需要维护进去，接口修改后也需要及时维护，当时我们在使用的时候遇到的最大的问题也是经常碰到接口没有及时维护的问题。

swagger

上面说的那些接口管理工具，其实都有一个很大的问题就是修改代码后需要同步维护接口文档，但是让程序员去修改文档是很难的，大部分程序员都比较讨厌维护各类文档。当我第一次了解到 swagger 的时候，发现这简直就是为程序员定制的接口管理工具，swagger 定义了很多注解，在对接口加上 swagger 相关的注解，当接口代码修改后，swagger 在工程启动后会根据代码自动生成最新的接口 html 文档，同时 swagger 提供了 mock 接口模拟的功能，也能够更加方便的模拟接口，并且还能够在 swagger 界面上直接发起接口调用，可以方便调用方在还没写代码的时候就能够尝试下接口调用后的结果。

看了那么多 swagger 的优点，下面也说说 swagger 的缺点，那就是 swagger 是跟随着每个工程一起启动的，这就导致每个工程都有一个 swagger 的访问地址，如果公司系统很多的话，那就会导致查看不同系统的接口都要到不同的地址去查看，每个开发都要自己收藏好各个系统的 swagger 地址。有些公司也自己开发了统一网关，将所有 swagger 的接口地址聚合起来，但是多少还是涉及到一些开发工作的，而且做的还不一定很完善。

Easy Mock

官网的这张图基本上介绍清楚了 easymock 的核心功能，这其中我最看重的功能有两块，一个是能够集成 swagger 接口并集中管理所有接口，另一个就是响应式数据。

EasyMock 能够根据 swagger 接口的地址自动导入所有 swagger 接口，非常方便，对于非 swagger 的接口也可以手工维护进去，这样可以很方便的做到全公司接口统一维护，而且也有比较完善的接口权限管理，方便分组管理。但缺点就是过于庞大，可能太适合小一点项目或团队。

上面提及到接口管理工具，大家可根据自己项目的规模、需求，进行实际选择，切记生搬硬套。

接口规范文档总结、接口管理工具推荐、如何写出完美的接口

写在前面：这是我最近整理的接口规范文档，无规矩不成方圆，为了 app 开发人员与后台接口开发人员更好的配合，我特意整理了这么一篇文档供大家参考学习，如有意见请在评论区留言谢谢。因部分内容涉及公司代码，我对本文档略有删减。

接口规范说起来大，其实也就那么几个部分，接口规范、接口管理工具、接口文档编写、开发文档编写。以下将详细介绍，下面进入正文：

接口规范文档

具体内容如下：

一：协议规范

二：域名规范

三：版本控制规范

四：API 路径规范

五：API 命名规范

六：请求参数规范

七：列表请求特殊规范
八：返回数据规范
九：接口文档规范
十：接口管理工具使用教程
参与编写
更新日期
版本
备注
AbyssKitty
2018/04/06
V1.1.0
无
V1.1.0 更新日志：
1. 新增协议规范、域名规范、版本控制规范、列表特殊规范。
2. 更新接口管理工具使用教程。
3. 美化排版。
正文：
一：协议规范
为进一步确保数据交互安全。正式地址（生产地址）必须遵循 HTTPS 协议。
二：域名规范
每个项目要有且仅有一个自己唯一的域名+端口。在项目配置文件中要添加静态变量专门进行存储。
如果一个域名满足不了要求，那么就需要再添加一个。
格式规范如下：
（java）public static final String URL_BASE = “https://127.0.0.1:8080/”;
（java）public static final String URL_BASE_SUB = “https://192.168.0.1:8080/”;
必须以 https 开头，并以“/”结尾。
三：API 路径规范
作为接口路径，为了和其他路径完美区分，必须在路径中添加 api 目录
格式规范如下：
（java）public static final String URL_API = “api/”;
（PHP）php 目录是加 index.php/api/
必须以字母开头，并以“/”结尾。
四：版本控制规范
项目正式上线后，正式版本要确定接口版本、并备份接口代码。
为方便管理，需要在接口路径中加入版本号信息。

格式规范如下：
（java）public static final String URL_VERSION =”v1/”;
必须以字母开头，并以“/”结尾。
更新版本后可以使用 v2 v3 等、依次递加。
五：API 命名规范
根据二：域名规范、三：API 路径规范、四：版本控制规范。项目中必须在配置文件中增加 BaseUrl 静态常量。值=三个相加。
格式规范如下：
（java）public static final String BASEURL=URL_BASE+URL_API+URL_VERSION;
具体代码如下：
BASEURL = [“https://127.0.0.1:8080/api/v1/”]
BASEURL = [“https://127.0.0.1:8080/api/v1/”]
BASEURL = [“https://127.0.0.1:8080/api/v1/”]
重要的事情说三遍。
根据业务需求，可以在 v1 版本文件夹里创建，一个或者多个接口文件。
一个的规范：
https://127.0.0.1:8080/api/v1/getBanner
这就是一个获取 banner 的接口。
多个的规范是根据业务需求来区分：
https://127.0.0.1:8080/api/v1/home/getBanner
https://127.0.0.1:8080/api/v1/user/userLogin
新建 user 文件，里面存放用户级别的操作：如登陆、注册、修改密码等等。
新建 sms 文件，里面存放对短信的接口操作：如发送验证码、验证手机号等等。
所以，接口方法文件必须要有自己的规范，命名必须统一使用驼峰命名法或者下划线拼接命名法。举个栗子：（upperCamelCase）（upper_camel_case）。所有接口命名方式，必须遵循如下规范。
（1）新增方法：如新增一个地址、新增一个联系人。
命名规范：
必须以“add”为前缀。例如 addAddress
事例地址：https://127.0.0.1:8080/api/v1/addAddress
（2）删除方法：如删除一个地址。
命名规范：
必须以“delete”为前缀。例如 deleteAddress
事例地址：https://127.0.0.1:8080/api/v1/deleteAddress
（3）修改方法：如修改一个地址。
命名规范：
必须以“updata”为前缀。例如 updataAddress

事例地址：https://127.0.0.1:8080/api/v1/updataAddress
（4）获取方法：如获取一个地址。
命名规范：
必须以“get”为前缀。例如 getAddress
事例地址：https://127.0.0.1:8080/api/v1/getAddress
（5）获取列表方法：如获取一个地址列表。
命名规范：
必须以“get”为前缀、“List”为后缀。例如 getAddressList
事例地址：https://127.0.0.1:8080/api/v1/getAddressList
其他规范：
发送验证码使用‘send’为前缀、保存一个数据以‘save’为前缀、上传图片以‘uploadImage’为名称等等。
具体地址就等于（BASEURL+“address/getAddressList”）
目的：一目了然、降低维护成本。
六：请求参数规范
请求方式：公共数据使用 get 方式请求，私有数据使用 post 方式请求。尽量全部是用 post。
请求头：请求头根据项目需求添加配置参数。如：accept=‘application/json
’等。请求头根据项目需求可以要求传入用户 token、app 名称版本、唯一验签码等加密数据。
　请求参数：
根据数据库字段进行命名、保持一致最省事。
七：列表请求特殊规范
列表请求，请求参数规范，必须传参：页数和每页个数的字段。并可包含查询等信息。
1. 列表接口必传字段（分页、使用小写字母）。
page ： 页数，从 1 开始。例如：{ “page”: 1 }
subnumber : 每页数量。
2. 列表接口选传字段。
只要是列表接口、一般情况下都会存在检索条件，例如淘宝商品检索。检索条件为选填。
后台需进行非空非 null 判断，选传字段为空为 null 默认查询全部。有值则需要接收，并进行 sql 查询。
规范事例：
普通列表接口：https://127.0.0.1:8080/api/v1/getBannerList
（不传 page、后台默认返回全部数据）
（banner 接口不需要使用检索条件）
需检索列表接口：https://127.0.0.1:8080/api/v1/getOrderList
（不传 page、后台默认返回全部数据）
（Order 订单接口需要检索条件，不传就不检索，只进行分页查询）
（如果有 time price 等检索条件，不传就不检索，传了就进行条件查询，并返回相应数据）

八：返回数据规范
注：列表数据返回，没有特殊情况的条件下，必须最新数据在上面，依次排序。
返回事例：
{
"list":[],
"object":{}, //"object":""
"status":"SUCCESS",
"message":"我是提示消息",
...
"page":1,
"subnumber":10,
}
必选-命名规范：驼峰命名法。
必选-新增键值规则：名字对应固定的格式（list 就是数组[]）。
举个栗子：比如一个"list":[]满足不了需求，那么可以新增一个"map":[]。
比如一个"object":{"name":"小明"}满足不了需求，那么可以新增一个"details":{"name":"小红"}。名字对应固定的格式，数组就是数组、实体类就是实体类、字段就是字段。不能 data 在这个接口返回的是实体类、另一个接口又返回数组了。需要特别注意。
必选-list：list 列表（数组）为空时显示[]。
必选-object：实体数据，json 键值对。
必选-status：状态信息=SUCCESS、ERROR 等静态变量。
必选-message：提示消息。（加载成功、）
可选-page：页数（分页查新时使用、显示第几页从一开始）。
可选-subnumber：每页的格式（分页查询时使用、显示当前页的个数）。
九：接口文档规范
接口文档需要包含以下部分：
文档名称。
版本号。
编写人。
编写、修改日期。
baseUrl 地址。
更新日志。
接口详情。（详情规范如下）
接口详情编辑规范：
一个完整的接口需要由以下几部分组成
1.请求地址 例如：https://127.0.0.1:8080/xxx/xxx/xxx

2.请求方式 例如：POST、GET 等
3.请求参数 例如：传 id：“1”，name：“小明”
4.返回参数 例如：{ json... } 【参考上面的接口规范】
5.返回事例 例如：{ json... }
十：接口管理工具使用教程
接口管理工具有很多，例如 RAP、eolinker 等等。
接口管理工具基本的作用都是用来管理接口的。这里简单介绍 eolinker 的使用方法。
使用方法步骤：
创建接口管理项目。
邀请开发者同事加入。
编写接口（接口地址、请求参数及备注、请求方式、返回参数及备注、返回事例、在线测试接口）。
开发者使用接口。
过程中灵活配合，接口可以灵活更新。
完成项目后可以导出接口文档。
附件：XXX 接口管理工具使用教程点击进入 eolinker 使用教程
RAP 的特色：
RAP 是一个 GUI 的 WEB 接口管理工具。在 RAP 中，您可定义接口的 URL、请求&响应细节格式等等。通过分析这些数据，RAP 提供 MOCK 服务、测试服务等自动化工具。
RAP 同时提供大量企业级功能，帮助企业和团队高效的工作。
在前后端分离的开发模式下，我们通常需要定义一份接口文档来规范接口的具体信息。如一个请求的地址、有几个参数、参数名称及类型含义等等。RAP 首先方便团队录入、查看和管理这些接口文档，并通过分析结构化的文档数据，重复利用并生成自测数据、提供自测控制台等等... 大幅度提升开发效率。
强大的 GUI 工具 给力的用户体验，你将会爱上使用 RAP 来管理您的 API 文档。
完善的 MOCK 服务 文档定义好的瞬间，所有接口已经准备就绪。有了 MockJS，无论您的业务模型有多复杂，它都能很好的满足。
庞大的用户群 RAP 在阿里巴巴有 200 多个大型项目在使用，也有许多著名的公司、开源人士在使用。RAP 跟随这些业务的成行而成长，专注细节，把握质量，经得住考验。
免费 + 专业的技术支持 RAP 是免费的，而且你的技术咨询都将在 24 小时内得到答复。大多数情况，在 1 小时内会得到答复。
RAP 是一个可视化接口管理工具 通过分析接口结构，动态生成模拟数据，校验真实接口正确性， 围绕接口定义，通过一系列自动化工具提升我们的协作效率。
完本。

API 接口规范(试行版)

1.协议
API 与用户的通信协议，总是使用 HTTPS 协议，确保交互数据的传输安全。
2.安全
为了保证接口接收到的数据不是被篡改以及防止信息泄露造成损失，对敏感数据进行加密及签名。
数据加密
api 接口请求参数一律采用 RSA 进行加解密，在客户端使用公钥对请求参数进行加密，在服务端使用对数私钥据进行解密，防止信息泄露。
签名
为了防止请求数据在网络传输过程中被恶意篡改，对所有非查询接口增加数字签名，签名原串为对请求参数进行自然排序，通过私钥加签后放入 sign 参数中。

时间戳
api 接口中增加时间戳 timestamp 字段，作用：固定时间范围内，减少同一请求被暴力调用的次数。
3.API 版本控制
API 的版本号统一放入 URL。
https://api.example.com/v{n}/
v{n}n 代表版本号，分为整形和浮点型
整形版本号：大功能版本发布形式；具有当前版本状态下的所有 API 接口，例如：v1,v2
浮点型：为小版本号，只具备补充 api 的功能，其他 api 都默认调用大版本号的 API，例如 v1.1,v1.2
4.API 路径规则
在 RESTful 架构中，每个网址代表一种资源(resource)，所以网址中不能有动词，只能有名词。名词尽量与数据库表格名对应。
例子:
https://api.example.com/v1/products
https://api.example.com/v1/users
https://api.example.com/v1/employees
5.HTTP 请求方式
对于资源的具体操作类型，由 HTTP 动词表示。
常用的 HTTP 动词由下面四个(括号里是对应的 SQL 命令)。
GET(SELECT):从服务器取出资源。
POST(CREATE):在服务器新建一个资源。
PUT(UPDATE):在服务器更新资源。
DELETE(DELETE):从服务器删除资源。
例子:
GET/product:列出所有商品
POST/product:新建一个商品
GET/product/ID:获取某个指定商品的信息
PUT/product/ID:更新某个指定商品的信息
DELETE/product/ID:删除某个商品
GET/product/ID/purchase:列出某个指定商品的所有投资者
GET/product/ID/purchase/ID:获取某个指定商品的指定投资者信息
6.请求数据
公共请求参数

参数名称	参数类型	是否必填	最大长度	描述	示例
charset	String	是	10	请求使用的编码格式如:utf-8,gbk	utf-8

参数名称	参数类型	是否必填	最大长度	描述	示例
sign_type	String	是	10	生成签名字符串所使用的算法类型	RSA
sign	String	是	344	请求参数签名串	djdu7dusufiusgfu
timestamp	String	是	14	发送请求的时间,格式:yyyyMMddHHmmss	20180505121212
requestSource	String	否	8	客户端请求来源 APP WAP PC	APP
token	String	否		鉴权标识，用于登录判断	
biz_content	String	是		请求参数集合，除公共参数外所有请求参数	

请求参数(biz_content)

参数名称	参数类型	是否必填	最大长度	描述	示例
phone	String	是	10	登录手机号	15088890908
phoneCode	String	是	6	验证码	234567

7.返回数据

为了保障前后端的数据交互的顺畅，统一接口返回模板如下:

```
{
    code:0000,
    data:{},
    msg:''
}
```

code:接口的执行状态

0000：表示成功

其他：不太异常

Data 接口的主数据

返回 JSON 对象

Msg 信息

当 code!=0000 都应该有错误信息

8.非 RESTful API 需求

由于实际业务开展过程中，可能会出现各种的 api 不是简单的 restful 规范能实现的，因此需要一些 api 突破 restful 规范原则。

8.1 页面级 API
把当前页面中需要用到的所有数据通过一个接口一次性返回全部数据。
例子:
api/v1/get-home-data 返回首页用到的所有数据
此类 API 存在缺陷:只要业务需求变动，该 api 就需要跟着变更。
8.2 自定义组合 API
把当前用户需要在第一时间内容加载的多个接口合并成一个请求发送到服务端，服务端根据请求内容，一次性把所有数据合并返回，相比于页面级 API，具备更高的灵活性，同时又能很容易实现页面级 API 功能。
规范
地址:api/v1/testApi
传入参数:

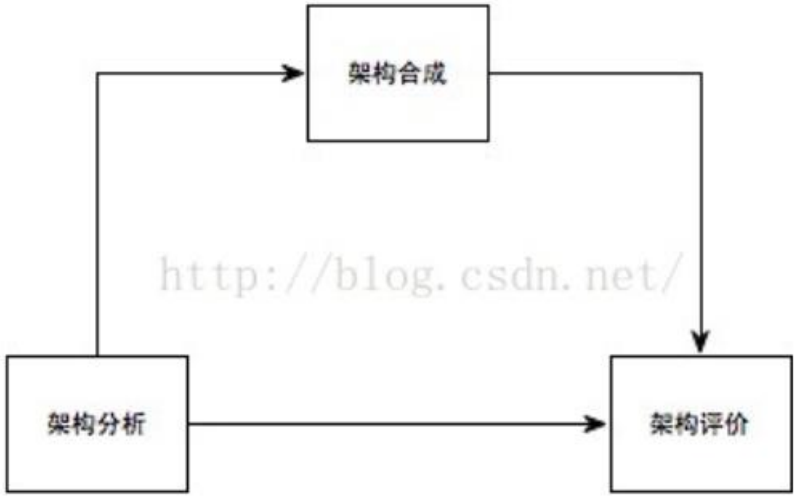
```
data:[
{url:'api1',type:'get',data:{}},
{url:'api2',type:'get',data:{}},
{url:'api3',type:'get',data:{}},
]
返回数据
{
   code:0000,
   msg:'',
   data:[
         {code:0000,msg:'',data:[]},
         {code:0000,msg:'',data:[]},
         {code:0000,msg:'',data:[]},
   ]
}
```

9.API 共建平台
RAP 是一个 GUI 的 WEB 接口管理工具。在 RAP 中，可定义接口的 URL、请求&响应格式等等。通过分析这些数据，RAP 提供 MOCK 服务、测试服务等自动化工具。
9.1 什么是 RAP?
在前后端分离的开发模式下，我们通常需要定义一份接口文档来规范接口的具体信息。如请求地址、有几个参数、参数名称及类型含义等等。RAP 首先方便团队录入、查看和管理这些接口文档，并通过分析结构化的文档数据，重复利用并生成自测数据、提供自测控制台等等。
9.2RAP 的特色
通过 RAP 来管理 API 文档。

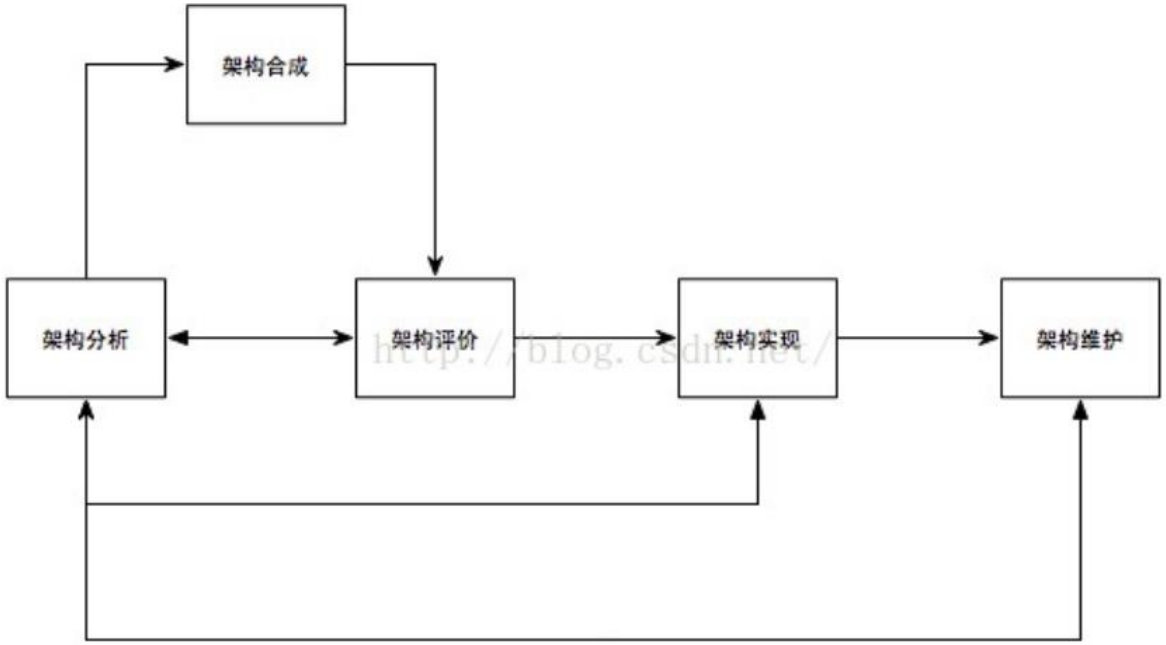
提供 MOCK 服务，通过 MockJS 创建 mock 测试数据。
参考文献
架构管理规范

软件架构管理过程

上一篇博客谈到架构的起源，也就是 Dewayne E. Perry 和 Alexander L. Wolf 在 1992 年发表的“Foundations for the Study of Software Architecture”，也说到了目前架构已经到了 3.0 版本，即架构 = 一系列的架构设计决策 + 这些决策背后的原理。这次来聊聊软件架构管理过程。
我认为目前来说，大部分的架构设计或者方法都有其生存的环境，比如项目环境、企业环境、行业环境等，适合你的不一定适合就适合我，很难找到一套通用的解决方案来应对所有变化，所以目前的架构解决方案呈现多样化。但是在各种各样的架构解决方案背后，却有着相同或者相似的架构管理过程。
架构管理过程的元素是活动，最初只有三个，由 Christine Hofmeister, Philippe Kruchten, Robert L. Nord, Henk Obbink, Alexander Ran，以及 Pierre America 提出：
Architectural Analysis（架构分析活动）：该活动用于定义一个软件架构必须解决的问题。
Architectural Synthesis（架构合成活动）：该活动用于建议能解决目标问题的潜在解决方案。
Architectural Evaluation（架构评价活动）：该活动用于评估潜在的解决方案，并确保架构决策的正确性。
这三个活动构成了最初的软件架构管理过程：



在 2010 年，Antony Tang, Paris Avgeriou, Anton Jansen, Rafael Capilla，以及 Muhammad Ali Babar，扩展了软件架构活动到五个。Architectural Analysis，Architectural Synthesis，Architectural Evaluation 仍然在列，同时他们增加了：
Architectural Implementation（架构实现）：该活动用于实现设计的架构，是一个从架构设计到详细设计的过程。
Architectural Maintenance（架构维护）：该活动用于维护架构，比如通过修改架构来解决发现的错误等。
于是软件架构过程演化为：



随着架构的发展，我们发现架构活动还可以包括架构演化、架构恢复、架构描述、架构理解、架构影响分析、架构复用以及架构重构，但是以上五个活动我认为是架构的核心活动，而其他活动则是一般性活动，这些一般性活动可以出现在整个架构生命周期的任意阶段，用以支持以上五个核心活动。

最有成就感的项目，学到经验，让自己比较踏实，让自己得到启发的项目？

接口文档怎么维护，如何维护三端（小程序，安卓，网页等）连接，以一套接口为原则，接口冲突怎么解决：利用 apidoc 维护 api 接口文档?

接口中解决默认方法冲突？

如果先在一个接口中将一个方法定义为默认方法，然后又在超类或者另一个接口中定义了同样的方法，那么同时实现这两个接口的类或者既继承了超类又实现了接口的类就会发生冲突。但 Java 提供相应的规则：

1.超类优先。如果超类提供一个具体方法，同名而且有相同参数类型的默认方法会被忽略。

2.接口冲突。如果一个超接口提供了一个默认方法，另一个接口提供了一个同名而且参数类型相同的方法，就必须解决冲突。

例子

1.假设 Person 是一个类（有 getName 方法），Named 是一个接口

```
interface Named{
    default String getName(){
        return getClass().getName()
```

```
    }
}
```

有一个类 Student 继承了 Person 和实现了 Named 接口：

```
class Student extends Person implements Named{...}
```

在这种情况下，只会考虑超类方法，接口的所有默认方法都会被忽略。接口可添加默认方法在 Java SE 8 中才有，使用“类优先”原则可以确保 Java SE 7 的兼容性，如果为一个接口添加默认方法，这对于有这个默认方法之前能正常工作的代码不会有任何影响。

2.假设 Person 是一个接口（默认实现了 getName 方法），Named 依旧是一个接口，类 Student 实现了这两个接口：

```
class Student implements Person,Named{...}
```

类会继承 Person 和 Named 接口中提供的两个不一致的 getName 方法，并不是从中选择一个，这样 Java 编译器就会报告一个错误，我们需要解决这个二义性。只需要在 Student 类中提供一个 getName 方法，在这个方法中，可以选择两个冲突方法中的一个：

```
class Student implement Person,Named{
    public String getName(){
        return Person.super.getName();
    }
}
```

常见的 Content-Type 类型

MediaType,即是 Internet Media Type,互联网媒体类型；也叫做 MIME 类型， 在 Http 协议消息头中，使用 Content-Type 来表示具体请求中的媒体类型信息。

常见的媒体格式类型如下

- text/html:HTML 格式
- text/plain:纯文本格式
- text/xml:XML 格式
- image/gif:gif 图片格式
- image/jpeg:jpg 图片格式
- image/png:png 图片格式

以 application 开头的媒体格式类型：

- application/xhtml+xml:XHTML 格式
- application/xml:XML 数据格式
- application/atom+xml:Atom XML 聚合格式

- application/json:JSON 数据格式
- application/pdf:pdf 格式
- application/msword:Word 文档格式
- application/octet-stream:二进制流数据（常见的文件下载)
- application/x-www-form-urlencoded:表单中默认的 encType,表单数据被编码为 key/value 格式发送到服务器
  另外一种常见的媒体格式是上传文件时使用：
- multipart/form-data:需要在表单中进行文件上传时，就需要使用该格式
  C#中的 HttpClient 如何设置这些 Content-Type，可看下这篇:
  HttpClient 的 Content-Type 设置

轮询(效率低，基本不用)

```
"""
原理
        让浏览器每隔几秒钟通过 ajax 朝服务端发送请求来获取数据
        eg:每隔 5s 中朝服务端发送一次请求
不足之处:
        消息延迟很高
        消耗资源较多
        请求次数太高
"""
```

长轮询(兼容性好)

```
"""
原理
        服务端给每个客户端建立队列，让浏览器通过 ajax 向后端偷偷的发送请求，去各自对应的队列中获取数据，如果没有数据回阻塞，但是不会一直阻塞，会通过 timeout 参数及一场处理的方式限制阻塞事件，比如 30s 后返回客户端触发回调函数让浏览器再次发送请求

相对于轮询
        消息基本没有延迟
        请求次数降低了
        资源消耗减少
"""

#  现在 web 版本的 qq 和微信还是基于长轮询实现的(大公司 web 项目可能都会使用)
```

基于 ajax 及队列自己实现简易版本的长轮询群聊功能

大公司一般情况下都会使用上面长轮询的方式，因为兼容性好

websocket(主流浏览器和框架都支持)

```
"""
http   网络协议   不加密传输
https   网络协议   加密传输
        上面这两个协议都是短链接
websocket   网络协议  加密传输
        websocket 的诞生  真正意义上实现了服务端给客户端主动推送消息
"""
```

内部原理

```
"""
内部原理
        1.握手环节(handshake)
                目的:验证服务端是否支持 websocket 协议
                客户端浏览器第一次访问服务端的时候
                浏览器内部会自动生成一个随机字符串，将该随机字符串发送给服务端(基于 http 协议)，自己也保留一份(请求头里面)

                服务端接受到随机字符串之后，会让它跟 magic string(全球统一)做字符串的拼接
                然后利用加密算法对拼接好的字符串做加密处理(sha1/base64)
                客户端也在对产生的随机字符串做上述的拼接和加密操作

                服务单将产生好的随机字符串发送给客户端浏览器(响应头里面)
                客户端浏览器会比对服务单发送的随机字符串很我浏览器本地操作完的随机字符串是否一致，如果一致说明该服务端支持 websocket，如果不一致则不支持
        2.收发数据(send/onmessage)
                验证成功之后就可以数据交互了  但是交互的数据是加密的  需要解密处理
                前提:
                        1.数据基于网络传输都是二进制格式
                        2.单位换算  8bit = 1bytes
                读取第二个字节的后七位称之为 payload
                1.根据 payload 大小决定不同的处理方式
                =127    再读取 8 个字节  作为数据报
```

```
                =126   再读取 2 个字节 作为数据报
                <=125 不再往后读了

                2.步骤 1 之后 会对剩下的数据再读取 4 个字节(masking-key)
                之后依据 masking-key 算出真实数据
                var DECODED = "";
                        for (var i = 0; i < ENCODED.length; i++) {
                        DECODED[i] = ENCODED[i] ^ MASK[i % 4];
                                }
"""
# 课下自己整理 说出重点即可    payload    masking-key
```

4.如何创建响应式布局?

响应式布局是通过@media 实现的

```
@media （min-width：768px）{
     .pg-header{
           background-color:green;
      }
}
@media   (min-width:992px){
     .pg-header{
            background-color:pink;
     }
}
```

代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
    <style>
        body{
```

```
            margin: 0;
        }
        .pg-header{
            background-color: red;
            height: 48px;
        }
        @media (min-width: 768px) {
            .pg-header{
                background-color: aqua;
            }
        }
        @media (min-width: 992px) {
            .pg-header{
                background-color: blueviolet;
            }
        }
    </style>
</head>
<body>
    <div class="pg-header"></div>
</body>
</html>
```

5.你曾经使用过哪些前端框架？

- jQuery
- Bootstrap
- Vue.js(与 vue 齐名的前端框架 React 和 Angular)

6.什么是 ajax 请求？并使用 jQuery 和 XMLHttpRequest 对象实现一个 ajax 请求？

https://www.cnblogs.com/wcwnina/p/10316919.html（Python 开发【第十六篇】：AJAX 全套）

概述

对于 WEB 应用程序：用户浏览器发送请求，服务器接收并处理请求，然后返回结果，往往返回就是字符串（HTML），浏览器将字符串（HTML）渲染并显示浏览器上。

1、传统的 Web 应用

一个简单操作需要重新加载全局数据

2、AJAX

AJAX，Asynchronous JavaScript and XML (异步的 JavaScript 和 XML)，一种创建交互式网页应用的网页开发技术方案。

- 异步的 JavaScript：
- 使用 【JavaScript 语言】 以及 相关【浏览器提供类库】 的功能向服务端发送请求，当服务端处理完请求之后，【自动执行某个 JavaScript 的回调函数】。
- PS：以上请求和响应的整个过程是【偷偷】进行的，页面上无任何感知。
- XML
- XML 是一种标记语言，是 Ajax 在和后台交互时传输数据的格式之一

利用 AJAX 可以做：

1、注册时，输入用户名自动检测用户是否已经存在。

2、登陆时，提示用户名密码错误

3、删除数据行时，将行 ID 发送到后台，后台在数据库中删除，数据库删除成功后，在页面 DOM 中将数据行也删除。（博客园）

“伪”AJAX

由于 HTML 标签的 iframe 标签具有局部加载内容的特性，所以可以使用其来伪造 Ajax 请求。

```
<!DOCTYPE html>
<html>

        <head lang="en">
                <meta charset="UTF-8">
                <title></title>
        </head>

        <body>

                <div>
                        <p>请输入要加载的地址：<span  id="currentTime"></span></p>
                        <p>
                                <input  id="url"  type="text"  />
                                <input  type="button"  value="刷新"  onclick="LoadPage();">
                        </p>
                </div>


                <div>
                        <h3>加载页面位置：</h3>
```

```
                        <iframe  id="iframePosition"  style="width: 100%;height: 500px;"></iframe>
                </div>


                <script  type="text/javascript">

                        window.onload=  function(){
                                var myDate  =  new Date();
                                document.getElementById('currentTime').innerText  =  myDate.getTime();

                        };

                        function LoadPage(){
                                var targetUrl  =    document.getElementById('url').value;
                                document.getElementById("iframePosition").src  =  targetUrl;
                        }

                </script>

        </body>
</html>
```

原生 AJAX

Ajax 主要就是使用 【XmlHttpRequest】对象来完成请求的操作，该对象在主流浏览器中均存在(除早起的 IE)，Ajax 首次出现 IE5.5 中存在（ActiveX 控件）。

1、XmlHttpRequest 对象介绍

XmlHttpRequest 对象的主要方法：

```
a. void  open(String method,String url,Boolen async)
      用于创建请求

      参数：
            method：  请求方式（字符串类型），如：POST、GET、DELETE...
            url：        要请求的地址（字符串类型）
```

```
                async:    是否异步（布尔类型）

b. void send(String body)
        用于发送请求

        参数：
                body:  要发送的数据（字符串类型）

c. void setRequestHeader(String header,String value)
        用于设置请求头

        参数：
                header:  请求头的 key（字符串类型）
                vlaue:    请求头的 value（字符串类型）

d. String getAllResponseHeaders()
        获取所有响应头

        返回值：
                响应头数据（字符串类型）

e. String getResponseHeader(String header)
        获取响应头中指定 header 的值

        参数：
                header:  响应头的 key（字符串类型）

        返回值：
                响应头中指定的 header 对应的值

f. void abort()

        终止请求
```

XmlHttpRequest 对象的主要属性：

```
a. Number readyState
      状态值（整数）

      详细：
            0-未初始化，尚未调用 open()方法；
            1-启动，调用了 open()方法，未调用 send()方法；
            2-发送，已经调用了 send()方法，未接收到响应；
            3-接收，已经接收到部分响应数据；
            4-完成，已经接收到全部响应数据；

b. Function onreadystatechange
      当 readyState 的值改变时自动触发执行其对应的函数（回调函数）

c. String responseText
      服务器返回的数据（字符串类型）

d. XmlDocument responseXML
      服务器返回的数据（Xml 对象）

e. Number states
      状态码（整数），如：200、404...

f. String statesText
      状态文本（字符串），如：OK、NotFound...
```

2、跨浏览器支持

- XmlHttpRequest
- IE7+, Firefox, Chrome, Opera, etc.
- ActiveXObject("Microsoft.XMLHTTP")
- IE6, IE5

```
基于原生 AJAX - - emo
```

IE6,IE5

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <h1>XMLHttpRequest - Ajax 请求</h1>
    <input type="button" onclick="XmlGetRequest();" value="Get 发送请求" />
    <input type="button" onclick="XmlPostRequest();" value="Post 发送请求" />
    <script src="/statics/jquery-1.12.4.js"></script>
    <script type="text/javascript">
        function GetXHR(){
            var xhr = null;
            if(XMLHttpRequest){
                xhr = new XMLHttpRequest();
            }else{
                xhr = new ActiveXObject("Microsoft.XMLHTTP");
            }
            return xhr;
    function XhrPostRequest(){
            var xhr = GetXHR();
            // 定义回调函数
            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4){
                    // 已经接收到全部响应数据，执行以下操作
                    var data = xhr.responseText;
                    console.log(data);
                }
            };
            // 指定连接方式和地址----文件方式
            xhr.open('POST', "/test/", true);
            // 设置请求头
```

```
            xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded; charset-UTF-8');
            // 发送请求
            xhr.send('n1=1;n2=2;');
        }
        function XhrGetRequest(){
            var xhr = GetXHR();
            // 定义回调函数
            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4){
                    // 已经接收到全部响应数据，执行以下操作
                    var data = xhr.responseText;
                    console.log(data);
                }
            };
            // 指定连接方式和地址----文件方式
            xhr.open('get', "/test/", true);
            // 发送请求
            xhr.send();
        }
    </script>
</body>
</html>
```

jQuery Ajax

jQuery 其实就是一个 JavaScript 的类库，其将复杂的功能做了上层封装，使得开发者可以在其基础上写更少的代码实现更多的功能。

- jQuery 不是生产者，而是大自然搬运工。
- jQuery Ajax 本质 XMLHttpRequest 或 ActiveXObject

注：2.+版本不再支持 IE9 以下的浏览器

jQuery Ajax 方法列表

jQuery.get(...)

所有参数：

url: 待载入页面的 URL 地址
data: 待发送 Key/value 参数。
success: 载入成功时回调函数。
dataType: 返回内容格式，xml, json, script, text, html

jQuery.post(...)
所有参数：
url: 待载入页面的 URL 地址
data: 待发送 Key/value 参数
success: 载入成功时回调函数
dataType: 返回内容格式，xml, json, script, text, html

jQuery.getJSON(...)
所有参数：
url: 待载入页面的 URL 地址
data: 待发送 Key/value 参数。
success: 载入成功时回调函数。

jQuery.getScript(...)
所有参数：
url: 待载入页面的 URL 地址
data: 待发送 Key/value 参数。
success: 载入成功时回调函数。

jQuery.ajax(...)

部分参数：

url：请求地址
type：请求方式，GET、POST（1.9.0 之后用 method）
headers：请求头
data：要发送的数据
contentType：即将发送信息至服务器的内容编码类型(默认: "application/x-www-form-urlencoded; charset=UTF-8")
async：是否异步
timeout：设置请求超时时间（毫秒）

beforeSend：发送请求前执行的函数(全局)
complete：完成之后执行的回调函数(全局)
success：成功之后执行的回调函数(全局)
error：失败之后执行的回调函数(全局)

accepts：通过请求头发送给服务器，告诉服务器当前客户端课接受的数据类型
dataType：将服务器端返回的数据转换成指定类型
"xml": 将服务器端返回的内容转换成 xml 格式
"text": 将服务器端返回的内容转换成普通文本格式
"html": 将服务器端返回的内容转换成普通文本格式，在插入 DOM 中时，如果包含 JavaScript 标签，则会尝试去执行。
"script": 尝试将返回值当作 JavaScript 去执行，然后再将服务器端返回的内容转换成普通文本格式
"json": 将服务器端返回的内容转换成相应的 JavaScript 对象
"jsonp": JSONP 格式
使用 JSONP 形式调用函数时，如 "myurl?callback=?" jQuery 将自动替换 ？ 为正确的函数名，以执行回调函数

如果不指定，jQuery 将自动根据 HTTP 包 MIME 信息返回相应类型(an XML MIME type will yield XML, in 1.4 JSON will yield a JavaScript object, in 1.4 script will execute the script, and anything else will be returned as a string

converters： 转换器，将服务器端的内容根据指定的 dataType 转换类型，并传值给 success 回调函数

```
$.ajax({
    accepts: {
```

```
            mycustomtype: 'application/x-some-custom-type'
        },

        // Expect a `mycustomtype` back from server
        dataType: 'mycustomtype'

        // Instructions for how to deserialize a `mycustomtype`
        converters: {
            'text mycustomtype': function(result) {
                // Do Stuff
                return newresult;
            }
        },
    });
```

基于 jQueryAjax - -emo

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

    <p>
        <input type="button" onclick="XmlSendRequest();" value='Ajax 请求' />
    </p>



    <script type="text/javascript" src="jquery-1.12.4.js"></script>
    <script>
```

```
        function JqSendRequest(){
            $.ajax({
                url: "http://c2.com:8000/test/",
                type: 'GET',
                dataType: 'text',
                success: function(data, statusText, xmlHttpRequest){
                    console.log(data);
                }
            })
        }


    </script>
</body>
</html>
```

跨域 AJAX

由于浏览器存在同源策略机制，同源策略阻止从一个源加载的文档或脚本获取或设置另一个源加载的文档的属性。

特别的：由于同源策略是浏览器的限制，所以请求的发送和响应是可以进行，只不过浏览器不接受罢了。

浏览器同源策略并不是对所有的请求均制约：

- 制约： XmlHttpRequest
- 不叼： img、iframe、script 等具有 src 属性的标签

跨域，跨域名访问，如：http://www.c1.com 域名向 http://www.c2.com 域名发送请求。

1、JSONP 实现跨域请求

JSONP（JSONP - JSON with Padding 是 JSON 的一种"使用模式"），利用 script 标签的 src 属性（浏览器允许 script 标签跨域）

基于 JSONP 实现跨域 Ajax - - emo

```
<!DOCTYPE html>
```

```
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title></title>
</head>
<body>

    <p>
        <input type="button" onclick="Jsonp1();"    value='提交'/>
    </p>

    <p>
        <input type="button" onclick="Jsonp2();" value='提交'/>
    </p>

    <script type="text/javascript" src="jquery-1.12.4.js"></script>
    <script>
        function Jsonp1(){
            var tag = document.createElement('script');
            tag.src = "http://c2.com:8000/test/";
            document.head.appendChild(tag);
            document.head.removeChild(tag);

        }

        function Jsonp2(){
            $.ajax({
                url: "http://c2.com:8000/test/",
                type: 'GET',
                dataType: 'JSONP',
                success: function(data, statusText, xmlHttpRequest){
```

```
                    console.log(data);
                }
            })
        }



    </script>
</body>
</html>
```

2、CORS

随着技术的发展，现在的浏览器可以支持主动设置从而允许跨域请求，即：跨域资源共享（CORS，Cross-Origin Resource Sharing），其本质是设置响应头，使得浏览器允许跨域请求。

* 简单请求 OR 非简单请求

```
条件：
        1、请求方式：HEAD、GET、POST
        2、请求头信息：
                Accept
                Accept-Language
                Content-Language
                Last-Event-ID
                Content-Type  对应的值是以下三个中的任意一个
                                                application/x-www-form-
urlencoded
                                                multipart/form-data
                                                text/plain

注意：同时满足以上两个条件时，则是简单请求，否则为复杂请求
```

* 简单请求和非简单请求的区别?

```
        简单请求：一次请求
```

	非简单请求：两次请求，在发送数据之前会先发一次请求用于做“预检”，只有“预检”通过后才再发送一次请求用于数据传输。

* 关于"预检"

```
- 请求方式：OPTIONS
- “预检”其实做检查，检查如果通过则允许传输数据，检查不通过则不再发送真正想要发送的消息
- 如何“预检”
        => 如果复杂请求是 PUT 等请求，则服务端需要设置允许某请求，否则“预检”不通过
            Access-Control-Request-Method
        => 如果复杂请求设置了请求头，则服务端需要设置允许某请求头，否则“预检”不通过
            Access-Control-Request-Headers
```

基于 cors 实现 AJAX 请求:

a、支持跨域，简单请求

服务器设置响应头：Access-Control-Allow-Origin = '域名' 或 '*'

HTML

服务器设置响应头：Access-Control-Allow-Origin = '域名' 或 '*'

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

    <p>
        <input type="submit" onclick="XmlSendRequest();" />
    </p>
```

```
<p>
	<input type="submit" onclick="JqSendRequest();" />
</p>

<script type="text/javascript" src="jquery-1.12.4.js"></script>
<script>
	function XmlSendRequest(){
		var xhr = new XMLHttpRequest();
		xhr.onreadystatechange = function(){
			if(xhr.readyState == 4) {
				var result = xhr.responseText;
				console.log(result);
			}
		};
		xhr.open('GET', "http://c2.com:8000/test/", true);
		xhr.send();
	}

	function JqSendRequest(){
		$.ajax({
			url: "http://c2.com:8000/test/",
			type: 'GET',
			dataType: 'text',
			success: function(data, statusText, xmlHttpRequest){
				console.log(data);
			}
		})
	}
```

```
    </script>
</body>
</html>
```

Torando

```
class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
        self.write('{"status": true, "data": "seven"}')
```

b、支持跨域，复杂请求

由于复杂请求时，首先会发送“预检”请求，如果“预检”成功，则发送真实数据。

- “预检”请求时，允许请求方式则需服务器设置响应头：Access-Control-Request-Method
- “预检”请求时，允许请求头则需服务器设置响应头：Access-Control-Request-Headers
- “预检”缓存时间，服务器设置响应头：Access-Control-Max-Age

HTML

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

    <p>
        <input type="submit" onclick="XmlSendRequest();" />
    </p>

    <p>
        <input type="submit" onclick="JqSendRequest();" />
```

```
</p>

<script type="text/javascript" src="jquery-1.12.4.js"></script>
<script>
    function XmlSendRequest(){
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function(){
            if(xhr.readyState == 4) {
                var result = xhr.responseText;
                console.log(result);
            }
        };
        xhr.open('PUT', "http://c2.com:8000/test/", true);
        xhr.setRequestHeader('k1', 'v1');
        xhr.send();
    }

    function JqSendRequest(){
        $.ajax({
            url: "http://c2.com:8000/test/",
            type: 'PUT',
            dataType: 'text',
            headers: {'k1': 'v1'},
            success: function(data, statusText, xmlHttpRequest){
                console.log(data);
            }
        })
    }


</script>
```

```
</body>
</html>
```

Tornado

```
class MainHandler(tornado.web.RequestHandler):

    def put(self):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
        self.write('{"status": true, "data": "seven"}')

    def options(self, *args, **kwargs):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
        self.set_header('Access-Control-Allow-Headers', "k1,k2")
        self.set_header('Access-Control-Allow-Methods', "PUT,DELETE")
        self.set_header('Access-Control-Max-Age', 10)
```

c、跨域获取响应头

默认获取到的所有响应头只有基本信息，如果想要获取自定义的响应头，则需要再服务器端设置 Access-Control-Expose-Headers。

HTML

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

    <p>
```

```
        <input type="submit" onclick="XmlSendRequest();" />
    </p>

    <p>
        <input type="submit" onclick="JqSendRequest();" />
    </p>

    <script type="text/javascript" src="jquery-1.12.4.js"></script>
    <script>
        function XmlSendRequest(){
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4) {
                    var result = xhr.responseText;
                    console.log(result);
                    //  获取响应头
                    console.log(xhr.getAllResponseHeaders());
                }
            };
            xhr.open('PUT', "http://c2.com:8000/test/", true);
            xhr.setRequestHeader('k1', 'v1');
            xhr.send();
        }

        function JqSendRequest(){
            $.ajax({
                url: "http://c2.com:8000/test/",
                type: 'PUT',
                dataType: 'text',
                headers: {'k1': 'v1'},
                success: function(data, statusText, xmlHttpRequest){
```

```
                        console.log(data);
                        // 获取响应头
                        console.log(xmlHttpRequest.getAllResponseHeaders());
                    }
                })
            }
        </script>
</body>
</html>
```

Tornado

```
class MainHandler(tornado.web.RequestHandler):

    def put(self):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")

        self.set_header('xxoo', "seven")
        self.set_header('bili', "daobidao")

        self.set_header('Access-Control-Expose-Headers', "xxoo,bili")



        self.write('{"status": true, "data": "seven"}')

    def options(self, *args, **kwargs):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
        self.set_header('Access-Control-Allow-Headers', "k1,k2")
        self.set_header('Access-Control-Allow-Methods', "PUT,DELETE")
        self.set_header('Access-Control-Max-Age', 10)
```

d、跨域传输 cookie

在跨域请求中，默认情况下，HTTP Authentication 信息，Cookie 头以及用户的 SSL 证书无论在预检请求中或是在实际请求都是不会被发送。

如果想要发送：

- 浏览器端：XMLHttpRequest 的 withCredentials 为 true
- 服务器端：Access-Control-Allow-Credentials 为 true
- 注意：服务器端响应的 Access-Control-Allow-Origin 不能是通配符 *

HTML

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

    <p>
        <input type="submit" onclick="XmlSendRequest();" />
    </p>

    <p>
        <input type="submit" onclick="JqSendRequest();" />
    </p>

    <script type="text/javascript" src="jquery-1.12.4.js"></script>
    <script>
        function XmlSendRequest(){
            var xhr = new XMLHttpRequest();
            xhr.onreadystatechange = function(){
                if(xhr.readyState == 4) {
                    var result = xhr.responseText;
                    console.log(result);
                }
            };
```

```
            xhr.withCredentials = true;

            xhr.open('PUT', "http://c2.com:8000/test/", true);
            xhr.setRequestHeader('k1', 'v1');
            xhr.send();
        }

        function JqSendRequest(){
            $.ajax({
                url: "http://c2.com:8000/test/",
                type: 'PUT',
                dataType: 'text',
                headers: {'k1': 'v1'},
                xhrFields:{withCredentials: true},
                success: function(data, statusText, xmlHttpRequest){
                    console.log(data);
                }
            })
        }
    </script>
</body>
</html>
```

Tornado

```
class MainHandler(tornado.web.RequestHandler):

    def put(self):
        self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
        self.set_header('Access-Control-Allow-Credentials', "true")

        self.set_header('xxoo', "seven")
```

```
            self.set_header('bili', "daobidao")
            self.set_header('Access-Control-Expose-Headers', "xxoo,bili")

            self.set_cookie('kkkkk', 'vvvvv');

            self.write('{"status": true, "data": "seven"}')

    def options(self, *args, **kwargs):
            self.set_header('Access-Control-Allow-Origin', "http://www.xxx.com")
            self.set_header('Access-Control-Allow-Headers', "k1,k2")
            self.set_header('Access-Control-Allow-Methods', "PUT,DELETE")
            self.set_header('Access-Control-Max-Age', 10)
```

7.如何在前端实现轮询？

轮询：通过定时器让程序每隔 n 秒执行一次操作。

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
</head>
<body>
    <h1>请选出最帅的男人</h1>
    <ul>
        {% for k,v in gg.items() %}
            <li>ID:{{ k }}, 姓名：{{ v.name }} , 票数：{{ v.count }}</li>
        {% endfor %}
    </ul>
    <script>
        setInterval(function () {
            location.reload();
```

```
        },2000)
    </script>
</body>
</html>
```

8.如何在前端实现长轮询？

客户端向服务器发送请求，服务器接到请求后 hang 住连接，等待 30 秒，直到有新消息，才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
</head>
<body>
    <h1>请选出最帅的男人</h1>
    <ul>
        {% for k,v in gg.items() %}
            <li style="cursor: pointer" id="user_{{ k }}" ondblclick="vote({{ k }});">ID:{{ k }}, 姓名：{{ v.name }} ，票数：<spa
n>{{ v.count }}</span></li>
        {% endfor %}
    </ul>

    <script src="/static/jquery-3.3.1.min.js"></script>
    <script>
        $(function () {
            get_new_count();
        });

        function get_new_count() {
            $.ajax({
                url: '/get_new_count',
                type:'GET',
```

```
                dataType:'JSON',
                success:function (arg) {
                    if (arg.status){
                        // 更新票数
                        var gid = "#user_" + arg.data.gid;
                        $(gid).find('span').text(arg.data.count);
                    }else{
                        // 10s 内没有人投票
                    }
                    get_new_count();
                }
            })
        }
        function vote(gid) {
            $.ajax({
                url: '/vote',
                type:'POST',
                data:{gid:gid},
                dataType:"JSON",
                success:function (arg) {
                }
            })
        }
    </script>
</body>
</html>
```

008、AJAX 是什么，如何使用 AJAX？

ajax(异步的 javascript 和 xml) 能够刷新局部网页数据而不是重新加载整个网页。
第一步，创建 xmlhttprequest 对象，var xmlhttp =new XMLHttpRequest ();XMLHttpRequest 对象用来和服务器交换数据。
第二步，使用 xmlhttprequest 对象的 open（）和 send（）方法发送资源请求给服务器。
第三步，使用 xmlhttprequest 对象的 responseText 或 responseXML 属性获得服务器的响应。
第四步，onreadystatechange 函数，当发送请求到服务器，我们想要服务器响应执行一些功能就需要使用 onreadystatechange 函数，每次 xmlhttprequest 对象的 readyState 发生改变都会触发 onreadystatechange 函数。

009、vuex 的作用？

多组件之间共享：vuex
补充 luffyvue
1：router-link / router-view
2：双向绑定，用户绑定 v-model
3：循环展示课程：v-for
4：路由系统，添加动态参数
5：cookie 操作：vue-cookies
6：多组件之间共享：vuex
7：发送 ajax 请求：axios（js 模块）

010、vue 中的路由的拦截器的作用？

vue-resource 的 interceptors 拦截器的作用正是解决此需求的妙方。
在每次 http 的请求响应之后，如果设置了拦截器如下，会优先执行拦截器函数，获取响应体，然后才会决定是否把 response 返回给 then 进行接收

011、axios 的作用？

发送 ajax 请求：axios（js 模块）

12.列举 vue 的常见指令

1、v-show 指令：条件渲染指令，无论返回的布尔值是 true 还是 false，元素都会存在在 html 中，只是 false 的元素会隐藏在 html 中，并不会删除.
2、v-if 指令：判断指令，根据表达式值得真假来插入或删除相应的值。
3、v-else 指令：配合 v-if 或 v-else 使用。
4、v-for 指令：循环指令，相当于遍历。
5、v-bind：给 DOM 绑定元素属性。
6、v-on 指令：监听 DOM 事件。

13.简述 jsonp 及实现原理？

JSONP 是 json 用来跨域的一个东西。原理是通过 script 标签的跨域特性来绕过同源策略。
JSONP 的简单实现：创建一个回调函数，然后在远程服务上调用这个函数并且将 JSON 数据作为参数传递，完成回调。

14.什么是 cors ？

CORS：跨域资源共享（CORS，Cross-Origin Resource Sharing），随着技术的发展，现在的浏览器可以支持主动设置从而允许跨域请求，其本质是设置响应头，使得浏览器允许跨域请求。
浏览器将 CORS 请求分成两类：简单请求和复杂请求
简单请求(同时满足以下两大条件)
（1）请求方法是以下三种方法之一:
HEAD
GET
POST
（2）HTTP 的头信息不超出以下几种字段：

Accept
Accept-Language
Content-Language
Last-Event-ID
Content-Type：只限于三个值 application/x-www-form-urlencoded、multipart/form-data、text/plain 凡是不同时满足上面两个条件，就属于非简单请求

15.列举 Http 请求中常见的请求方式？

GET、POST、PUT、DELETE
PATCH（修改数据）
HEAD（类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头）

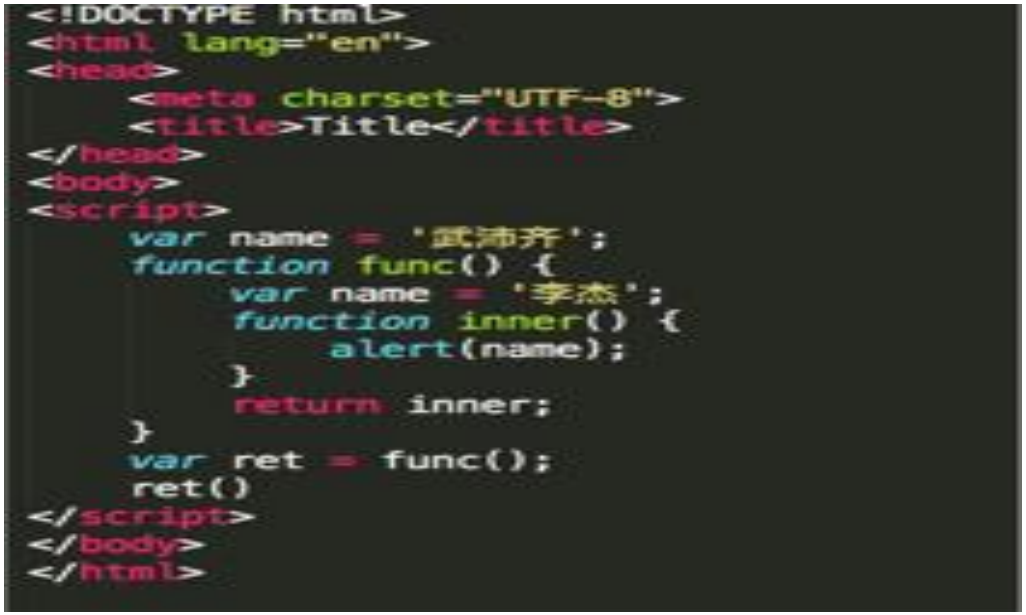
16.列举 Http 请求中的状态码？

分类：
1** 信息，服务器收到请求，需要请求者继续执行操作
2** 成功，操作被成功接收并处理
3** 重定向，需要进一步的操作以完成请求
4** 客户端错误，请求包含语法错误或无法完成请求
5** 服务器错误，服务器在处理请求的过程中发生了错误
常见的状态码
200 - 请求成功
202 - 已接受请求，尚未处理
204 - 请求成功，且不需返回内容
301 - 资源（网页等）被永久转移到其他 url
400 - 请求的语义或是参数有错
403 - 服务器拒绝请求
404 - 请求资源（网页）不存在
500 - 内部服务器错误
502 - 网关错误，一般是服务器压力过大导致连接超时
503 - 由于超载或系统维护，服务器暂时的无法处理客户端的请求

17.列举 Http 请求中常见的请求头？

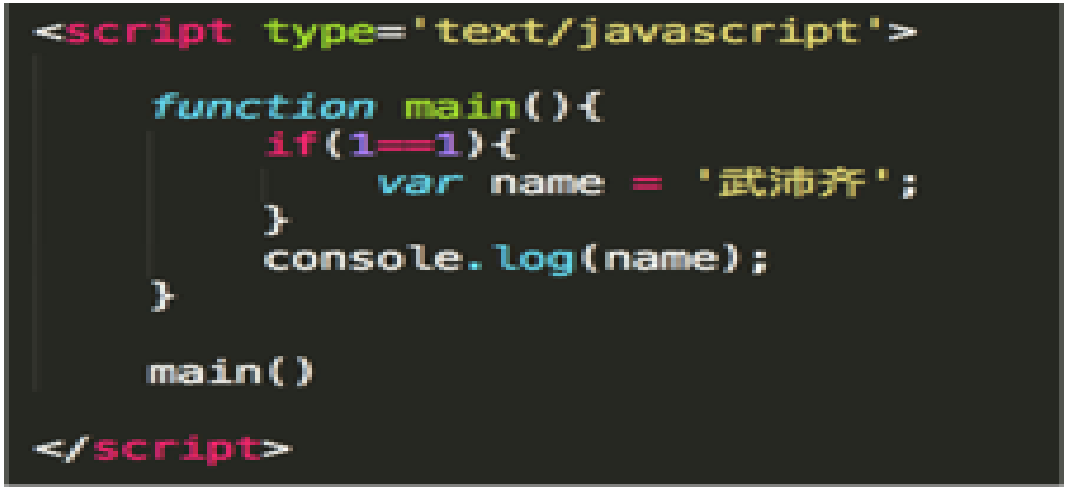
- user-agent （代理）
- host
- referer
- cookie
- content-type

18.看图写结果（js）：



李杰

看图写结果（js）：



武沛奇

看图写结果：（**js**）

```
<script>
    xo = 'Alex';

    function func() {
        var xo = "武沛齐";

        function inner() {
            var xo = '老男孩';
            console.log(xo);
        }

        inner();
    }

    func();
</script>
```

老男孩

看图写结果：（**js**）

```
<script>

    function Foo() {
        console.log(xo);
        var xo = '武沛齐';
    }

    Foo();

</script>
```

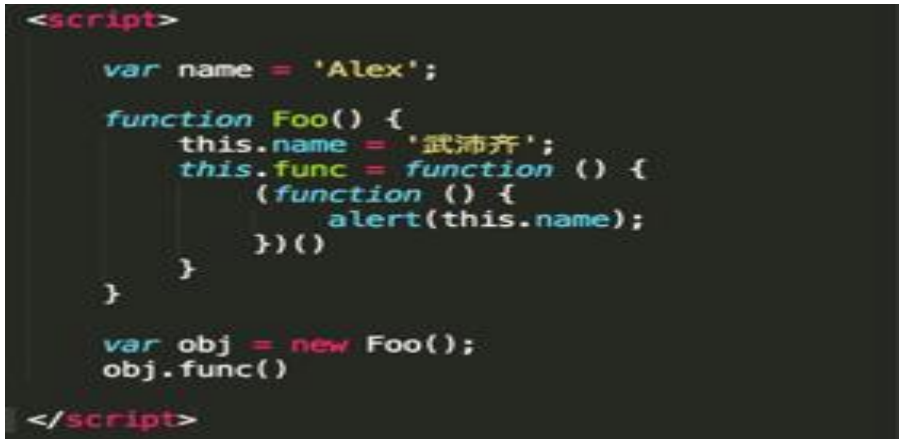
undefined

看图写结果：（**js**）

```
<script>

    var name = 'Alex';

    function Foo() {
        this.name = '武沛齐';
        this.func = function () {
            alert(this.name);
        }
    }

    var obj = new Foo();
    obj.func()

</script>
```

武沛奇

看图写结果：（js）



```
<script>
    var name = 'Alex';

    function Foo() {
        this.name = '武沛齐';
        this.func = function () {
            (function () {
                alert(this.name);
            })()
        }
    }

    var obj = new Foo();
    obj.func()
</script>
```

Alex

vue 前端框架

什么是 vue.js

1.vue 是目前最火的一个前端框架,react 是最流行的前端框架(react 除了开发网站,还可以开发手机 APP,vue 语法也是可以进行手机 app 开发的,需要借助于 weex)

2.vue.js 是前端的主流框架之一,和 angular.js react.js 一起 并成为前端三大主流框架

3.vue.js 是一套构建用户界面的框架,值关注视图层,不仅易于上手,还可便于第三方库或既有项目整合.(vue 有配套的第三方库,可以整合起来做带向项目的开发)

4.前端的主要工作?主要负责 MVC 的 v 这一层 主要工作就是和界面交道

为什么要学习流行框架

企业为了提高开发效率,在企业中,事件就是效率,效率就是金钱

提高开发的发展历程,原生 js -> jquery 类的类库 -> 前端模板引擎 -> angular/vue/

在 vue 中,一个核心的概念,就是让用户不在操作 DOM 操作元素,解放了用户的双手,让程序员可以更多的时间去关注业务逻辑

增强自己就业时候的竞争力

框架和库 的却别:

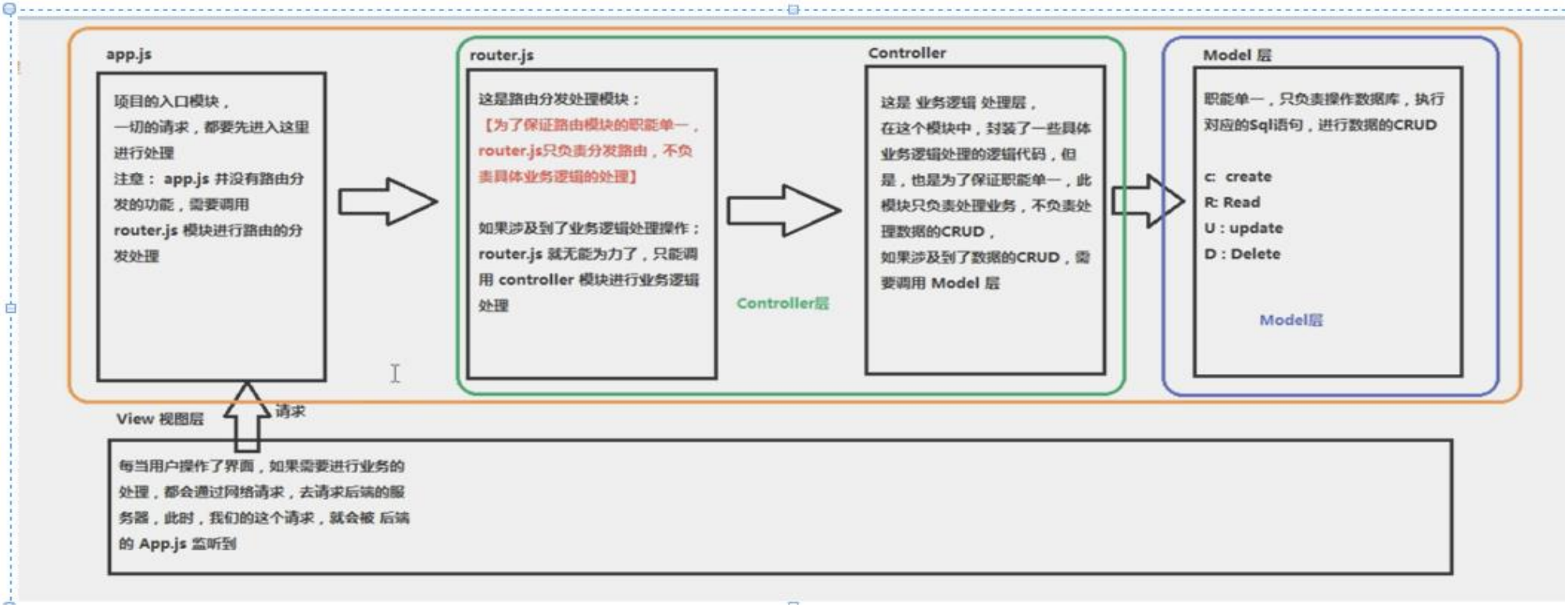
框架:是一台完整的解决方案,对项目的侵入性较大,项目如果需要更换框架,则需要重新架构整个项目

node 中的 express

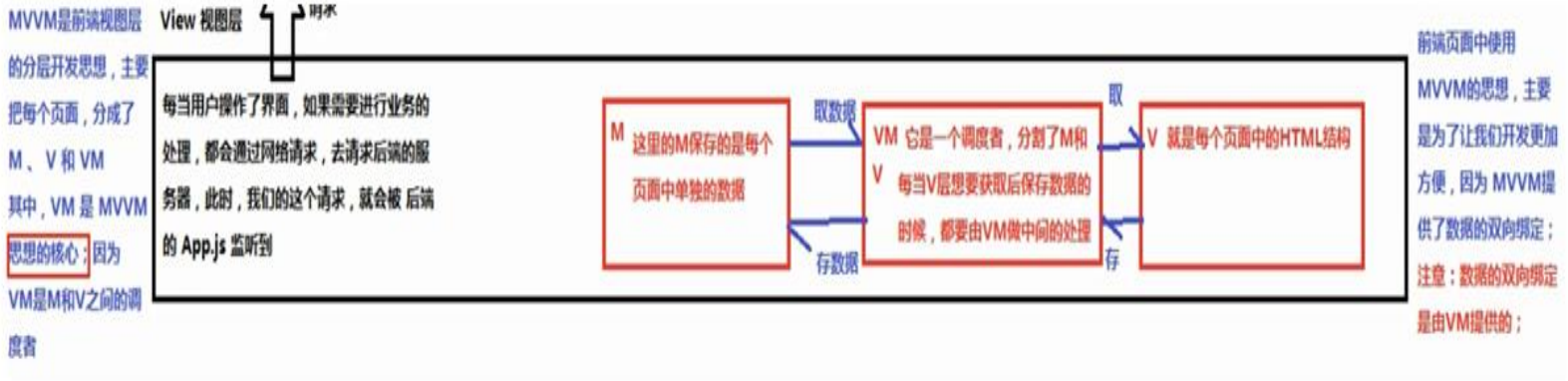
库(插件):提供某一个小功能,对项目的侵入性小,如果某个库完成某些需求,可以很容易切换到其他库的实现要求.

node(后端) 中的 MVC 与 前端 MVVM 之间的区别:

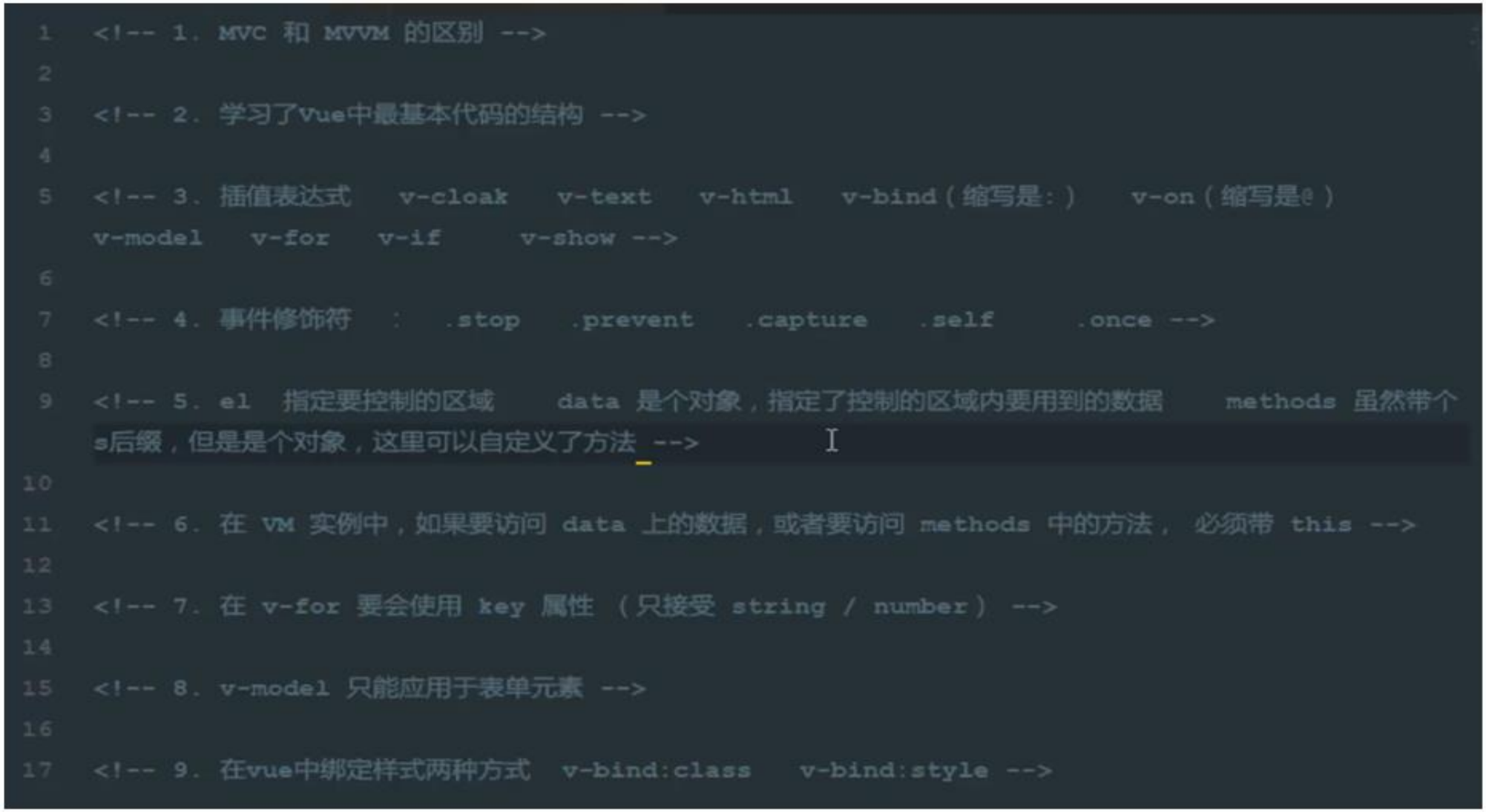
MVC (后端概念)


app.js
项目的入口模块，
一切的请求，都要先进入这里进行处理
注意： app.js 并没有路由分发的功能，需要调用 router.js 模块进行路由的分发处理
router.js
这是路由分发处理模块：
【为了保证路由模块的职能单一，router.js只负责分发路由，不负责具体业务逻辑的处理】
如果涉及到了业务逻辑处理操作；router.js 就无能为力了，只能调用 controller 模块进行业务逻辑处理
Controller层
Controller
这是 业务逻辑 处理层，
在这个模块中，封装了一些具体业务逻辑处理的逻辑代码，但是，也是为了保证职能单一，此模块只负责处理业务，不负责处理数据的CRUD，
如果涉及到了数据的CRUD，需要调用 Model 层
Model 层
职能单一，只负责操作数据库，执行对应的Sql语句，进行数据的CRUD
c: create
R: Read
U : update
D : Delete
Model层
View 视图层
请求
每当用户操作了界面，如果需要进行业务的处理，都会通过网络请求，去请求后端的服务器，此时，我们的这个请求，就会被 后端的 App.js 监听到

MVVM 前端概念:


MVVM是前端视图层的分层开发思想，主要把每个页面，分成了M、V 和 VM
其中，VM 是 MVVM 思想的核心；因为VM是M和V之间的调度者
View 视图层
请求
每当用户操作了界面，如果需要进行业务的处理，都会通过网络请求，去请求后端的服务器，此时，我们的这个请求，就会被 后端的 App.js 监听到
M 这里的M保存的是每个页面中单独的数据
取数据
存数据
VM 它是一个调度者，分割了M和V 每当V层想要获取后保存数据的时候，都要由VM做中间的处理
取
存
V 就是每个页面中的HTML结构
前端页面中使用MVVM的思想，主要是为了让我们开发更加方便，因为 MVVM提供了数据的双向绑定；
注意：数据的双向绑定是由VM提供的；

小结:



vue 第一话: 实例化 vue

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>vue firstone</title>
```

```
        <script src="vue.js"></script>
    </head>
    <body>
        <div id="app">
            <p>{{ msg }}</p>
        </div>


        <script>
            var vm = new Vue({
                el:    '#app',
                data:{
                    msg:'hello world!',
                },
            })
        </script>
    </body>
</html>
```

需要引入 vue.js

vue 第二话: v-cloak v-text v-html v-on v-bind

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>
        <style>
            [v-cloak]{
                display: none;
            }
        </style>
    </head>
    <body>
        <div id="app">
```

```
        <!-- 文件显示以及解决闪烁问题 cloak/text/html 的应用方式 -->
        <p v-cloak>00000{{ msg }}0000</p>
        <h3 v-text="msg">00000000</h3>
        <p v-text="msg"></p>
        <p v-html="msg"></p>
        <!-- ------------------------------- -->
        <!-- 事件绑定 v-bind 绑定 js 表达式-->
        <p>
            <input type="button" value="按钮" v-bind:title="msg2" />
            <input type="button" value="按钮" v-bind:title="msg2 + '添加字符'" />
            <input type="button" value="按钮" :title="msg2 + '简写方式!'" />
        </p>
        <!-- 事件绑定 v-on 绑定 methods 事件-->
        <p>
            <input type="button" value="v-on 绑定 methods" v-on:click="show" />
            <input type="button" value="v-on 绑定 methods 简写方式" @click="show" />
            <input type="button" value="v-on 绑定 methods 其他事件" v-on:mouseover="show" />
        </p>
    </div>


    <script>
        var vm = new Vue({
            el:'#app',
            data:{
                msg:'<h1>hello</h1>',
                msg2:'(v-bind 属性!)this button!'
            },
            methods:{
                show:function(){
                    alert("OK")
                },
            }
        })
```

```
        </script>
    </body>
</html>
```

vue 第三话: 跑马灯练习

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>
    </head>
    <body>
        <div id="app">
            <input type="button" value="启动" @click="start" />
            <input type="button"  value="停止"@click="stop" />
            <input type="text" :value="info" />

            <h1>{{ info }}</h1>
        </div>

        <script>
            var vm = new Vue({
                el:'#app',
                data:{
                    info:'hello world !',
                    Intervalid:null,

                },
                methods:{
                    start(){
//                      var _this = this
//                      setInterval(function(){
//                          // console.log(this.info)
//                          var start0 = _this.info.substring(0,1)
```

```
//                              var end0 = _this.info.substring(1)
//                              _this.info = end0 + start0
//                          },400)

                        // => 说明函数体外部的 this 参数 传递至函数体内部!
                        if(this.Intervalid != null) return;
                        this.Intervalid = setInterval( () => {
                            // console.log(this.info)
                            var start0 = this.info.substring(0,1)
                            var end0 = this.info.substring(1)
                            this.info = end0 + start0
                        },400)

                    },
                    stop(){
                        clearInterval(this.Intervalid);
                        this.Intervalid = null;
                    },
                }
            })

        </script>
    </body>
</html>
```

vue 第四话 v-on 事件修饰符

.stop 阻止冒泡

.prevent 阻止默认时间

.capture 添加时间侦听器使用事件模式

.self 只当前事件在该元素本身(比如不是子元素)触发时触发回调

.once 事件只触发一次

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
    <title></title>
    <script></script>
    <script src="vue.js"></script>
    <style>
        .inner{
            height: 200px;
            width: 200px;
            background-color: aqua;
        }
        .inner2{
            height: 200px;
            width: 200px;
            background-color: firebrick;
        }
        .inner3{
            height: 200px;
            width: 200px;
            background-color:greenyellow;
        }
        .inner4{
            height: 200px;
            width: 200px;
            background-color:gray;
        }
    </style>
</head>
<body>
    <div id="app">
        <!-- 此时点击按钮,同时也会触发 div 的点击事件. -->
        <div class="inner" @click="div0">
            <input type="button" value="点击" @click="inp0"/>
            <!-- .stop -->
            <input type="button" value="点击(.stop)" @click.stop="inp0"/>
            <!-- .prevent 阻止默认行为 -->
```

```
            <p><a href="http://www.baidu.com" @click.prevent="bdclick">百度链接</a></p>
        </div>

        <!-- .capture 捕获机制-->
        <div class="inner2" @click.capture="div0">
            <input type="button" value="点击" @click="inp0"/>
        </div>
        <!-- .self 点击当前元素时触发-->
        <div class="inner3" @click.self="div0">
            <input type="button" value="点击" @click.self="inp0"/>
        </div>
        <!-- .once 只触发一次 -->
        <div class="inner4" @click.once="div0">
            <input type="button" value="点击" @click.once="inp0"/>
        </div>

    <!-- 方法之间可以相互叠加调用! -->
    <!--
    .self .stop 区别:
    .self 只阻止 自己元素上的冒泡,并不阻止自己外层的元素冒泡
    .stop 阻止除自己以外的其他元素冒泡
    -->
    </div>

    <script>
        var ve = new Vue({
            el:'#app',
            data:{
            },
            methods:{
                div0(){
                    console.log('this div0!')
                },
                inp0(){
```

```
                    console.log("this inp0!")
                },
                bdclick(){
                    console.log('baidu 链接事件!')
                }
            },
        })
    </script>

</body>
</html>
```

vue 第五话 双向数据绑定 v-model

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script></script>
        <script src="vue.js"></script>
    </head>
    <body>
        <div id="app">
            <!--
            使用 v-model 可以是先,表单元素和 model 中数据的双向书绑定
            但是注意!
            v-model 只能运用在 表单元素中
            -->
            <input type="text" v-model="msg" />
            <h3>{{ msg }}</h3>
        </div>

        <script>
            var ve = new Vue({
                el:'#app',
```

```
                data:{
                    msg:'hello world ! '
                },
                methods:{

                },
            })

        </script>

    </body>
</html>
```

vue 第 6 话 简易计算器练习

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script></script>
        <script src="vue.js"></script>
    </head>
    <body>
        <div id="app">
            <input type="text"  v-model="num1" />
            <select v-model="opt">
                <option value ="+">+</option>
                <option value ="-">-</option>
                <option value ="*">*</option>
                <option value ="/">/</option>
            </select>
            <input type="text"  v-model="num2" />
            <input type="button" value="=" @click='jisuan' />
            <input type="text"  v-model="result" />
        </div>
```

```
        <script>
            var ve = new Vue({
                el:'#app',
                data:{
                    num1:0,
                    opt:'+',
                    num2:0,
                    result:0,
                },
                methods:{
                    jisuan(){
//                      switch(this.opt){
//                          case '+':
//                              this.result = parseInt(this.num1) + parseInt(this.num2)
//                          break;
//                          case '-':
//                              this.result = parseInt(this.num1) - parseInt(this.num2)
//                          break;
//                          case '*':
//                              this.result = parseInt(this.num1) * parseInt(this.num2)
//                          break;
//                          case '/':
//                              this.result = parseInt(this.num1) / parseInt(this.num2)
//                          break;
//                      }

                        this.result = eval(this.num1 + this.opt + this.num2)


                    },
                },
            })
```

```
        </script>

    </body>
</html>
```

vue 第七话 样式选择

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>
        <style>
            .st1{
                    width: 200px;
                    height: 200px;
            }
            .st2{
                background-color: #ADFF2F;
            }
            .active{
                display: none;
            }

        </style>

    </head>
    <body>
        <div id="app">
            <input type="button" value="显示" @click='a'/>
            <input type="button" value="隐藏" @click='b' />

            <p><h1 :class="['st1','st2']">aaaaaaaaaaa !!!</h1></p>

            <p><h1 :class="['st1','st2',dis?'active':'']">bbbbbbbbbbb !!!</h1></p>
```

```
            <p><h1 :class="['st1','st2',{'active':dis}]">ccccccccccc !!!</h1></p>

            <p><h1 :class="{st1:true,st2:true,active:dis}">ddddddddddd !!!</h1></p>

            <p><h1 :class="sobj">EEEEEEEEEEEEE !!!</h1></p>

        </div>

        <script>
            var vm = new Vue({
                el: '#app',
                data: {
                    dis:false,
                    sobj:{st1:true,st2:true,active:false},
                },
                methods: {
                    a(){
                        this.dis = false
                    },
                    b(){
                        this.dis = true
                    },
                }
            })
        </script>
    </body>
</html>
```

vue 第八话 内联样式

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

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

    </head>
    <body>
        <div id="app">
            <h1 :style="{color:'red','font-weight':200}">asdadsada</h1>
            <h1 :style="sty1">asdadsada</h1>
            <h1 :style="[sty1 , sty2]">asdadsada</h1>


        </div>

        <script>
            var vm = new Vue({
                el: '#app',
                data: {
                    sty1:{color:'green','font-weight':100},
                    sty2:{'font-style':'italic'},
                },
                methods: {
                }
            })
        </script>
    </body>
</html>
```

vue 第九话 v-for 循环

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>

    </head>
```

```
<body>
    <div id="app">
        <p>{{ list[0] }}</p>
        <p>{{ list[1] }}</p>
        <p>{{ list }}</p>

        <!-- 循环列表 -->
        <p v-for='i in list'>{{ i }}</p>

        <!-- 循环列表,以及列表索引 -->
        <p v-for='i,index in list'>{{ i }} - {{ index }}</p>

        <!-- 循环列表对象 -->
        <p v-for='i,j in list2'>{{ i.id }} - {{ i.name }} - {{ j }}</p>

        <!-- 循环对象 -->
        <p v-for='i,j,l in user'>{{ i }}- {{ j }} - {{ l }}</p>

        <!-- 循环数据 -->
        <p v-for='count in 15'>{{ count }}</p>

    </div>
    <script>
        var vm = new Vue({
            el: '#app',
            data: {
                list:[1,2,3,4,5,6,7,8,9],
                list2:[
                    {id:1,name:'a1'},
                    {id:2,name:'a2'},
                    {id:3,name:'a3'},
                    {id:4,name:'a4'},
                    {id:5,name:'a5'},
                    {id:6,name:'a6'},
```

```
                    ],
                    user:{
                        id:1,
                        name:'alex',
                        gender:'男',
                    },
                },
                methods: {
                }
            })
        </script>
    </body>
</html>
```

vue 第十话 v-for key 值 数据绑定

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>

    </head>
    <body>
        <div id="app">
            <p>
                <label>id </label><input type="text" v-model="id"/>
                <label>name</label> <input type="text" v-model="name"/>
                <input type="button" value="add" @click="add" />
            </p>

            <label>序号</label>    -    <label>姓名</label>
            <p v-for='i in info' :key='i.id'><input type="checkbox" />{{ i.id }} ------- {{ i.name }}</p>

            <!--
```

```
            注意:!!!
            v-for 循环的时候 key 属性只能使用 num 或 str 数据类型绑定
            key 在使用的时候,必须要使用 v-bind 属性绑定的形式,指定 key 的值
            在组件中 使用 v-for 循环的时候 或者在一些特殊的情况中
            如果 v-for 有问题 必须在使用 v-for 的是同时指定唯一的字符串/数字 类型:key 值
            -->

        </div>

        <script>
            var vm = new Vue({
                el: '#app',
                data: {
                    id:'',
                    name:'',
                    info:[
                        {id:1,name:'a1'},
                        {id:2,name:'a2'},
                        {id:3,name:'a3'},
                        {id:4,name:'a4'},
                        {id:5,name:'a5'},
                        {id:6,name:'a6'},
                    ],
                },
                methods: {
                    add(){
                        // this.info.push({id:this.id,name:this.name})
                        this.info.unshift({id:this.id,name:this.name})
                    },
                }
            })
        </script>
    </body>
</html>
```

vue 第十一话 v-if 以及 v-show

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="vue.js"></script>

    </head>
    <body>
        <div id="app">
            <input type="button" value="切换" @click="qiehuan"/>
            <h3 v-if="flag">v-if 语句</h3>
            <h3 v-show="flag">v-show 语句</h3>
        </div>

        <!--
        v-if 特点:每次都会重新删除或创建元素
        有较高的切换性能消耗
        如果元素设计到频繁的切换,最好不要使用 v-if 而推荐使用 v-show
        v-show 特点:只是切换了元素的 display:none 的样式
        有较高的初始渲染消耗
        如果元素可能永远也不会被显示出来被用户看到,则推荐使用
        v-if -->
        <script>
            var vm = new Vue({
                el: '#app',
                data: {
                    flag:true,
                },
                methods: {
                    qiehuan(){
                        this.flag = !this.flag
                    }
```

```
                }
            })
        </script>
    </body>
</html>
```

vue 前端框架 (二) 表格增加搜索

本章知识点 归纳:

1.定义全局过滤器 以及 私有过滤器

2.定义全局指令 以及 定义私有指令

3.定义键盘修饰符

4.v-for 的函数引入

5.字符串的 incluede 方法,.toString().padStart(2,'0') 的字符串补全方法

6.通过过滤器定义日期格式

需要 vue.js 以及 bootstrap

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <script src="js/vue.js"></script>
    </head>
    <body>
        <!--
        过滤器
        vue 允许你自定义过滤器,可被用作一些常见文本格式化,过滤可以用在两个地方,
        mustachc 差值 , v-bind 表达式,过滤器应该添加在 javascript 表达式的尾部,由管道符指示

        过滤器分为 全局过滤器以及私有过滤器
        全局过滤拉起是定义在 script 中
        而私有过滤器是定义在 vue 实例当中的 filters 方法中的

        定义过滤器 有两个条件[过滤器名称和处理函数]
```

```
过滤器调用的时候,采用的就近原则,如果私有过滤器和全局过滤器名称一致了,
这时候优先调用私有过滤器

-->

<div id="app">
    <div class="panel panel-primary">
        <div class="panel-heading">
            <h3 class="pannel-title">添加品牌</h3>
        </div>
        <div class="panel-body form-inline">
            <label>
                ID:
                <input type="text" class="form-control" v-model="id">
            </label>
            <label>
                Name:
                <!-- <input type="text" class="form-control" v-model="name" @keyup.enter="add"> -->

                <!-- keyup.113 113 对应 F2 的键盘码 -->
                <!-- <input type="text" class="form-control" v-model="name" @keyup.113="add"> -->

                <input type="text" class="form-control" v-model="name" @keyup.f2="add">

            </label>
            <input type="button" value="添加" class="btn btn-primary" @click="add">
            <label>
                Search:
                <input type="text" class="form-control" v-model="searchname" v-focus v-color='"blue"'>
            </label>
        </div>
    </div>
    <br>
    <table class="table table-bordered table-hover table-striped">
```

```
            <tr>
                <th><input type="checkbox"></th>
                <th>ID</th>
                <th>Name</th>
                <th>Ctime</th>
                <th>action</th>
            </tr>
            <tr v-for='i in search(searchname)' :key='i.id'>
                <td><input type="checkbox"></td>
                <td >{{i.id}}</td>
                <td v-color='"blue"'>{{i.name}}</td>
                <td >{{i.ctime | dateformat }}</td>
                <td ><a href="#" @click.prevent="del(i.id)">删除</a></td>
            </tr>
        </table>
        <br>

    </div>
    <div id='app2'>
        {{msg|infodata}}
    </div>

    <script>
        // 全局过滤器 定义以及使用
        Vue.filter('dateformat',function(datestr,pattern=''){
            var dt = new Date(datestr)
            var y = dt.getFullYear()
            var m = dt.getMonth()+1
            var d = dt.getDate().toString().padStart(2,'0')
            // return y+'-'+ m +'-'+ d

            if(pattern.toLowerCase() === 'yyyy-mm-dd'){
                return `${y}-${m}-${d}`
            }else{
```

```
                var hh = dt.getHours().toString().padStart(2,'0')
                var mm = dt.getMinutes().toString().padStart(2,'0')
                var ss = dt.getSeconds().toString().padStart(2,'0')
                return `${y}-${m}-${d} ${hh}:${mm}:${ss}`
            }
        });

        // 1.X 版本的 自定义全局键盘码(键盘修饰符)
        Vue.config.keyCodes.f2 = 113;
        // 使用 Vue.directive()定义全局的指令 v-focus
        Vue.directive('focus',{
            // 和行为有关的操作交给 inserted 操作
            inserted(el){
                el.focus()
            },
        })

        Vue.directive('color',{
            //跟样式有关的可以交给 bind 操作
            bind(el,binding){
                el.style.color = binding.value
            }
        })
//          使用 directive() 定义全局的指令
//          其中参数:
//          1.指令名称,注意在定义的时候,指令的名称前面,都不要加 v-前缀,但是调用的时候必须在指令名称前面加上 v- 前缀进行调用
//          2.对象,这个对象身上,有一些指令的函数,这些函数可以在特定的阶段,执行相关的操作
//          对象中,5 个钩子函数:
//          bind,inserted,updated,componentUpdated,unbind

        // 钩子函数参数:
        // el,binding( name,value,oldvalue,expression,arg modifiers ),vnode,oldvnode

        // 实例一
```

```
            var vm =  new Vue({
                el:'#app',
                data:{
                    id:'',
                    name:'',
                    searchname:'',
                    list:[
                        {id:1,name:'alex',ctime:new Date()},
                        {id:2,name:'anex',ctime:new Date()},
                        {id:3,name:'aexk',ctime:new Date()},
                    ],
                    msg:'abc',
                },
                methods:{
                    add(){
                        this.list.push({id:this.id,name:this.name,ctime:new Date})
                        this.name = this.id = ''
                    },
                    del(id){

//                          方式一(循环 list 比对值)
//                          this.list.some((item,i)=>{
//                              if(item.id == id){
//                                  this.list.splice(i,1)
//                                  return true;
//
// //                                  .splice(a,b,c)
// //                                  参数 a 为下标值,参数 b 为步长,参数 c 为替换的值
//                              }
//                          })

                        // 方式二(查找对应的值得方法)
                        var index = this.list.findIndex(i =>{
                            if(i.id == id){
```

```
                            return true
                        }
                    })
                    this.list.splice(index,1)
                },
                search(searchname){

//                      // 方式一
//                      var newlist = []
//                      this.list.forEach(k => {
//                          if(k.name.indexOf(searchname) != -1){
//                              newlist.push(k)
//                          }
//                      })
//                      return newlist

                    // 方式二
                    return this.list.filter(k => {
                        if(k.name.includes(searchname)){
                            return k
                        }
                    })

//                  注意:forEach some filter findIndex 这些都属于数组的新方法,
//                  都会对数组中的每一项 进行遍历 执行相关的操作
//
//                  ES6 中 为了字符串提供了一个新方法,叫做 String.prototype.includes(包含字符串)
//                  包含 则返回 true
//                  不包含 则返回 false

                },
            },
            directives:{
                'color0':{
```

```
                    bind(el,bd){
                        el.style.color = db.value
                    }
                },
            },

        })

        // 实例二
        var vu = new Vue({
            el:'#app2',
            data:{
                msg:'ahello !',
            },
            methods:{

            },

            // 私有过滤器(局部过滤器)
            filters:{
                infodata(msg){
                    return msg.replace('a','b')
                },
            }
        })

    </script>
</body>
</html>
```

vue 前端框架 (三)

VUE 生命周期

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

```
    <meta charset="utf-8">
    <title></title>
    <script type="text/javascript" src="js/vue.js"></script>
    <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css"/>
</head>
<body>
    <div id="app"></div>




    <script>
        var vm = Vue({
            el:'#app',
            data:{},
            methods:{},
            filters:{},
            directives:{},

            // 什么是生命周期:VUE 实例创建/运行/销毁 ,总是伴随着各种各样的事件,
            // 这些事件,统称为生命周期

            // 生命周期钩子:就是生命周期事件的别名,

            // 主要的生命周期函数分为:

            // 创建期间的生命周期函数:
            // vue 第一个生命周期函数
            beforeCreate(){}
            // 实例刚在内存中被创建出来,此时,还没有初始化 data 和 methods 属性

            // vue 第二个生命周期函数
            created(){}
            // 实例已经在内存中创建,此时 data 和 mothods 已经创建好了,此时还没有开始编译模板
```

```
                // vue 第三个生命周期函数
                beforeMount() {}
                // 此时已经完成了模板的编译,但是还没有挂载到页面中

                // vue 第四个生命周期函数
                mounted(){}
                // 此时,已经编译好了模板,挂载到页面指定容器中显示了

                // 运行期间的生命周期函数:
                // vue 第五个生命周期函数
                beforeUpdate(){}
                // 状态更新之前执行此函数,此时 data 中的状态值是最新的,但是界面上显示的数据还是旧的,因为此时还没有开始重新渲染 DOM
节点

                // vue 第六个生命周期函数
                updated(){}
                // 实例更新完毕之后调用此函数,此时 data 中的状态值和界面上显示的数据,都已经完成了更新,界面已经被重新渲染好了.

                // 销毁期间的生命周期函数:
                // vue 第七个生命周期函数
                beforeDestroy(){}
                // 实例销毁之前调用,这一步,实例乃然可用

                // vue 第八个生命周期函数
                destroyed(){}
                // Vue 实例销毁后调用,调用后,vue 实例指示的所有东西都会解绑定,所有的事件监听器都会被移除,所有的子实例也会被销
毁.__constructor__()


            })
        </script>
    </body>
</html>
```

vue-resource 获取后端数据

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script type="text/javascript" src="js/vue.js"></script>
        <script src="js/vue-resource.js"></script>
        <link rel="stylesheet" type="text/css" href="css/bootstrap.min.css"/>
    </head>
    <body>
        <div id="app">
            <input type="button" value="get 点击" @click="getinfo" />
            <input type="button" value="post 点击" @click="postinfo" />
            <input type="button" value="jsonp 点击" @click="jsonpinfo" />
        </div>

        <script>
            var vm = new Vue({
                el:'#app',
                data:{},
                methods:{
                    getinfo(){
                        this.$http.get('http://127.0.0.1:5000/infoapi').then(function(result){
                            console.log(result.body.msg)
                        })
                    },
                    postinfo(){
                        var data = {'zhuangtai':'OK'}
                        this.$http.post('http://127.0.0.1:5000/infoapi',{data},{emulateJSON:true}).then(result =>{
                            console.log(result.body.msg)
                            console.log(typeof(result))
                        })
                    },
```

```
                    jsonpinfo(){
                        this.$http.jsonp('http://127.0.0.1:5000/infoapi').then(result => {
                            console.log(result.body)
                        })
                    },
                },
            })
        </script>
    </body>
</html>
```

获取后端数据 练习

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-resource.js"></script>
        <link rel="stylesheet" href="css/bootstrap.min.css">
    </head>
    <body>
        <div id='app'>
            <div class="panel panel-primary">
                <div class="panel-heading">
                    <h3 class="pannel-title">添加品牌</h3>
                </div>
                <div class="panel-body form-inline">
                    <label>
                        Name:
                        <input type="text" class="form-control" v-model="name" @keyup.enter="add">

                    </label>
                    <input type="button" value="添加" class="btn btn-primary" @click="add">
                </div>
```

```
        </div>
        <br>
        <table class="table table-bordered table-hover table-striped" >
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>操作</th>
            </tr>
            <tr v-for='i in info' :key='i.id'>
                <td>{{ i.id }}</td>
                <td>{{ i.name }}</td>
                <td><a href="#" @click.prevent='delinfo(i.id)'>删除</a></td>
            </tr>
        </table>

    </div>
    <script>
        Vue.http.options.emulateJSON = true;
        Vue.http.options.root = 'http://172.16.3.104:5000/';

        var vm = new Vue({
            el:'#app',
            data:{
                name:'',
                searchname:'',
                info:[],
            },
            methods:{
                getinfo(){
                    this.$http.get('infoapi').then( data => {
                        this.info = data.body
                    })
                },
                add(){
```

```
                    this.$http.post('addinfoapi',{name:this.name},{}).then( data => {
                        if(data.body.status == 'OK'){
                            this.name = ''
                            this.getinfo()
                        }else{
                            alert('添加失败')
                        }
                    })
                },
                delinfo(id){
                    this.$http.post('delinfoapi',{id:id},{}).then( data => {
                        if(data.body.status == 'OK'){
                            this.getinfo()
                        }else{
                            alert('删除失败')
                        }
                    })
                },
            },
            created(){
                this.getinfo()
            },
        });
    </script>
</body>
</html>
```

VUE 动画效果

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-resource.js"></script>
```

```
        <link rel="stylesheet" href="css/bootstrap.min.css">

        <style>
            .v-enter,.v-leave-to{
                opacity: 0;
                transform: translateX(150px);
            }
            .v-enter-active,.v-leave-active{
                transition: all 2s ease
            }
            .my-enter,.my-leave-to{
                opacity: 0;
                transform: translateX(300px) translateY(150px);
            }
            .my-enter-active,.my-leave-active{
                transition: all 1s ease
            }



        </style>
    </head>
    <body>
        <div id='app'>
            <input type="button" @click="istrue=!istrue" value="点击">
            <transition>
                <p v-show="istrue">啊啊啊啊啊啊啊啊啊啊啊</p>
            </transition>
            <br>
            <input type="button" @click="istrue2 =!istrue2" value="点击">
            <transition name='my'>
                <p v-show="istrue2">啊啊啊啊啊啊啊啊啊啊啊</p>
            </transition>
<!--          点击按钮,显示 p 标签 再次点击隐藏皮标签
            1.transition 元素将 p 标签包裹
```

```
            2.通过修改 transition 所提供的样式修改
            .v-enter,.v-enter-to  入场前,入场后
            .v-leave,.v-leave-to  离场前,离场后
            .v-enter-active,     入场动作
            .v-leave-active,     离场动作 -->


        </div>

        <script>
            Vue.http.options.emulateJSON = true;
            Vue.http.options.root = 'http://172.16.3.104:5000/';
            var vm = new Vue({
                el:'#app',
                data:{
                    istrue:false,
                    istrue2:false,
                },
                methods:{},
            });
        </script>
    </body>
</html>
```

第三方 css 动画效果

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-resource.js"></script>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <link rel="stylesheet" href="css/animate.css">
    </head>
```

```
    <body>
        <div id='app'>
            <input type="button" @click="istrue=!istrue" value="点击">
            <transition
            enter-active-class='animated bounceIn'
            leave-active-class='animated bounceOut'
            :duration="{ enter:200, leave:500 }">

                <p v-show="istrue">我来了啦!!</p>
            </transition>
        </div>

        <script>

            var vm = new Vue({
                el:'#app',
                data:{
                    istrue:false,
                },
                methods:{},
            });
        </script>
    </body>
</html>
```

VUE 动画钩子函数

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-resource.js"></script>
        <link rel="stylesheet" href="css/bootstrap.min.css">
        <style>
```

```
        .v-enter,.v-leave-to{
            opacity: 0;
            transform: translateY(400px);
        }

        .v-enter-active,
        .v-leave-active{
            transition: all 0.8s ease;
        }

        /* 这里必须两个属性连用要不不起效果 */
        .v-move{
            transition: all 1s ease;
        }
        .v-leave-active{
            position: absolute;
        }
    </style>
</head>
<body>
    <div id='app'>
        <div class="panel panel-primary">
            <div class="panel-heading">
                <h3 class="pannel-title">添加品牌</h3>
            </div>
            <div class="panel-body form-inline">
                <label>
                    Name:
                    <input type="text" class="form-control" v-model="name" @keyup.enter="add">

                </label>
                <input type="button" value="添加" class="btn btn-primary" @click="add">
            </div>
        </div>
```

```
        <br>
        <table class="table table-bordered table-hover table-striped" >
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Name</th>
                    <th>操作</th>
                </tr>
            </thead>
            <tbody is="transition-group">
                <tr v-for='i in info' :key='i.id'>
                    <td>{{ i.id }}</td>
                    <td>{{ i.name }}</td>
                    <td><a href="#" @click.prevent='delinfo(i.id)'>删除</a></td>
                </tr>
            </tbody>
        </table>

    </div>
    <script>
        Vue.http.options.emulateJSON = true;
        Vue.http.options.root = 'http://172.16.3.104:5000/';

        var vm = new Vue({
            el:'#app',
            data:{
                name:'',
                searchname:'',
                info:[],
            },
            methods:{
                getinfo(){
                    this.$http.get('infoapi').then( data => {
                        this.info = data.body
```

```
                    })
                },
                add(){
                    this.$http.post('addinfoapi',{name:this.name},{}).then( data => {
                        if(data.body.status == 'OK'){
                            this.name = ''
                            this.getinfo()
                        }else{
                            alert('添加失败')
                        }
                    })
                },
                delinfo(id){
                    this.$http.post('delinfoapi',{id:id},{}).then( data => {
                        if(data.body.status == 'OK'){
                            this.getinfo()
                        }else{
                            alert('删除失败')
                        }
                    })
                },
            },
            created(){
                this.getinfo()
            },
        });
    </script>
</body>
</html>
```

VUE 全局组件

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
        <title>定义 vue 组件</title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <!--
        什么是组件:
        组件的组件的出现就是为了拆分 vue 实例的代码量,能够让我们以不同的组件来划分不同的功能模块,将我们需要什么样的功能,就可以去调用对应的组件即可.

        组件和模块的区别:
        组件化:
        是从代码逻辑的角度进行划分,方便 diamante 分层开发,保证每个功能模块的职能单一;
        组件化:
        是从 UI 界面的角度进行划分的,前端组件化,方便组件 UI 的重用
        -->
        <div id="app">
                <!-- 第一种方式 -->
                <!-- <mycom1></mycom1> -->
                <!-- <my-com2></my-com2> -->

                <!-- 第二种方式 -->
                <mycom1></mycom1>

        </div>

        <template  id="muban">
            <div>
                <h1>你好 中国!</h1>
                <p>这是在 app 实例外定义的一个模板 id=muban</p>
            </div>
        </template>

        <script>
//              // 第一种创建全局组件方式
```

```
//              //使用 vue.extend 来创建全局的 vue 组件
//              var com1 = Vue.extend({
//                  template:"<h1>hello world1!!</h1>"
//              })
//
//              //使用 vue.component 定义全局组件的时候,
//              //组件名称使用了 驼峰命名,则需要把大写的驼峰改为小写的字母,
//              //同时两个单词之前 使用'-'链接
//              Vue.component('mycom1',com1)
//              Vue.component('myCom2',com1)
//

//              // 上面的简写方式
//              Vue.component('mycom1',Vue.extend({
//                  template:"<h1>hello world1!!</h1>"
//              }))

//              //上面的再简写方式
//              Vue.component('mycom1',{
//                  template:"<h1>hello world1!!</h1>"
//              })

            // 注意 无论是哪种方式 template 属性指向的模板内容,必须有且只能有唯一的一个根元素.

            // 第二种创建全局模板的方式
            // 在#app 实例外创建一个 template 元素模板,然后引入 app 实例内部
            Vue.component('mycom1',{
                template:'#muban'
            })

            // 需要创建 vue 实例,得到 viewmodel 才能渲染组件
            var vm = new Vue({
                el:'#app',
            })
```

```
        </script>
    </body>
</html>
```

Vue 私有组件

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>定义 vue 组件</title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <div id="app">
                <muban></muban>
                <muban2></muban2>
        </div>

        <template id='muban2'>
            <div >
                <h1>这是私有组件!</h1>
            </div>
        </template>
        <script>
            var vm = new Vue({
                el:'#app',
                data:{},
                methods:{},
                filters:{},
                directives:{},

                components:{
                    muban:{
                        template:'<div><h1>你好 世界!</h1></div>'
```

```
                    },
                    muban2:{
                        template:'#muban2'
                    },
                },

                beforeCreate(){},
                created(){},
                beforeMount(){},
                mounted(){},
                beforeUpdate(){},
                updated(){},
                beforeDestroy(){},
                destroyed(){},

            })

        </script>
    </body>
</html>
```

VUe 组件应用 data 属性

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>定义 vue 组件</title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <div id="app">
                <muban></muban>
                <muban></muban>
                <muban2></muban2>
```

```
        </div>

        <template id='muban2'>
            <div >
                <h1>这是私有组件!</h1>
            </div>
        </template>

        <script>
            var vm = new Vue({
                el:'#app',
                data:{},
                methods:{},
                filters:{},
                directives:{},

                components:{
                    muban:{
//                      1.组件可以有自己 data 数据
//                      2.组件的 data 和实例 data 有些不一样,实例中的 data 可以为一个对象,但是组件中的 data 必须是一个方法
//                      3.组件中 data 除了必须为一个方法之外,这个方法内部,还必须返回一个对象才行;
//                      4.组件中的 data 数据,使用方式和实例中的 data 使用方式完全一样.

                        template:'<div><input type="button" value="+1" @click="add" ><h1>{{count}}</h1></div>',
                        data(){
                            return{count:0}
                        },
                        methods:{
                            add(){
                                this.count ++
                            }
                        },
```

```
                    },
                    muban2:{
                        template:'#muban2'
                    },
                },

                beforeCreate(){},
                created(){},
                beforeMount(){},
                mounted(){},
                beforeUpdate(){},
                updated(){},
                beforeDestroy(){},
                destroyed(){},

            })

        </script>
    </body>
</html>
```

Vue 组件切换

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <div id="app">
            <!-- 第一种组件切换的方式 -->
            <a href="#" @click.prevent="flag=true">登录</a>
            <a href="#" @click.prevent="flag=false">注册</a>
```

```
            <login v-if='flag'></login>
            <register v-else='flag'></register>

            <!-- 第二种组件切换的方式 -->
            <a href="#" @click.prevent="flag2='login'">登录</a>
            <a href="#" @click.prevent="flag2='register'">注册</a>

            <component :is='flag2'></component>

<!--      vue 提供的标签 回顾!
        component,
        template,
        transition,
        transition-group -->

        </div>

        <script>
            Vue.component('login',{
                template:'<h1>登录组件</h1>'
            })
            Vue.component('register',{
                template:'<h1>注册组件</h1>'
            })

            var vm = new Vue({
                el:'#app',
                data:{
                    flag:true,
                    flag2:'login',
                }


            })
```

```
        </script>
    </body>
</html>
```

VUE 组件切换+ 动画效果

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <style type="text/css">
            .v-enter,.v-leave-to{
                opacity: 0;
                transform: translateX(100px);
            }

            .v-enter-active,.v-leave-active{
                transition: all 1s ease;
            }
        </style>
    </head>
    <body>
        <div id="app">
            <!-- 第一种组件切换的方式 -->
            <a href="#" @click.prevent="flag=true">登录</a>
            <a href="#" @click.prevent="flag=false">注册</a>
            <login v-if='flag'></login>
            <register v-else='flag'></register>

            <!-- 第二种组件切换的方式 -->
            <a href="#" @click.prevent="flag2='login'">登录</a>
            <a href="#" @click.prevent="flag2='register'">注册</a>
```

```
            <!-- 通过 mode 属性 设置组件切换时候的模式 -->
            <transition mode='out-in'>
                <component :is='flag2'></component>
            </transition>

<!--     vue 提供的标签 回顾!
        component,
        template,
        transition,
        transition-group -->



        </div>



        <script>
            Vue.component('login',{
                template:'<h1>登录组件</h1>'
            })
            Vue.component('register',{
                template:'<h1>注册组件</h1>'
            })

            var vm = new Vue({
                el:'#app',
                data:{
                    flag:true,
                    flag2:'login',
                }


            })
```

```
        </script>
    </body>
</html>
```

VUe 动画小球

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <style type="text/css">
            #ball{
                width: 15px;
                height: 15px;
                background-color: #269ABC;
                border-radius: 50%;
            }

        </style>
    </head>
    <body>
        <div id="app">
            <input type="button" value="按钮" @click="flag=!flag" />

            <transition
                @before-enter = "benter"
                @enter = 'enter'
                @after-enter='fenter'
            >
                <div id="ball" v-show="flag"></div>
            </transition>

        </div>
```

```
        <script>
            var vm = new Vue({
                el:'#app',
                data:{
                    flag:false,
                },
                methods:{
                    benter(el){
                        el.style.transform = 'translate(0,0)'
                    },
                    enter(el,done){
                        el.offsetWidth
                        el.style.transform = 'translate(150px,450px)'
                        el.style.transition='all 1s ease'
                        done()
                    },
                    fenter(el){
                        this.flag = !this.flag
                    },
                }
            })

        </script>
    </body>
</html>
```

VUE 组件向子组件传值

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
    </head>
    <body>
```

```
        <div id="app">
            <com1 :zzz='msg'></com1>
        </div>
        <script>
//              子组件中要访问父类数据,
//              1.首先父类数据要绑定到模板上(传递数据给模板)
//              2.子组件 要引入父类数据
//              3.子组件 应用 引入的父类自定义的数据名

            var vm = new Vue({
                el:'#app',
                data:{
                    msg:'这是父类 数据!'
                },
                methods:{},
                components:{
                    com1:{
                        template:'<h1>{{info}}这是私有组件!-{{zzz}}</h1>',
                        // 组件中 data 数据 对于组件是可以读写
                        data(){
                            return {
                                info:'--这是组件数据--',
                            }
                        },
                        // 组件中定义的父类数据 是只读的
                        props:['zzz'],
                    }
                }
            })

        </script>
    </body>
</html>
```

VUE 父组件方法传递给子组件/共享子组件属性

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <div id="app">
            <com1 @zzz='show'></com1>
        </div>
        <script>
            var vm = new Vue({
                el:'#app',
                data:{
                    msg:'这是父类 数据!',
                    sonmsg:null,
                },
                methods:{
                    // **传递参数
//                    show(d1){
//                        console.log('this 父组件 ! OK!' + d1)
//                    },
                    show(d1){
                        console.log('this 父组件 ! OK!')
                        this.sonmsg = d1
                    },


                },
                components:{
                    com1:{
                        template:'<h1>这是私有组件!<input type="button" value="按钮" @click="fshow"></h1>',
                        data(){
```

```
                        return {
                            sonmsg:{name:'sonname'},
                        }
                    },
                    methods:{
                        fshow(){
                            // **传递参数,第二的位置传递参数
                            // this.$emit('zzz','123')

                            // 将子组件的 data 传递至方法然后父组件可以从方法获取子组件数据
                            this.$emit('zzz',this.sonmsg)

                        }
                    }
                }
            }
        })

    </script>
</body>
</html>
```

VUE 评论练习

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
    </head>
    <body>
        <div id="app">
            <muban0 @ftj='tj' ></muban0>
            <div>
                <table>
```

```
            <tr v-for='i in list'>
                <td>{{i.pl}}</td>
                <td>{{i.name}}</td>
            </tr>
        </table>
    </div>


</div>


<template id="muban0">
    <div>
        <input type="text" placeholder="评论人" v-model="name"/>
        <input type="text" placeholder="评论" v-model="pl"/>
        <input type="button" value="发表评论" @click="fbpl" />
    </div>
</template>
<script>
    var vm = new Vue({
        el:'#app',
        data:{
            list:[
                {name:'a',pl:'aaaa'},
            ]
        },
        created(){
            this.tj()
        },
        methods:{
            tj(){
                var list = JSON.parse(localStorage.getItem('cmts') || '[]')
                this.list = list
            },
```

```
            },
            components:{
                muban0:{
                    data(){
                        return {
                            name:'',
                            pl:'',
                        }
                    },
                    template:'#muban0',
                    methods:{
                        fbpl(){
                            var aaa = {name:this.name,pl:this.pl}
                            var list = JSON.parse(localStorage.getItem('cmts') || '[]')
                            list.unshift(aaa)
                            localStorage.setItem('cmts',JSON.stringify(list))
                            this.name = this.pl = ''

                            this.$emit('ftj')
                        }
                    },
                }
            }
        })

    </script>
</body>
</html>
```

vue $ref 绑定 DOM 元素以及元素组件

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>ref</title>
```

```
    <script src="js/vue.js"></script>
</head>
<body>
    <div id="app">
            <muban ref='r1'></muban>

            <p ref='r2'>这是 p 标签</p>

            <br>
            <input type="button" value="r1 点击获取组件数据以及处罚组件方法" @click="r1click"/>
            <input type="button" value="r2 点击获取 r2DOM 对象" @click="r2click" />

    </div>

    <script>
        var vm = new Vue({
            el:'#app',
            methods:{
                r1click(){
                    console.log(this.$refs.r1.msg)
                    this.$refs.r1.show()
                },
                r2click(){
                    console.log(this.$refs.r2)
                },
            },
            components:{
                muban:{
                    template:'<div><h1>你好 世界!</h1></div>',
                    data(){
                        return {
                            msg:'私有组件数据!'
                        }
                    },
```

```
                        methods:{
                            show(){
                                console.log("这是私有组件,方法")
                            },
                        }
                    },
                },
            })

        </script>
    </body>
</html>
```

VUE 前端路由

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-router.js"></script>
        <link href="css/bootstrap.min.css"/>
        <style type="text/css">

            /* router-link 默认的 class 类名 */
            /* 可以在 router 实例化中 linkActiveCLASS 属性自定义 class 名名 */
            .router-link-active,.mylinkactive{
                color: orangered;
                font-weight: 500;
                font-size: 80px;
                font-style: italic;
            }

            .v-enter,v-leave-to{
                opacity: 0;
```

```
            transform: translateX(140px);
        }

        .v-enter-active,v-leave-active{
            transition:all 1s ease ;
        }

    </style>
</head>
<body>
    <!-- 前端路由概念:-->
    <!-- 对于单个应用程序来说,主要通过 URL 中的 hash 来实现不同页面之间的切换,同时,hash 有一个特点:
    http 请求中不会包含 hash 相关的内容,若依,单页面程序中的页面跳转主要用 hash 实现.
    在单页面应用程序中,这种通过 hash 改变来切换页面的方式,称为前端路由 -->

    <!-- 这一节总结: -->
    <!--
    应用 vue-route 步骤:
    1.创建 Vue 实例
    2.创建 Vuerouter 实例
    3.创建组件
    4.vueroute 实例中,配置 path 路径 将组件变量名注册至 path 路径中
    5.在 vue 实例中,router 属性中 注册 vuerouter 实例
    6.在 vue el 中 router-view 展现组件内容,可选项 router-link 模拟动态切换组件
     -->

    <!--
    URL 中的 hash(#)
    1.# 的含义:#代表网页中的一个位置，其右边的字符，就是该位置的标识符。
    2.HTTP 请求不包含#:#号是用来指导浏览器动作的，对服务器端完全无用。所以，HTTP 请求中不包含#。
    3.#后面的字符:在第一个#后面出现的任何字符，都会被浏览器解读为位置标识符。这意味着，这些字符都不会被发送到服务器端。
    4.改变#不触发网页重载:单单改变#后的内容，浏览器只会滚动到相应位置，不会重新加载网页。浏览器不会重新向服务器请求页面。
    5.改变#会改变浏览器的访问历史:每一次改变#后的部分，都会在浏览器的访问历史中增加一个记录，使用"后退"按钮，就可以回到上一个
位置。对于 ajax 应用程序特别有用，可以用不同的#值，表示不同的访问状态，然后向用户给出可以访问某个状态的链接。
```

值得注意的是，上述规则对 IE 6 和 IE 7 不成立，它们不会因为#的改变而增加历史记录。

6.window.location.hash 读取#值:window.location.hash 这个属性可读可写。读取时，可以用来判断网页状态是否改变；写入时，则会在不重载网页的前提下，创造一条访问历史记录。

7.onhashchange 事件:

这是一个 HTML 5 新增的事件，当#值发生变化时，就会触发这个事件。IE8+、Firefox 3.6+、Chrome 5+、Safari 4.0+支持该事件。

它的使用方法有三种：

1.window.onhashchange = func;

2.<body onhashchange="func();">

3.window.addEventListener("hashchange", func, false);

8.Google 抓取#的机制

默认情况下，Google 的网络蜘蛛忽视 URL 的#部分。

但是，Google 还规定，如果你希望 Ajax 生成的内容被浏览引擎读取，那么 URL 中可以使用"#!"，Google 会自动将其后面的内容转成查询字符串_escaped_fragment_的值。

比如，Google 发现新版 twitter 的 URL 如下：

http://twitter.com/#!/username

就会自动抓取另一个 URL：

http://twitter.com/?_escaped_fragment_=/username

通过这种机制，Google 就可以索引动态的 Ajax 内容。

-->

```
<div id="app">

<!--     不推荐
    <a href="#/login">登录</a>
    <a href="#/register">注册</a>
    -->

    <!-- router-link 默认渲染成 A 标签 -->
    <router-link to='login' tag='span'>登录</router-link>
    <router-link to='register' tag='span'>注册</router-link>


    <!-- vue-router 提供的元素,专门用来 当做占位符,将组件展示到 router-view 中 -->
    <transition mode='out-in'>
```

```
            <router-view></router-view>
        </transition>


    </div>



    <script>
        var login = {
            template:"<h1>登录组件</h1>"
        }
        var register = {
            template:"<h1>注册组件</h1>"
        }
        var routerobj = new VueRouter({
            // route 代表匹配规则
            routes:[
//                      每个路由规则,都是一个对象,这个规则对象,身上有两个必须的属性
//                      属性 1 是 path,表示监听的那个路由链接地址
//                      属性 2 是 component 表示如果路由前面匹配到了 path,则展示 componnent 对应的那个组件
                {path:'/',redirect:'/login'},
                {path:'/login',component:login},
                {path:'/register',component:register},
                // 注意,component 的属性值,逆序是一个组件的模板对象,而不能是引用组件名称
            ],
            linkActiveClass:'mylinkactive',
        })
        var vm = new Vue({
            el:'#app',
            router:routerobj,
            // 将路由规则对象,注册到 vm 实例上,用来监听 URL 地址的变化,然后展示对应的组件
        })
```

```
            </script>
    </body>
</html>
```

vue 前端路由参数传递

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-router.js"></script>
        <link href="css/bootstrap.min.css"/>
        <style type="text/css">
            .router-link-active,.mylinkactive{
                color: orangered;
                font-weight: 500;
                font-size: 80px;
                font-style: italic;
            }
            .v-enter,v-leave-to{
                opacity: 0;
                transform: translateX(140px);
            }
            .v-enter-active,v-leave-active{
                transition:all 1s ease ;
            }
        </style>
    </head>
    <body>
        <div id="app">
            <router-link to='/login' tag='span'>登录</router-link>
            <!-- 第一种传递参数方式: -->
            <router-link to='/login?id=10&name=alex' tag='span'>第一种传递参数方式</router-link>
            <!-- 第二种传递参数方式: 使用过程需要严格匹配路由 path-->
```

```
        <router-link to='/login/12' tag='span'>第二种传递参数方式</router-link>
        <router-link to='/register' tag='span'>注册</router-link>

        <transition mode='out-in'>
            <router-view></router-view>
        </transition>
    </div>
    <script>
        var login = {
            template:"<h1>登录组件-{{$route.query.id}}-{{$route.params.id}}-</h1>",
            created(){

                console.log('这是$route 实例'+this.$route)
                // 第一种传递参数的数据获取:
                // 这是 url?参数的方式 传递数据至$route.query 中,多个参数值也是在 query 中提取.
                console.log('这是第一种方式传参:'+this.$route.query.id)

                // 第二种传递参数的数据获取,需要匹配路由规则!!!
                console.log('这是第二种方式传参:'+this.$route.params.id)
            },
        }
        var register = {
            template:"<h1>注册组件</h1>"
        }
        var routerobj = new VueRouter({
            routes:[
                {path:'/',redirect:'/login'},
                {path:'/login',component:login},
                {path:'/login/:id',component:login},
                {path:'/register',component:register},
            ],
            linkActiveClass:'mylinkactive',
        })
        var vm = new Vue({
```

```
                el:'#app',
                router:routerobj,
            })
        </script>
    </body>
</html>
```

vue 路由嵌套

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-router.js"></script>
        <link href="css/bootstrap.min.css"/>
    </head>
    <body>
        <div id="app">
            <router-link to='/index'>首页</router-link>
            <router-view></router-view>
        </div>

        <template id="shouye">
            <div>
                <h1>首页</h1>
<!--             注意:这子路由节点,加/ 或者 不加 / 直接影响匹配的路由路径,
                当不加的/ 时 访问路径为 /父路径/子路径
                加了/ 时 访问的路径为 /子路径 -->

                <!-- 这种是不加斜线的方式 -->
                <router-link to='/index/login'>登录</router-link>
                <router-link to='/index/register'>注册</router-link>

                <!-- 这种是加斜线的方式 -->
```

```
<!--             <router-link to='/login'>登录</router-link>
                <router-link to='/register'>注册</router-link> -->

                <router-view></router-view>
            </div>

        </template>
        <script>
            var index = {
                template:'#shouye',
            }
            var login = {
                template:"<h1>登录组件</h1>",
            }
            var register = {
                template:"<h1>注册组件</h1>",
            }
            var routerobj = new VueRouter({
                routes:[
                    {
                        path:'/index',
                        component:index,
                        children:[

//                          注意:这子路由节点,加/ 或者 不加 / 直接影响匹配的路由路径,
//                          当不加的/ 时 访问路径为 /父路径/子路径
//                          加了/ 时 访问的路径为 /子路径
                            // 推荐 子路由不加斜线
                            // 归纳总结:一个是相对路径方式,一个是绝对路径的方式

                            {path:'login',component:login},
                            {path:'register',component:register},

//                              {path:'/login',component:login},
```

```
//                              {path:'/register',component:register},
                        ],
                    },
                ],
            })
            var vm = new Vue({
                el:'#app',
                router:routerobj,
            })
        </script>
    </body>
</html>
```

vue 命名视图 实现经典布局

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-router.js"></script>
        <link href="css/bootstrap.min.css"/>
        <style type="text/css">
            body{
                margin: 0 auto;
            }

            .header{
                height: 160px;
                width: 100%;
                background-color: #00FFFF;
            }
            .container{
                display: flex;
                height: 700px;
```

```
            }
            .left{
                background-color: firebrick;
                flex: 2;
            }
            .main{
                background-color: yellowgreen;
                flex: 8;
            }
        </style>
    </head>
    <body>
        <div id="app">
            <router-view name='default'></router-view>
            <div class='container'>
                <router-view name='left'></router-view>
                <router-view name='main'></router-view>
            </div>
        </div>

        <script>
            var indextop = {
                template:'<div class="header">banner 条</div>',
            }
            var indexlelf = {
                template:"<div class='left'><ul><li>首页</li><li>事件 1</li><li>事件 2</li><li>事件 3</li><li>事件
4</li></ul></div>",
            }
            var indexmain = {
                template:"<div class='main'>main 内容区域 <router-view></router-view> </div>",
            }
            var routerobj = new VueRouter({
```

```
//                  注意!!!!
//                  指向单个路由对象使用的是:component,
//                  指向多个路由对象使用的是:components,
//                  注意是否加 s

                routes:[
                    {
                        path:'/',
                        components:{
                            'default':indextop,
                            'left':indexlelf,
                            'main':indexmain,
                        },
                    },
                ],
            })
            var vm = new Vue({
                el:'#app',
                router:routerobj,

            })
        </script>
    </body>
</html>
```

VUe 监控属性

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <script src="js/vue.js"></script>
        <script src="js/vue-router.js"></script>
        <link href="css/bootstrap.min.css"/>
```

```
</head>
<body>
    <!--
    wathch、computer 和 methods 之间的对比:
    1.computed 属性的结果会被缓存,除非依赖的响应属性变化才会重新计算,主要当做属性来使用;
    2.methods 方法表示一个具体的操作,主要书写业务逻辑
    3.watch 一个对象,键是需要观察的表达式,值是对应回调函数,主要用来监听某些特定数据的变化,从而进行某些具体的业务逻辑操作;
    可以看做 computed 和 methods 的结合体. watch 提供了新旧值的记录
     -->
    <div id="app">
        <div>
            <p>第一种实现方式:(keyup + methods 方式)</p>
            <input type="text" v-model="n1" @keyup='getn3' />
            +
            <input type="text" v-model="n2" @keyup='getn3' />
            =
            <input type="text" v-model="n3" />
        </div>

        <div>
            <p>第二种实现方式:(watch 方式)</p>
            <input type="text" v-model="nn1" />
            +
            <input type="text" v-model="nn2" />
            =
            <input type="text" v-model="nn3"/>
        </div>

        <div>
            <!-- 在 computed 中,可以定一些属性,这些属性叫做 计算属性.
            本质上是一个方法,但是在使用这些计算属性的时候 是把它的名称直接当做属性来使用
            并不会把计算属性,当做方法去调用 -->
```

```
            <p>第三种实现方式:(computed 方式)</p>
            <p>注意这里 data 属性中没有定义 nnn3 属性,nnn3 属性是 computed 提供的,computed 需要返回值</p>
            <input type="text" v-model="nnn1" />
            +
            <input type="text" v-model="nnn2" />
            =
            <input type="text" v-model="nnn3"/>
        </div>

    </div>

    <script>
        var vm = new Vue({
            el:'#app',
            data:{
                n1:'',
                n2:'',
                n3:'',

                nn1:'',
                nn2:'',
                nn3:'',

                nnn1:'',
                nnn2:'',

            },
            methods:{
                getn3(){
                    this.n3 = this.n1 + '-' + this.n2
                },
            },
            watch:{
                nn1(){
```

```
                    this.nn3 = this.nn1 + '-' + this.nn2
                },
                nn2(){
                    this.nn3 = this.nn1 + '-' + this.nn2
                },
                // watch 提供了新旧值的记录
                nn3(newvalue,oldvlue){
                    console.log(newvalue,'----',oldvlue)
                },
            },
            computed:{
//                  'nnn3':function(){
//                      return this.nnn1 + '-' + this.nnn2
//                  },

                nnn3(){
                    return this.nnn1 + '-' + this.nnn2
                },
            },

        })
    </script>
</body>
</html>
```

VUE render 函数 模板渲染

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
    <script src="js/vue.js"></script>
```

```
</head>
<body>
    <div id="app">
    </div>
    <script>
        var login = {
            template:'<h1>登录组件</h1>'
        };
        var vm = new Vue({
            el:'#app',
            data:{},
            methods: {},
            components:{},
            render(h) {
                return h(login)
            },
        });
    </script>
</body>
</html>
```

Vue 到底是怎样个框架？

一、Vue 介绍

过去的十年，我们的网页变得更加动态化和强大，是因为有 JAVAScript，我们已经把很多传统服务端代码放到浏览器中这样就产生了成千上万行的 JavaScript 代码，他们链接了各种各样的 HTML 和 CSS 文件，但缺乏正规的组织形式，这也是为什么越来越多的开发者使用 JavaScript 框架，例如：angular、REACT 和 VUE 这样的框架。

这里我们就说一下 vue，vue 是一款有好的、多用途且高性能的 JavaScript 框架，它能够帮助你创建可维护性和课测试性更强的代码库，vue 是渐进式的 JavaScript 框架，也就是说如果你已经有一个现成的服务端应用你可以将 vue 作为该应用的一部分嵌入其中，带来更加丰富的交互体验或者如果你希望将更多的业务逻辑放到前端来实现那么 vue 的核心库机器生态系统也可以满足你的各式需求。

与其他框架相同，vue 允许你讲一个网页分割成可复用的组件，每个组件都包含属于自己的 HTML、CSS、JavaScript 以用来渲染网页中相应的地方。

二、使用 VUE

1、引入 vue.js 文件

```
//下载后导入
<script src=vue.js></script>
```

2、通过下面的代码展示用，创建一个 Vue 的实例，然后通过应用的 id 嵌入根元素，将数据放入一个对象 data 中，并且将表达式传入 div 中{{msg}}（一定注意必须是双大括号）

```
//展示的 HTML
<div id="app">
  {{ msg}}
</div>
//建立 vue 对象，固定格式
new Vue({
  el: '#app',    //通过 id 嵌入元素，el 是元素 ELEMENT 的缩写
  data: {
    msg: 'Holle Word!'
  }
})
```

三 、指令

指令：带有前缀 v-，以表示它们是 Vue 提供的特殊特性，通过属性来操作元素。

1、v-model

```
v-model:实现了数据和视图的双向绑定
 分成了 3 步：
 1）把元素的值和数据相互绑定
 2）当输入内容时，数据同步发生变化，视图  ---数据的驱动
 3）当改变数据时，输入内容也会发生变化，数据-》视图的驱动
例子：利用 vue 中的 v-model 把 input 标签与 data 数据中 msg 属性进行双向绑定，msg 可以有默认值也可以默认为空！
<div id="app">
    <input type="text" v-model="msg">
    <h1>{{msg}}</h1>
</div>

<script>
    new Vue({
        el:"#app",
```

```
        data:{
            msg:"Holle Word !"
        }
    })
```

2、v-bind

```
// 绑定元素的属性来执行相应的操作，鼠标悬浮几秒后显示出绑定的提示信息
<div id="app">
    <a v-on:href="url">我想去百度</a>
</div>
<script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                url: "http://www.qq.com"
            },
        })
    </script>
```

3、v-text

```
// v-text 在元素中插入文本，比较单一
<div id="app">
    <h1 v-text="msg">{{msg}}</h1>
</div>
<script>
    new Vue({
        el:"#app",
        data:{
            msg:"Holle Word !"
        }
    })
```

4、v-html

```
 // v-html:在元素不中不仅可以插入文本，还可以插入标签，多样化
<div id="app">
<h1 v-html="hd"></h1>
</div>
```

```
<script>
    new Vue({
        el:"#app",
        data:{
            hd: "<input type='button' value='提交'>",
            str: "我要发财!"
        }
    })
```

5、v-if -- v-show -- v-on

// v-if: 根据表达式的真假值来动态插入和移除元素，下面的例子演示了我们不仅可以把数据绑定到 DOM 文本或特性，还可以绑定到 DOM 结构。此外，Vue 也提供一个强大的过渡效果系统，可以在 Vue 插入/更新/移除元素时自动应用过渡效果。

// v-show:根据表达式的真假值来隐藏和显示元素

代码显示

```
<div id="app">
        <p v-if="pick">我是刘德华</p>
        <p v-else>我是张学友</p>

        <p v-show="temp">我是赵本山</p>
        <p v-show="ok">你喜欢我吗？</p>

</div>
<script>
    var vm = new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                pick: false,
                temp: true,
                ok: true
            }
        })
        <!--不属于 vue，单纯是 js 的语法，没过多少时间就会改变当前状态-->
```

```
        window.setInterval(function(){
            vm.ok = !vm.ok;
        },1000)
```

v-on 和 v-show 合用删除事件

// 对循环的数据进行删除 v-on 点击事件（也可以写成：@click）

```
<div id="app">
        <input type="button" value="点我吧!" v-on:click="show()">
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                arr: [11,22,3344,55],
            },
            methods: {
                show: function () {
                    this.arr.pop();
                }
            }
        })
    </script>
```

6、v-for

根据变量的值来循环渲染元素

```
<div id="app">
  <ul>
    <li v-for="item in todos">
      {{ item.text }}
    </li>
  </ul>
</div>
```

```
var app = new Vue({
  el: '#app',
  data: {
    todos: [
      { text: '学习 JavaScript' },
      { text: '学习 Vue' },
      { text: '整个牛项目' }
    ]
  }
})
```

```
// 循环列表有两个参数，一个为元素中的数据，另一个为索引值
 <div id="app">
        <ul>
            <li v-for="(item,index2) in arr">
                {{item}}: {{index2}}
            </li>
        </ul>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                arr: [11,22,33,44,55],
            }
        })
    </script>
// 循环字典时，有三个参数，元素中的 value 值，key，索引值
<div id="app">
          <ul>
            <li v-for="(item,key,index) in obj">
                {{item}}: {{key}}:{{index}}
            </li>
        </ul>
```

```
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                obj: {a:"张三",b:"李四",c:"王大拿",d:"谢大脚"},
            },
        })
    </script>
// 循环数组中的对象，点出来（理解但是不知道怎么表达，参考之前学的知识）
<div id="app">
          <ul>
            <li v-for="item in obj2">
                {{item.username}}
                {{item.age}}
                {{item.sex}}
            </li>
        </ul>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                obj2:[
                    {username: "jason"},
                    {age: 20},
                    {sex: "male"}
                ],
            },
        })
    </script>
```

对数组的操作：

```
push
pop
```

```
shift
unshift
splice
reverse
```

7、输入信息表（low 逼）

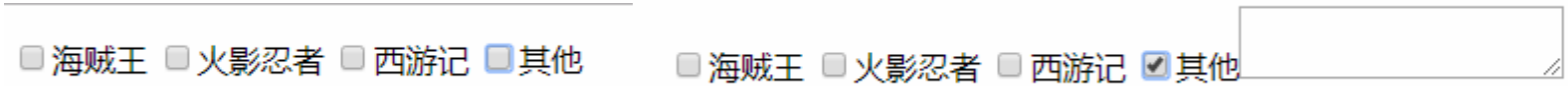
```
<div id="app">
    <div>
    <p><input type="text" placeholder="姓名" v-model="username"></p>
    <p><input type="text" placeholder="年龄" v-model="age"></p>
    <p><input type="button" value="提交" v-on:click="add"></p>
    </div>
    <div>
        <table cellpadding="1" border="1" >
            <tr v-for="(item,index) in arr">
                <td><input type="text" class="txt" v-model="item.username"></td>
                <td>{{item.age}}</td>
                <td>{{index}}</td>
                <td><input type="text" class="txt"></td>
                <td><input type="button" value="删除" v-on:click="del(index)"></td>
            </tr>
        </table>
    </div>
</div>
<script>
    new Vue({
        el:"#app",
        data:{
            username:"",
            age:"",
            arr:[]
        },
        methods:{
            add:function () {
                this.arr.push({username:this.username,age:this.age})
```

```
                console.log(this.arr);
            },
            del:function (index) {
                this.arr.splice(index,1);
            }
        }
    })
</script>
```

四、相关代码展示

利用所学知识点做一些简单的页面在展示（烂到家了）

1、显示动态生成

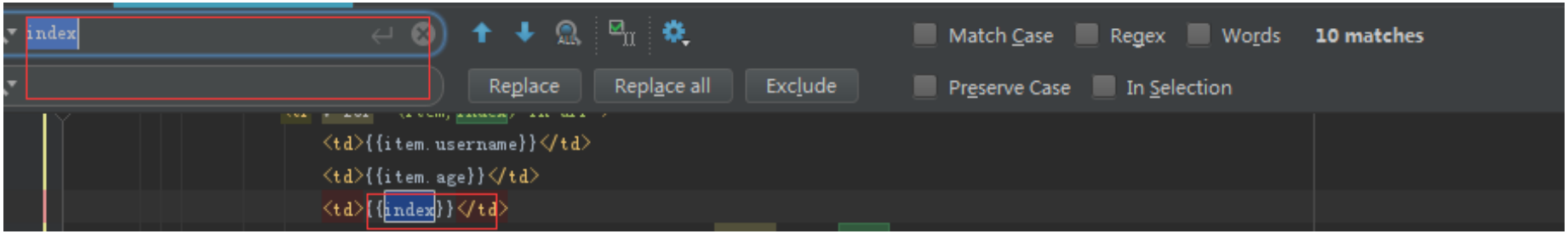



```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script src="vue.js"></script>
    <style>
        /*<!--把 li 标签的黑点去掉-->*/
        ul li {
            display: inline-block;
        }
    </style>
</head>
<body>
<div id="app">
    <ul>
        <!--CheckBox 是选项框框-->
        <li><input type="checkbox">海贼王</li>
```

```
        <li><input type="checkbox">火影忍者</li>
        <li><input type="checkbox">西游记</li>
        <li>
            <!--对 create 添加一个点击事件，后面的 focus 是一个自定义指令，HTML 和 show 指令-->
            <input type="checkbox" v-on:click="create()">其他<li v-focus v-html="htmlstr" v-show="test"></li>
        </li>
    </ul>
</div>

<script>
//    <!--定义一个 vue 对象，内部生成 n 个方法-->
    var vm = new Vue({
        el: "#app",    //表示在当前这个元素内开始使用 VUE
        data: {
            htmlstr: "<textarea></textarea>",   //生成一个标签，就是那个框框
            test: false
        },
        methods: {
            create: function () {
                this.test = !this.test;
            }
        }
    })
</script>
</body>
</html>
```

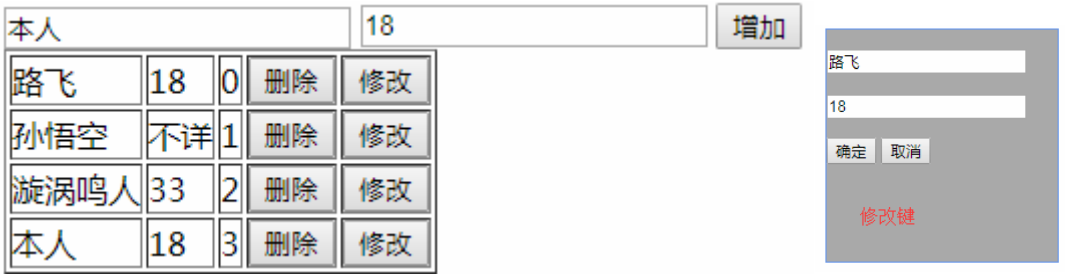
2、增加信息表功能

插曲：



只想说一句 MMP，以为索引英文是别的单词，找了 半天 pycharm 替换快捷键，最后有道一下，索引英文竟然是 index。

正文：



通过点击增加按钮，增加数据，基本的增删改，以上页面通过多个知识点 进行拼接而成。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
   <script src="vue.js"></script>
    <style>
        ul li{
            list-style: none;
        }
        .tipbox{
            width: 200px;
            height:200px;
            border: 1px solid cornflowerblue;
            position: absolute;
```

```
            background-color: #aaaaaa;
            top: 200px;
            left: 600px;

        }
    </style>
</head>
<body>
      <div id="app">
        <div>
            <input type="text" placeholder="姓名" v-model="username">
            <input type="text" placeholder="年龄" v-model="age">
            <input type="button" value="增加" @click="add">
        </div>
          <div>
                <table cellpadding="0" border="1">
                    <tr v-for="(item,index) in arr">
                        <td>{{item.username}}</td>
                        <td>{{item.age}}</td>
                        <td>{{index}}</td>
                        <td><input type="button" value="删除" @click="del(index)"></td>
                        <td><input type="button" value="修改" @click="showBox(index)"></td>
                    </tr>
                </table>
          </div>
          <div class="tipbox" v-show="isShow">
              <p><input type="text" placeholder="姓名" v-model="m_username"></p>
            <p><input type="text" placeholder="年龄" v-model="m_age"></p>
            <p>
                <input type="button" value="确定" @click="save()">
                <input type="button" value="取消" @click="cancel()">
            </p>
          </div>
    </div>
```

```
<script>
    new Vue({
        el: "#app", //表示在当前这个元素内开始使用 VUE
        data:{
            username: "",
            age: "",
            arr: [],
            isShow:false,
            m_username: "",
            m_age: "",
            n: 0
        },
        methods: {
            add: function () {
                this.arr.push({username:this.username,age: this.age});
                console.log(this.arr);
            },
            // 删除
            del: function (index) {
                this.arr.splice(index,1);
            },
            // 修改
            showBox: function (index) {
                this.isShow = true;
                this.n = index;
                this.m_username = this.arr[index].username;
                this.m_age = this.arr[index].age;
            },
            cancel: function () {
                this.isShow = false
            },
            save: function () {
                this.arr[this.n].username = this.m_username;
                this.arr[this.n].age = this.m_age;
```

```
                this.isShow = false
            }
        }
    })
</script>
</body>
</html>
```

还有很多实例和知识点，请看官网 https://cn.vuejs.org/v2/guide/index.html

相关博客：http://www.cnblogs.com/keepfool/p/5619070.html

http://www.cnblogs.com/zhaodagang8/category/1134528.html

前端 VUE 框架

一、什么是 VUE?

它是一个构建用户界面的 JAVASCRIPt 框架

vue 不关心你页面上的是什么标签，它操作的是变量或属性

为什么要使用 VUE?

在前后端分离的时候，后端只返回 json 数据，再没有 render 方法，前端发送 ajax 请求（api=url）得到数据后，要在页面渲染数据，如果你 js+css 实现就太麻烦了，这时候 VUE 就出现了。

二、怎么样使用 VUE

1）引入 vue.js

```
<script src=vue.js></script>
```

2) 展示 html

```
<div id="app">
    <input type="text" v-model="msg">
    <p>{{msg}}</p>
</div>
```

3)建立 vue 对象

```
new Vue({
    el: "#app", //表示在当前这个元素内开始使用 VUE
    data:{
        msg: ""
    }
})
```

三、VUE 指令

1. v-model：实现了数据和视图的双向绑定

分成了 3 步：

1）把元素的值和数据相绑定

2）当输入内容时，数据同步发生变化，视图 --->数据的驱动

3）当改变数据时，输入内容也会发生变化，数据--->视图的驱动

```
<body>
      <div id="app">
        <input v-model="msg">
          <p>{{msg}}</p>
          <input type="button" value="变化" @click="change">
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
              msg: "aaaaa"
            },
            methods: {
                change: function () {
                    this.msg = 8888888;
                }
            }
        })
    </script>
</body>
```

2. v-on:监听元素事件，并执行相应的操作； @是对 v-on 的简写

```
<style>
      ul li{
              list-style: none;
              display: inline-block;
              border: 1px solid cornflowerblue;
              height:40px;
              line-height: 40px;
```

```
            width: 120px;
            text-align: center;
        }
</style>

<body>
    <div id="mybox">
        <ul>
            <li>
                <span v-on:click="qh(true)">二唯码登录</span>
            </li>
            <li>
                <span v-on:click="qh(false)">邮箱登录</span>
            </li>
        </ul>
        <div v-show="temp">
            <img src="erma.jpg">
        </div>
        <div v-show="!temp">
            <form action="http://mail.163.com" method="post">
                <p><input type="email"></p>
                <p><input type="password"></p>
                <p><input type="submit" value="登录"></p>
            </form>
        </div>
    </div>
    <script>
        new Vue({
            el: "#mybox",
            data: {
                temp: true
            },
            methods: {
                qh: function (t) {
```

```
                    this.temp = t
                }
            }
        })
    </script>
</body>
```

补充：display=none 时 如果你的盒子没有宽高，那它就不占位

3. v-bind:绑定元素的属性来执行相应的操作； ：是对 v-bind 的简写

```
<style>
        .bk_1{
            background-color: cornflowerblue;
            width: 200px;
            height: 200px;
        }
        .bk_2{
            background-color: red;
            width: 200px;
             height: 200px;
        }
        .bk_3{
            border: 5px solid #000;
        }
</style>

<body>
    <div id="app">
        <a href="http://www.qq.com" v-bind:title="msg">腾讯</a>
        <div :class="bk"></div>
        <div :class="bk2"></div>
        <div :class="{bk_2:temp}">fdjjdjdkdkkeedd</div>
        <div :class="[bk2,bk3]">8888888888</div>
    </div>
    <script>
       var vm =  new Vue({
```

```
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                msg: "我想去腾讯公司上班",
                bk:"bk_1",
                bk2:"bk_2",
                bk3:"bk_3",
                temp: false
            }
        })
    </script>
</body>
```

4. v-for:根据变量的值来循环渲染元素

```
<body>
      <div id="app">
        <ul>
            <li v-for="(item,index2) in arr">
                {{item}}: {{index2}}
            </li>
        </ul>

          <ul>
            <li v-for="(item,key,index) in obj">
                {{item}}: {{key}}:{{index}}
            </li>
        </ul>
          <ul>
            <li v-for="item in obj2">
                {{item.username}}
                {{item.age}}
                {{item.sex}}
            </li>
        </ul>
          <div v-for="i in 8">
              {{i}}
```

```
        </div>
        <input type="button" value="点我吧!" @click="show()">
        <a :href="url">我想去百度</a>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                arr: [11,22,3344,55],
                obj: {a:"张三",b:"李四",c:"王大拿",d:"谢大脚"},
                obj2:[
                    {username: "jason"},
                    {age: 20},
                    {sex: "male"}
                ],
                str: "我来了",
                url: "http://www.qq.com"
            },
            methods: {
                show: function () {
                    this.arr.pop();
                }
            }
        })
    </script>

</body>
```

5. v-if / show

v-if: 根据表达式的真假值来动态插入和移除元素

v-show:根据表达式的真假值来隐藏和显示元素

```
<body>
      <div id="app">
       <p v-if="pick">我是刘德华</p>
       <p v-else>我是张学友</p>
```

```
        <p v-show="temp">我是赵本山</p>
        <p v-show="ok">你喜欢我吗？</p>
    </div>
    <script>
        var vm = new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                pick: false,
                temp: true,
                ok: true
            }
        })
         window.setInterval(function(){
             vm.ok = !vm.ok;
         },1000)
    </script>
</body>
```

6. v-html:在元素中不仅可以插入文本，还可以插入标签

```
<body>
      <div id="app">
        <ul>
            <li>
                <input type="checkbox">苹果
            </li>
            <li>
                <input type="checkbox">香蕉
            </li>
            <li>
                <input type="checkbox">香梨
            </li>
            <li>
                <input type="checkbox" v-on:click="create()">其它
            </li>
            <li v-html="htmlstr" v-show="test">
```

```
            </li>
        </ul>
    </div>
    <script>
        var vm = new Vue({
            el: "app", //表示在当前这个元素内开始使用 VUE
            data:{
                htmlstr: "<textarea></textarea>",
                test: false
            },
            methods: {
                create: function () {
                    this.test = !this.test;
                }
            }
        })
    </script>
</body>
```

7. 模板对象

```
<body>
      <div id="app">
        <p>{{msg}}</p>
        <p>{{80+2}}</p>
        <p>{{20>30}}</p>
        {{msg}}
          我是：<h1 v-text="msg">{{str}}</h1>
          你是：<h1 v-text="msg">2222222222222</h1>

          <h2 v-html="hd"></h2>
          <h2 v-html="str"></h2>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
```

```
            data:{
                msg: "我是老大",
                hd: "<input type='button' value='你是小三'>",
                str: "我要发财!"
            }
        })
    </script>
</body>
```

8. 计算属性

```
<body>
    <div id="app">
        <p>{{msg}}</p>
    </div>
    <script>
        var vm = new Vue({
            el: "#app",
            data: {
                temp: 1001
            },
           computed: {
               msg: function () {
                   if(this.temp > 1000){
                       return parseInt(this.temp/10)-1
                   } else {
                       return this.temp-1
                   }
               }
           }
        })
    </script>
</body>
```

9. 小综合练习

对数组的操作：

push 数组中最后添加一个值

```
    pop         数组中删除最后一个值
    shift       删除数组头一个元素
    unshift     向开头添加一个或多个元素
    splice      删除其中一个对象
    reverse     反转
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
   <script src="vue.js"></script>
    <style>
        ul li{
            list-style: none;
        }
    </style>
</head>
<body>
     <div id="app">
       <div>
           <input type="text" placeholder="姓名" v-model="username">
           <input type="text" placeholder="年龄" v-model="age">
           <input type="button" value="增加" @click="add">
       </div>
         <div>
              <table cellpadding="0" border="1">
                  <tr v-for="(item,index) in arr">
                      <td><input type="text" class="txt" v-model="item.username"> </td>
                      <td>{{item.age}}</td>
                      <td>{{index}}</td>
                      <td><input type="text" class="txt"></td>
                      <td><input type="button" value="删除" @click="del(index)"></td>
                  </tr>
```

```
            </table>
        </div>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
                username: "",
                age: "",
                arr: []
            },
            methods: {
                add: function () {
                    this.arr.push({username:this.username,age: this.age});
                    console.log(this.arr);
                },
                del: function (index) {
                    this.arr.splice(index,1);
                }
            }
        })


    </script>

</body>
</html>
```

10. 自定义指令：相关网址 https://cn.vuejs.org/v2/guide/custom-directive.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
   <script src="vue.js"></script>
```

```
    <style>
        ul li{
            list-style: none;
        }
    </style>
</head>
<body>
      <div id="app">
            <input type="text" v-focus>
    </div>
    <script>
        new Vue({
            el: "#app", //表示在当前这个元素内开始使用 VUE
            data:{
            },
            directives: {
                focus: {    //指令的名字
                    //当被绑定的元素插入到 DOM 中时
                    inserted: function (tt) {
                        tt.focus();
                        tt.style.backgroundColor = "blue";
                        tt.style.color = "#fff"
                    }
                }
            }
        })

    </script>

</body>
</html>
```

EMCAScript 6 又叫 es2015

1、常量和变量

常量： const a = "hello"
常量不能修改和重复定义
变量：
let:定义一个块级作用域的变量
需要先定义再使用；(不存在变量提升)
不能重复定义
可以被修改
var:定义一个变量
存在变量提升
变量提升： 先使用后定义和赋值
// console.log(b); undefined
// var b = 123456;
详解：
// var b;
// console.log(b); undefined
// b = 123456;
js 的数据类型：
string array number null undefined boolean object
基本数据类型：string number null undefined boolean
引用类型：array object

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        const a = "hello";
        console.log(a);

//        console.log(b);
```

```
//        var b = 123456;

        //变量提升

//        var b;
//         console.log(b);
//        b = 123456;

        //let c = 100;
        if(10> 9){
            let c=200;
            console.log(c);
        }
         console.log(c);
        var  c = 300
        let d = 888;
        d = 999
        console.log(d);

        var i=10;
        var arr = [22,33,44,55]
        for(let i=0;i< arr.length;i++){

        }

        if(i>5){
            console.log(i+10);
        }

        const obj = {
            name: "谢小二",
            age: 22
        }
        var obj2 = obj;
```

```
        obj2.age = 90
        console.log(obj.age);

    </script>
</head>
<body>
</body>
</html>
```

2、模板字符串

通过反引号来使用，字符串当中可以使用变量

可以当作普通字符串来处理

可以使用多行字符串

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
        <ul id="list_1">
        </ul>
        <script>
        var name = `小三`;
        console.log(`她的名字叫${name}`);
        document.getElementById("list_1").innerHTML = `
        <li>11</li>
        <li>22</li>
        <li>33</li>
        <li>44</li>
        `
    </script>
</body>
</html>
```

3、解构变量

解构变量的结构要一样，结构对象时被赋值的变量要和对象内的 key 一样

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
//        let arr = [89,90,99];
//        let a = arr[0];
//        let b = arr[1];
//        let c = arr[2];
        let [a,b,c,[d]] = [89,90,99,[100]];
        console.log(a);
        console.log(c);
        console.log(d);
        let obj = {
            "a1":"json",
            a2: 23
        };
        let {a1,a2} = obj;
        console.log(a1);
    </script>
</head>
<body>

</body>
</html>
```

4、对象的扩展

对象当中的属性可以简写

对象当中的方法也可以简写

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        let username = '谢小闲';

        let obj = {
            username,       //username='谢小闲'
            fun() {
                alert(999);
            }
        };
        console.log(obj.username);
        obj.fun();

//用法举例：
//        var useranme = $("#text1").val();
//        var password = $("#text2").val();

//        $.get(url,{useranme,password},function(){
//
//
//        })
    </script>
</head>
<body>

</body>
</html>
```

5、函数的扩展

可以给函数默认参数

剩余参数：function fun2(x=3,...y) {

console.log(x);

```
                        console.log(y);    //  [3.4.5]
                        }
            fun2(x=2,y=3,z=4,5)
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <script>
        function fun(x=100) {
            alert(x);
        }
        //fun();
        function fun2(x=3,...y) {
            console.log(x);
            console.log(y);    //  [3.4.5]
        }
        fun2(x=2,y=3,z=4,5)
    </script>
</head>
<body>
</body>
</html>
```

6、数组的扩展

1)判断数组当中是否存在某个数值

```
     console.log(arr.indexOf(1000))     //没有打印 -1 ，有则打印数值的索引
     console.log(arr.includes(201))     // false 或 true
```

2）对数组的遍历

```
    var arr = [78,89,90,21];
    arr.forEach(function (value,index) {
            console.log(value);
        })
```

```
    var arr2 = arr.map(function (value,index) {
            return value+1          //必须有返回值
        })
    console.log(arr2);              //[79, 90, 91, 22]
    let arr3 = [11,22,33]
    for(var i in arr3){             // in 方法   i 表示索引
            console.log(arr3[i]);
        }
    for(var i of arr3){             // of 方法   i 表示值
            console.log(i);
        }
```

3)对数组的过滤

```
    var arr4 = arr.filter(function (value,index) {
            return value > 50       //必须有返回值
        })
        console.log(arr4);          // [78, 89, 90]
```

7、类扩展

```
<script>
        var age2 = 99;
        Object.prototype.age2 = age2;
        function Person(name,age){
            this.name = name;
            this.age = age;
            this.run = function () {
                alert(`${this.name}能跑`);
            }
        }
        Person.prototype.sing = function () {      //prototype 是对每一个类的扩展，如果类内有，那拓展就失效了
            alert(`${this.name}能唱歌`);
        };
        let man = new Person("小秋",19);
        console.log(man.name);       //首先在类中查找
        man.run();
        man.sing();                  //其次到类的 prototype 中查找
```

```
        console.log(man.age2);        //再到 Object 中查找
</script>
```

Vue.js 面试题整理

Vue 项目结构介绍

- build 文件夹：用于存放 webpack 相关配置和脚本。
- config 文件夹：主要存放配置文件，比如配置开发环境的端口号、开启热加载或开启 gzip 压缩等。
- dist 文件夹：默认命令打包生成的静态资源文件。
- node_modules：存放 npm 命令下载的开发环境和生产环境的依赖包。
- src: 存放项目源码及需要引用的资源文件。
- src 下 assets：存放项目中需要用到的资源文件，css、js、images 等。
- src 下 componets：存放 vue 开发中一些公共组件。
- src 下 emit：自己配置的 vue 集中式事件管理机制。
- src 下 router：vue-router vue 路由的配置文件。
- src 下 service：自己配置的 vue 请求后台接口方法。
- src 下 page：存在 vue 页面组件的文件夹。
- src 下 util：存放 vue 开发过程中一些公共的 js 方法。
- src 下 vuex：存放 vuex 为 vue 专门开发的状态管理器。
- src 下 app.vue：整个工程的 vue 根组件。
- src 下 main.js：工程的入口文件。
- index.html：设置项目的一些 meta 头信息和提供 html 元素节点，用于挂载 vue。
- package.json：对项目的描述以及对项目部署和启动、打包的 npm 命令管理。

Vue 常用指令

- v-model 多用于表单元素实现双向数据绑定（同 angular 中的 ng-model）
- v-for 格式： v-for="字段名 in(of) 数组 json" 循环数组或 json(同 angular 中的 ng-repeat),需要注意从 vue2 开始取消了$index
- v-show 显示内容 （同 angular 中的 ng-show）
- v-hide 隐藏内容（同 angular 中的 ng-hide）
- v-if 显示与隐藏 （dom 元素的删除添加 同 angular 中的 ng-if 默认值为 false）
- v-else-if 必须和 v-if 连用
- v-else 必须和 v-if 连用 不能单独使用 否则报错 模板编译错误
- v-bind 动态绑定 作用： 及时对页面的数据进行更改
- v-on:click 给标签绑定函数，可以缩写为@，例如绑定一个点击函数 函数必须写在 methods 里面
- v-text 解析文本
- v-html 解析 html 标签
- v-bind:class 三种绑定方法 1、对象型 '{red:isred}' 2、三元型 'isred?"red":"blue"' 3、数组型 '[{red:"isred"},{blue:"isblue"}]'

- v-once 进入页面时 只渲染一次 不在进行渲染
- v-cloak 防止闪烁
- v-pre 把标签内部的元素原位输出

v-for 与 v-if 的优先级

当它们处于同一节点，v-for 的优先级比 v-if 更高。

vue 中 keep-alive 组件的作用

keep-alive：主要用于保留组件状态或避免重新渲染。两个重要属性，include 缓存组件名称，exclude 不需要缓存的组件名称。

v-if 和 v-show 有什么区别

共同点：

v-if 和 v-show 都可以用来动态显示 DOM 元素。

区别：

- 编译过程： 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-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。
- 编译条件： v-if 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。v-show 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。
- 性能消耗： v-if 有更高的切换消耗。v-show 有更高的初始渲染消耗。
- 应用场景： v-if 适合运行时条件很少改变时使用。v-show 适合频繁切换。

v-on 可以监听多个方法吗?

v-on 可以监听多个方法，但是同一种事件类型的方法，v-on 只能监听一个。

vue 中 key 值的作用

用于 管理可复用的元素。因为 Vue 会尽可能高效地渲染元素，通常会复用已有元素而不是从头开始渲染。当 Vue.js 用 v-for 正在更新已渲染过的元素列表时，它默认用“就地复用”策略。如果数据项的顺序被改变，Vue 将不会移动 DOM 元素来匹配数据项的顺序， 而是简单复用此处每个元素，并且确保它在特定索引下显示已被渲染过的每个元素。key 的作用主要是为了高效的更新虚拟 DOM

vue $nextTick 作用是什么?

官方解释：在下次 DOM 更新循环结束之后执行延迟回调。在修改数据之后立即使用这个方法，获取更新后的 DOM。简单的说就是再 DOM 操作时，vue 的更新是异步的，$nextTick 是用来知道什么时候 DOM 更新完成的。

举例：我们给一个 div 设置显示隐藏，当点击 button 的时候 #textDiv 先是被渲染出来，接着我们获取 #textDiv 的 html 内容

```
<div id="app">
    <div id="textDiv" v-if="isShow">这是一段文本</div>
    <button @click="getText">获取 div 内容</button>
</div>
<script>
var app = new Vue({
```

```
    el : "#app",
    data:{
        isShow: false
    },
    methods:{
        getText:function(){
            this.isShow= true;
            var text = document.getElementById('textDiv').innnerHTML;
             console.log(text);
        }
    }
})
</script>
```

这段代码表面上看不会有问题，但实际上点击报错，提示获取不到 div 元素，这里就涉及到 Vue 异步更新队列。

Vue 异步更新队列

Vue 执行 DOM 更新时，只要观察到数据变化，就会自动开启一个队列，并缓冲在同一个事件循环中发生的所以数据改变。在缓冲时会去除重复数据，从而避免不必要的计算和 DOM 操作。以上面的代码举例，你用一个 for 循环来动态改变 isShow 100 次，其实它只会应用最后一次改变，如果没有这种机制，DOM 就要重绘 100 次，这固然是一个很大的开销。所以执行 this.isShow= true 时，#textDiv 还没有被创建出来，直到下一个 Vue 事件循环时，才开始创建。

上面的代码应修改为：

```
getText:function(){
    this.showDiv = true;
    this.$nextTick(function(){
            var text = document.getElementById('div').innnerHTML;
            console.log(text);
    });
}
```

Vue 如何注册组件

组件系统是 Vue.js 其中一个重要的概念，它提供了一种抽象，让我们可以使用独立可复用的小组件来构建大型应用，任意类型的应用界面都可以抽象为一个组件树

全局注册

- 在 src 文件夹中新建 utils 文件夹，在 utils 文件夹中新建 components.js 文件
- 在 components.js 文件引入所有要注册的全局组件
- 在 main.js 中引入 components.js 文件并使用 Vue.use() 全局注册

- 举例:

组件 diyHeader.vue

```
<template>
    <div>
        <div  id="header" @click="fun"></div>
    </div>
</template>

<script>
export default {
    methods:{
        fun(){
            alert(1);
        }
    }
}
</script>

<style>
#header{
    height: 100px;
    background: red;
}
</style>
```

util 下 components.js

```
import DiyHeader from "../components/diyHeader.vue"
export default (Vue)=>{
    Vue.component("DiyHeader", DiyHeader)
  }
```

main.js

```
import components from "./util/components"
Vue.use(components);
```

之后就可以直接在项目中使用 DiyHeader 这个组件

局部注册

在页面里导入组件然后放到 components 中就可以使用了

```
import DiyHeader from "../components/diyHeader.vue"
export default {
  name: "App",
  components: { DiyHeader },
}
```

全局注册指令

- 在 src 文件夹中新建 utils 文件夹，在 utils 文件夹中新建 directives.js 文件
- 在 directives.js 文件引入所有要注册的全局指令
- 在 main.js 中引入 directives.js 文件并使用 Vue.use() 全局注册

```
directives.js
export default (Vue)=>{
    Vue.directive("focus", {
        inserted: function (el) {
            el.focus();
        }
    })
}
main.js
import directives from '@/utils/directives.js'
Vue.use(directives
```

什么是 vue 生命周期和生命周期钩子函数?

vue 的生命周期是： vue 实例从创建到销毁，也就是从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、卸载等一系列过程。

vue 生命周期钩子函数有哪些?

beforeCreate:

在实例初始化之后，数据观测 (data observer) 和 event/watcher 事件配置之前被调用。

created:

在实例创建完成后被立即调用。在这一步，实例已完成以下的配置：数据观测 (data observer)， 属性和方法的运算，watch/event 事件回调。然而，挂载阶段还没开始，$el 属性目前不可见。

beforeMount:

在挂载开始之前被调用：相关的 render 函数首次被调用。

mounted:

el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

beforeUpdate：
数据更新时调用，发生在虚拟 DOM 打补丁之前。这里适合在更新之前访问现有的 DOM，比如手动移除已添加的事件监听器。该钩子在服务器端渲染期间不被调用，因为只有初次渲染会在服务端进行。
updated：
由于数据更改导致的虚拟 DOM 重新渲染和打补丁，在这之后会调用该钩子。
activated
keep-alive 组件激活时调用。该钩子在服务器端渲染期间不被调用。
deactivated
keep-alive 组件停用时调用。该钩子在服务器端渲染期间不被调用。
beforeDestroy
实例销毁之前调用。在这一步，实例仍然完全可用。该钩子在服务器端渲染期间不被调用。
destroyed
Vue 实例销毁后调用。调用后，Vue 实例指示的所有东西都会解绑定，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。
errorCaptured（2.5.0+ 新增）
当捕获一个来自子孙组件的错误时被调用。此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播。

vue 等单页面应用及其优缺点

单页 Web 应用:
就是只有一张 Web 页面的应用。单页应用程序（SPA）是加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面的 Web 应用程序。浏览器一开始会加载必需的 HTML、CSS 和 JavaScript，所有的操作都在这张页面上完成，都由 JavaScript 来控制。因此，对单页应用来说模块化的开发和设计显得相当重要。
单页 Web 应用的优点：

- 提供了更加吸引人的用户体验：具有桌面应用的即时性、网站的可移植性和可访问性。
- 单页应用的内容的改变不需要重新加载整个页面，web 应用更具响应性和更令人着迷。
- 单页应用没有页面之间的切换，就不会出现“白屏现象”,也不会出现假死并有“闪烁”现象
- 单页应用相对服务器压力小，服务器只用出数据就可以，不用管展示逻辑和页面合成，吞吐能力会提高几倍。
- 良好的前后端分离。后端不再负责模板渲染、输出页面工作，后端 API 通用化，即同一套后端程序代码，不用修改就可以用于 Web 界面、手机、平板等多种客户端。

单页 Web 应用的缺点：

- 首次加载耗时比较多。
- SEO 问题，不利于百度，360 等搜索引擎收录。
- 容易造成 Css 命名冲突。
- 前进、后退、地址栏、书签等，都需要程序进行管理，页面的复杂度很高，需要一定的技能水平和开发成本高。

自定义指令的几个钩子函数

bind：只调用一次，指令第一次绑定到元素时调用。在这里可以进行一次性的初始化设置。
inserted：被绑定元素插入父节点时调用（仅保证父节点存在，但不一定已被插入文档中)。
update：所在组件的 VNode 更新时调用，但是可能发生在其子 VNode 更新之前。指令的值可能发生了改变，也可能没有。但是你可以通过比较更新前后的值来忽略不必要的模板更新 。
componentUpdated：指令所在组件的 VNode 及其子 VNode 全部更新后调用。
unbind：只调用一次，指令与元素解绑时调用。

13. Vue 双向绑定实现的原理

采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty（）来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。当把一个普通 Javascript 对象传给 Vue 实例来作为它的 data 选项时，Vue 将遍历它的属性，用 Object.defineProperty 将它们转为 getter/setter。用户看不到 getter/setter，但是在内部它们让 Vue 追踪依赖，在属性被访问和修改时通知变化。

active-class 是哪个组件的属性

vue-router 模块 的 router-link 组件

vue-router 有哪几种导航钩子

全局守卫： router.beforeEach
全局解析守卫： router.beforeResolve
全局后置钩子： router.afterEach
路由独享的守卫： beforeEnter
组件内的守卫： beforeRouteEnter、beforeRouteUpdate (2.2 新增)、beforeRouteLeave

vuex 是什么？怎么使用？哪种功能场景使用它?

Vuex 是一个专为 Vue.js 应用程序开发的状态管理器，采用集中式存储管理应用的所有组件的状态，主要是为了多页面、多组件之间的通信。
Vuex 有 5 个重要的属性，分别是 State、Getter、Mutation、Action、Module，由 view 层发起一个 Action 给 Mutation，在 Mutation 中修改状态，返回新的状态，通过 Getter 暴露给 view 层的组件或者页面，页面监测到状态改变于是更新页面。如果你的项目很简单，最好不要使用 Vuex，对于大型项目，Vuex 能够更好的帮助我们管理组件外部的状态，一般可以运用在购物车、登录状态、播放等场景中。

$route 和$router 的区别

$route 是“路由信息对象”，包括 path，params，hash，query，fullPath，matched，name 等路由信息参数。而$router 是“路由实例”对象包括了路由的跳转方法，钩子函数等

什么是 MVVM?

答：MVVM 是是 Model-View-ViewModel 的缩写，Model 代表数据模型，定义数据操作的业务逻辑，View 代表视图层，负责将数据模型渲染到页面上，ViewModel 通过双向绑定把 View 和 Model 进行同步交互，不需要手动操作 DOM 的一种设计思想。

怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数?

答：在 router 目录下的 index.js 文件中，对 path 属性加上/:id。 使用 router 对象的 params.id

vue 项目优化解决方案

使用 mini-css-extract-plugin 插件抽离 css

配置 optimization 把公共的 js 代码抽离出来
通过 Webpack 处理文件压缩
不打包框架、库文件，通过 cdn 的方式引入
小图片使用 base64
配置项目文件懒加载
UI 库配置按需加载
开启 Gzip 压缩

Vue 路由的实现

hash 模式
通过用 window.location.hash 监听页面的 hash 值变化，切换对于的内容，hash 变化不会重载页面。
history 模式
history 采用 HTML5 的新特性；且提供了两个新方法：pushState（），replaceState（）可以对浏览器历史记录栈进行修改，以及 popState 事件的监听到状态变更。

Vue 父子组件通信

父组件向子组件传值通过 props
子组件向父组件传递，子组件使用$emit 传递，父组件使用 on 监听。

Vue minix（混入）的用法

minix（混入） 是 Vue 中的高级用法，混入（mixin）提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。比如我们做一个下拉加载，很多组件都需要用到下拉加载，我们就可以把下拉加载封装成一个 minix，然后需要下拉加载功能的页面都去导入这个 minix，minix 里面的属性或者方法就会被混合到当前组件本身的属性上。简单的说，minix B 有个 C 方法（下拉加载），页面 A 需要下拉加载于是就导入了 minix B，这时候页面 A 也就拥有了 C 方法。如果页面 A 本身有个 D 方法，这时页面 A 就会既有 C 方法也有本身的 D 方法。

关于 Vue.use() 的理解

Vue.use() 是 Vue 的一个全局注册方法，主要用来注册插件，默认第一个参数是它接受的参数类型必须是 Function 或者是 Object，如果是个对象，必须提供 install 方法，install 方法默认第一个参数为 Vue,其后的参数为注册时传入的 arguments。如果是 Function 那么这个函数就被当做 install 方法。同一个插件 Vue.use 会自动阻止多次注册。除了在注册插件中使用 Vue.use 外，我们还可以在 directive 注册、filters 注册、components 注册等条件下使用。
有的时候我们会遇到某些时候引入插件是并没有使用 Vue.use ，比如使用 axios 的时候，原因是 axios 没有 install 方法，所以也就不需要使用 Vue.use 来全局注册。

vue.js 面试题整理

一、什么是 MVVM？

MVVM 是 Model-View-ViewModel 的缩写。MVVM 是一种设计思想。Model 层代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑；View 代表 UI 组件，它负责将数据模型转化成 UI 展现出来，ViewModel 是一个同步 View 和 Model 的对象（桥梁）。

在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 ViewModel 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接了起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作 DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

二、mvvm 和 mvc 区别？它和其它框架（jquery）的区别是什么？哪些场景适合？

mvc 和 mvvm 其实区别并不大。都是一种设计思想。主要就是 mvc 中 Controller 演变成 mvvm 中的 viewModel。mvvm 主要解决了 mvc 中大量的 DOM 操作使页面渲染性能降低，加载速度变慢，影响用户体验。

区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

三、vue 的优点是什么？

· 低耦合。视图（View）可以独立于 Model 变化和修改，一个 ViewModel 可以绑定到不同的"View"上，当 View 变化的时候 Model 可以不变，当 Model 变化的时候 View 也可以不变。

· 可重用性。你可以把一些视图逻辑放在一个 ViewModel 里面，让很多 view 重用这段视图逻辑。

· 独立开发。开发人员可以专注于业务逻辑和数据的开发（ViewModel），设计人员可以专注于页面设计。

· 可测试。界面素来是比较难于测试的，而现在测试可以针对 ViewModel 来写。

四、组件之间的传值？

· 父组件与子组件传值

父组件通过标签上面定义传值 ：eg='data' 父组件中 data(){return {data:'egdata'}}

子组件通过 props 方法接受数据 props:['eg'] 在 props 中添加了元素之后，就不需要在 data 中再添加变量了

· 子组件向父组件传递数据

子组件通过$emit 方法传递参数

子组件部分:

```
<label>
  <span>用户名: </span>
  <input v-model="username" @change="setUser" />
</label>
```

```
methods: {
  setUser: function () {
    this.$emit('transferUser', this.username)
  }
}
```

父组件中：

```
<template>
  <div id="app">
    <LoginDiv @transferUser="getUser"></LoginDiv>
    <p>用户名为: {{user}}</p>
  </div>
</template>
```

```
  },
  methods: {
    getUser (msg) {
      this.user = msg
    }
  },
  components: {
    LoginDiv
```

子组件向子组件传递数据

Vue 没有直接子对子传参的方法，建议将需要传递数据的子组件，都合并为一个组件。如果一定需要子对子传参，可以先从传到父组件，再传到子组件。

为了便于开发，Vue 推出了一个状态管理工具 Vuex，可以很方便实现组件之间的参数传递

五、路由之间跳转

声明式（标签跳转） 编程式（ js 跳转）

1.直接修改地址栏中的路由地址 ②通过 router-link 实现跳转

```
<router-link to="/myRegister">注册</router-link>
```

③通过 js 的编程的方式

```
</div>
<script>
  var testLogin = Vue.component("login",{
    template:`
      <div>
        <h1>这是我的登录页面</h1>
        <router-link to="/myRegister">注册</router-link>
      </div>
    `
    /*to后面是路由地址*/
  })
  var testRegister = Vue.component("register",{
    methods:{
      jumpToLogin:function(){
        this.$router.push('/myLogin');
      }
    },
    template:`
      <div>
        <h1>这是我的注册页面</h1>
        <button @click="jumpToLogin">注册完成,去登录</button>
      </div>
    `

  })
  //配置路由词典
  const  myRoutes =[
    {path:'',component:testLogin},
    {path:'/myLogin',component:testLogin},
    {path:'/myRegister',component:testRegister}
  ]

  const myRouter = new VueRouter({
    routes:myRoutes
  })
  new Vue({
    router:myRouter,
    el:"#container",
    data:{
      msg:"Hello VueJs"
```

六、vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

- 第一步：在 components 目录新建你的组件文件（如：indexPage.vue），script 一定要 export default {}
- 第二步：在需要用的页面（组件）中导入：import indexPage from '@/components/indexPage.vue'
- 第三步：注入到 vue 的子组件的 components 属性上面,components:{indexPage}
- 第四步：在 template 视图 view 中使用，

例如有 indexPage 命名，使用的时候则 index-page

七、vue 如何实现按需加载配合 webpack 设置

webpack 中提供了 require.ensure()来实现按需加载。以前引入路由是通过 import 这样的方式引入，改为 const 定义的方式进行引入。

不进行页面按需加载引入方式：import home from '../../common/home.vue'

进行页面按需加载的引入方式：const home = r => require.ensure([], () => r (require('../../common/home.vue')))

在音乐 app 中使用的路由懒加载方式为：

```
const Recommend = (resolve) => {
  import('components/recommend/recommend').then((module) => {
    resolve(module)
  })
}
const Singer = (resolve) => {
  import('components/singer/singer').then((module) => {
    resolve(module)
  })
}
```

八、vuex 面试相关

（1）vuex 是什么？怎么使用？哪种功能场景使用它？

vue 框架中状态管理。在 main.js 引入 store，注入。新建一个目录 store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

main.js:

```
import store from './store'
new Vue({
el:'#app',
store
})
```

（2）vuex 有哪几种属性？

有五种，分别是 State、 Getter、Mutation 、Action、 Module

· vuex 的 State 特性

A、Vuex 就是一个仓库，仓库里面放了很多对象。其中 state 就是数据源存放地，对应于一般 Vue 对象里面的 data

B、state 里面存放的数据是响应式的，Vue 组件从 store 中读取数据，若是 store 中的数据发生改变，依赖这个数据的组件也会发生更新

C、它通过 mapState 把全局的 state 和 getters 映射到当前组件的 computed 计算属性中

· vuex 的 Getter 特性

A、getters 可以对 State 进行计算操作，它就是 Store 的计算属性

B、 虽然在组件内也可以做计算属性，但是 getters 可以在多组件之间复用

C、 如果一个状态只在一个组件内使用，是可以不用 getters

· vuex 的 Mutation 特性

Action 类似于 mutation，不同在于：Action 提交的是 mutation，而不是直接变更状态；Action 可以包含任意异步操作。

（3）不用 Vuex 会带来什么问题？

· 可维护性会下降，想修改数据要维护三个地方；

· 可读性会下降，因为一个组件里的数据，根本就看不出来是从哪来的；

· 增加耦合，大量的上传派发，会让耦合性大大增加，本来 Vue 用 Component 就是为了减少耦合，现在这么用，和组件化的初衷相背。

九、 v-show 和 v-if 指令的共同点和不同点

· v-show 指令是通过修改元素的 display 的 CSS 属性让其显示或者隐藏

· v-if 指令是直接销毁和重建 DOM 达到让元素显示和隐藏的效果

十、如何让 CSS 只在当前组件中起作用

将当前组件的<style>修改为<style scoped>

十一、<keep-alive></keep-alive>的作用是什么？

<keep-alive></keep-alive> 包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染。

十二、Vue 中引入组件的步骤？

1）采用 ES6 的 import ... from ...语法或 CommonJS 的 require()方法引入组件

2）对组件进行注册,代码如下

```
1. // 注册
2. Vue.component('my-component', {
3.   template: '<div>A custom component!</div>'
4. })
5.
```

3）使用组件<my-component></my-component>

十三、指令 v-el 的作用是什么？

提供一个在页面上已存在的 DOM 元素作为 Vue 实例的挂载目标.可以是 CSS 选择器，也可以是一个 HTMLElement 实例

十四、在 Vue 中使用插件的步骤

· 采用 ES6 的 import ... from ...语法或 CommonJSd 的 require()方法引入插件

· 使用全局方法 Vue.use(plugin)使用插件,可以传入一个选项对象 Vue.use(MyPlugin, { someOption: true })

如使用懒加载插件：

```
Vue.use(VueLazyload, {
  loading: require('common/image/default.png')
})
```

十五、请列举出 3 个 Vue 中常用的生命周期钩子函数 （见博客）

· created: 实例已经创建完成之后调用,在这一步,实例已经完成数据观测, 属性和方法的运算, watch/event 事件回调. 然而, 挂载阶段还没有开始, $el 属性目前还不可见

· mounted: el 被新创建的 vm.$el 替换，并挂载到实例上去之后调用该钩子。如果 root 实例挂载了一个文档内元素，当 mounted 被调用时 vm.$el 也在文档内。

· activated: keep-alive 组件激活时调用

十六、active-class 是哪个组件的属性？

vue-router 模块的 router-link 组件。

十七、怎么定义 vue-router 的动态路由以及如何获取传过来的动态参数？

在 router 目录下的 index.js 文件中，对 path 属性加上/:id。

使用 router 对象的 params.id。

十八、vue-router 有哪几种导航钩子？（见博客）

三种，一种是全局导航钩子：router.beforeEach(to,from,next)，作用：跳转前进行判断拦截。

第二种：组件内的钩子；

第三种：单独路由独享组件

十九、生命周期相关面试题

总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

· 创建前/后： 在 beforeCreate 阶段，vue 实例的挂载元素 el 和数据对象 data 都为 undefined，还未初始化。在 created 阶段，vue 实例的数据对象 data 有了，el 还没有。

· 载入前/后：在 beforeMount 阶段，vue 实例的$el 和 data 都初始化了，但还是挂载之前为虚拟的 dom 节点，data.message 还未替换。在 mounted 阶段，vue 实例挂载完成，data.message 成功渲染。

· 更新前/后：当 data 变化时，会触发 beforeUpdate 和 updated 方法。

· 销毁前/后：在执行 destroy 方法后，对 data 的改变不会再触发周期函数，说明此时 vue 实例已经解除了事件监听以及和 dom 的绑定，但是 dom 结构依然存在

（1）、什么是 vue 生命周期

答： Vue 实例从创建到销毁的过程，就是生命周期。也就是从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、卸载等一系列过程，我们称这是 Vue 的生命周期。

（2）、vue 生命周期的作用是什么

答：它的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。

（3）、第一次页面加载会触发哪几个钩子

答：第一次页面加载时会触发 beforeCreate, created, beforeMount, mounted 这几个钩子

（4）、DOM 渲染在哪个周期中就已经完成

答：DOM 渲染在 mounted 中就已经完成了。

（5）、简单描述每个周期具体适合哪些场景

答：生命周期钩子的一些使用方法：

· beforecreate : 可以在这加个 loading 事件，在加载实例时触发

- created : 初始化完成时的事件写在这里，如在这结束 loading 事件，异步请求也适宜在这里调用
- mounted : 挂载元素，获取到 DOM 节点
- updated : 如果对数据统一处理，在这里写上相应函数
- beforeDestroy : 可以做一个确认停止事件的确认框
- nextTick : 更新数据后立即操作 dom

二十、说出至少 4 种 vue 当中的指令和它的用法？

v-if：判断是否隐藏；v-for：数据循环；v-bind:class：绑定一个属性；v-model：实现双向绑定

二十一、vue-loader 是什么？使用它的用途有哪些？

解析.vue 文件的一个加载器。（深入理解见 https://www.jb51.net/article/115480.htm）

用途：js 可以写 es6、style 样式可以 scss 或 less、template 可以加 jade 等

根据官网的定义，vue-loader 是 webpack 的一个 loader，用于处理 .vue 文件。

其次，使用 vue-cli 脚手架，作者已经配置好了基本的配置，开箱及用，你需要做的就是 npm install 安装下依赖，然后就可以开发业务代码了。当然，如果你想进阶，最好熟悉下 vue-loader 的具体配置，而不要依赖脚手架

二十二、scss 是什么？在 vue.cli 中的安装使用步骤是？有哪几大特性？

答：css 的预编译。（scss 是 sass 的一个升级版本，完全兼容 sass 之前的功能，又有了些新增能力，最主要的就是 sass 是靠缩进表示嵌套关系，scss 是花括号）

使用步骤：

第一步：先装 css-loader、node-loader、sass-loader 等加载器模块

第二步：在 build 目录找到 webpack.base.config.js，在那个 extends 属性中加一个拓展.scss

第三步：在同一个文件，配置一个 module 属性

第四步：然后在组件的 style 标签加上 lang 属性 ，例如：lang=”scss”

特性:

- 可以用变量，例如（$变量名称=值）；
- 可以用混合器，混入@mixin 可以传变量
- 可以嵌套

继承@extend 不可以传变量，相同样式直接继承，不会造成代码冗余；基类未被继承时，也会被编译成 css 代码

二十三、为什么使用 key?

当有相同标签名的元素切换时，需要通过 key 特性设置唯一的值来标记以让 Vue 区分它们，否则 Vue 为了效率只会替换相同标签内部的内容。

二十四、为什么避免 v-if 和 v-for 用在一起

当 Vue 处理指令时，v-for 比 v-if 具有更高的优先级，这意味着 v-if 将分别重复运行于每个 v-for 循环中。通过 v-if 移动到容器元素，不会再重复遍历列表中的每个值。取而代之的是，我们只检查它一次，且不会在 v-if 为否的时候运算 v-for。

二十五、VNode 是什么？虚拟 DOM 是什么？ 详情见 https://www.jb51.net/article/105221.htm

Vue 在 页面上渲染的节点，及其子节点称为“虚拟节点 (Virtual Node)”，简写为“VNode”。“虚拟 DOM”是由 Vue 组件树建立起来的整个 VNode 树的称呼。

2018 vue 前端面试题

1、active-class 是哪个组件的属性？嵌套路由怎么定义？

答：vue-router 模块的 router-link 组件。

嵌套路由顾名思义就是路由的多层嵌套。 一级路由里面使用 children 数组配置子路由，就是嵌套路由。

2、怎么定义 vue-router 的动态路由？怎么获取传过来的动态参数？

答：在 router 目录下的 index.js 文件中，对 path 属性加上/:id。 使用 router 对象的 params.id

4、scss 是什么？安装使用的步骤是？有哪几大特性？

答：预处理 css，把 css 当前函数编写，定义变量,嵌套。 先装 css-loader、node-loader、sass-loader 等加载器模块，在 webpack-base.config.js 配置文件中加多一个拓展:extenstion，再加多一个模块：module 里面 test、loader

5、mint-ui 是什么？怎么使用？说出至少三个组件使用方法？ https://www.cnblogs.com/stella1024/p/7771334.html

答：基于 vue 的前端组件库。npm 安装，然后 import 样式和 js，vue.use（mintUi）全局引入。在单个组件局部引入：import {Toast} from 'mint-ui'。组件一：Toast('登录成功')；组件二：mint-header；组件三：mint-swiper

6、v-model 是什么？怎么使用？ vue 中标签怎么绑定事件？

答：可以实现双向绑定，指令（v-class、v-for、v-if、v-show、v-on）。vue 的 model 层的 data 属性。绑定事件：<input @click=doLog() />

7、axios 是什么？怎么使用？描述使用它实现登录功能的流程？

答：请求后台资源的模块。npm install axios -S 装好，然后发送的是跨域，需在配置文件中 config/index.js 进行设置。后台如果是 Tp5 则定义一个资源路由。js 中使用 import 进来，然后.get 或.post。返回在.then 函数中如果成功，失败则是在.catch 函数中

Vue.js 1.0 我们常使用 vue-resource (官方 ajax 库), Vue 2.0 发布后作者宣告不再对 vue-resource 进行更新， 推荐我们使用 axios (基于 Promise 的 HTTP 请求客户端，可同时在浏览器和 node.js 中使用)

8、axios+tp5 进阶中，调用 axios.post('api/user')是进行的什么操作？axios.put('api/user/8')呢？

答：跨域，添加用户操作，更新操作。

9、什么是 RESTful API？怎么使用?

答：是一个 api 的标准，无状态请求。请求的路由地址是固定的，如果是 tp5 则先路由配置中把资源路由配置好。标准有：.post .put .delete

10、vuex 是什么？怎么使用？哪种功能场景使用它？

答：vue 框架中状态管理。在 main.js 引入 store，注入。新建了一个目录 store，….. export 。场景有：单页应用中，组件之间的状态。音乐播放、登录状态、加入购物车

11、mvvm 框架是什么？它和其它框架（jquery）的区别是什么？哪些场景适合？

答：一个 model+view+viewModel 框架，数据模型 model，viewModel 连接两个

区别：vue 数据驱动，通过数据来显示视图层而不是节点操作。

场景：数据操作比较多的场景，更加便捷

12、自定义指令（v-check、v-focus）的方法有哪些？它有哪些钩子函数？还有哪些钩子函数参数？

答：全局定义指令：在 vue 对象的 directive 方法里面有两个参数，一个是指令名称，另外一个是函数。组件内定义指令：directives

钩子函数：bind（绑定事件触发）、inserted(节点插入的时候触发)、update（组件内相关更新）

钩子函数参数：el、binding

14、vue-router 是什么？它有哪些组件？

答：vue 用来写路由一个插件。router-link、router-view

15、导航钩子有哪些？它们有哪些参数？

答：导航钩子有：a/全局钩子和组件内独享的钩子。b/beforeRouteEnter、afterEnter、beforeRouterUpdate、beforeRouteLeave

参数：有 to（去的那个路由）、from（离开的路由）、next（一定要用这个函数才能去到下一个路由，如果不用就拦截）最常用就这几种

16、Vue 的双向数据绑定原理是什么？

答：vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

具体步骤：

第一步：需要 observe 的数据对象进行递归遍历，包括子属性对象的属性，都加上 setter 和 getter

这样的话，给这个对象的某个值赋值，就会触发 setter，那么就能监听到了数据变化

第二步：compile 解析模板指令，将模板中的变量替换成数据，然后初始化渲染页面视图，并将每个指令对应的节点绑定更新函数，添加监听数据的订阅者，一旦数据有变动，收到通知，更新视图

第三步：Watcher 订阅者是 Observer 和 Compile 之间通信的桥梁，主要做的事情是:

1、在自身实例化时往属性订阅器(dep)里面添加自己

2、自身必须有一个 update()方法

3、待属性变动 dep.notice()通知时，能调用自身的 update()方法，并触发 Compile 中绑定的回调，则功成身退。

第四步：MVVM 作为数据绑定的入口，整合 Observer、Compile 和 Watcher 三者，通过 Observer 来监听自己的 model 数据变化，通过 Compile 来解析编译模板指令，最终利用 Watcher 搭起 Observer 和 Compile 之间的通信桥梁，达到数据变化 -> 视图更新；视图交互变化(input) -> 数据 model 变更的双向绑定效果。

ps：16 题答案同样适合”**vue data 是怎么实现的？**”此面试题。

18、请说下封装 vue 组件的过程？

答：首先，组件可以提升整个项目的开发效率。能够把页面抽象成多个相对独立的模块，解决了我们传统项目开发：**效率低**、**难维护**、**复用性**等问题。

然后，使用 Vue.extend 方法创建一个组件，然后使用 Vue.component 方法注册组件。子组件需要数据，可以在 props 中接受定义。而子组件修改好数据后，想把数据传递给父组件。可以采用 emit 方法。

19、你是怎么认识 vuex 的？

答：vuex 可以理解为一种开发模式或框架。比如 PHP 有 thinkphp，java 有 spring 等。

通过状态（数据源）集中管理驱动组件的变化（好比 spring 的 IOC 容器对 bean 进行集中管理）。

应用级的状态集中放在 store 中； 改变状态的方式是提交 mutations，这是个同步的事物； 异步逻辑应该封装在 action 中。

21、请说出 vue.cli 项目中 src 目录每个文件夹和文件的用法？

答：assets 文件夹是放静态资源；components 是放组件；router 是定义路由相关的配置;view 视图；app.vue 是一个应用主组件；main.js 是入口文件

22、vue.cli 中怎样使用自定义的组件？有遇到过哪些问题吗？

答：第一步：在 components 目录新建你的组件文件（smithButton.vue），script 一定要 export default {

第二步：在需要用的页面（组件）中导入：import smithButton from ‘../components/smithButton.vue’

第三步：注入到 vue 的子组件的 components 属性上面,components:{smithButton}

第四步：在 template 视图 view 中使用，<smith-button> </smith-button>

问题有：smithButton 命名，使用的时候则 smith-button。

23、聊聊你对 Vue.js 的 template 编译的理解？ https://www.jianshu.com/p/e1669afa30b8

答：简而言之，就是先转化成 AST 树，再得到的 render 函数返回 VNode（Vue 的虚拟 DOM 节点）

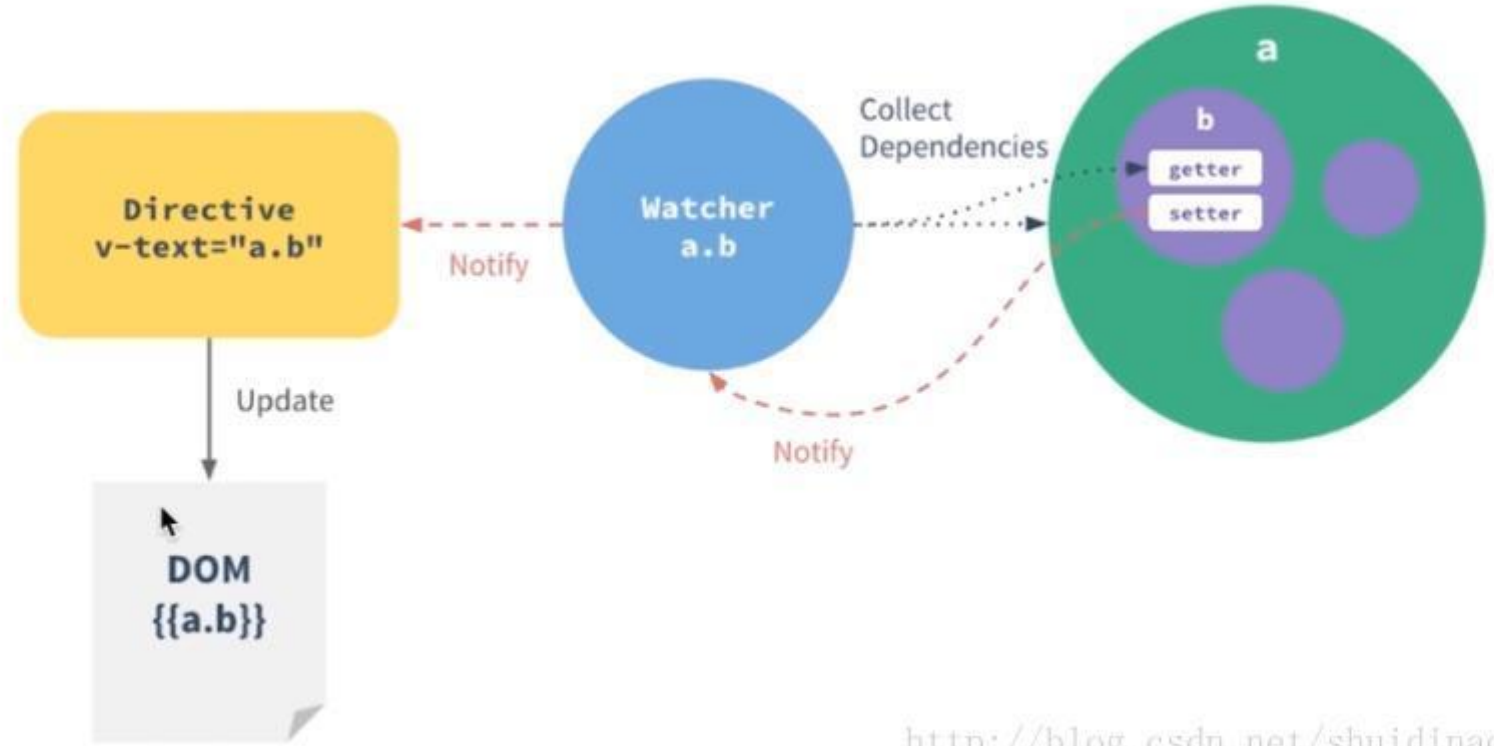
详情步骤：

首先，通过 compile 编译器把 template 编译成 AST 语法树（abstract syntax tree 抽象语法树 即 源代码的抽象语法结构的树状表现形式），compile 是 createCompiler 的返回值，createCompiler 是用以创建编译器的。另外 compile 还负责合并 option。

然后，AST 会经过 generate（将 AST 语法树转化成 render funtion 字符串的过程）得到 render 函数，render 的返回值是 VNode，VNode 是 Vue 的虚拟 DOM 节点，里面有（标签名、子节点、文本等等）

挑战一下：

1、vue 响应式原理？https://blog.csdn.net/GitChat/article/details/78516752 看这篇 https://blog.csdn.net/aaa333qwe/article/details/80093810



有一个数据 a.b,在 vue 对象实例化过程中，会给 a,b 通过 ES5 的 defineProperty()方法，添加 getter 和 setter 方法，同时 vue.js 会对模板做编译，解析生成一个指令对象，比如 v-text 指令，每个指令对象都会关联一个 watcher，当对指令对象求值时，就会触发 getter，并将依赖收集到 watcher 中；当再次改变 a.b 值时，就会触发 setter 方法，会通知到对应关联的 watcher,watcher 则再次对 a.b 求值，计算对比新旧值，当值改变时，watcher 会通知到指令，调用指令的 update 方法，由于指令是对 dom 的封装，所以会调用原生 dom 的方法，去更新视图。

2、vue-router 实现原理？

https://www.cnblogs.com/yanze/p/7644631.html https://segmentfault.com/a/1190000015123061

3、为什么要选 vue？与其它框架对比的优势和劣势？

(1)vue.js 更轻量，gzip 后大小只有 20K+,React gzip 后大小为 44k，Angular gzip 后大小有 56k，所以对于移动端来说，vue.js 更适合；

(2)vue.js**更易上手，学习曲线平稳，而 Angular 入门较难，概念较多(比如依赖注入)，它使用 java 写的**，很多思想沿用了后台的技术，react 需学习较多东西，附带 react 全家桶，

(3)吸收两家之长，借用了 angular 的指令(比如 v-show,v-hide 对应 angular 的 ng-show,ng-hide)和 react 的组件化(将一个页面抽成一个组件，组件具有完整的生命周期)

(4)vue 还有自己的特点，比如计算属性

4、vue 如何实现父子组件通信，以及非父子组件通信？

http://www.cnblogs.com/sichaoyun/p/6690322.html#4021894

5、vuejs 与 angularjs 以及 react 的区别？

6、vuex 是用来做什么的？

7、vue 源码结构

组件的设计原则

(1)页面上每个独立的**可视/可交互区域**视为一个组件(比如页面的头部，尾部，可复用的区块)

(2)每个组件对应一个工程目录，组件所需要的**各种资源**在这个目录下就近维护(组件的就近维护思想体现了前端的**工程化**思想，为前端开发提供了很好的分治策略，在 vue.js 中，通过.vue 文件将组件依赖的模板，js，样式写在一个文件中)

(每个开发者清楚开发维护的功能单元，它的代码必然存在在对应的组件目录中，在该目录下，可以找到功能单元所有的内部逻辑)

(3)页面不过是**组件的容器**，组件可以嵌套自由组合成完整的页面

Virtual DOM 算法，简单总结下包括几个步骤： https://www.cnblogs.com/aaronjs/p/7274965.html

1. 用 JS 对象描述出 DOM 树的结构，然后在初始化构建中，用这个描述树去构建真正的 DOM，并实际展现到页面中
2. 当有数据状态变更时，重新构建一个新的 JS 的 DOM 树，通过新旧对比 DOM 数的变化 diff，并记录两棵树差异
3. 把步骤 2 中对应的差异通过步骤 1 重新构建真正的 DOM，并重新渲染到页面中，这样整个虚拟 DOM 的操作就完成了，视图也就更新了

Vue 面试题

一：什么是 MVVM

MVVM 是是 Model-View-ViewModel 的缩写，Model 代表数据模型，定义数据操作的业务逻辑，View 代表视图层，负责将数据模型渲染到页面上，ViewModel 通过双向绑定把 View 和 Model 进行同步交互，不需要手动操作 DOM 的一种设计思想。

二：MVVM 和 MVC 区别？和其他框架(jquery)区别？那些场景适用？

MVVM 和 MVC 都是一种设计思想，主要就是 MVC 中的 Controller 演变成 ViewModel,，MVVM 主要通过数据来显示视图层而不是操作节点，解决了 MVC 中大量的 DOM 操作使页面渲染性能降低，加载速度慢，影响用户体验问题。主要用于数据操作比较多的场景。

三：Vue 的优缺点是什么

优点：低耦合，可重用性，独立开发，可测试，渐进式

缺点：不利于 SEO，社区维护力度不强，相比还不够成熟

三：组件之间传值

父向子传值：属性传值，父组件通过给子组件标签上定义属性，子组件通过 props 方法接收数据；

子向父传值：事件传值，子组件通过$emit(‘自定义事件名’，值)，父组件通过子组件上的@自定义事件名=“函数”接收
非父子组件传值：全局定义 bus，var bus=new Vue()；发送者， bus.$emit(‘自定义名’，值) ；接受者，bus.$on(‘自定义名’，(值)=>{})
四：路由之间传参
路由字典中：routes={path:’/detail/:id’,component:Detail}
标签中：<router-link :to=”/detail/’+item.id ”>
Js 中：this.$route.params.id
五：axios 的特点和使用
特点：基于 promise 的 Http 库，支持 promise 的所有 API，用来请求和响应数据的，而且对响应回来的数据自动转化为 json 类型，安全性较高，客户端支持防御 XRSF(跨站请求伪造)，默认不携带 cookie;
使用：下载包后引入 import axios from ‘axios’ , 让其携带 cookie ，axios.defaults.withCredentials=true, 然后添加到 prototype 中，Vue.prototype.$axios=axios ,组建中不用引入直接使用 this.$axios.get(url,{params:{id:1}})。
六：Vuex 是什么？怎么使用？用于哪些场景？
Vuex 是框架中状态管理；新建目录 store...export,然后 main.js 引入 store 再注入到 vue 实例中；用于购物车，登录状态，单页应用等。
七：Vuex 有哪几种属性？
五种：state，action，mutation，getter，module
State：数据源存放地，数据是响应式的
Action：逻辑处理，提交的是 mutation,包含任意异步操作
Mutation：修改 state 里的公共数据
Getter：相当于计算属性，可以复用，可缓存
Module：模块化
八：Vuex 取值

```
This.$store.state.city
This.$store.commit( ‘getData’ )
this.$store.dispath( ‘getData’ )
This.$store.getters.getData
```

九：不使用 vuex 会带来什么问题？
可维护性下降，可读性下降，增加耦合
十：v-show 和 v-if 指令的共同点和不同点
V-show 指令是通过修改元素的 display 的 css 样式使其显示隐藏
V-if 指令是销毁和重建 DOM 达到让元素显示隐藏
十一：如何让 css 只在当前组件中起作用?
将当前组件的<style>修改为<style scoped>
十二：<keep-alive></keep-alive>的作用是什么，如何使用？
包裹动态组件时，会缓存不活动的组件实例，主要用于保留组件状态或避免重新渲染；
使用：简单页面时

缓存： <keep-alive include="组件名"></keep-alive>
不缓存：<keep-alive exclude="组件名"></keep-alive>
使用：复杂项目时
路由字典中定义{path:'/detail',meta:{keepAlive:false/true}} 是否缓存
根目录中：
<keep-alive><router-view v-if="$route.meta.keepAlive"></router-view></keep-alive>
<keep-alive><router-view v-if=" ! $route.meta.keepAlive"></router-view></keep-alive>
十三：Vue 数据双向绑定原理
Vue 数据双向绑定是通过数据劫持结合发布者-订阅者模式方式来实现的，语法主要有{{}}和 v-model。首先用递归方法遍历所有的属性值，再用 Object.defineProperty()给属性绑定 getter 和 setter 方法添加一个 observe(val)监听器对数据进行劫持监听;然后创建一个订阅器来在 getter 里收集订阅者并创建和绑定 watcher，如果数据变化，订阅者就会执行自己对应的更新函数;watcher 就是在自身实例化的时候往订阅器里添加自己，自身必须有一个 update 的数据，是监听器和模板渲染的通信桥梁;最后创建解析模板指令 compile，替换数据，初始化视图。最终 observer 来监听自己的 model 数据变化通过 compile 解析编译模板指令，利用 watcher 搭起 observer 和 compile 之间的通信桥梁，达到数据变化->视图更新双向绑定效果。
十四：Vue 生命周期
vue 生命周期就是从开始创建，初始化数据，编译模板，挂载 DOM，渲染->更新->渲染，销毁等一系列过程。生命周期钩子如下：
组件实例周期：
BeforeCreate：实例初始化后，无法访问方法和数据；
Created：实例创建完成，可访问数据和方法，注意，假如有某些数据必须获取才允许进入页面的话，并不适合；
beforeMonut：挂载 DOM 之前
Mounted :el 被新创建的 vm.$el 替换，可获取 dom，改变 data，注意，beforeRouterEnter 的 next 的钩子比 mountend 触发靠后；
beforeUpdate：数据更新时调用，发生在虚拟 DOM 重新渲染前；
Updated：数据更改后，可以执行依赖于 DOM 的操作，注意，应该避免在此期间更改状态，可能会导致更新无限循环；
beforeDestroy：实例销毁之前，这一步还可以用 this 获取实例，一般在这一步做重置操作，比如清定时器监听 dom 事件；
Destroyed：实例销毁后，会解除绑定，移除监听。
十五：路由钩子
全局路由钩子：
Router.beforeEach((to,from,next)=>{... next()})
注意：一定要调用 next(),否则页面卡在那，一般用于对路由跳转前进行拦截，参数：
To：即将要进入的目标路由对象 From：当前导航正要离开的路由
Next()：跳转下一个页面 next('/path')：跳转指定路由
Next(false)：返回原来页面 next((vm)=>{})：且在 beforeRouterEnter 用
比如根据登录状态跳转页面判断(to.name->name 是路由名)
Router.beforeEach(function(to,from,next){if(to.name=='login'){..}next();})
Router.afterEach((to,from)=>{}) 跳转后调用没有 next 方法
组件路有钩子：

beforeRouteEnter(to,from,next){next(vm=>{...})} 路由跳转时
注意：此钩子在 beforeCreate 之前执行，但是 next 在组件 mounted 周期之后,无法直接调用 this 访问组件实例，可用 next 访问 vm 实例，修改数据；
beforeRouteLeave(to,from,next){...next()} 离开路由时
注意：可以直接访问 this,next 不可回调
beforeRouteUpdate 路由切换时
十六：指令周期
Bind：一次初始化调用　　inserted：被绑定元素插入父节点调用
Update：模板更新调用　　unbind：指令与元素解绑时
Vue.nextTick：在 dom 更新后执行，一般用于 dom 操作
Vue.$nextTick：一直到真实的 dom 渲染结束后执行
Ex:created(){this.$nextTick(()=>{...})}
十七：生命周期的作用是什么？
它的生命周期有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。
十八：第一次加载会触发哪几个钩子？
会触发 beforeCreate , created ,beforeMount ,mounted
十九：说出至少 4 种 vue 当中的指令和用法？
V-if：判断是否隐藏　　v-for：数据循环　　v-bind：绑定属性
v-model：双向绑定　　v-is：动态组件特殊特性 v-on：事件绑定
二十：vue-loader 是什么？用途有哪些？
是解析 vue 文件的一个加载器，用途是 js 可以写 es6，style 样式可以 scss 或 less，template 可以加 jade
二十一：active-class 是那个组件属性？
Vue-router 模块的 router-link 组件,设置激活时样式
二十二：vue 中使用插件的步骤是什么？
Inport ... from ... 引入插件，Vue.use(...)全局注册
二十三：为什么使用 key？
当有相同标签名和元素切换时，需要通过 key 特性设置唯一的值来标记让 vue 区分它们，否则 vue 为了效率只会替换相同标签内部的内容
二十三：为什么避免 v-if 和 v-for 用在一起？
当 vue 处理指令时，v-for 比 v-if 具有更高的优先级，通过 v-if 移动到容器的元素，不会在重复遍历列表中的每个值，取而代之的是，我们只检查它一次，且不会 v-if 为否的时候运算 v-for
二十四：VNode 是什么？虚拟 DOM 是什么？
Vue 在页面上渲染的节点，及其子节点称为虚拟节点，简称 VNode；虚拟 DOM 时由组件树建立起来的整个 VNode 树的称呼
二十五：scss 是什么？有哪些特性？怎么使用？
是 css 的预编译，特新有可以用变量($变量名=值)，可以用混合器()，可以嵌套，可以继承，可以运算，安装先装 css-loader，node-loader，sass-loader，在 webpack.base 配置，style 标签上加 lang=”scss”
二十五：Vue router 如何实现跳转

<router-link></router-link> router.push('/') router.go(0)
二十六：vue router 跳转和 location.href 有什么区别？
Router 是 hash 改变；location.href 是页面跳转，刷新页面
二十七：v-model 原理
```
<input v-model="sth">
```
==相当于通过 oninput(用户输入时触发)把表单值给到变量
```
<input v-bind:value="sth" v-on:input="sth=$event.target.value">
```
二十八：vue 的 template 的理解
通过 compile 编译 template 生成 AST 语法树，AST 语法树经过 generate 转化为 render function 字符串后返回虚拟 DOM 节点(Vnode)的过程
二十九：vue 和 react 区别
相同点：
都鼓励组件化，都有'props'的概念，都有自己的构建工具，Reat 与 Vue 只有框架的骨架，其他的功能如路由、状态管理等是框架分离的组件。
不同点：
React：数据流单向，语法—JSX，在 React 中你需要使用 setState()方法去更新状态
Vue：数据双向绑定，语法--HTML，state 对象并不是必须的，数据由 data 属性在 Vue 对象中进行管理。适用于小型应用，但对于对于大型应用而言不太适合。
三十：单页面和多页面的区别
单页面：
整个项目中只有一个完整的 HTML 页面，其它"页面"只是一段 HTML 片断而已
优：请求少
缺：不利于搜索引擎优化
页面跳转本质：把当前 DOM 树中某个 DIV 删除，下载并挂载另一个 div 片断
多页面：
项目中有多个独立的完整的 HTML 页面
缺：请求次数多，效率低
页面跳转本质：
删除旧的 DOM 树，重新下载新的 DOM 树
三十一：Vue 的 SPA 如何优化加载速度
减少入口文件体积，静态资源本地缓存，开启 Gzip 压缩，使用 SSR，nuxt.js
三十二：Axios 发送 post 请求
```
Import qs from  'qs'
Var data=qs.stringify({
Number : ' 1'
})
Axios.post(url,data).then()
```
VUE 如何自定义属性

全局自定义：

```
Vue.directive( ‘focus’ ,{
                Inserted:function(el){
                                el.focus()    //聚焦函数
}
})
```

三十三：组件自定义

```
directive{
                inserted:function(el){
                                el.focus()
}
}
```

三十四：Vue 和 vuex 有什么区别

Vue 是框架，vuex 是插件，vuex 是专门为 vue 应用程序开发的状态管理模式

三十五：.Vuex 中 actions 和 mutations 的区别

Mutations 的更改是同步更改，用于用户执行直接数据更改，this.$store.commit(‘名’)触发

Actions 的更改是异步操作，用于需要与后端交互的数据更改,this.$store.dispath(“名”)触发

注意：

1)：定义 actions 方法创建一个更改函数时，这个函数必须携带一个 context 参数，用于触发 mutations 方法，context.commit(‘修改函数名’ , ’异步请求值’)；

2)：mutations 第一个参数必须传入 state，第二个参数是新值

vue 前端面试题知识点整理

1. 说一下 Vue 的双向绑定数据的原理

vue 实现数据双向绑定主要是：采用数据劫持结合发布者-订阅者模式的方式，通过 Object.defineProperty() 来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调

2. 解释单向数据流和双向数据绑定

单向数据流： 顾名思义，数据流是单向的。数据流动方向可以跟踪，流动单一，追查问题的时候可以更快捷。缺点就是写起来不太方便。要使 UI 发生变更就必须创建各种 action 来维护对应的 state

双向数据绑定：数据之间是相通的，将数据变更的操作隐藏在框架内部。优点是在表单交互较多的场景下，会简化大量与业务无关的代码。缺点就是无法追踪局部状态的变化，增加了出错时 debug 的难度

3. Vue 如何去除 url 中的

vue-router 默认使用 hash 模式，所以在路由加载的时候，项目中的 url 会自带 #。如果不想使用 #， 可以使用 vue-router 的另一种模式 history

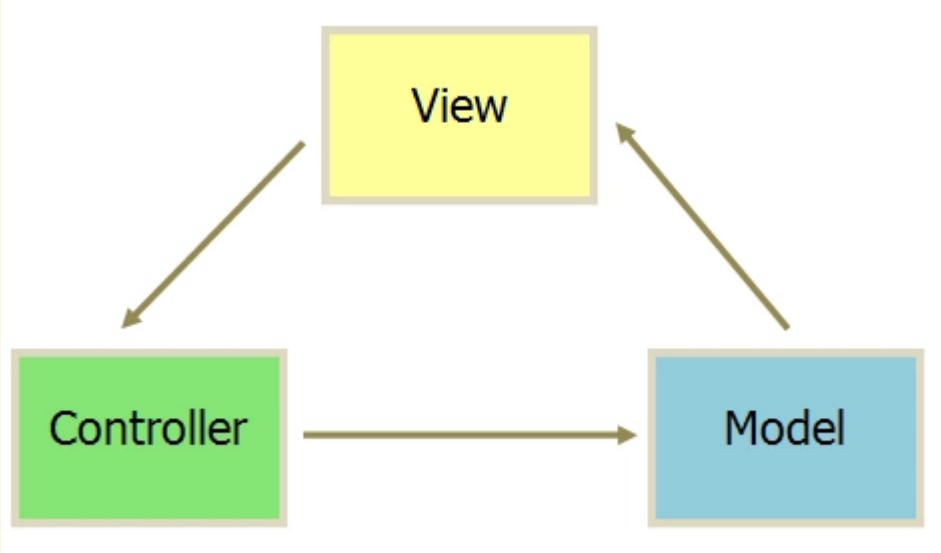
```
new Router({
  mode: ’history’,
  routes: [ ]
```

```
})
```

需要注意的是，当我们启用 history 模式的时候，由于我们的项目是一个单页面应用，所以在路由跳转的时候，就会出现访问不到静态资源而出现 404 的情况，这时候就需要服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个 index.html 页面

4. 对 MVC、MVVM 的理解

MVC

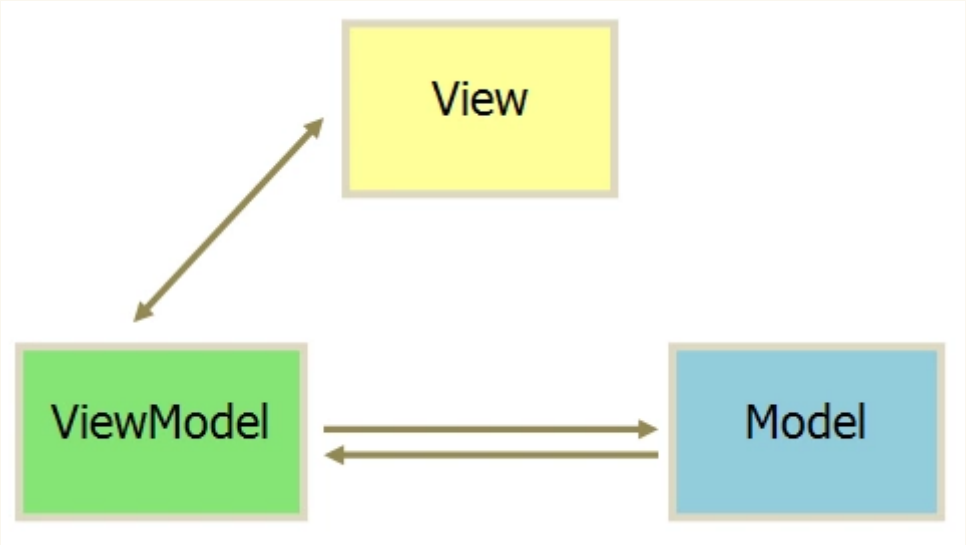


特点：

1. View 传送指令到 Controller
2. Controller 完成业务逻辑后，要求 Model 改变状态
3. Model 将新的数据发送到 View，用户得到反馈

所有通信都是单向的

MVVM



特点：

1. 各部分之间的通信，都是双向的
2. 采用双向绑定：View 的变动，自动反映在 ViewModel，反之亦然

具体请移步 这里

5. 介绍虚拟 DOM

参考这里

6. vue 生命周期的理解

vue 实例有一个完整的生命周期，生命周期也就是指一个实例从开始创建到销毁的这个过程

1、beforeCreated() 在实例创建之间执行，数据未加载状态
2、created() 在实例创建、数据加载后，能初始化数据，dom 渲染之前执行
3、beforeMount() 虚拟 dom 已创建完成，在数据渲染前最后一次更改数据
4、mounted() 页面、数据渲染完成，真实 dom 挂载完成
5、beforeUpadate() 重新渲染之前触发
6、updated() 数据已经更改完成，dom 也重新 render 完成,更改数据会陷入死循环
7、beforeDestory() 和 destoryed() 前者是销毁前执行（实例仍然完全可用），后者则是销毁后执行

7. 组件通信

父组件向子组件通信
子组件通过 props 属性，绑定父组件数据，实现双方通信
子组件向父组件通信
将父组件的事件在子组件中通过 $emit 触发

非父子组件、兄弟组件之间的数据传递

```
/*新建一个 Vue 实例作为中央事件总嫌*/
let event = new Vue();

/*监听事件*/
event.$on('eventName', (val) => {
    //......do something
});
/*触发事件*/
event.$emit('eventName', 'this is a message.')
```

Vuex 数据管理

8. vue-router 路由实现

路由就是用来跟后端服务器进行交互的一种方式，通过不同的路径，来请求不同的资源，请求不同的页面是路由的其中一种功能

参考 这里

9. v-if 和 v-show 区别

使用了 v-if 的时候，如果值为 false ，那么页面将不会有这个 html 标签生成。

v-show 则是不管值为 true 还是 false ，html 元素都会存在，只是 CSS 中的 display 显示或隐藏

10. $route 和$router 的区别

$router 为 VueRouter 实例，想要导航到不同 URL，则使用 $router.push 方法

$route 为当前 router 跳转对象里面可以获取 name 、 path 、 query 、 params 等

11. NextTick 是做什么的

$nextTick 是在下次 DOM 更新循环结束之后执行延迟回调，在修改数据之后使用 $nextTick，则可以在回调中获取更新后的 DOM

具体可参考官方文档 深入响应式原理

12. Vue 组件 data 为什么必须是函数

因为 js 本身的特性带来的，如果 data 是一个对象，那么由于对象本身属于引用类型，当我们修改其中的一个属性时，会影响到所有 Vue 实例的数据。如果将 data 作为一个函数返回一个对象，那么每一个实例的 data 属性都是独立的，不会相互影响了

13. 计算属性 computed 和事件 methods 有什么区别

我们可以将同一函数定义为一个 method 或者一个计算属性。对于最终的结果，两种方式是相同的

不同点：

computed: 计算属性是基于它们的依赖进行缓存的,只有在它的相关依赖发生改变时才会重新求值

对于 method ， 只要发生重新渲染，method 调用总会执行该函数

14. 对比 jQuery ， Vue 有什么不同

jQuery 专注视图层，通过操作 DOM 去实现页面的一些逻辑渲染； Vue 专注于数据层，通过数据的双向绑定，最终表现在 DOM 层面，减少了 DOM 操作

Vue 使用了组件化思想，使得项目子集职责清晰，提高了开发效率，方便重复利用，便于协同开发

15. Vue 中怎么自定义指令

全局注册

```
// 注册一个全局自定义指令 `v-focus`
Vue.directive('focus', {
  // 当被绑定的元素插入到 DOM 中时……
  inserted: function (el) {
    // 聚焦元素
    el.focus()
  }
})
```

局部注册

```
directives: {
  focus: {
    // 指令的定义
    inserted: function (el) {
      el.focus()
    }
  }
}
```

参考 官方文档-自定义指令

16. Vue 中怎么自定义过滤器

可以用全局方法 Vue.filter() 注册一个自定义过滤器，它接收两个参数：过滤器 ID 和过滤器函数。过滤器函数以值为参数，返回转换后的值

```
Vue.filter('reverse', function (value) {
    return value.split('').reverse().join('')
})
<!-- 'abc' => 'cba' -->
<span v-text="message | reverse"></span>
```

过滤器也同样接受全局注册和局部注册

17. 对 keep-alive 的了解

keep-alive 是 Vue 内置的一个组件，可以使被包含的组件保留状态，或避免重新渲染

```
<keep-alive>
  <component>
    <!-- 该组件将被缓存！ -->
  </component>
```

</keep-alive>
可以使用 API 提供的 props，实现组件的动态缓存
具体参考 官方 API

18. Vue 中 key 的作用

key 的特殊属性主要用在 Vue 的虚拟 DOM 算法，在新旧 nodes 对比时辨识 VNodes。如果不使用 key，Vue 会使用一种最大限度减少动态元素并且尽可能的尝试修复/再利用相同类型元素的算法。使用 key，它会基于 key 的变化重新排列元素顺序，并且会移除 key 不存在的元素。
有相同父元素的子元素必须有独特的 key。重复的 key 会造成渲染错误
具体参考 官方 API

19. Vue 的核心是什么

数据驱动 组件系统

20. vue 等单页面应用的优缺点

优点：

- 良好的交互体验
- 良好的前后端工作分离模式
- 减轻服务器压力

缺点：

- SEO 难度较高
- 前进、后退管理
- 初次加载耗时多

vue 面试题 2019

vue 核心知识点

1、对于 Vue 是一套渐进式框架的理解
渐进式代表的含义是：主张最少。
Vue 可能有些方面是不如 React，不如 Angular，但它是渐进的，没有强主张，你可以在原有大系统的上面，把一两个组件改用它实现，当 jQuery 用；也可以整个用它全家桶开发，当 Angular 用；还可以用它的视图，搭配你自己设计的整个下层用。你可以在底层数据逻辑的地方用 OO 和设计模式的那套理念，也可以函数式，都可以，它只是个轻量视图而已，只做了自己该做的事，没有做不该做的事，仅此而已。
渐进式的含义，我的理解是：没有多做职责之外的事。
2、vue.js 的两个核心是什么？
数据驱动和组件化
3、请问 v-if 和 v-show 有什么区别？
相同点： 两者都是在判断 DOM 节点是否要显示
不同点：

a.实现方式： v-if 是根据后面数据的真假值判断直接从 Dom 树上删除或重建元素节点。 v-show 只是在修改元素的 css 样式，也就是 display 的属性值，元素始终在 Dom 树上。
b.编译过程：v-if 切换有一个局部编译/卸载的过程，切换过程中合适地销毁和重建内部的事件监听和子组件； v-show 只是简单的基于 css 切换；
c.编译条件：v-if 是惰性的，如果初始条件为假，则什么也不做；只有在条件第一次变为真时才开始局部编译； v-show 是在任何条件下（首次条件是否为真）都被编译，然后被缓存，而且 DOM 元素始终被保留；
d.性能消耗：v-if 有更高的切换消耗，不适合做频繁的切换； v-show 有更高的初始渲染消耗，适合做频繁的额切换；

4、vue 常用的修饰符

a、按键修饰符

如：.delete（捕获"删除"和"退格"键） 用法上和事件修饰符一样，挂载在 v-on:后面，语法：v-on:keyup.xxx=’yyy’ <input class = ’aaa’ v-model="inputValue" @keyup.delete="onKey"/>

b、系统修饰符

可以用如下修饰符来实现仅在按下相应按键时才触发鼠标或键盘事件的监听器

```
1、.ctrl
2、.alt
3、.shift
4、.meta
```

c、鼠标按钮修饰符

```
1、.left
2、.right
3、.middle
```

这些修饰符会限制处理函数仅响应特定的鼠标按钮。如：<button @click.middle ="onClick">A</button> 鼠标滚轮单击触发 Click 默认是鼠标左键单击

d、其他修饰符

- .lazy
- 在默认情况下，v-model 在每次 input 事件触发后将输入框的值与数据进行同步 ，我们可以添加 lazy 修饰符，从而转变为使用 change 事件进行同步：

```
<!-- 在“change”时而非“input”时更新 -->
<input v-model.lazy="msg" >
```

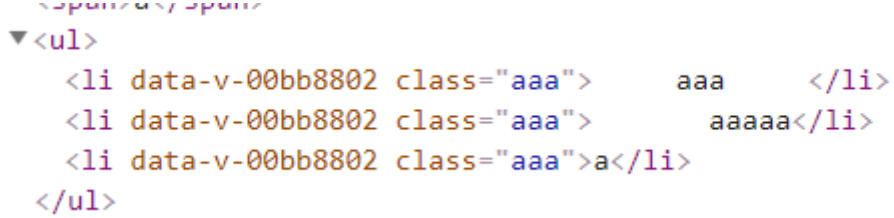
- .number
- 如果想自动将用户的输入值转为数值类型，可以给 v-model 添加 .number 修饰符：

```
<input v-model.number="age" type="number">
```

这通常很有用，因为即使在 type="number"时，HTML 输入元素的值也总会返回字符串。如果这个值无法被 parseFloat()解析，则会返回原始的值。

- .trim
- 如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符：

```
<input v-model.trim="msg">
```



同样前面都有空格加上.trim 后 将前后空格都去掉了

5、v-on 可以监听多个方法吗？

可以

6、vue 中 key 值的作用

使用 key 来给每个节点做一个唯一标识

key 的作用主要是为了高效的更新虚拟 DOM。另外 vue 中在使用相同标签名元素的过渡切换时，也会使用到 key 属性，其目的也是为了让 vue 可以区分它们，否则 vue 只会替换其内部属性而不会触发过渡效果。

7、Vue 组件中 data 为什么必须是函数

在 new Vue() 中，data 是可以作为一个对象进行操作的，然而在 component 中，data 只能以函数的形式存在，不能直接将对象赋值给它。

当 data 选项是一个函数的时候，每个实例可以维护一份被返回对象的独立的拷贝，这样各个实例中的 data 不会相互影响，是独立的

8、v-for 与 v-if 的优先级

v-for 的优先级比 v-if 高

9、vue 中子组件调用父组件的方法

参考： https://www.cnblogs.com/jin-zhe/p/9523782.html

第一种方法是直接在子组件中通过 this.$parent.event 来调用父组件的方法

第二种方法是在子组件里用$emit 向父组件触发一个事件，父组件监听这个事件就行了。

第三种是父组件把方法传入子组件中，在子组件里直接调用这个方法

10、vue 生命周期钩子函数有哪些？

参考： https://www.jianshu.com/p/8b7373362b4c

总共分为 8 个阶段创建前/后，载入前/后，更新前/后，销毁前/后。

创建前/后

在 beforeCreated 阶段，vue 实例的挂载元素$el 和数据对象 data 都为 undefined，还未初始化。

在 created 阶段，vue 实例的数据对象 data 有了，$el 还没有。

载入前/后

在 beforeMount 阶段，vue 实例的$el 和 data 都初始化了，但还是挂载之前为虚拟的 dom 节点，data.message 还未替换。

在 mounted 阶段，vue 实例挂载完成，data.message 成功渲染。

更新前/后

当 data 变化时，会触发 beforeUpdate 和 updated 方法。

销毁前/后

在执行 destroy 方法后，对 data 的改变不会再触发周期函数，说明此时 vue 实例已经解除了事件监听以及和 dom 的绑定，但是 dom 结构依然存在

11、说出至少 4 种 vue 当中的指令和它的用法

v-if(判断是否隐藏)、v-for(把数据遍历出来)、v-bind(绑定属性)、v-model(实现双向绑定)

12、vue 的双向绑定的原理是什么

参考：https://www.cnblogs.com/libin-1/p/6893712.html

vue 数据双向绑定是通过数据劫持结合发布者-订阅者模式的方式来实现的。

Vue 面试经常会被问到的面试题

一、对于 MVVM 的理解

MVVM 是 Model-View-ViewModel 的缩写。

- **Model** 代表数据模型，也可以在 Model 中定义数据修改和操作的业务逻辑。
- **View** 代表视图模型，它负责将数据模型转化成 UI 展现出来。View 层不负责处理状态，View 层做的是 数据绑定的声明、 指令的声明、 事件绑定的声明。
- **ViewModel** 监听数据的改变和控制视图行为、处理用户交互，它就是一个同步 View 和 Model 的对象，连接 Model 和 View。

在 MVVM 架构下，View 和 Model 之间并没有直接的联系，而是通过 ViewModel 进行交互，Model 和 View 之间的交互是双向的， 因此 View 数据的变化会同步到 Model 中，而 Model 数据的变化也会立即反应到 View 上。

ViewModel 通过双向数据绑定把 View 层和 Model 层连接起来，而 View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需要关注业务逻辑，不需要手动操作 DOM，不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理

MVVM 优缺点

- 优点：
- 1）分离视图（View）和模型（Model）,降低代码耦合，提高视图或者逻辑的重用性
- 2）提高可测试性： ViewModel 的存在可以帮助开发者更好地编写测试代码
- 3）自动更新 dom: 利用双向绑定,数据更新后视图自动更新,让开发者从繁琐的 dom 操作中解放
- 缺点：
- 1）Bug 很难被调试：数据绑定使得一个位置的 Bug 被快速传递到别的位置，要定位原始出问题的地方就变得不那么容易了。另外，数据绑定的声明是指令式地写在 View 的模版当中的，这些内容是没办法去打断点 debug 的
- 2）一个大的模块中 model 也会很大，虽然使用方便也并保证了数据的一致性，当时长期持有，不释放内存就造成了花费更多的内存
- 3）对于大型的图形应用程序，视图状态较多，ViewModel 的构建和维护的成本都会比较高

vue 对 css 的操作可以通过绑定 class 或者绑定 style。

二、vue 框架与 jQuery 类库的区别

- Vue 直接操作视图层，它通过 Vue 对象将数据和 View 完全分离开来了。对数据进行操作不需要引用相应的 DOM 节点，只需要关注逻辑，完全实现了视图层和逻辑层的解耦；
- Jquery 的操作是基于 DOM 节点的操作，jQuery 是使用选择器（$）选取 DOM 对象，对其进行赋值、取值、事件绑定等操作，其实和原生的 js 的区别只在于可以更方便的选取和操作 DOM 对象，而数据和界面是在一起的。它的优势在于良好的封装和兼容，使调用简单方便。

三、vue 的生命周期

Vue 实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载 Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。vue 的生命周期中有多个事件钩子，让我们在控制整个 Vue 实例的过程时更容易形成好的逻辑。**它可以总共分为 8 个阶段：创建前/后，载入前/后,更新前/后,销毁前/销毁后。**

第一次页面加载会触发 beforeCreate, created, beforeMount, mounted 这几个钩子函数；**DOM 渲染**在 mounted 中就已经完成了。官方实例的**异步请求**是在 mounted 生命周期中调用的，而实际上也可以在 created 生命周期中调用。

beforeCreate(创建前)：数据观测和初始化事件还未开始

created(创建后)：完成数据观测，属性和方法的运算，初始化事件，el 属性还没有显示出来。

beforeMount(挂载前)：在挂载开始之前被调用，相关 render 函数首次被调用。实例完成以下配置：编译模板，把 data 里面的数据和模板生成 html，但是没有挂载 html 到页面中。

mounted(挂载后)：挂载到实例之后被调用，实例完成以下配置：用上面编译好的 html 内容替换 el 属性指向的 DOM 对象。完成模板中的 html 渲染到 html 页面中。此过程中进行 ajax 交互。

beforeUpdate(更新前)：在数据更新之前调用，发生在虚拟 DOM 重新渲染和打补丁之前。可以在该钩子中进一步地更改状态，不会触发附加的重渲染过程。

updated(更新后)：在由于数据更改导致的虚拟 DOM 重新渲染和打补丁之后调用。调用时，组件 DOM 已经更新，所以可以执行依赖于 DOM 的操作。然而在大多数情况下，应该避免在此期间更改状态，因为这可能会导致更新无限循环。该钩子在服务器端渲染期间不被调用。

beforeDestroy(销毁前)： 在实例销毁之前调用。实例仍然完全可用。

destroyed(销毁后)： 在实例销毁之后调用。调用后，所有的事件监听器会被移除，所有的子实例也会被销毁。该钩子在服务器端渲染期间不被调用。

activited：keep-alive 专属，组件被激活时调用

deactivated：keep-alive 专属，组件被激活时调用

四、Vue 实现数据双向绑定的原理：Object.defineProperty（）

vue 实现数据双向绑定主要是：采用**数据劫持**结合**发布者-订阅者模式**的方式，通过 Object.defineProperty()来劫持各个属性的 setter，getter，在数据变动时发布消息给订阅者，触发相应监听回调。

要想实现 mvvm，主要包含两个方面，视图变化更新数据，数据变化更新视图.

- 1）view 变化更新 data：可以通过事件监听实现
- 2）data 变化更新 view：通过 Object.defineProperty( )对属性设置一个 set 函数，当属性变化时就会触发这个函数，所以我们只需要将一些更新的方法放在 set 函数中就可以实现 data 变化更新 view 了.
- 3）**具体过程：**
- 首先要对数据进行劫持监听，所以要设置一个**监听器 Observer**,用来监听所有的属性，当属性变化时，就通知**订阅者 Watcher**,看是否需要更新。属性可能是多个，所以会有多个订阅者，故需要一个**消息订阅器 Dep** 来专门收集这些订阅者，并在监听器 Observer 和订阅者 Watcher 之间进行统一的管理。

因为在节点元素上可能存在一些指令，所以还需要一个**指令解析器 Compile**，对每个节点元素进行扫描和解析，将相关指令初始化成一个订阅者 Watcher，并替换模板数据并绑定相应的函数，这时候当订阅者 Watcher 接受到相应属性的变化，就会执行相对应的更新函数，从而更新视图.

总结：

1.实现一个监听器 Observer，用来劫持并监听所有属性，如果有变动的，就通知订阅者。

2.实现一个订阅者 Watcher，可以收到属性的变化通知并执行相应的函数，从而更新视图。

3.实现一个解析器 Compile，可以扫描和解析每个节点的相关指令，并根据初始化模板数据以及初始化相应的订阅器。

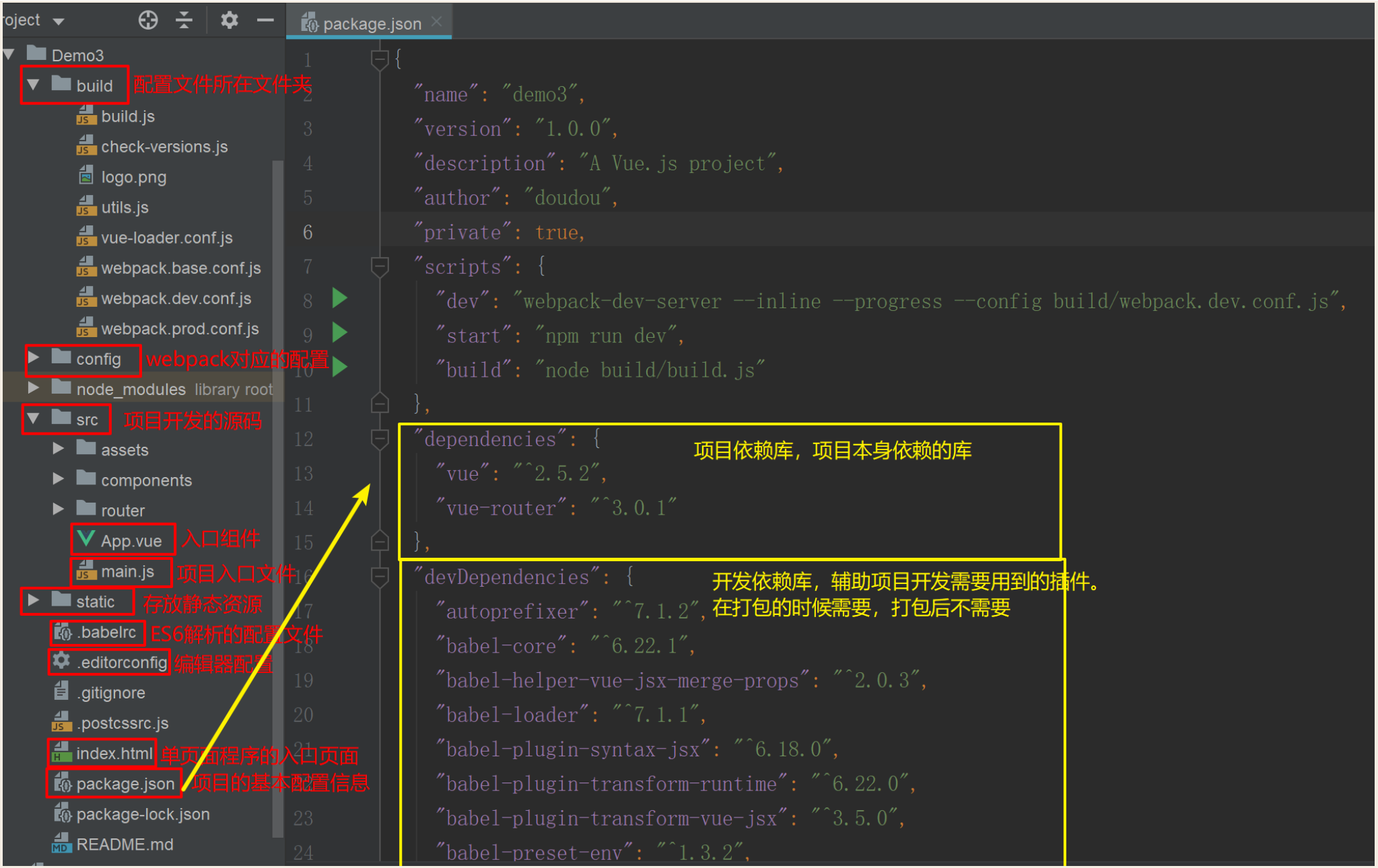
详情请读：

https://www.jianshu.com/p/0c0a4513d2a6

五、vue-cli 是什么？

Vue. js 提供一个官方命令行工具，可用于快速搭建大型单页应用（在一个完成的应用或者站点中，只有一个完整的 HTML 页面，这个页面有一个容器，可以把需要加载的代码（以组件的方式）插入到该容器中）。

该工具提供开箱即用的构建工具配置，带来现代化的前端开发流程。只需几分钟即可创建并启动一个带热重载、保存时静态检查以及可用于生产环境的构建配置的项目。



assets 文件夹是放静态资源；components 是放组件；router 是定义路由相关的配置;view 视图；app.vue 是一个应用主组件；main.js 是入口文件等等

六、Vue 组件如何通信

- props/$emit+v-on: 通过 props 将数据自上而下传递，而通过$emit 和 v-on 来向上传递信息。

- 中央事件总线 EventBus: 通过 EventBus 进行信息的发布与订阅，实现非父子组件之间的通信
- vuex: 是全局数据管理库，可以通过 vuex 管理全局的数据流
- v-model 方式：直接绑定父组件变量，把数据从子组件传回父组件

七、vuex 是什么？

Vuex 类似 Redux 的状态管理器，用来管理 Vue 的所有组件状态。

八、vue 组件中的 data 为什么是一个函数？

组件是可复用的 vue 实例，一个组件被创建好之后，就可能被用在各个地方，而组件不管被复用了多少次，组件中的 data 数据都应该是相互隔离，互不影响的，基于这一理念，组件每复用一次，data 数据就应该被复制一次，之后，当某一处复用的地方组件内 data 数据被改变时，其他复用地方组件的 data 数据不受影响。

类似于给每个组件实例创建一个私有的数据空间，让各个组件实例维护各自的数据。而单纯的写成对象形式，就使得所有组件实例共用了一份 data，就会造成一个变了全都会变的结果。

九、computed 和 watch 有什么区别?

- **computed:**

1、computed 是计算属性,也就是计算值

2、computed 具有缓存性,computed 的值在 getter 执行后是会缓存的，只有在它依赖的属性值改变之后，下一次获取 computed 的值时才会重新调用对应的 getter 来计算

3、computed 适用于计算比较消耗性能的计算场景

```
data: {
    message: 'Hello'
  },
 computed: {
    // 计算属性的 getter
    reversedMessage: function () {
      // `this` 指向 vm 实例
      return this.message.split('').reverse().join('')
    }
  }
```

- ** watch**

1、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。

2、无缓存性，页面重新渲染时值不变化也会执行

```
  watch: {
    // 如果 `question` 发生改变，这个函数就会运行
    question: function (newQuestion, oldQuestion) {
      this.answer = 'Waiting for you to stop typing...'
      this.debouncedGetAnswer()
    }
  },
```

1.谈谈你对 http 协议的认识。

浏览器本质,socket 客户端遵循 Http 协议
HTTP 协议本质：通过\r\n 分割的规范+ 请求响应之后断开链接 == > 无状态、 短连接
具体：
Http 协议是建立在 tcp 之上的，是一种规范，它规范定了发送的数据的数据格式，
然而这个数据格式是通过\r\n 进行分割的，请求头与请求体也是通过 2 个\r\n 分割的，响应的时候，
响应头与响应体也是通过\r\n 分割，并且还规定已请求已响应就会断开链接
即---> 短连接、无状态

2.谈谈你对 websocket 协议的认识。

websocket 是给浏览器新建的一套（类似与 http）协议，协议规定：（\r\n 分割）浏览器和服务器连接之后不断开，
以此完成：服务端向客户端主动推送消息。

websocket 协议额外做的一些操作
握手 ----> 连接钱进行校验
加密 ----> payload_len=127/126/<=125 --> mask key
本质
创建一个连接后不断开的 socket
当连接成功之后：
客户端（浏览器）会自动向服务端发送消息，包含： Sec-WebSocket-Key: iyRe1KMHi4S4QXzcoboMmw==
服务端接收之后，会对于该数据进行加密：base64(sha1(swk + magic_string))
构造响应头：

```
HTTP/1.1 101 Switching Protocols\r\n
Upgrade:websocket\r\n
Connection: Upgrade\r\n
Sec-WebSocket-Accept: 加密后的值\r\n
WebSocket-Location: ws://127.0.0.1:8002\r\n\r\n
```

发给客户端（浏览器）
建立：双工通道，接下来就可以进行收发数据
发送数据是加密，解密，根据 payload_len 的值进行处理
payload_len <= 125
payload_len == 126
payload_len == 127
获取内容：

mask_key
数据
根据 mask_key 和数据进行位运算，就可以把值解析出来。

3.什么是 magic string ？

客户端向服务端发送消息时，会有一个'sec-websocket-key'和'magic string'的随机字符串(魔法字符串)
服务端接收到消息后会把他们连接成一个新的 key 串，进行编码、加密，确保信息的安全性

4.如何创建响应式布局？

响应式布局是通过@media 实现的

```
@media （min-width：768px）{
     .pg-header{
           background-color:green;
      }
}
@media   (min-width:992px){
     .pg-header{
            background-color:pink;
     }
}
```

代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
    <style>
        body{
            margin: 0;
        }
        .pg-header{
            background-color: red;
```

```
            height: 48px;
        }

        @media (min-width: 768px) {
            .pg-header{
                background-color: aqua;
            }
        }
        @media (min-width: 992px) {
            .pg-header{
                background-color: blueviolet;
            }
        }
    </style>
</head>
<body>
    <div class="pg-header"></div>
</body>
</html>
```

5.你曾经使用过哪些前端框架？

```
jQuery
- BootStrap
- Vue.js(与 vue 齐名的前端框架 React 和 Angular)
```

6.什么是 ajax 请求？并使用 jQuery 和 XMLHttpRequest 对象实现一个 ajax 请求。

```
⊞基于原生 AJAX - Demo
⊞基于 jQueryAjax - Demo
http://www.cnblogs.com/wupeiqi/articles/5703697.html
```

7.如何在前端实现轮训？

```
 轮询：通过定时器让程序每隔 n 秒执行一次操作。
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
</head>
<body>
    <h1>请选出最帅的男人</h1>
    <ul>
        {% for k,v in gg.items() %}
            <li>ID:{{ k }}, 姓名：{{ v.name }} ，票数：{{ v.count }}</li>
        {% endfor %}
    </ul>

    <script>
        setInterval(function () {
            location.reload();
        },2000)
    </script>
</body>
</html>
```

8.如何在前端实现长轮训？

客户端向服务器发送请求，服务器接到请求后 hang 住连接，等待 30 秒，30s 过后再重新发起请求，
直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。

```
<!DOCTYPE html>
<html lang="zh-cn">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Title</title>
</head>
<body>
    <h1>请选出最帅的男人</h1>
    <ul>
        {% for k,v in gg.items() %}
```

```
            <li style="cursor: pointer" id="user_{{ k }}" ondblclick="vote({{ k }});">ID:{{ k }}, 姓名：{{ v.name }} , 票数：<spa
n>{{ v.count }}</span></li>
        {% endfor %}
    </ul>

    <script src="/static/jquery-3.3.1.min.js"></script>
    <script>
        $(function () {
            get_new_count();
        });

        function get_new_count() {
            $.ajax({
                url: '/get_new_count',
                type:'GET',
                dataType:'JSON',
                success:function (arg) {
                    if (arg.status){
                        // 更新票数
                        var gid = "#user_" + arg.data.gid;
                        $(gid).find('span').text(arg.data.count);
                    }else{
                        // 10s 内没有人投票
                    }
                    get_new_count();

                }
            })
        }

        function vote(gid) {
            $.ajax({
                url: '/vote',
                type:'POST',
```

```
                data:{gid:gid},
                dataType:"JSON",
                success:function (arg) {

                }
            })
        }
    </script>
</body>
</html>
```

9.vuex 的作用？

多组件之间共享：vuex

补充 luffyvue
1:router-link / router-view
2:双向绑定，用户绑定 v-model
3：循环展示课程：v-for
4：路由系统，添加动态参数
5：cookie 操作：vue-cookies
6：多组件之间共享：vuex
7：发送 ajax 请求：axios（js 模块）

10.vue 中的路由的拦截器的作用？

vue-resource 的 interceptors 拦截器的作用正是解决此需求的妙方。
在每次 http 的请求响应之后，如果设置了拦截器如下，会优先执行拦截器函数，获取响应体，然后才会决定是否把 response 返回给 then 进行接收

11.axios 的作用？

发送 ajax 请求：axios（js 模块）

12.列举 vue 的常见指令。

1、v-if 指令:判断指令，根据表达式值得真假来插入或删除相应的值。
2、v-show 指令:条件渲染指令，无论返回的布尔值是 true 还是 false，元素都会存在在 html 中，只是 false 的元素会隐藏在 html 中，并不会删除.
3、v-else 指令:配合 v-if 或 v-else 使用。
4、v-for 指令:循环指令，相当于遍历。
5、v-bind:给 DOM 绑定元素属性。

6、v-on 指令:监听 DOM 事件。

13.简述 jsonp 及实现原理？

JSONP
jsonp 是 json 用来跨域的一个东西。原理是通过 script 标签的跨域特性来绕过同源策略。
JSONP 的简单实现模式，或者说是 JSONP 的原型：创建一个回调函数，然后在远程服务上调用这个函数并且将 JSON 数据形式作为参数传递，完成回调。

14.什么是 cors ？

CORS
浏览器将 CORS 请求分成两类：简单请求和赋复杂请求

简单请求(同时满足以下两大条件)
（1）请求方法是以下三种方法之一:
HEAD
GET
POST
（2）HTTP 的头信息不超出以下几种字段：
Accept
Accept-Language
Content-Language
Last-Event-ID
Content-Type ：只限于三个值 application/x-www-form-urlencoded、multipart/form-data、text/plain

凡是不同时满足上面两个条件，就属于非简单请求

15.列举 Http 请求中常见的请求方式？

GET、POST、
PUT、patch（修改数据）
HEAD(类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头)
DELETE

传值代码 f
Request.QueryString 方法针对控件 id
Request.Form 方法针对控件名称 name

16.列举 Http 请求中的状态码？

分类：
1** 信息，服务器收到请求，需要请求者继续执行操作
2** 成功，操作被成功接收并处理
3** 重定向，需要进一步的操作以完成请求
4** 客户端错误，请求包含语法错误或无法完成请求
5** 服务器错误，服务器在处理请求的过程中发生了错误

常见的状态码
200 -请求成功
202 -已接受请求，尚未处理
204 -请求成功，且不需返回内容
301 - 资源（网页等）被永久转移到其他 url
400 - 请求的语义或是参数有错
403 - 服务器拒绝请求
404 - 请求资源（网页）不存在

500 - 内部服务器错误
502 - 网关错误，一般是服务器压力过大导致连接超时
503 - 由于超载或系统维护，服务器暂时的无法处理客户端的请求。

17.列举 Http 请求中常见的请求头？

- user-agent
- host
- referer
- cookie
- content-type

18.看图写结果（js）：

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<script>
    var name = '武沛齐';
    function func() {
        var name = '李杰';
        function inner() {
            alert(name);
        }
        return inner;
    }
    var ret = func();
    ret()
</script>
</body>
</html>
```

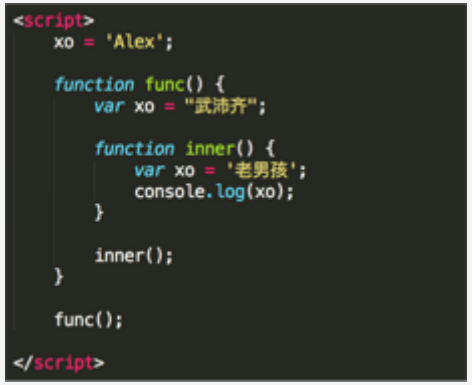
李杰

看图写结果（js）：

```html
<script type='text/javascript'>

    function main(){
        if(1==1){
            var name = '武沛齐';
        }
        console.log(name);
    }

    main()

</script>
```

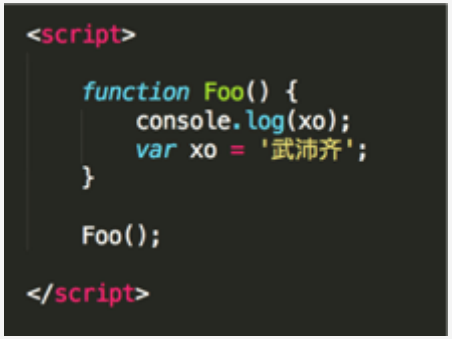
武沛奇

看图写结果：（js）



老男孩

看图写结果：（js）



undefined

看图写结果：（js）

```
<script>

    var name = 'Alex';

    function Foo() {
        this.name = '武沛齐';
        this.func = function () {
            alert(this.name);
        }
    }

    var obj = new Foo();
    obj.func()

</script>
```

武沛奇

看图写结果：（js）

```
<script>

    var name = 'Alex';

    function Foo() {
        this.name = '武沛齐';
        this.func = function () {
            (function () {
                alert(this.name);
            })()
        }
    }

    var obj = new Foo();
    obj.func()

</script>
```

Alex