js基础知识必备

变量常量

常量：

一旦申明不得更改，建议大写

变量命名：

不得以数字开头，由字母、数字、$ 下划线组成，下划线命名，驼峰命名

变量注意

申明未赋值,此时的值是undefine,使用未声明的变量会报错

数据类型

基本数值类型

数值型、字符串、布尔型、未定义(undefined)、空(null)

数据类型转换

隐式转换

1.数字+字符串：

```
1+'a' //'1a'
```

2.数字+布尔型: `true->1 false->0`
3.布尔型+字符串： `true+'hello'//'truehello'`
4.减乘除：那么对于除了加法的运算符来说，
只要其中一方
是数字, 那么另一方就会被转为数字

```
4 * '3' // 12
4 * [] // 0
4 * [1, 2] // NaN
```

强制转换

①将任意类型转为整型：

`parseInt('1.5a')//1字母开头变NAN`

②将任意类型转为浮点型:

`parseFloat('1.5a')//1.5与①类似`

③将任意类型转为数值型：

`Number('1.5a')//NaN`

④数值型和布尔型转为字符串型：

`toString()`

运算符

算数运算符

> 算数+ - * / % ++ --
>
> > 自增a++可将a='1'隐式转换为数值型
> > console.log(num++) //先打印num的值，再执行自增
> > console.log(++num) //先执行自增，再打印num的值

比较运算符

> < ,>=，<=,,!= ， =(全等于) ， !==(不全等于)
>
> > 返回一个布尔型的结果
> > == 只是比较两个值是否相同
> > === 不仅比较值，还会比较类型是否相同
> > 3>'10' //false数字和字符串比较,字符串转成数字。
> > '3'>'10' //true 比较首个字符的Unicode码,如果首个字符相同，则比较第二个字符
> > '3'->51 '1' -> 49
> > 3>'10a' //false

逻辑运算符

> && 并且， || 或者， ! 非 返回一个布尔型的结果

> ! 取反 !false -> true !true -> false
> 练习：声明两个变量保存用户名和密码，如果用户名是'root'，
> 并且密码是'123456'，打印true，否则打印false；
> 练习:声明一个变量保存年龄,如果年龄大于90岁,或者年龄小
> 于3岁，打印true，否则打印false

位运算符

按位与&，按位或|。按位异或^, 按位右移>>,按位左移<<

赋值运算

= += -= *= /= %=

三目运算

单目运 算

> **a++ , a--, !false**

双目运算

> 需要两个数据或者表达式&& || & | ^ >> << = += -=
> *= /= %= + - * / % > < >= <= == != === !==

三目运算

> 条件表达式 ? 表达式1 : 表达式2
> 课后任务：
> (1)复习今天内容，整理当天的思维导图
> (2)练习： 声明一个变量保存年份，判断这个年份是否为闰年，
> 如果是打印“是闰年”，否则打印“不是闰年” 闰年：4年一个闰年
> (能被4整除，和4 取余为0)，并且不能被100整除，或者被400整除。

```
var year=2001;
var years=year%4==0
&&year%100!=0||year%400==0 ? '闰年':'平年';
 console.log(years);
```

逻辑结构

函数对象

常见错误

ES6

JSbasic习题集

VUE

使用

1.定义页面的HTML内容
要求: 必须包含在一个 `<div id="app"></div>` 内
必须用{{变量}}标记处，要使用数据的位置
如有事件处理函数，就用@click绑定事件处理函数
比如:

```
<div id="app">
<button @click="add">click me({{n}})
</button>
```

2.在自定义程序中先定义页面所需的所有数据。
要求: 必须包含在一个data:{}对象中
比如: 页面上有一个{{n}}，表示一处n需要发生变化，所以

```
var data={ n:0; }
```

3. 创建new Vue()对象示例，将数据和页面元素绑定起来

```
var vm=new Vue({
 el:"#app",
 data: data,
 //结果: data中的变量是什么值，页面中就显示什么值
 //data中的变量被改成什么值，页面中就自动变成什么值
 //如果需要事件处理函数，都要定义在methods:{}中
 methods:{
   add:function(){//当单击按钮时，自动触发add函数
     this.n++;//修改data中的n+1,页面上的n自动跟着变化
   }
 }
})
```

MVVM模式

绑定语法

1.找页面可能发生变化的地方
2.模型中定义同名变量

```
new Vue({
        el:"#app",
        data:{
              变量名:值,
            ... : ...,
       }
      })
```

强调: HTML中有几处变化，data对象中就要有几个变量与之对应。

3.绑定语法定义变量{{}}

指令

1绑定属性.v-bind
何时: 只要属性值需要根据变量动态变化时，
就要用v-bind或:简写
如何:

```
<img v-bind:src="pm25<100?'img/1.png':
                 pm25<200?'img/2.png':
                 pm25<300?'img/3.png':
                          'img/4.png'">
```

去{{}}换v-bind:
其实可简写: `<img v-bind:src="...` 去{{}}换:
2.元素显示/隐藏：
单个元素： v-show="判断条件"
多个元素:v-if="判断条件"
3.由数组生成HTML

1. data中必须先定义一个可遍历的数组
2. 在HTML中使用v-for遍历数组，反复生成多个相同结构的元素，并动态绑定元素的内容。
   语法:
   <要反复生成的元素 `v-for="(elem,i) of` 数组" :key="i"
   强调: 特例: v-for中的循环变量可被v-for自己或子元素用于绑定！

4.事件绑定:v-on:事件名="处理函数" 简写为@事件名="处理函数"

3. 处理函数必须定义在:

```
new Vue({
    el:"#app",
    data:{ ... },
    methods:{
        处理函数(){
            this.data中变量
      }
    }
  })
```

中的methods:{}结构中

2. 其实可以传参: @事件名="处理函数(实参值)"

3. 也可以用事件对象e：用法和DOM中完全一样

获得

```
e: methods:{
               事件处理函数(e){ ... }
               }
```

后续:

获得目标元素，实现事件委托： `e.target`

取消冒泡: `e.stopPropagation()`

阻止默认行为: `e.preventDefault();`

键盘事件中获得键盘号: `e.keyCode`

获得鼠标坐标位置: `e.screenX, e.screenY`

`e.clientX, e.clientY`

`e.offsetX, e.offsetY`

5. 避免用户短暂看到{{}}用：v-cloak

解决: 只能自己手写！用属性选择器:

`[v-cloak]{display:none}`

强调: v-cloak不能改名！改名了，vue就找不到了！

其实，除了**v-cloak**外，还可用**v-text**代替{{}}绑定元素内容，

避免短暂看到{{}}

```
<h1 v-text="`姓名: ${uname}`"></h1>
<h2 v-text="`性别: ${sex==1?'男':'女'}`"></h2>
```

6. 绑定HTML片段:v-html

问题: 使用{{}}绑定HTML片段，VUE不会解析HTML片段为网页内容。而是原样显示HTML 代码

解决: 今后，只要绑定一段HTML片段，必须用v-html。

如何: `<ANY v-html="变量"></ANY>`

7.只做一次绑定：v-once

8.让{{}}原样显示。v-pre

####this指向问题汇总

1. 普通函数或匿名函数自调中的this->window

严格模式下: this->undefined

2. obj.fun() fun中的this->obj

3. new Fun() Fun中的this->正在创建的新对象

4. 原型对象中的函数里的this->将来调用当前函数的.前的当前类型的子对象。

5. btn.onclick=function(){... } this->btn

6. 回调函数:

```
arr.forEach(function(){ ... })
arr.map(function(){ ... })
setInterval(function(){ ... })
setTimeout(function(){ ... })
$.ajax({
    ...
    success:function(){ ... }
}).then(function(){
    ...
})
this->window// 所有回调函数，真正被调用时，前边是没
//有任何"对象."前缀，回调函数立足于window
//所以，通常如果希望回调函数中的this不指window，
//而是跟外部的this保持一致，都要改为箭头函数
```

7. jQuery中的回调函数:

```
$().each(function(){ ... })
$().animate({ ... }, ms, function(){ ... })
```

jquery中的多数回调函数,this->当前正在操作的dom元素,还要注意是否有

阻止冒泡;

8. 不考虑之前已经总结的情况，vue中一切this都指向当前new Vue()对象。

所以在vue js中访问任何变量都要加this.变量名。但是html中绑定变量名不用加this！

####双向绑定v-model:value="变量"

1. radio:
   特殊在: value任何情况下是固定的
   变的是checked属性，也就是选中与不选中状态
   比如: 性别:

```
<input type="radio" name="sex" v-model="sex"
 value="1" checked >男
<input type="radio" name="sex" v-model="sex"
 value="0" checked >女

data:{
      sex:1  0
    }
```

绑定时: 用data中sex的值和每个radio的提前写死的value做比较。如果那个radio的value值刚好等于data中sex的值就选中其余不选中修改时: 用当前选中的radio的写死的value，反向赋值回data中的sex上

2. select:

特殊: 每个option的value都是提前写死的变得只是option的selected属性，变的只是选中的option是哪个。

比如:

```
<select v-model="city">
     <option value="bj.png">北京</option>
     <option value="hz.png">杭州</option>
     <option value="sh.png">上海</option>
  </select>
  data:{
   city:"bj.png"
  }
```

3. checkbox:

```
如何: <input type="checkbox" v-model="isAgree">同意
     data: { isAgree:false  }
```

> 双向绑定中可用watch机制，实时获得修改的新变量值:
> 何时: 只要希望边修改边实时执行操作时
> 如何:

```
data: { kwords:"" }
 watch:{
   kwords:function(){
    //执行操作
   }
 }
```

只要通过任何手段修改一次kwords变量的值，都会立刻
触发一次kwords()函数，执行一次操作。

绑定样式

1.将整个style属性值当做一个大的字符串去绑定

> 问题: 如果使用style绑定样式，说明经常会单独修改一
> 个样css属性值。而如果style是一个字符串，非常不便于
> 程序仅修改其中一个css属性。
> 解决: style支持对象形式绑定

2. 以对象形式绑定style

> 2步:
> 1. 在data中定义一个内嵌对象结构，保存style中每个css属性
> 比如: 希望div#pop的位置上下左右变化，可定义对象:

```
data:{
   popStyle:{
     left:"100px",//必须带单位
     top:"50px"
   }
 }
```

> 2. 在HTML元素的style上，用冒号绑定data中的样式对
> 象，比如：
> `<div id="pop" :style="popStyle"></div>`
> 运行时: new Vue()会自动将
> popStyle:{
> left:"100px",//必须带单位
> top:"50px"

> }
> 转化为字符串: "left:100px; top:50px"
> 结果还是: style="left:100px; top:50px"
> 说明: 如果div中还有部分固定的内联样式，可将普通style和:style同时使用。比如:

```
<div id="pop" style="width:100px;
 height:100px:style="popStyle"></div>
//运行时: 先将:style中的对象编译为字符串，再和固定不变
//的普通style内容合并到一个style中。
//结果:
<div id="pop" style="width: 100px;
 height: 100px; left: 100px; top: 50px;"></div>
```

绑定class: 2种方式:

1. 以字符串绑定class--不好
2. 以对象方式绑定class

> 2步:
> 1. data中定义个对象，包含所有要显示的class作为属性名。用true/false控制哪个class应用，哪个class不用。比如:

```
data:{
   phoneClass:{
     vali_info:true,//默认启用
     vali_success:false,//验证通过启用
     vali_fail:false//验证失败才启用
   }
 }

 //2. html中用:class，绑定data中的对象
 //比如: <span :class="phoneClass">1[3-9]\d{9}</span>
 //结果: new Vue()自动将phoneClass中，所有标记为true的
 //class名，拼接
 //为字符串，替换元素中class的内容,应用指定的class。比如
   phoneClass:{
     vali_info:true,
     vali_success:true,
     vali_fail:false
   }
```

```
 // 会被翻译为
 "vali_info vali_success"
  //最终:
  <span class="vali_info vali_success"
//其实: :class="对象"中变化的类名，也可以和其他固定
//不变的类名同时存在:
//比如:
<span class="vali" :class="phoneClass">
// 结果: 先翻译: phoneClass为:
        "vali_info vali_success"
//  再和class="vali"组合形成一个class
// 最终:
 <span class="vali vali_info vali_success"
```

计算属性

1. 在new Vue中定义计算属性的逻辑

```
//1. 在new Vue中定义计算属性的逻辑
   new Vue({
     el:"#app",
     data:{  变量1, 变量2, ... },
     computed:{
       汇总的属性名(){
         return 汇总结果
       }
     }
   })
   //比如：定义计算属性，动态计算购物车中商品的总价:
    data:{
     cart:[
      {pname:"iphonex",price:5566,count:2},
      {pname:"iphonexs",price:8888,count:3},
      {pname:"iphonexs max",price:9999,count:1}
     ]
   },
   computed:{
    total(){
      console.log("计算了一次总价");
      //先定义变量
      var sum=0;
      //遍历cart中每个商品
      for(var p of this.cart){
        //每遍历一个商品，就就计算小计，并汇总到结果中
        sum+=p.price*p.count;
      }
      return sum;
    }
   }
```

2. 页面绑定

```
//在页面元素中，使用绑定语法，直接绑定计算属性的函数名:
   <ANY>{{汇总的属性名}}</ANY>
   //不要加()
   //比如:
   <td>总价:¥{{total.toFixed(2)}}</td>
```

自定义指令:

1. 创建:

```
//创建：一个名为v-focus的指令
     Vue.directive("focus",{//强调：不要加v-前缀
       inserted(dom_elem){
         dom_elem.DOM函数()
       }
     })
```

2. 使用

> 原理:
> 当new Vue()时，Vue框架会扫描原始DOM中的元素
> 只要发现带自定义指令的元素，就触发自定义的inserted()
> 函数，对当前DOM元素，执行预先定义好的DOM操作。

过滤器

1. 创建过滤器函数

```
1. 创建过滤器函数：比如，创建将性别0和1转为女和男的过滤器
   Vue.filter(
    "sexFilter", //过滤器函数名
    function(oldValue){//过滤器函数本身
      //至少有一个形参接收来自data中变量的原始值
      //对原始值加工后返回新值
      return oldValue==1?"男":"女";
      //必须有返回值
    }
  )
```

2. 使用过滤器

> 2. 在绑定语法中使用过滤器
>
> ```
> <h1>性别:{{sex|sexFilter}}</h1>
> ```
>
> 其中: 使用|将过滤器函数连接到sex原始变量的后边。
> 执行时，sex的值，自动传给过滤器sexFilter的函数的形参
> oldValue，经过过滤加工，将返回的新值显示到绑定位置。
> 其实，过滤器可以带参数，来选择不同的返回值样式：
> 比如：定义可根据语言选择返回不同语言性别的过滤器

```
Vue.filter(
 "sexFilter",
 function(oldValue,lang="cn"){//en cn
   if(lang=="cn"){//如果传入cn或不传入，则默认都返
   //回中文的男女
     return oldValue==1?"男":"女"
   }else{//否则，返回英文的男女。
     return oldValue==1?"Male":"Female"
   }
 }
)
```

使用带参数的过滤器:

```
<h1>性别:{{sex|sexFilter("en")}}</h1>
<h1>性别:{{sex|sexFilter("cn")}}</h1>
<h1>性别:{{sex|sexFilter}}</h1>
```

强调: 虽然定义形参lang时，lang处于第二个形参，但是使用过滤器时，因为原始值自动占了第一个形参oldValue，所以，自定义传入的实参值cn或en，就可直接作为第一个实参传入。实际给的确实第二个形参lang。

axois

1. 引入axios
2. 发送请求:

```
// get方式:
 axios.get("url",{
   params:{
     //请求参数: 参数值
   }
 }).then(function(/*返回结果result*/){
   result.data //才是服务器返回的结果
 })
 //比如: 用id查询一个商品
 axios.get("/products/getById",{
   params:{
     lid:5
   }
 })
 //http://localhost:3000/products/getById?lid=5 ->
 //{ lid:5,title: macbook,subtitle:优惠酬宾,...} <-
 .then(function(result){
   var product=result.data;
 })
```

new vue 的生命周期

1. 创建(create)阶段:

> 创建new Vue()对象和data()对象
> **已经有data对象了，但是没有虚拟DOM树**
> 可以发送ajax请求
> 因为没有虚拟DOM树，所以暂时不能用DOM操作
> 后: created(){ ... }
> 前: beforeMount() { ... }

2. 挂载(mount)阶段:

> 创建虚拟DOM树，将data中的变量值开始向DOM树上绑定
> 即有data对象，又有虚拟DOM树
> **即可发送ajax请求，又可执行DOM操作**
> 后: mounted(){ ... }
> //后两个阶段不是必须:
> 前: beforeUpdate(){ ... }

3. 更新(update)阶段:

> 当data中的变量值被改变时才触发
> 后: updated(){ ...}
> 前: beforeDestroy(){ ... }

4. 销毁(destroy)阶段:

> 当调用专门的$destroy函数销毁一个组件时，才触发
> 后: destroyed(){ ... }

组件封装

比如: //创建一个名为my-counter的组件
//将来反复在主页面中使用

```
Vue.component("my-counter",{
  template:`<div>
    <button @click="sub">-</button>
    <span>{{n}}</span>
    <button @click="add">+</button>
  </div>`,
  data:function(){
    return { n:1 }
  },
  methods:{
    sub(){ this.n--; },
    add(){ this.n++; }
  }
```

反复使用组件:

组件其实就是一个可重用的自定义HTML标签而已。
<组件名></组件名>
比如:

组件化开发 组件间的数据传递

父->子的通信:

1. 子: 为组件定义自定义属性，将来用于从父对象中
   绑定获得父对象中的变量。

```
var 子={
     template:`... `,
     data(){
        //仅限于子组件自己使用的内部变量
     },
     //定义 自定义属性
    props:[ "tasks" ]
   }
```

2. 父: 通过绑定子组件的自定义属性，将自己的变量
   交给子组件

> `<子 :tasks="tasks"></子>`
> 孩子的兜 <-- 爹的变量

4. SPA: Single Page Application

> 多页面
> 1.一个应用中包含**多个.html ****
> **2.每切换一次页面，都会重新发送请求——请求次数多**
> **3.每次切换页面时，都要重建整棵DOM树——效率低**
> **4.对于共用的资源，每次**切换页面，都要重复请求。**
> 单页面
> 1.一个应用中只有**一个.html**
> 2.所有组件都是在首页第一次加载时，已经全部加载过来了。每切换一次页面，不需要向服务器重新发起页面请求。只是在客户端本地挑选匹配的组件替换页面内容而已。
> ——**请求次数少**
> 3.每次切换界面时，不需要整棵重建DOM树，只要替换局部的节点对象即可。——**效率高**
> 4.每次切换界面时，因为中的引用没变，所以
> -----**不会重复请求共享的资源。**

1. 划分"页面"组件
2. 创建组件
3. 定义唯一的完整的.html页面

引用公共资源

```
`<link rel="stylesheet" href="css/bootstrap.css">`
 ...
引入vue.js和vue-router.js
引入所有组件对象的所在的js文件备用
<body>中
  <div id="app">
    <my-header/>页头组件，所有页面共用
    <router-view/>临时占位，将来会根据路由地址
    不同，动态选择不同的组件template替换
    <router-view>所在位置。
```

vue总结

```
var vm=new Vue({
  el:"#app",//引入html元素，形成虚拟DOM树
  data:{ //保存页面需要的所有模型变量
    //页面中有几处变化，data中就要有几个变量支持
    uname:,
    sex:,
    popStyle:{
      css属性: 属性值,
          ... : ...
    }
    phoneClass:{
      样式类: true,
          ... : ...
    }
  },
  methods:{ //所有函数
    处理函数(){
      this->new Vue()
      this.变量
    },
    自定义函数(){ ... }
  },
  watch:{ //所有对data中变量的监听函数
    变量名(){ //变量名要和data中的某个变量同名
    }
  },
  computed:{//计算属性
    total(){
      return
    }
  }
})
```

1.安装使用方法

下载安装二种方式

(1)方式一:(学习)

mint-ui官方网站下载css/js/fonts 加载当前html文件中

(2)方式二:

vue cli 脚手架工具下载配置

在脚手架当前目录下 npm i mint-ui -S

#以上指令下载安装

验证: node_modules/mint-ui/lib 所有组件

package.json

启动脚手架指令

npm run serve
#其中 serve 可以配置package.json

组件库的引入

(1)引入方式一:按需引入
(2)引入方式二:完整引入(唯一正确)
-必须先下载 mint-ui库
-在main.js [引入其它库并且配置库]

```
import MintUI from "mint-ui" #完整引入mint-ui库
import "mint-ui/lib/style.css"  #mint-ui 样式文件
Vue.use(MintUI);
```

#将mint-ui库所有组件注册vue实例中
#"mint-ui" node_modules下目录名称
#MintUI 为组件起名
#只引mint-ui组件库库文件还有一个文件需要
#单独引入 mint-ui库有一个样式文件

调用

短消息提供框:常用 删除成功;登录成功;用户名有误
基本语法: this.$toast({message:"",....});
参数
message:提示消息文本的内容
position:位置 top;bottom;middle
duration:持续时间(单位毫秒) 默认3000
className:类名为其添加样式
iconClass:图标类名

组件库--创建新组件

(1)创建空组件

```
src/components/exam/Exam01.vue
```

(2)为空组件指定路径
src/router.js path:'/Exam01'..
(3)运行组件:在浏览器地址栏中输入空组件地址回车

```
http://127.0.0.1:8080/#/Exam01
```

(4)添加内容再运行

表单

1.开关

```
<mt-switch v-model="val1">开关</mt-switch>
```

2.单选框列表

```
<mt-radio
title="单选框列表"
 v-model="val2"
 :options="['选项a','选项b','选项c']">
</mt-radio>
```

3.复选框列表

```
<mt-radio
title="单选框列表"
v-model="val2"
:options="['选项a','选项b','选项c']">
</mt-radio>
```

父子面板

面板,可切换显示子页面

```
<mt-tab-container v-model="active"> <--!父元素(面板)-->
   <mt-tab-container-item id="tab1"><--!子面板1-->
     <mt-cell v-for=""></mt-cell><--!单元格-->
   </mt-tab-container-item>

   <mt-tab-container-item id="tab2"><--!子面板2-->
     <mt-cell v-for=""></mt-cell><--!单元格-->
   </mt-tab-container-item>
</mt-tab-container>
```

注意:改变active值与子面板id一致即显示对应子面板

底部选项栏

底部选项卡,点击tab会切换显示页面,

```
<mt-tabbar v-model="selected" fixed>
  <mt-tab-item id="外卖">
      <img slot="icon" src="../asserts/100*100.png" />
    外卖
  </mt-tab-item>
  <mt-tab-item id="订单">
      <img slot="icon" src="../asserts/100*100.png" />
    外卖
  </mt-tab-item>
</mt-tabbar>
```

fixed 固定在页面底部
-value 返回当前选中**tab-item**的**id**

父子间通讯

子组件**S09.vue**

```
<h3>{{msg}}</h3> //父组件传 msg
<img :src="img_url" />//父组件传 img_url
<button @click="add3"></button> //父组件传 add
<script>
  export default {
      props:{    //接收父组件数据
        msg:{default:""},    //msg默认空字符串
        img_url:{default:""}, //img_url默认空字符串
        add3:{type:Function} //add 函数
      }
  }
</script>
```

父组件: **F10.vue**

```
<script>
//1:引入子组件文件
import S09 from "./S09.vue";
//2:注册子组件,起标签名
components:{
  "s09": S09
}
myadd(){
}
</script>
<--! 调用子组件-->3:
<template>
    <s09
    msg="微信(11)"
    :img_url="require('../../asseets/01.png')"
    :add3="myadd" >
    </s09>
</template>
```

mint-ui组件库-应用

2.2:mint-ui组件库-应用-分析功能

> (1)顶部标题栏(完全自定义)
> (2)面板(主体功能使用mint-ui面板组件)
> (3)底部导航条(主体功能使用mint-ui tabbar组件)

解决小问题:
(1)如何动态读取文件中图片路径
(2)图片路径一定是在服务器端 示例: http://127.0.0.1:3000/1.jpg

```
<img :src="item.img" />
```

(3)本地图片路径报错

注意：vue事件有一些修饰符 native:原生事件

> 如果自定义属性直使用事件绑定即可,如果调用组件库中组件
> 直接绑定失败情况,清加修饰符 native 启动原生事件绑定
> 父元素给子元素绑定事件 native
> #注意:去除默认边框
> -App.vue 根组件它的内补丁影响我们组件
> -normalize.css->n.css 通用样式文件
> public/index.html

session-会话-(前端后端程序)

(1)session一个用户操作过程

> -会话开始:当用户打开某个网站第一个网页
> -会话中:中间用户可以打开此网站多个网页
> -会话结束:当用户关闭浏览器

(2)session对象(在服务器为了保存此次会话专有数据对象)

> session对象是保存当前会话中使用数据,在当前会话
> 中所有网页可以共享此数据，但是当会话结束session数据
> 失效
> 为什么使用session 对象,项目中有一些数据必须保存
> session对象比如:当前登录用户编号 uid
> 解决问题: