2017.12.08

vue实战todolist：渲染数据列表

//静态页面 渲染任务列表，定义数据格式(v-for将数据渲染到页面) //创建vue实例，挂载html元素作为模板，可以使用vue语法，代理数据到实例中

//指令语法： v-for="value,item in list" 事件绑定v-on:keyup.13="addTodo" 条件渲染v-show="list.length"

动态class设置 :class="{completed: item.isChecked,editing: item === edtorTodos}" 表单元素双向数据绑定v-model = "item.title"

//vue实例属性：watch，监控属性，当属性对应的值发生变化就会执行函数 computed，计算属性，函数中的数据发生改变调用函数 directives，自定义vue指令

//事件处理函数中的this指向的是，当前这个根实例，当调用data里面的数据时候要加上this，但是vue模板中调用数据的时候不用加this

//数据添加、删除、编辑(编辑成功和取消编辑) //存取localStorage中的数据，监控hash值变化，通过hash值变化筛选数据

数据从根实例传到custom-select组件，在传到custom-list组件，父级组件向子级传递数据：在父级模板标签上自定义属性绑定数据，子级props接收数据；

2017.12.04

闭包和作用域、上下文环境

给执行上下文环境下一个通俗的定义——在执行代码之前，把将要用到的所有的变量都事先拿出来，有的直接赋值了，有的先用undefined占个空。全局代码的上下文环境数据：普通变量（包括函数表达式）、函数声明、this。函数体代码段，在此基础上需要附加：arguments变量和函数参数以及自由变量的取值作用域

在函数中this到底取何值，是在函数真正被调用执行的时候确定的，函数定义的时候确定不了。因为this的取值是执行上下文环境的一部分，每次调用函数，都会产生一个新的执行上下文环境。this的取值，分四种情况：构造函数、函数作为对象的一个属性、函数用call或者apply调用、全局 & 调用普通函数

在Fn.prototype.getName函数中，this指向的是f1对象。因此可以通过this.name获取f1.name的值。其实，不仅仅是构造函数的prototype，即便是在整个原型链中，this代表的也都是当前对象的值。

执行全局代码时，会产生一个执行上下文环境，每次调用函数都又会产生执行上下文环境。当函数调用完成时，这个上下文环境以及其中的数据都会被消除，再重新回到全局上下文环境。处于活动状态的执行上下文环境只有一个。其实这是一个压栈出栈的过程——执行上下文栈。

执行到第5行，调用了fn函数。进入fn函数，在执行函数体语句之前，会创建fn函数的执行上下文环境，并压栈，设置为活动状态。待第5行执行完毕，即fn函数执行完毕后，此次调用fn所生成的上下文环境出栈，并且被销毁（已经用完了，就要及时销毁，释放内存）。有一种情况即闭包，可以不用销毁

javascript除了全局作用域之外，只有函数可以创建的作用域。所以，我们在声明变量时，全局代码要在代码前端声明，函数中要在函数体一开始就声明好。除了这两个地方，其他地方都不要出现变量声明。而且建议用“单var”形式。

作用域有上下级的关系，上下级关系的确定就看函数是在哪个作用域下创建的。作用域最大的用处就是隔离变量，不同作用域下同名变量不会有冲突。

作用域中变量的值是在执行过程中产生的确定的，而作用域却是在函数创建时就确定了。所以，如果要查找一个作用域下某个变量的值，就需要找到这个作用域对应的执行上下文环境，再在其中寻找变量的值。

在A作用域中使用的变量x，却没有在A作用域中声明（即在其他作用域中声明的），对于A作用域来说，x就是一个自由变量。要到创建这个函数的那个作用域中取值——是“创建”，而不是“调用”，切记切记。

上面描述的只是跨一步作用域去寻找，要一直跨到全局作用域为止。要是在全局作用域中都没有找到，那就是真的没有了。这个一步一步“跨”的路线，我们称之为——作用域链。

闭包应用的两种情况——函数作为返回值，函数作为参数传递。有些情况下，函数调用完成之后，其执行上下文环境不会接着被销毁，这就是需要理解闭包的核心内容。闭包作用：变量常驻内存、函数外部可以获取函数内部变量。

fn返回的函数体中，有自由变量max要引用fn作用域下的fn()上下文环境中的max。因此，这个max不能被销毁，销毁了之后bar函数中的max就找不到值了。

