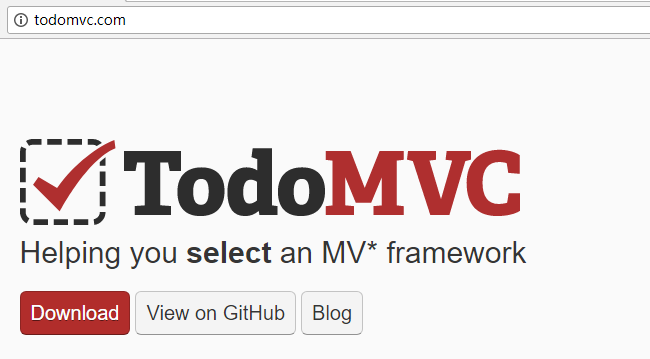
# 案例



在已经有html和css的基础上，通过vue来实现具体功能。

准备采用：vue + vuex +webpack + es6 + localstorge

官网上的例子是用了vue-router的，我们这里不准备使用。

## 分析

### 数据项

准备两个数据项。放在vuex中，供组件去使用。

todos：[{text:”运行一下vue-cli”,done:false},{text:”上传自己的npm包”,done:false}]

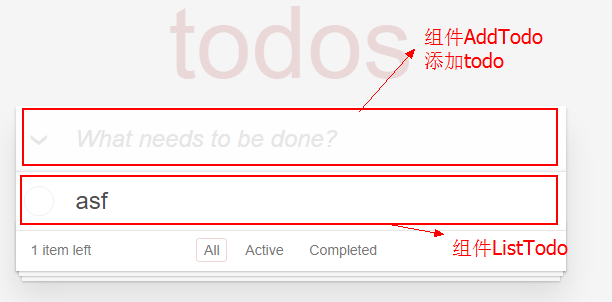
Visibilty:，筛选的条件 :all completed active

### 功能

* 显示todos数据
* 添加一条todos数据
* 删除todos数据
* 修改某一项的状态：完成，没有完成之间切换
* 批量修改状态:全部完成，全部未完成
* 统计没有完成的数量
* 批量删除已经完成的
* 三种状态的筛选
* 编辑todos的内容
* 本地存储

### 组件

在脚手架工具的基础上，额外设置两个组件。



## 通过脚手架创建项目

### Vue create vuetodos初始化

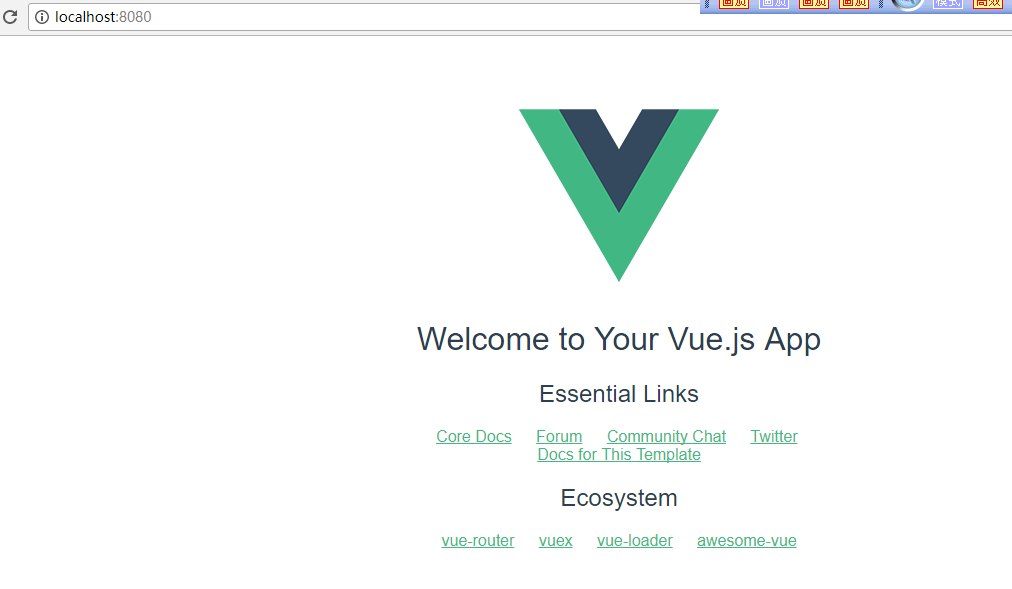
通过命令vue create vuetodos创建一个项目。

设置一项选项：注意，不需要路由

### 安装vuex



### 运行起来



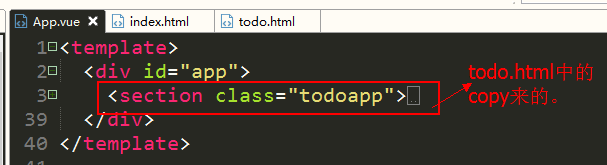
## 具体实现

### 把静态资源加入app.vue中



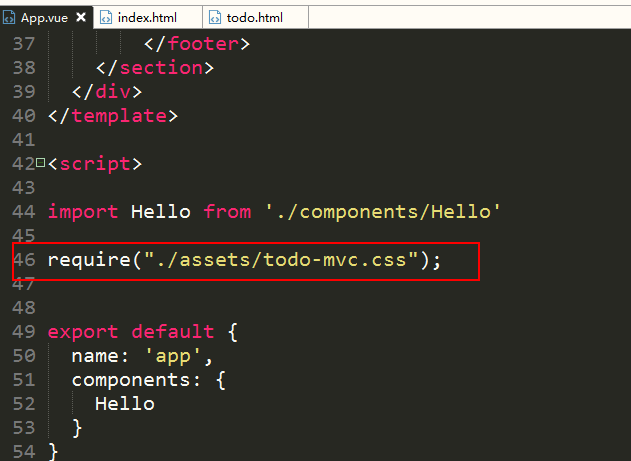
todo-mvc.css拷贝到 src/assets/目录下，

把app.vue中的template的内容用todo.html的内容来代替。

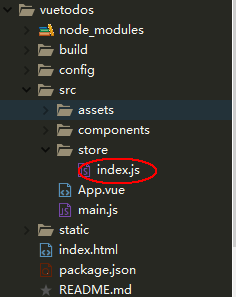


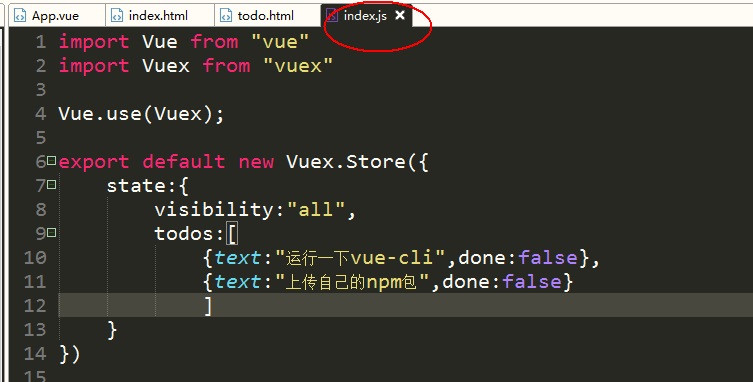
把app.vue中的样式去掉。

通过require去引入todo-mvc.css

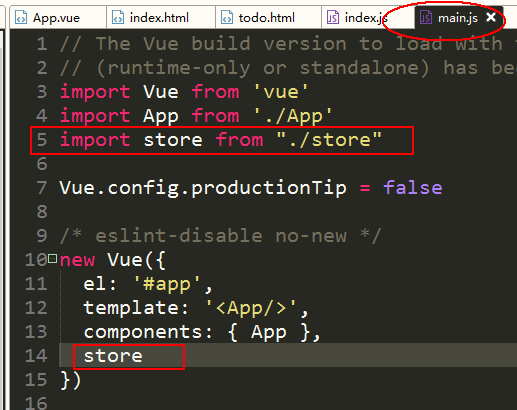


### 创建并引入vuex

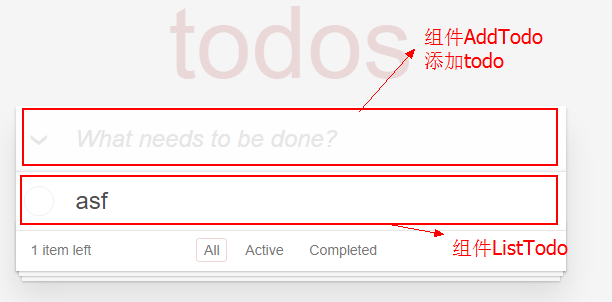




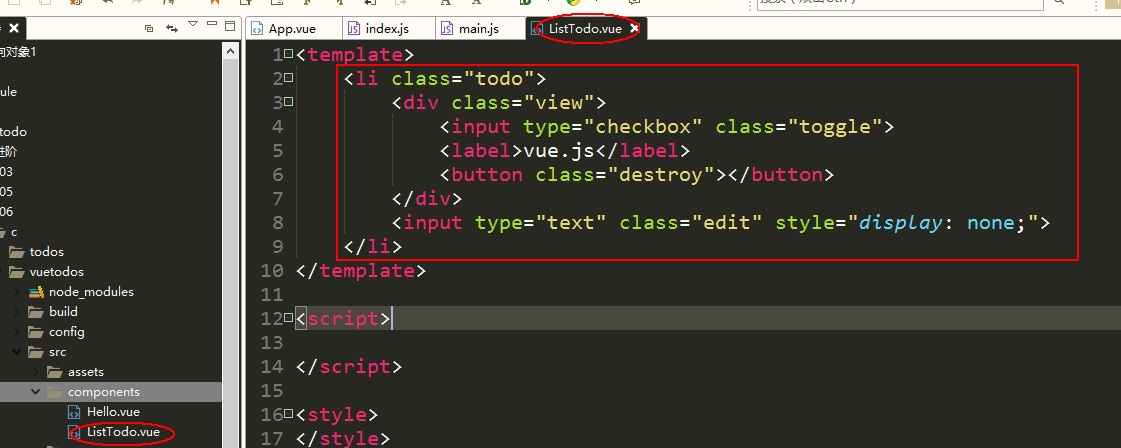
去main.js中去引入



### 拆分组件-- 显示todo数据

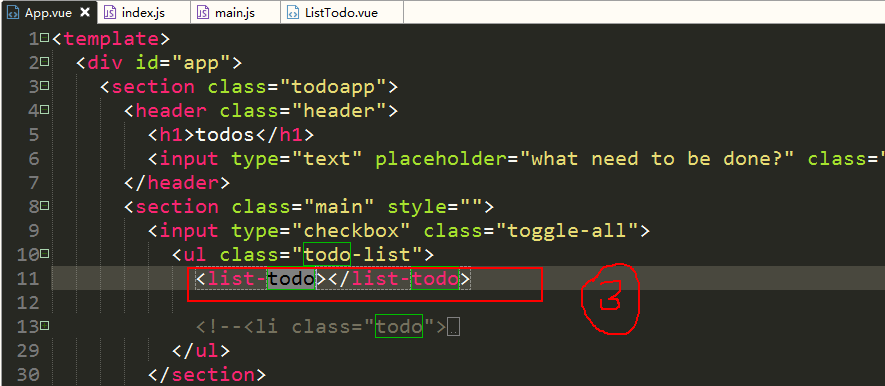


#### listTodo基本的使用



在app.vue组件中，使用listtodo组件：



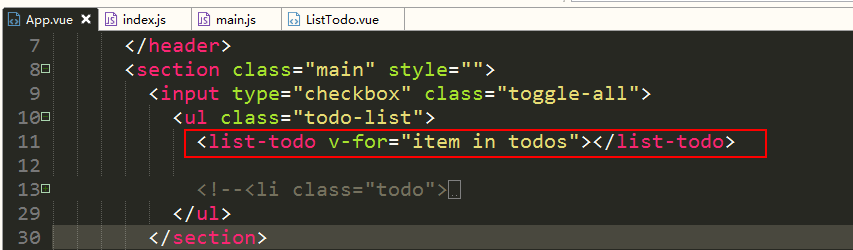


#### 在app.vue组件中给listTodo传递数据，显示todos

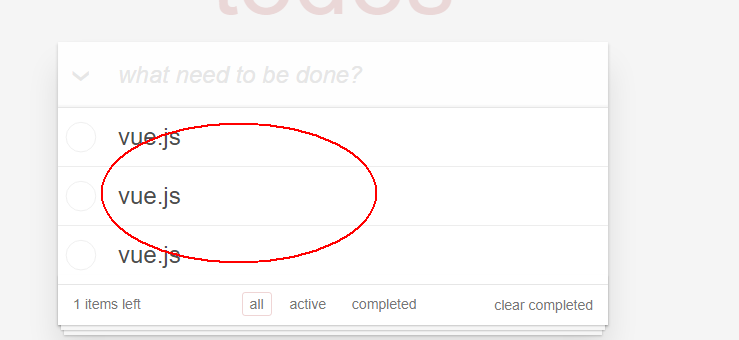
在app.vue中获取vuex中的数据项。



现在对list-todo进行循环



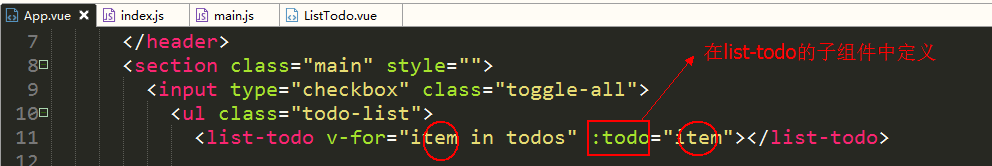
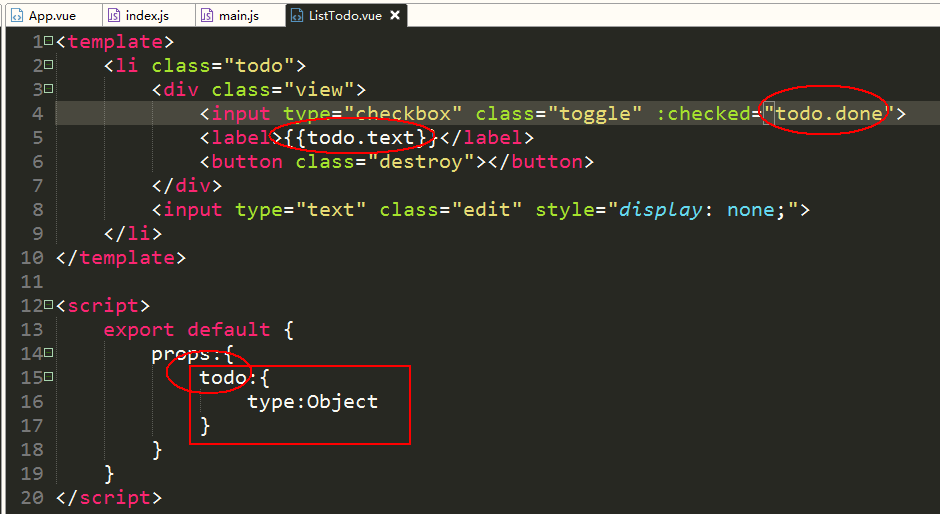
效果如下：



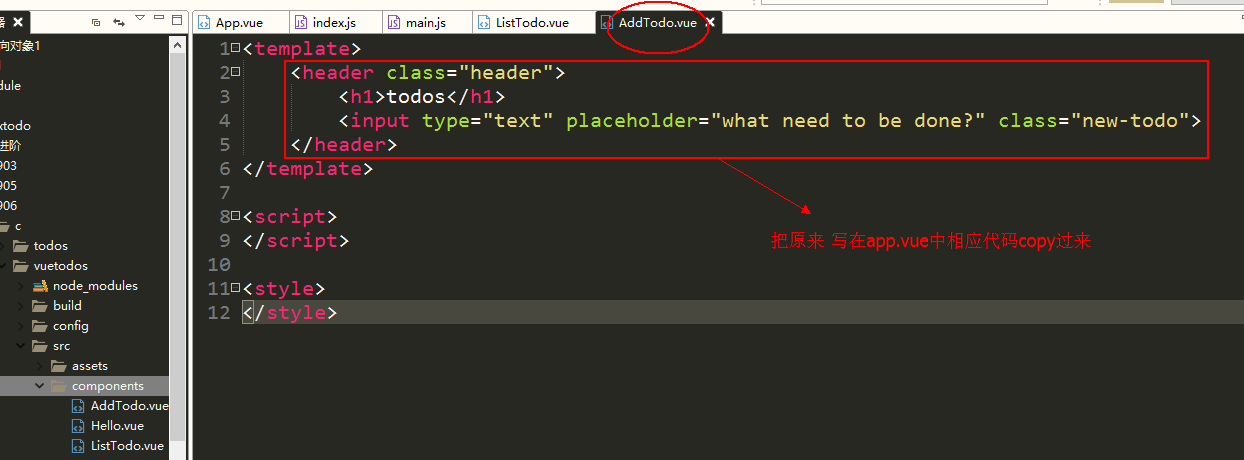
#### 通app.vue这个父组件给list-todo这个子组件进行数据传递

因为我们需要把一个一个todos具体内容传递list-todo，让它来显示具体的信息。

回顾：数据从父到子的传递。

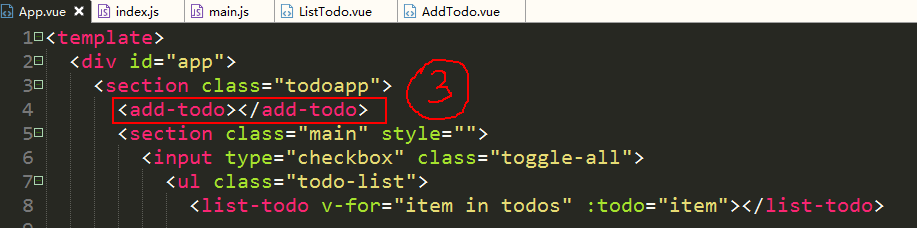
1. 在使用子组件时，设置自定义属性  
   
2. 在子组件中，设置props  
   

### 添加一条todos数据 -AddTodo组件



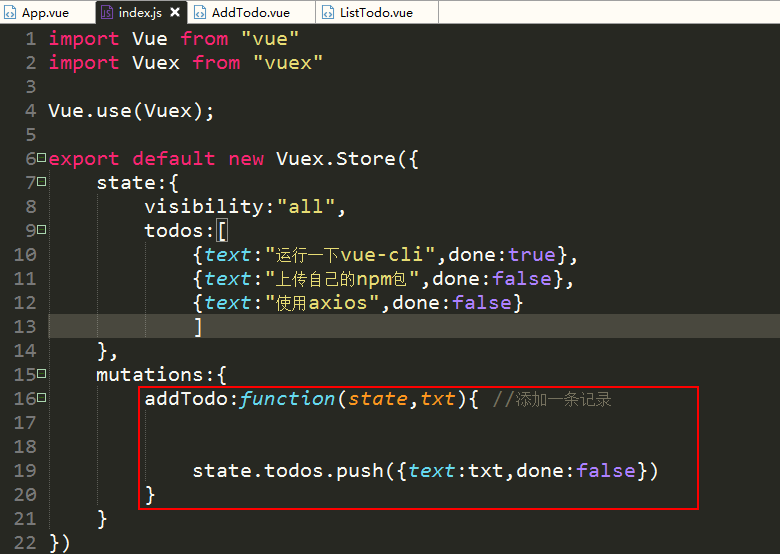
在app.vue中使用



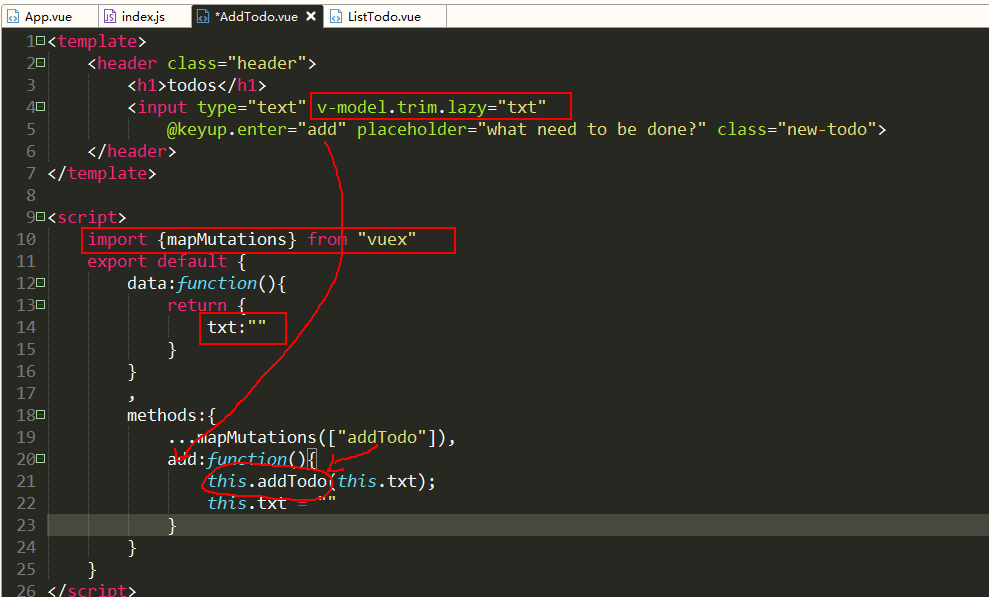


下面就要具体去实现添加一条todo的功能

本质上需去vuex.store.state.todos中加一条记录，这里就是涉及修改vuex中的数据，必须要用mutations.所以，我们应该先去vuex中去创建好一个mutations。

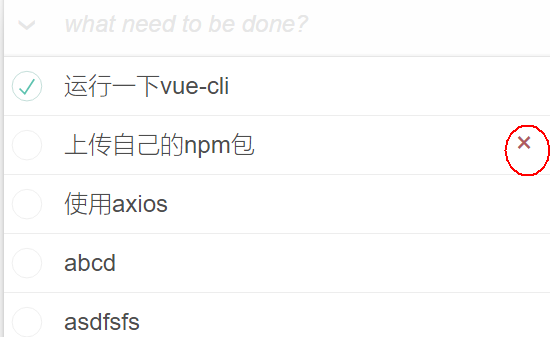


把这个方法直接映射到addTodo组件的methods中

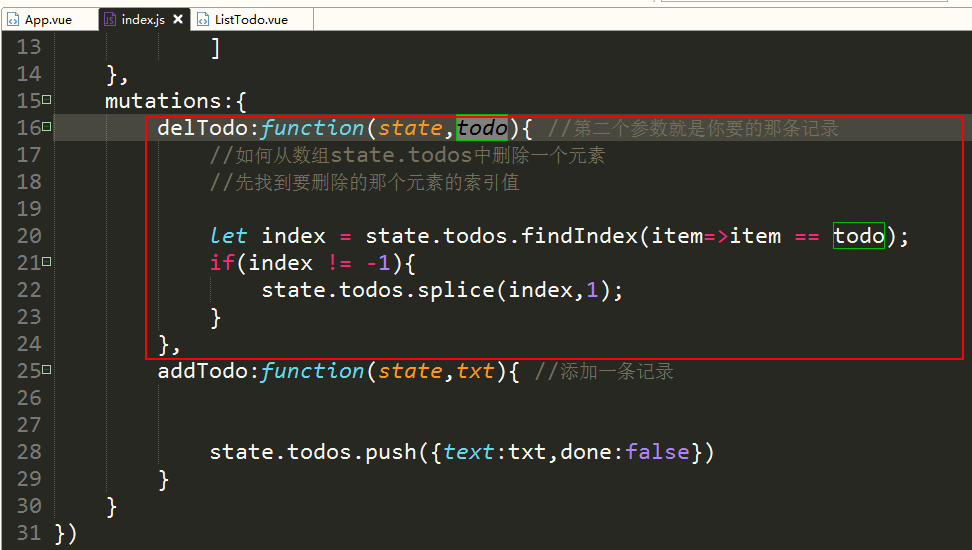


就实现了添加的效果。

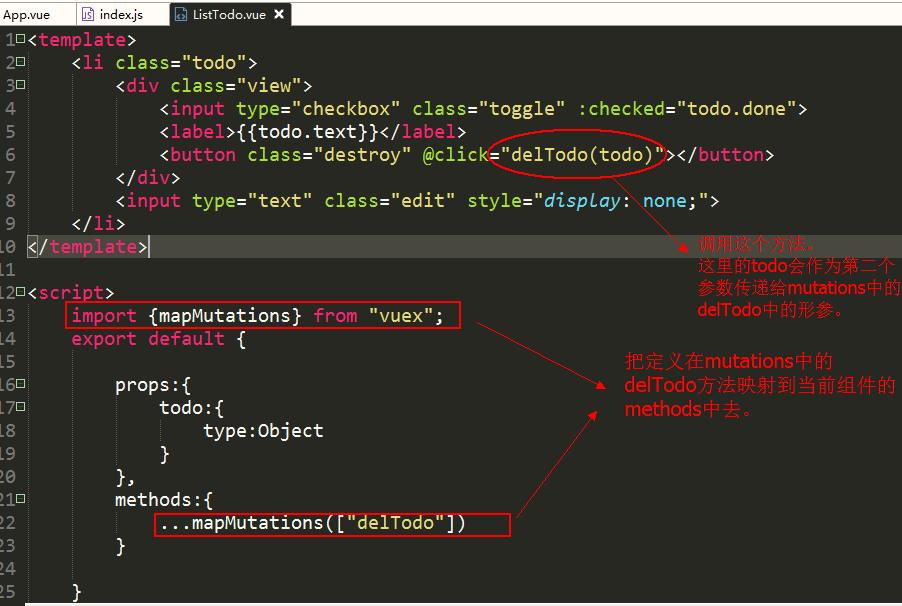
### 删除一条todos数据



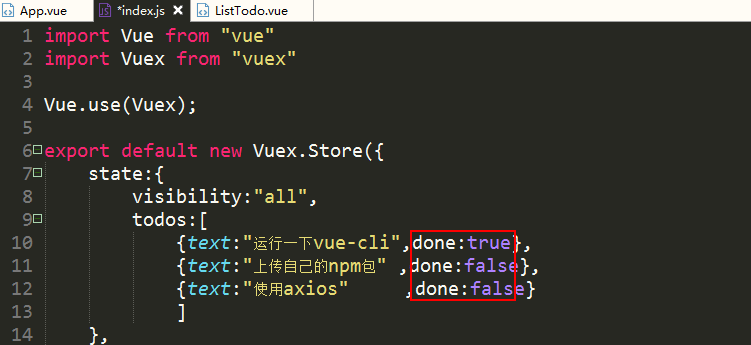
删除操作也是要去修改数据，则对应的也要建立一个mutations。



这个删除操作是在listTodo组件中完成的。在 listTodo组件中给删除的按钮绑定事件：



### 修改某一项的状态：完成，没有完成之间切换



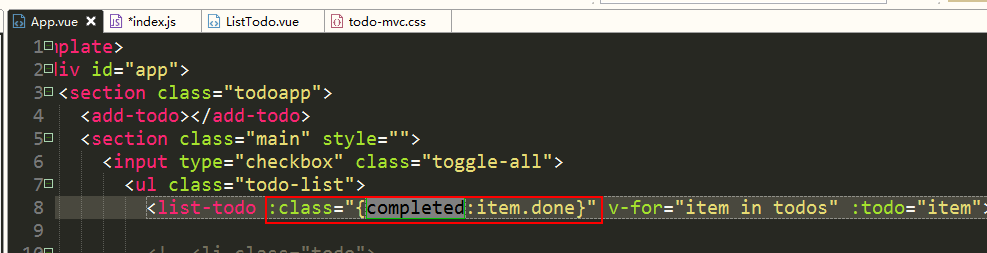
这个操作，本质就是要去修改某一个todo项的done属性。所以也需要在mutations中去写好对应的方法。



这个工作是在listTodo组件完成的，我们去给按钮加点击事件。



在app.vue组件，我们根据todo是否已经完成，设置一个特殊的类：



完成了修改当前项的状态的功能。

### 批量修改状态:全部完成，全部未完成

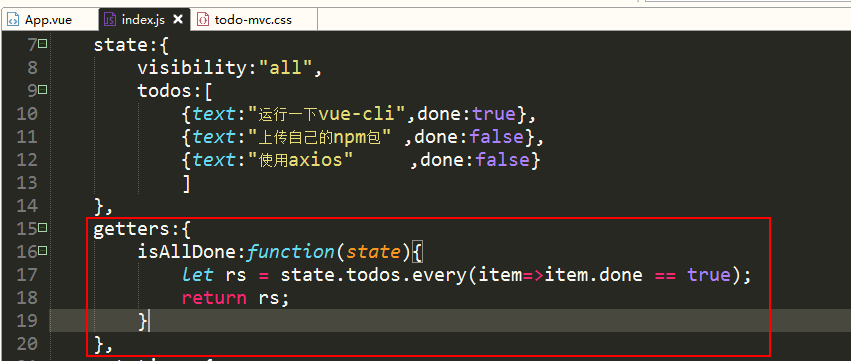


我们通过input的check属性来设置css样式。如果它是处于check，则说明全部的任务已经完成，则它的样式应该是“高亮”的。

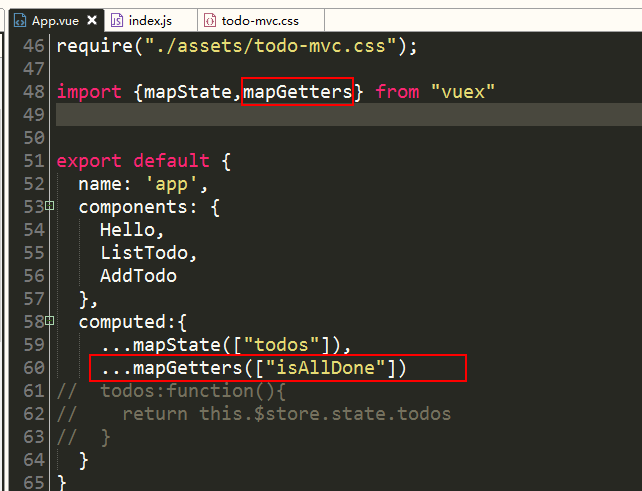
首先，我们要去判断当前情况下，是否所有的任务已经全部完成。如果全部完成，则它应该要处于check状态。

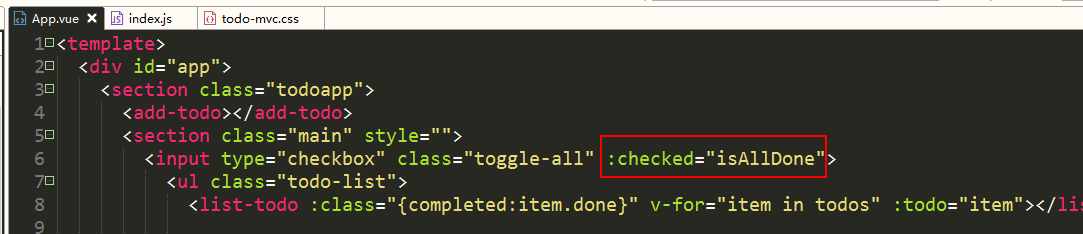
现在，我们需要在原始数据的基础，加工得到一个新数据：是否全部完成。

所以，应该要在vuex.store中的加一个getters



在app.vue中使用：



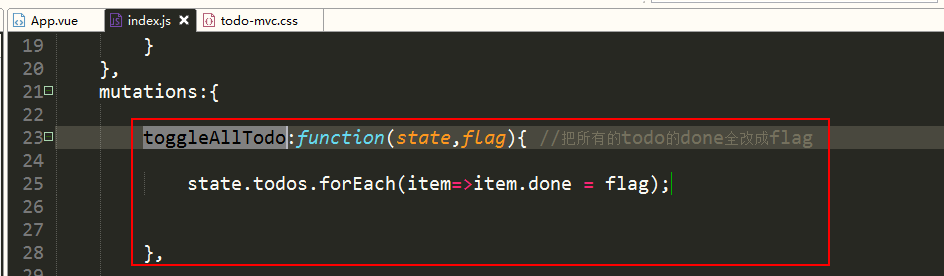


现在只是实现了：根据是否全部完成来设置input的checked。

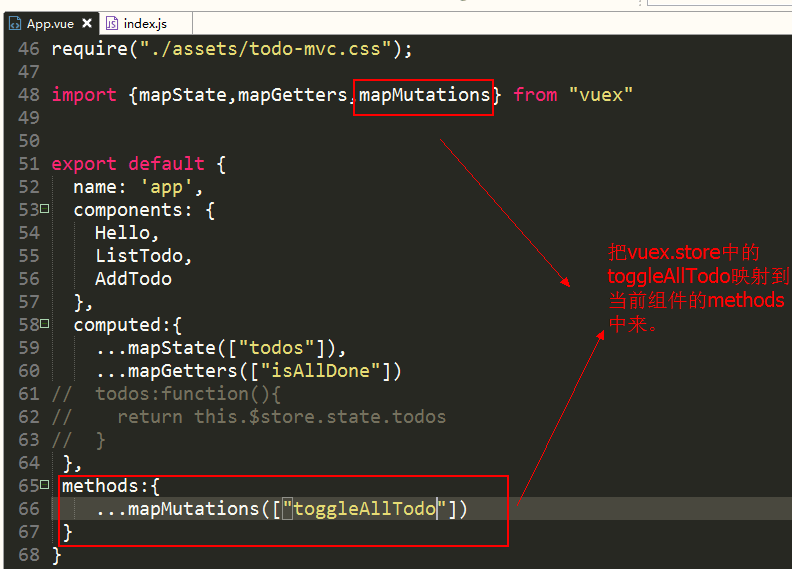
下面，我们还要给它加点击事件：

1. 如果当前是处于checked ，则点击事件发生，要全部改成done:false
2. 如果当前是没有选中，则点击事件发生，要全部改成done:true

上面的操作，也是要修改数据，则对应地要去建立mutations.



在app.vue中：



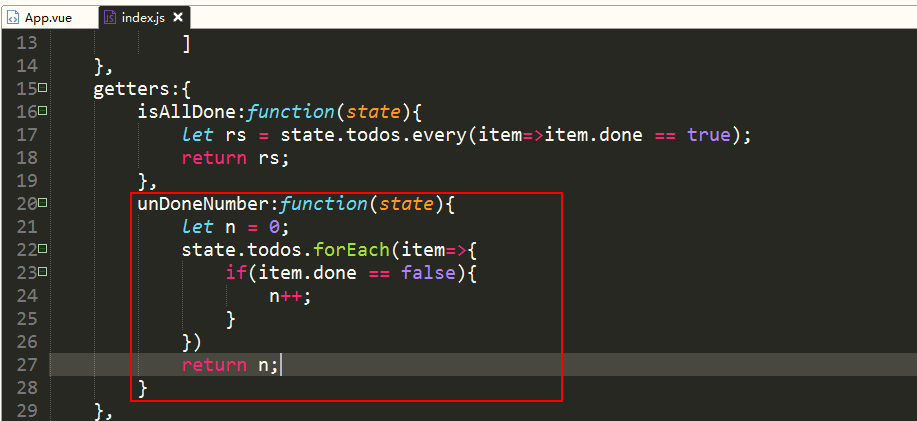
使用toggleAllTodo



### 统计没有完成的数量

不需要对数据进行修改，只是在原始数据的基础上，加工得到新数据。

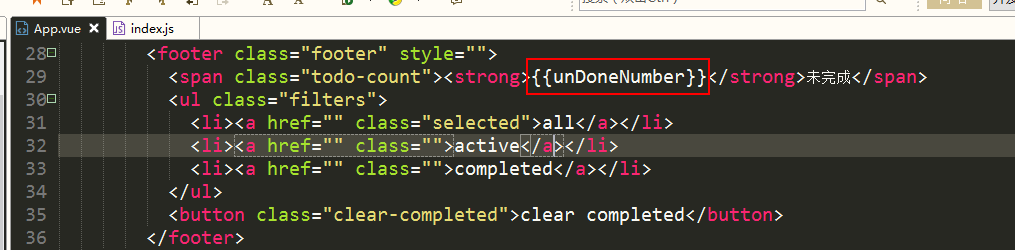
去vuex.store中设置一个 Getters



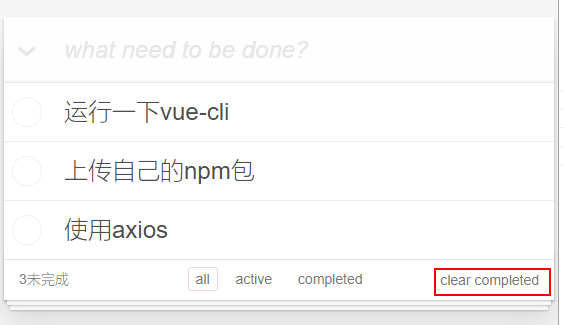
把它map到app.vue组件中，成为一个计算属性。



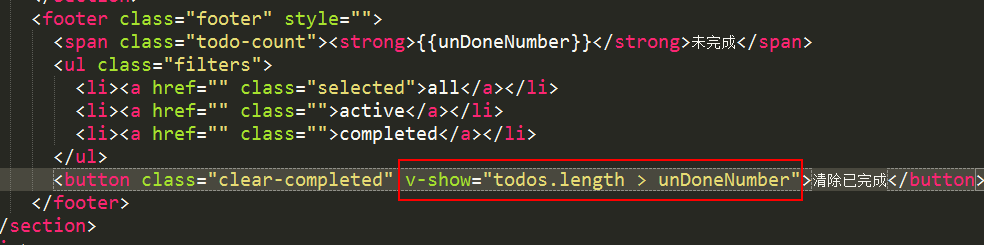
使用：



### 批量删除已经完成的



这个按钮，只有在“完成了数量>0”的情况下才可见。



Todos.length是所有任务的数量

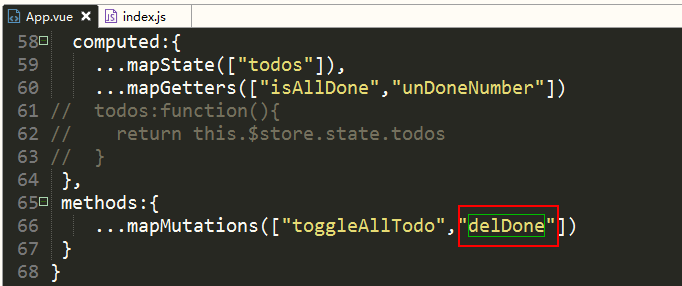
unDoneNumber:还没有完成的数量

当Todos.length >unDoneNumber 说明，有一部分是已经完成了的。此时应该要显示这个按钮.

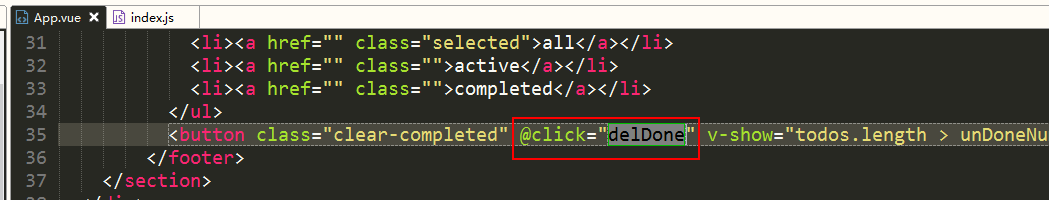
下面，接给它添加点击事件，点击之后，要把已经完成的任务从整个的todos当中删除掉。由于这里涉及修改数据 ，我们需要去建立 一个mutations。



下面，把这个功能映射到app.vue组件的methods 中。



给按钮加点击事件：

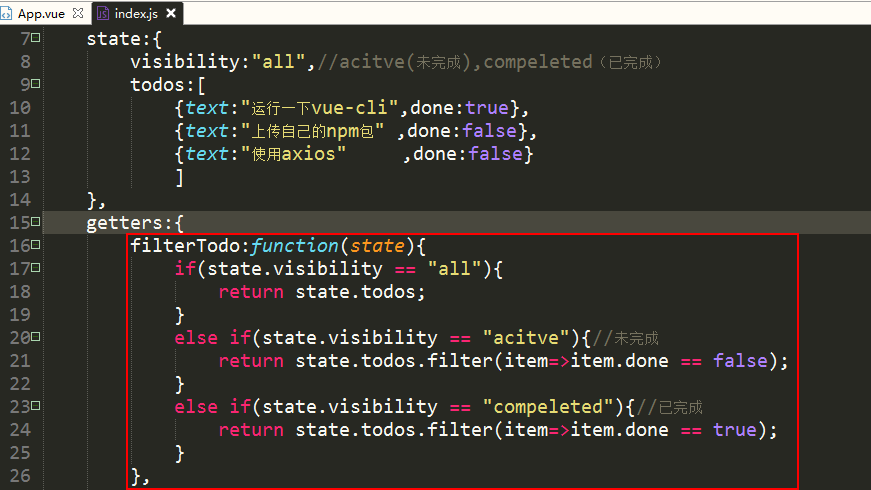


### 三种状态的筛选

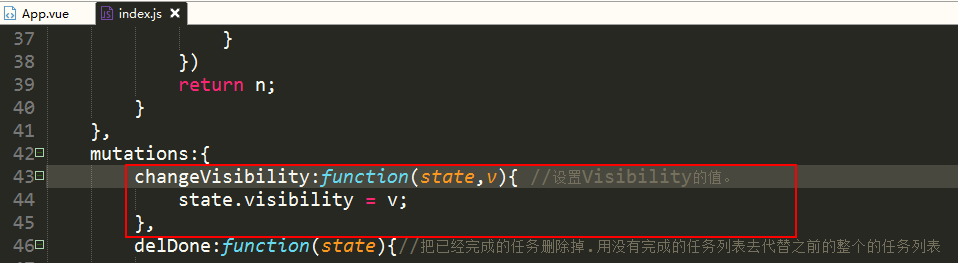


这个操作不会更改数据，只是在原数据的基础上进行筛选，得到新数据。

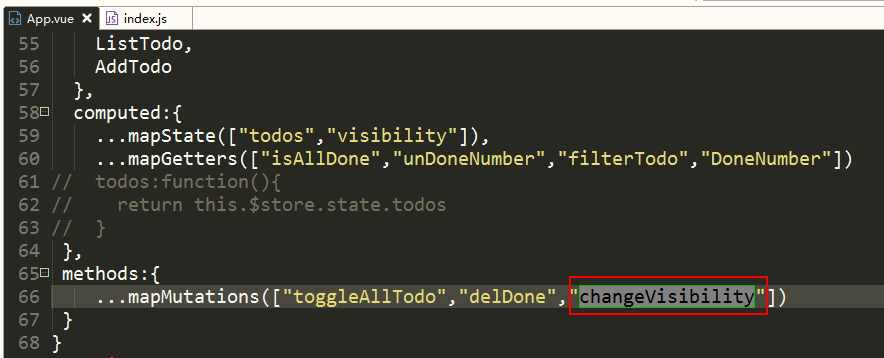
所以，我们去创建getters



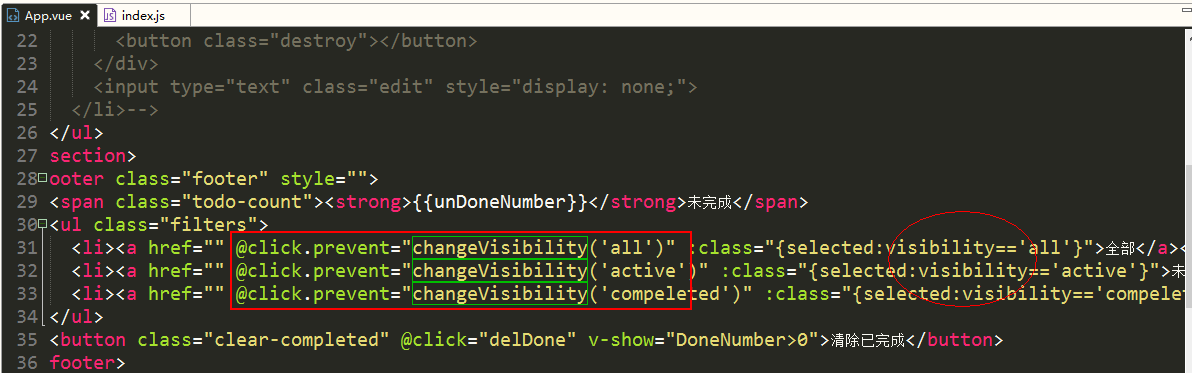
每次点击不同的按钮，就相当于是要去修改visibility，所以，我们要去设置一个mutations



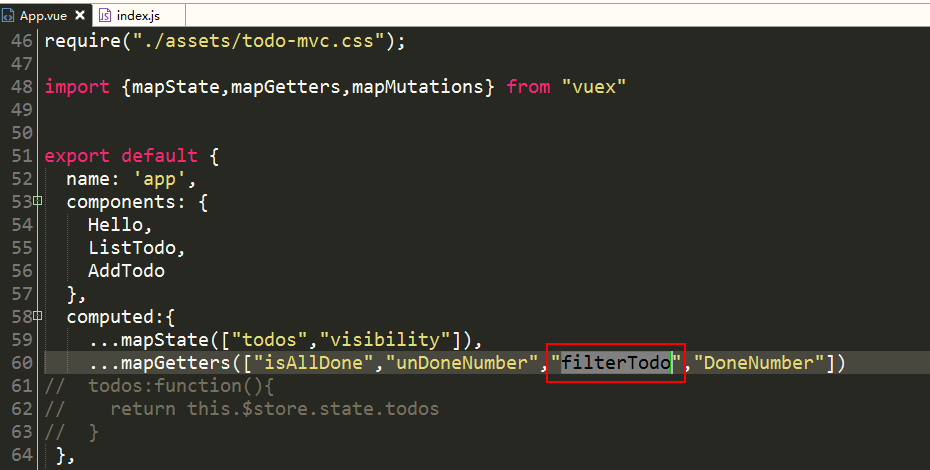
把上面的changeVisibility映射到app.vue组件的方法中去。



使用：



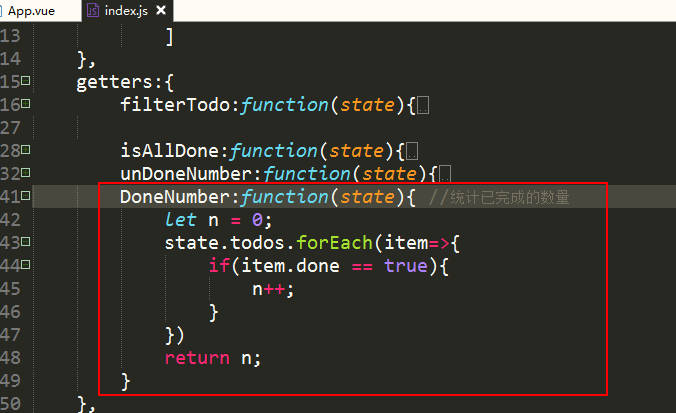
最重要的一点：现在的数据的来源不是整个的todos，而是根据当前条件过滤的数据：

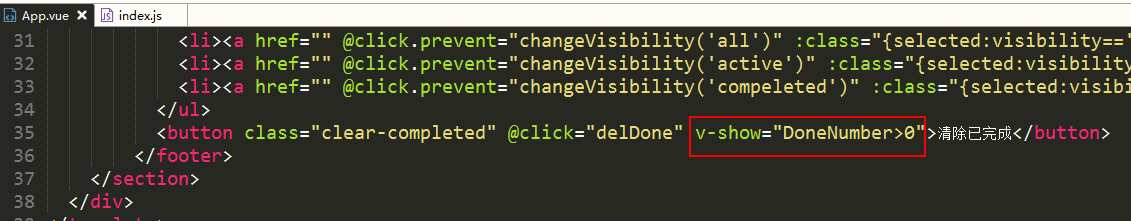




把原来的todos改成了filterTodo。

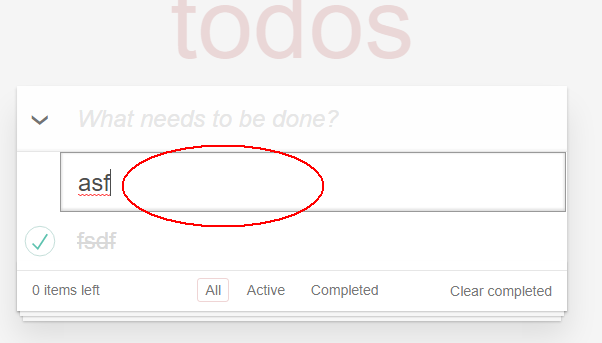
由于对于数据的来源产生的变化，所以我们重新去写一个getter来处理是否显示“清除已完成”





### 编辑todos的内容

当我们在todos的内容上双击时，则会处于编辑状态:



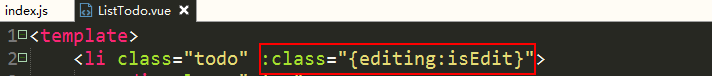
开启编辑状态：加一个editing这个类.

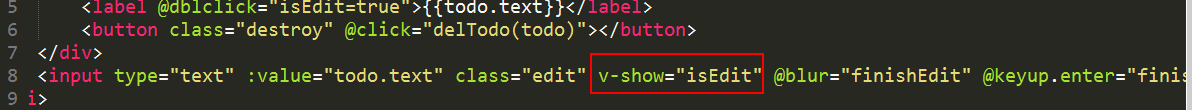
去listtodo中，设置双击加一个类的效果。

给listTodo这个组件加一个isEdit数据项，用它来表示当前是否处于编辑状态。

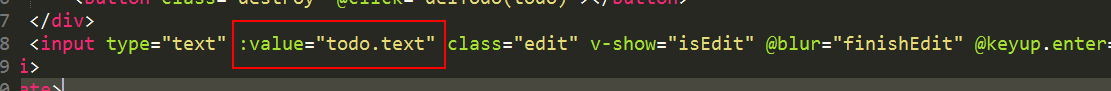
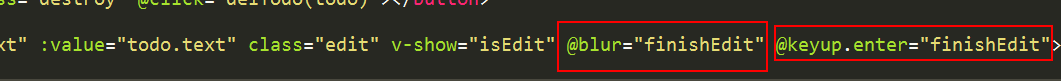


对文件框和整个的li，它们在编辑状态会有不同的样式：

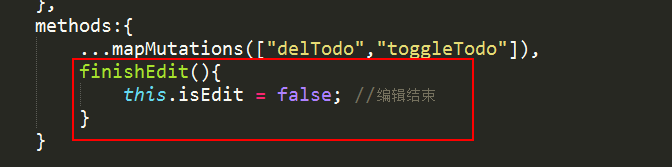




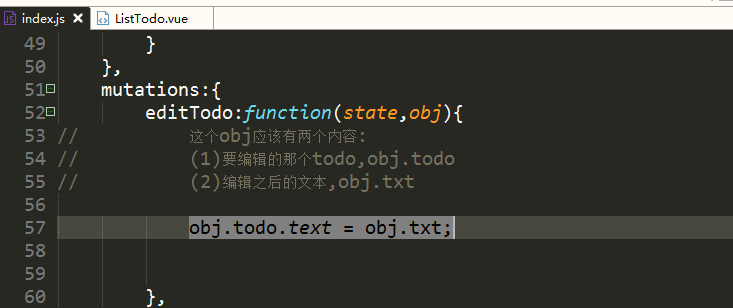
对文本框来说：

1. 初始值就是当前的todo的text.  
   
2. 它失去焦点，或者是回车时，都应该结束编辑  
   

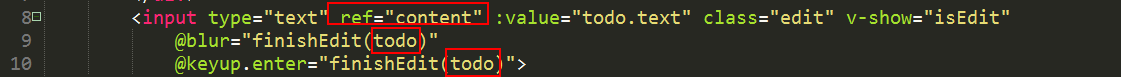
上面的finishEdit是一个方法，如下：



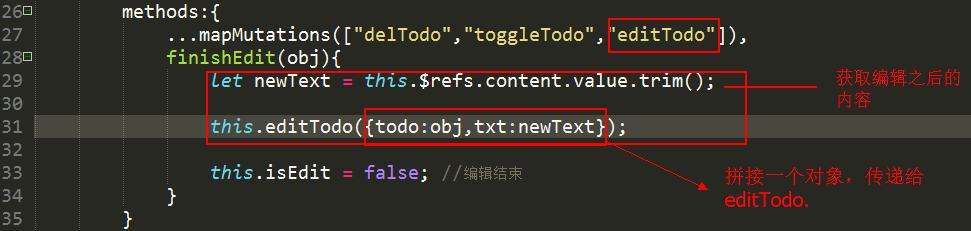
但，结束编辑时，应该去修改数据。我们应该定义 一个mutations去处理这种情况。



下面，我们把这个mutations映射到listTodo组件的methods中去。



修改finishEdit方法:



### 本地存储

我们希望把这个todos的数据保存在本地，下次打开时还可以继续使用。

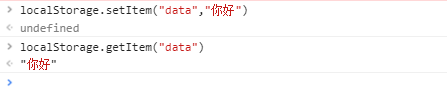
涉及要用到localStorage

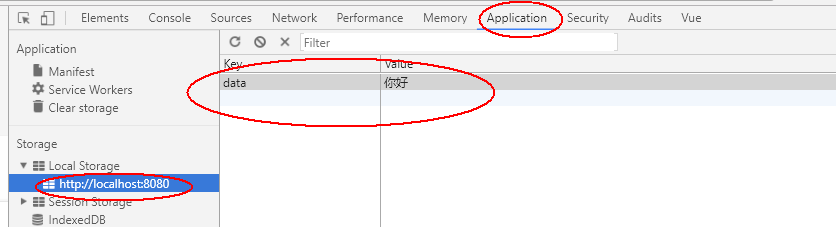
#### localStorage

只有两个方法：

setItem(key,value)

getItem(key)



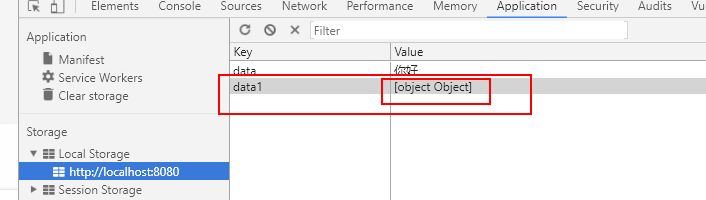


#### setItem(key,value):这里的value必须只能是字符串。

如果你试图去保存整个对象 ，如下 ：



你实际保存的是：

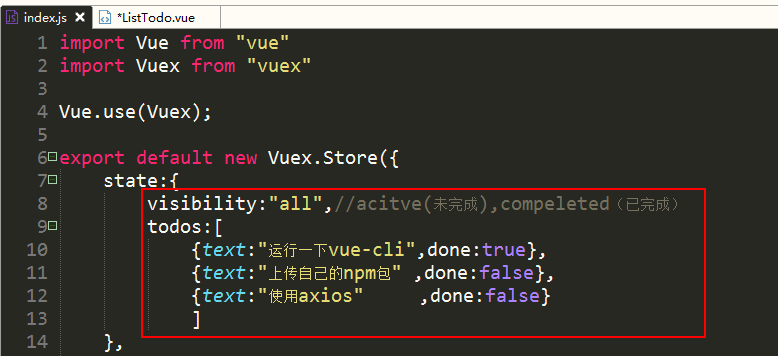


你可以再次获取：



#### 我们这里就有一个大问题

我们的数据是一个标准的对象 ，我们希望把整个对象（包括具体的数据）全部保存下来。



但，localStorage只能保存字符串！！！！

怎么办？

#### 解决方案

1. 在保存之前，把对象转成字符串。
2. 获取之后，把字符串转成对象。

恰好有一对工具可以实现从字符串到对象之间的转换。JSON是js中的内置对象。

* JSON.stringify：把一个对象转成json格式的字符串。
* JSON.parse：把一个json格式的字符串转成一个对象。

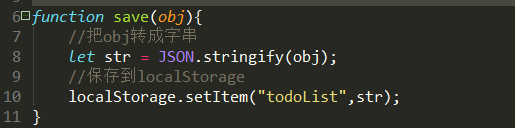
示例如下：

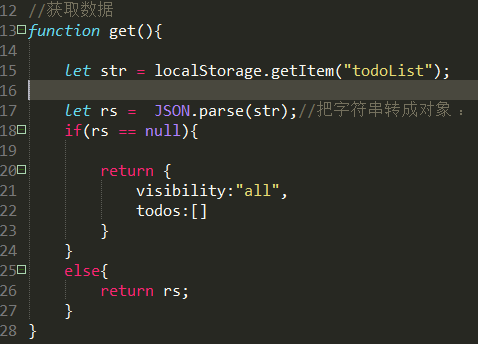




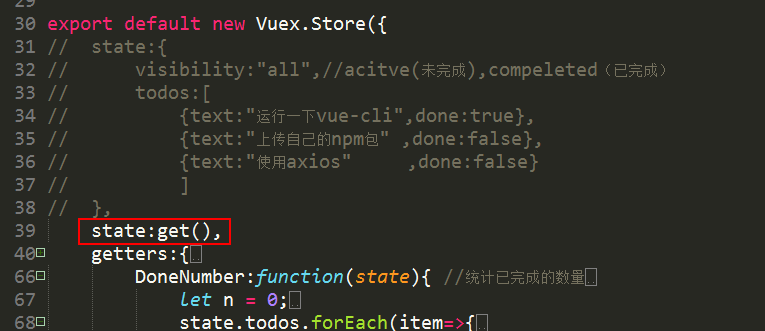
#### 就用这两个工具去实现本地存储

在store/index.js中添加两个函数

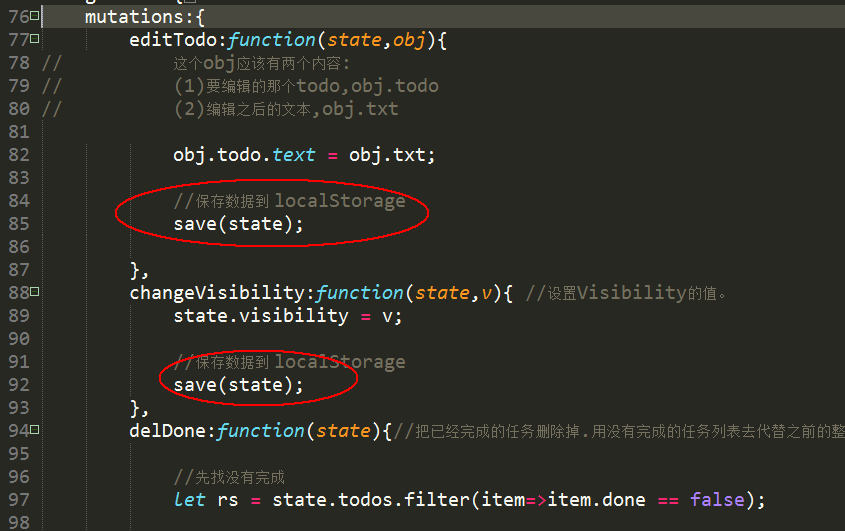




在初始化，通过get去获取数据：

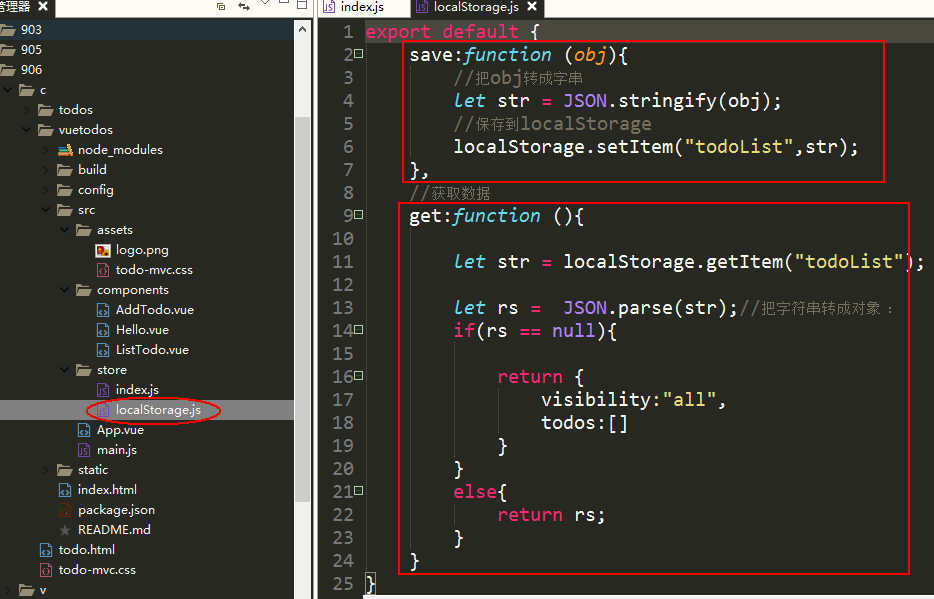


在每一个mutations中都要去save()一下。



#### 用模块化整理代码

新建一个模块



在store/index.js中引入这个模块

