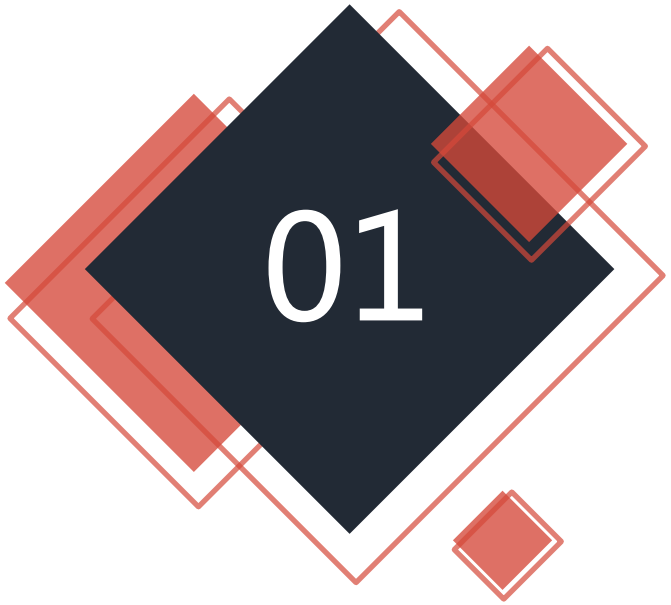
ToDoList

1552643 陈楠



基本功能

基本功能

新增

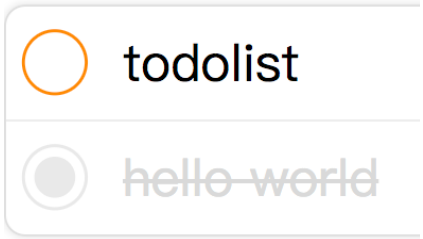
可通过点击回车，点击加号，

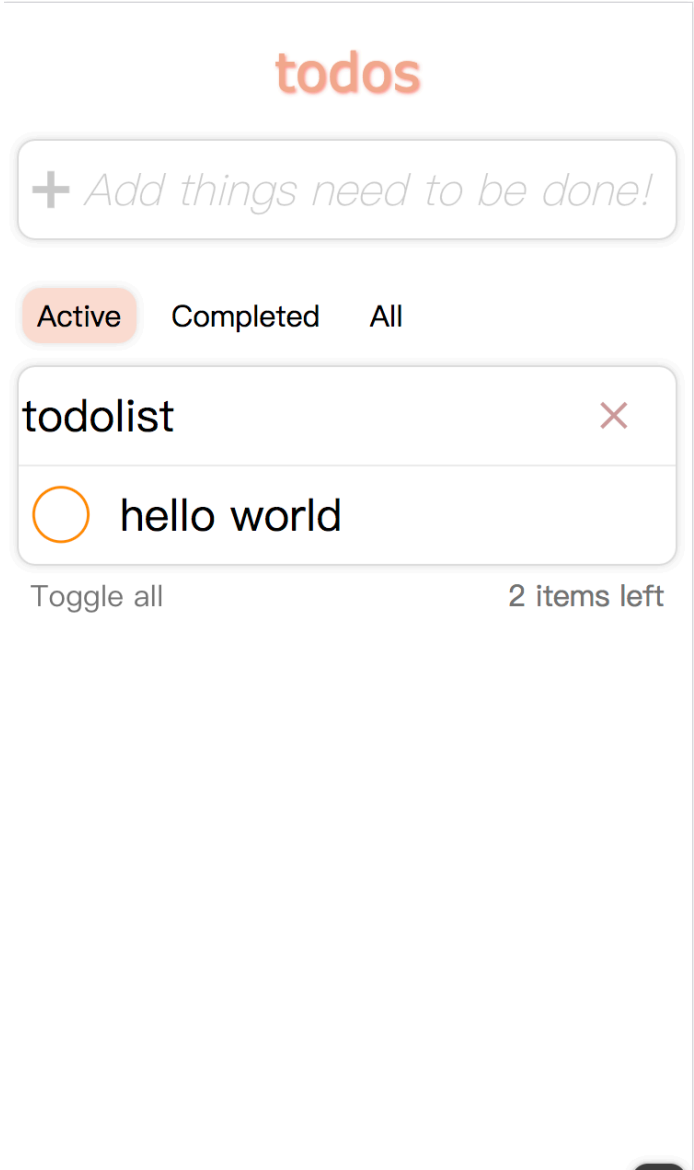
或使其失去焦点

完成\取消

点击圆圈

完成了的项会显示在列表的底部





删除

左划

点击叉号实现删除

展示列表

垂直排列

未完成的项显示在上面

基本功能

todos

+ Add things need to be done!

Active Completed All

周六答辩

todolist

~~hello world~~

~~yeah~~

Toggle all 2 items left Clear completed

yeah

Toggle all 4 items left

全部完成\取消

点击toggle all 全部完成

点击cancel toggle 取消

~~yeah~~

Cancel toggle No item left Clear completed

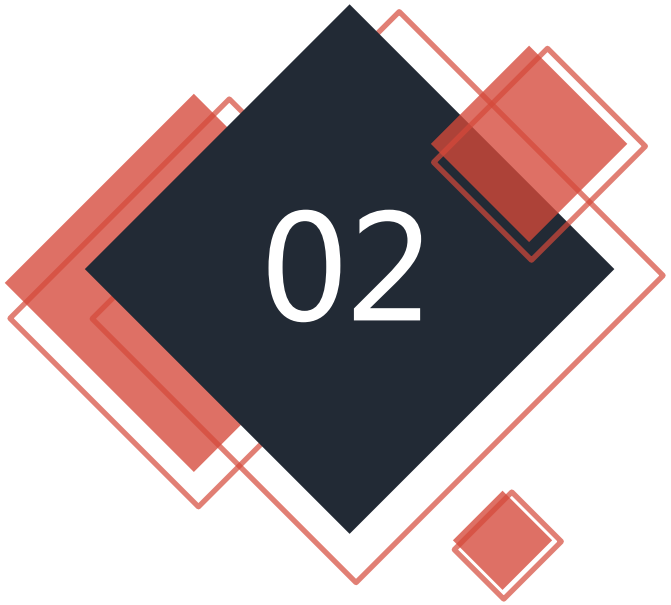
删除已完成

点击clear completed

没有完成项时不显示

保留页面内容

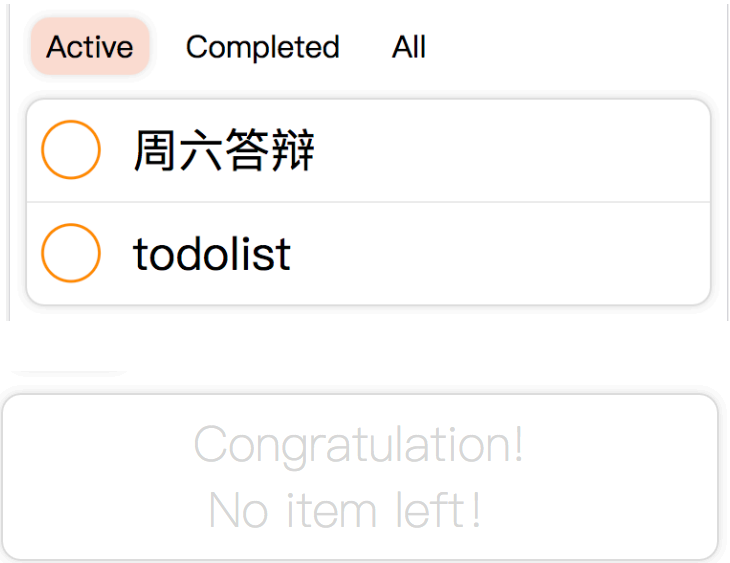
localStorage



高级功能

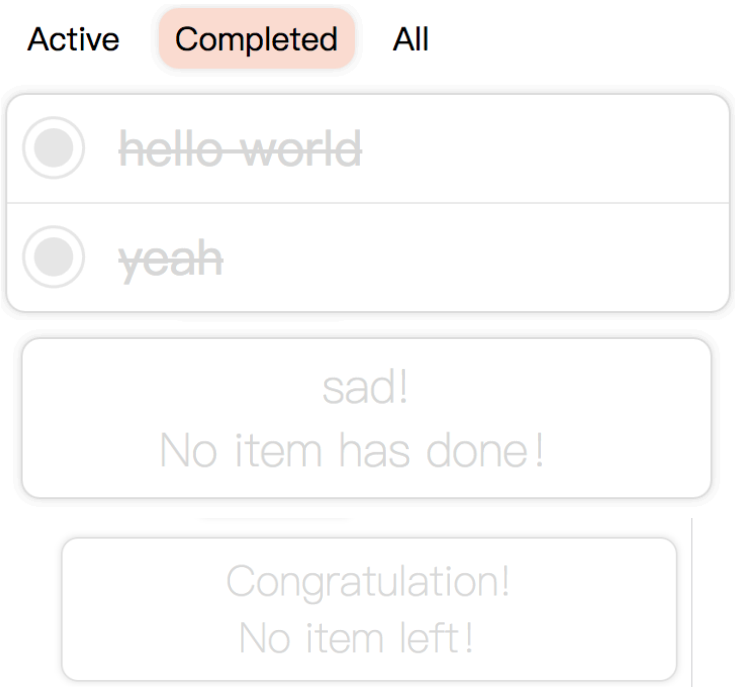
过滤

Active



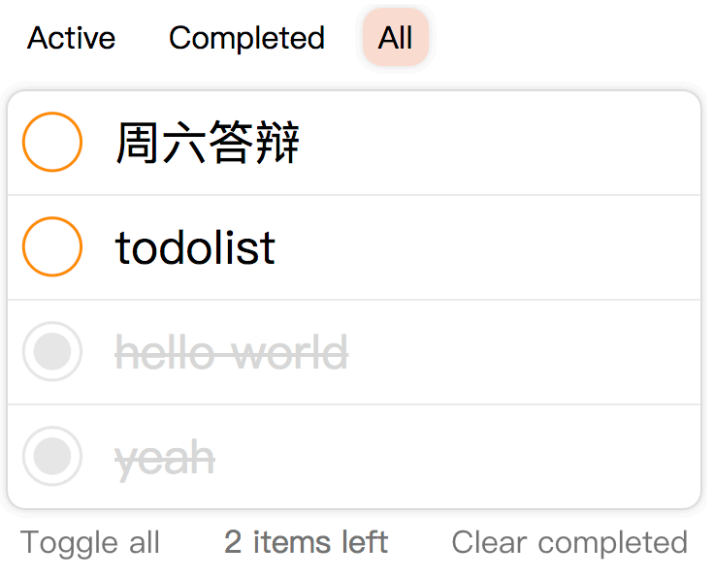
默认显示未完成事项
当所有事项完成后显示祝贺

Completed



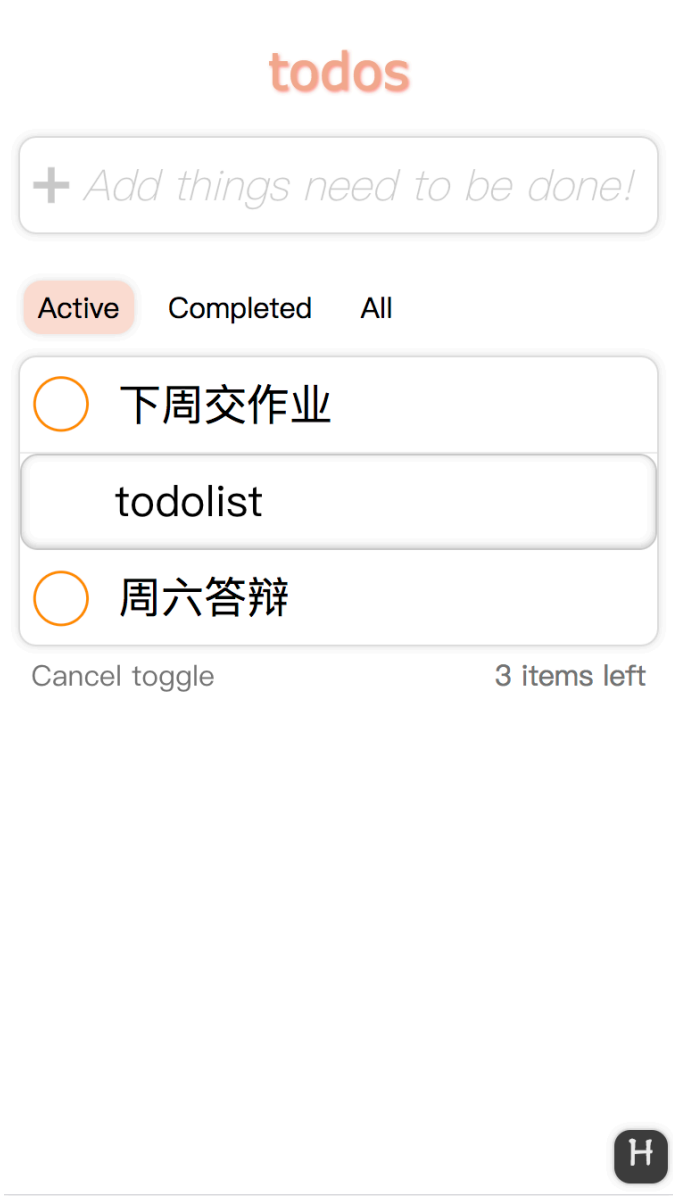
当没有事项完成时表遗憾。
但若此时没有任何事项需要完成表祝贺

All



显示所有事项

编辑



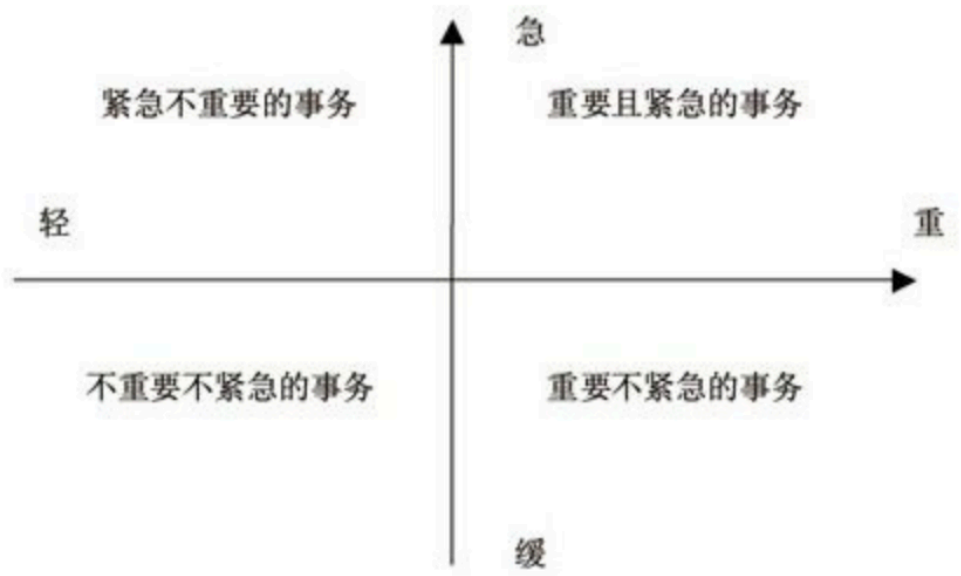
编辑

长按该条事项进入编辑状态

使其失去焦点或按回车完成编辑

亮点

四象限法则

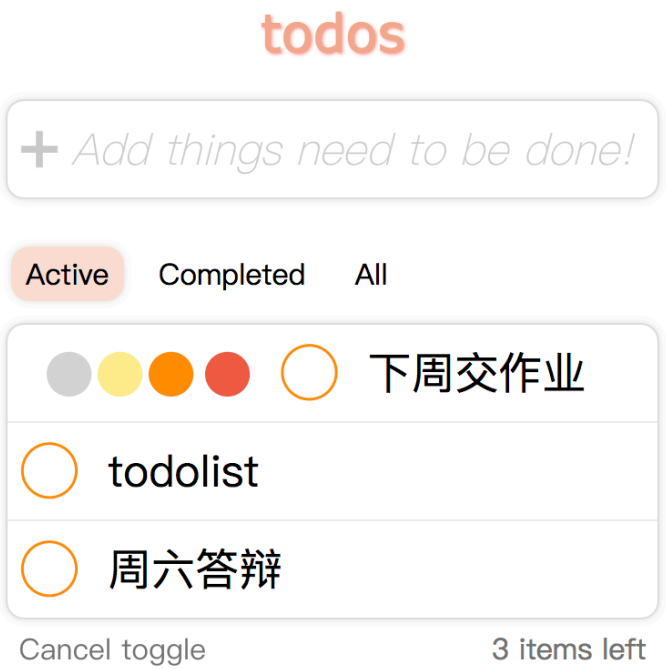


时间管理坐标体系

颜色标识

- urgent and important
- urgent but not important
- important but not urgent
- not urgent and not important

亮点



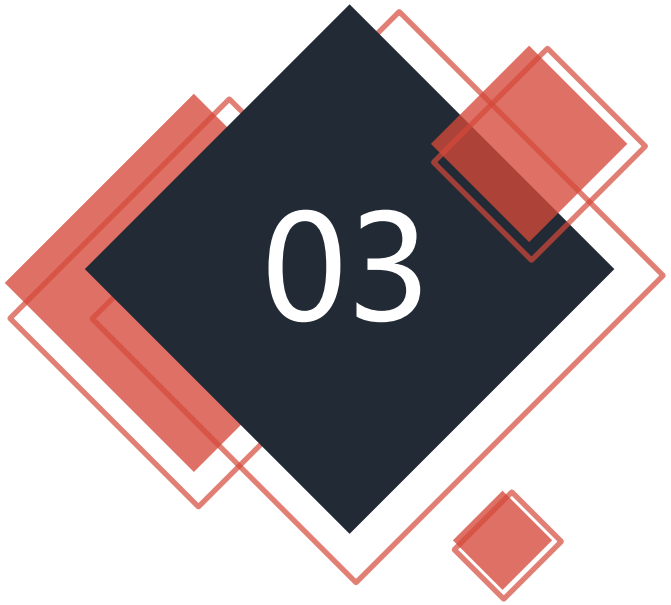
四象限法则

右划可对分类进行修改

（已完成的项目不支持修改）

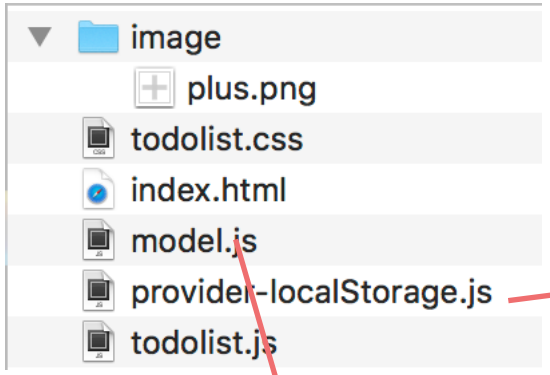
列表会根据紧急且重要、紧急但不重要、重要但不紧急、不重要也不紧急的顺序进行排列

- urgent and important
- urgent but not important
- importan but not urgent
- not urgent and not important



代码讲解

组成



引入model层统一数据层：

```
window.model = {
  data: {
    items: [
      // {msg:'', completed: false,type:''}
    ],
    msg: '',
    filter: 'Active'
  },
  TOKEN: 'Todolist'
  //init(),
  //flush()
};
```

localStorage 实现model中的 provider 接口：

```
Object.assign(model, {
  //从localstorage中获取model的值
  init: function(callback) {
    var data = storage.getItem(model.TOKEN);
    try {
      if (data) model.data = JSON.parse(data);
    }
    catch (e) {
      storage.setItem(model.TOKEN, '');
      console.log(e);
    }
    //若有回调函数吗，则执行回调函数
    if (callback) callback();
  },
  //将model和localstorage存的数据同步
  flush: function(callback) {
    try {
      storage.setItem(model.TOKEN, JSON.stringify(model.data));
    }
    catch (e) {
      console.log(e);
    }
    if (callback) callback();
  }
});
```

添加

```
var newTodo = $('.newmsg');
//
newTodo.addEventListener('change', function() {
  data.msg = newTodo.value;
  model.flush();
});
//添加新todo
function addNewTodo() {
  if (data.msg == '') {
    console.log('input msg is empty');
    return;
  }
  data.items.push({
    msg: data.msg,
    completed: false,
    type: "orange"
  });
  data.msg = '';
  data.items.sort(compare('type', 'completed'));
  update();
}
```

```
//回车时添加
newTodo.addEventListener('keyup', function(ev) {
  if (ev.keyCode != 13) return; // Enter
  addNewTodo();
}, false);
//失去焦点添加
newTodo.addEventListener('blur', function() {
  addNewTodo();
}, false);
//按加号键添加
var plus = $('.plus');
plus.addEventListener('touchstart', function() {
  addNewTodo();
}, false);
```

删除已完成

```
/*清空所有已完成的*/
var clearCompleted = $('.clear-completed');
clearCompleted.addEventListener('touchstart', function() {
  var list = []
  data.items.forEach(function(itemData, index) {
    if (!itemData.completed) {
      //将所有未完成的任务都存入一个新的列表
      list.push(itemData);
    }
  });
  data.items = list;
  update();
}, false);
```

全部完成/取消

```
/*全部完成/取消*/
var toggleAll = $('.toggle-all');
toggleAll.addEventListener('click', function() {
  var completed = false;
  data.items.forEach(function(itemData) {
    if (itemData.completed == false) {
      //存在未完成的，就把所有都变成已完成
      completed = true;
      return
    }
  });
  //修改按钮文字
  if (!completed) toggleAll.innerHTML = "Toggle all";
  else toggleAll.innerHTML = "Cancel toggle";
  //修改任务完成状态
  data.items.forEach(function(itemData) {
    itemData.completed = completed;
  });
  update();
}, false);
```

过滤

```
/*过滤，未完成，已完成，全部*/
var filters = $All('.filters li a');
filters.forEach(function(filter) {
  //根据存储的filter，来设置当前被选中的过滤器
  if(data.filter==filter.innerHTML)
  {
    filter.classList.add(TODO_SELECTED);
  }
  else{
    filter.classList.remove(TODO_SELECTED);
  }
  //点击选择过滤器
  filter.addEventListener('click', function() {
    data.filter = filter.innerHTML;
    filters.forEach(function(filter) {
      filter.classList.remove(TODO_SELECTED);
    });
    filter.classList.add(TODO_SELECTED);
    update();
  }, false);
});
```

显示列表

```
//过滤显示
if (
  (data.filter == 'Active' && !itemData.completed) ||
  (data.filter == 'Completed' && itemData.completed)||
   data.filter == 'All'
) {
  var item = document.createElement('li');
  if (itemData.completed) item.classList.add(TODO_COMPLETED);
  item.innerHTML = [
    '<div class="view">',
    '  <div class="choose">',
    '     <i class="grey"></i>',
    '     <i class="yellow"></i>',
    '     <i class="orange"></i>',
    '     <i class="red"></i>',
    '  </div>',
    '  <input class="toggle" type="checkbox">',
    '  <label class="todo-label">' + itemData.msg + '</label>',
    '  <button class="destroy"></button>',
    '</div>'
  ].join('');
  //type color
  var toggle = item.querySelector('.toggle');
  toggle.classList.add(itemData.type);
```

```
//列表中没有要显示的内容
if ((data.filter == 'All' && !data.items.length) ||
  ((data.filter == 'Active' && !activeCount)) ||
  (data.filter == 'Completed' && !totalCount)) {
  var li = document.createElement('li');
  li.className = "congratulation"
  li.innerHTML = "Congratulation!<br> No item left! "
  todoList.appendChild(li);
} else if (data.filter == 'Completed' && !completedCount) {
  var li = document.createElement('li');
  li.className = "congratulation"
  li.innerHTML = "sad!<br> No item has done! "
  todoList.appendChild(li);
}
```

完成/取消

```
//是否完成这个todo
var itemToggle = item.querySelector('.toggle');
itemToggle.checked = itemData.completed;
itemToggle.addEventListener('change', function() {
  itemData.completed = !itemData.completed;
  //完成了的追加到列表最后
  if(itemData.completed){
  data.items.unshift(itemData);
  data.items.splice(index + 1, 1);}
  else{
    data.items.sort(compare('type','completed'));
  }
  update();
}, false);
```

编辑todo

```
//文本编辑
label.addEventListener('touchstart', function(e) {
  if (editing) //如果有正在编辑中的就删除
  {
    editing.querySelector(".edit").blur();
    editing = null;
  }
  //长按半秒钟编辑文本
  timeOutEvent = setTimeout(function() {
    editItem(item, label, itemData)
    editing = item;
  }, 500);
});
```

```
function clearTime() {
  if (timeOutEvent) {
    clearTimeout(timeOutEvent);
    timeOutEvent = 0;
  }
}
//不到半秒钟
label.addEventListener('touchend', function() {
  clearTime();
});
```

```
//编辑项
function editItem(item, label, itemData) {
  item.classList.add(TODO_EDITING);
  //添加输入框，并聚焦
  var edit = document.createElement('input');
  edit.setAttribute('type', 'text');
  edit.setAttribute('class', 'edit');
  edit.setAttribute('value', label.innerHTML);
  item.appendChild(edit);
  //ios无法自动聚焦，需用户再轻触一下
  edit.focus();

  function editTodo(obj) {
    label.innerHTML = obj.value;
    itemData.msg = obj.value;
    editing=null;
    update();
  }
  //失去焦点添加
  edit.addEventListener('blur', function() {
    editTodo(this);
  }, false);
  //回车时添加
  edit.addEventListener('keyup', function(ev) {
    if (ev.keyCode == 13) {
      editTodo(this);
    }
  }, false);

}
```

左右滑动

```
/*左右滑动*/
label.addEventListener('touchstart', function(e) {
  x = event.changedTouches[0].pageX;
  y = event.changedTouches[0].pageY;
  swipeX = true;
  swipeY = true;
  if (expansion) { //判断是否展开，如果展开则收起
    expansion.classList.remove("swipeleft");
    expansion.classList.remove("swiperight");
    expansion = null;
  }
  e.preventDefault();
}, false);
```

```
view.addEventListener('touchmove', function(e) {
  //如果之前的moving还在就移除
  if ($('.moving'))
    $('.moving').classList.remove('moving');
  //开始移动后的话，就不属于长按事件
  clearTime();
  //开始滑动
  item.classList.add('moving');
  X = e.changedTouches[0].pageX;
  Y = e.changedTouches[0].pageY;
  // 左右滑动
  if (swipeX && Math.abs(X - x) - Math.abs(Y - y) > 0) {
    // 阻止事件冒泡
    e.stopPropagation();
    if (X - x > 10) { //右滑
      e.preventDefault();
      if (expansion && this.classList.contains("swipeleft")) { //右滑收起
        this.classList.remove("swipeleft");
        expansion = null;
      } else if ((X - x > 30) && !toggle.checked) { //没有完成的可以展开
        this.classList.add("swiperight"); //右滑展开
        expansion = this;
      }
    } else if ((x - X > 10)) { //左滑
      e.preventDefault();
      if (expansion && this.classList.contains('swiperight')) {
        this.classList.remove("swiperight");
        expansion = null;
      } else if (x - X > 30) {
        this.classList.add("swipeleft"); //左滑展开
        expansion = this;
      }
    }
    swipeY = false;
  }
  // 上下滑动
  else if (swipeY && Math.abs(X - x) - Math.abs(Y - y) < 0) {
    swipeX = false;
  }
}, true);
```

删除

```
//删除todo
item.querySelector('.destroy').addEventListener('touchstart', function() {
  data.items.splice(index, 1);
  update();
}, false);
```

类别

```
//修改type
var types = item.querySelectorAll('.choose i');
types.forEach(function(type) {
  type.addEventListener('click', function() {
    //修改type
    toggle.className = "toggle " + type.className;
    itemData.type = type.className;
    //按类别排序
    data.items.sort(compare('type','completed'));
    update();
  });
});
```

谢谢!