Lab8

Deadline: 2015-05-04 23:59:59

答案（仅供参考）会在 Deadline 后上传到 FTP

如有疑问，联系 TA（11302010035@fudan.edu.cn）

Part A

本部分主要让大家接触了解 jQuery，并用 jQuery 实现一个简单的选项卡效果。

jQuery 官网：http://jquery.com/

jQuery API：http://api.jquery.com/

W3School jQuery 教程：http://www.w3school.com.cn/jquery/index.asp

jQuery 是一个优秀的 JavaScript 库，通过使用 jQuery，可以非常方便地进行 DOM 操作、事件处理、动画效果、AJAX 等。不仅如此，jQuery 还提供了一些非常有用的工具方法和工具类。除此之外，jQuery 还提供了简单的接口，从而可以方便地进行插件开发。jQuery 对不同浏览器之间的各种差异进行了封装，因此有着良好的浏览器兼容性。其 1.x 版本兼容 IE6+，2.x 版本兼容 IE9+。

1. 使用 jQuery 选择元素

通过 jQuery，可以非常方便地进行元素选择。例如：

```
jQuery('#header') // 选中 ID 为 header 的元素
```

通常情况下，为了书写简便，在 jQuery 中，使用$来代替 jQuery，因此上面的操作可以写为：

```
$('#header')
```

jQuery 支持 CSS 中的各种选择器，因此进行元素选择非常容易，例如：

```
$('.example')
$('p+ul.list>li')
$('input[type="text"]')
$('div.item:last-child a')
```

2. jQuery 对象与原生 DOM 对象的区别

jQuery 选择元素得到的结果是一个类数组对象。类数组通常有如下结构：

```
{
    0 : 'prop 0',
    1 : 'prop 1',
    2 : 'prop 2',
    length : 3,
    // other props (optional)
}
```

即有一个 length 属性（必须），同时有名为 0，1，2，...的属性。其余属性可选。通过 jQuery 选择元素得到的 jQuery 对象，就是这样的一个类数组结构，其中类数组部分的每一项都是一个原生的 DOM 对象。除此之外，它还有一些其它的属性。

3. 基本的 DOM 操作

jQuery 中提供了大量 DOM 操作相关的方法，例如：

```
$('#example').html()
// 相当于 document.getElementById('example').innerHTML

$('#example').html('<h1>header</h1>')
// 相当于 document.getElementById('example').innerHTML =
// '<h1>header</h1>'
```

还有一些别的方法，例如：

```
$().text()
$().css()
$().attr()
$().prop()
$().addClass()
$().hasClass()
$().removeClass()
... ... ... ...
```

具体的方法和使用请大家查阅 API 文档。

以上是对 jQuery 的简单介绍。在该部分中中，选项卡的 DOM 结构和样式已经定

义好了，不需要做任何改动。只需要在 script.js 中相应位置添加 JavaScript 代码即可。

基本要求：不能对 index.html 和 style.css 有任何改变，也不要自己添加任何样式。只需要 JavaScript 代码。

Part B 实现一个 EventEmitter

在 JavaScript 程序中，经常会涉及到事件处理（不仅仅是 DOM 事件），其一般流程是：

1. 注册回调函数
2. 触发相应事件
3. 该事件相应的回调函数被执行

在该部分，需要你实现一个 EventEmitter 类，该类有两个方法：on 和 emit。其基本描述如下：

```
function EventEmitter() {
    // Your codes here
}

// 为事件（name）注册回调函数（callback）
// 第一个参数 name 为字符串，表示事件名
// 第二个参数 callback 为回调函数，当该事件发生的时候被调用
// 该方法可以链式调用，即 obj.on(a, b).on(c, d)…
EventEmitter.prototype.on = function(name, callback) {
    // Your codes here
};

// 触发事件（name），同时向该事件的回调函数传递参数（data）
EventEmitter.prototype.emit = function(name, data) {
    // Your codes here
};
```

打开 Console 查看测试输出信息，Part B 部分正确的输出信息为：

```
test info for part B
chain method passed
from event1 handler1: event1 passed
from event1 handler2: event1 passed
from event2 handler1: event2 passed
```

Part C 实现一个 currying（柯里化）函数

该部分主要涉及如下概念：作用域和作用域链、闭包（closure）、柯里化（currying）。其中作用域和作用域链是闭包的基础，闭包又是柯里化的基础。因此需要首先明白这三个概念，可以查阅 WIKI，或则参考《JavaScript 高级程序设计》中的讲解。（该书电子版之前已经上传在 FTP Materials 目录下。强烈建议阅读。4.2 讲解作用域和作用域链，7.2 讲解闭包，22.1.5 讲解柯里化）

该部分需要实现一个通用的柯里化函数。例如有如下函数：

```
function add5(a, b, c, d, e) {
    return a + b + c + d + e;
}
```

需要你实现一个 currying 函数：

```
function currying(fn) {
    // Your codes here
}
```

该函数接收若干个参数，但是第一个参数一定为一个函数。通过使用该函数，我们希望可以进行如下调用：

```
currying(add5, 1, 2, 3, 4, 5) // 15
currying(add5, 1, 2, 3)(4, 5) // 15
currying(add5, 1, 2)(3, 4, 5) // 15
currying(add5)(1, 2, 3, 4, 5) // 15
currying(add5)(1, 2, 3)(4, 5) // 15
currying(add5)(1)(2)(3)(4)(5) // 15
… … … …
```

即当实际接收到的参数小于应当接收的参数个数时，就可以继续接收参数，否则返回最终的计算结果。

打开 Console 可以看到该部分的测试输出信息。

提示：

1. 一个函数的应当接收的参数个数可以通过 fn.length 得到，例如
   ```
   add5.length==5
   ```
2. 一个函数的实际接收的参数可以通过 arguments 对象得到，例如：
   ```
   function test(){
       console.log(arguments);
   }
   test(1, 2) // [1, 2]
   ```
   需要注意的是 arguments 是一个类数组结构，因此并不能直接调用数组的方法，但可以通过 call 或 apply 方式调用，例如：
   ```
   Array.prototype.slice.call(arguments, 0, 2)
   [].slice.call(arguments, 0, 2)
   Array.prototype.slice.apply(arguments, [0, 2])
   [].slice.apply(arguments, [0, 2])
   ```
   注意区分 call 和 apply 在使用上的不同。