**基于Vue+SpringBoot框架的**

**个性化选课网站**

编

码

规

范

课 程： 软件工程

题 目：基于Vue+SpringBoot框架的个性化选课网站

专 业： 软件工程

班 级： 1802

项目小组： G16小组

目录

[目录](#_Toc9691039)

[参考资料 3](#_Toc9691040)

[一、 命名规范： 4](#_Toc9691041)

[二、 格式规范： 4](#_Toc9691042)

[三、 代码规范： 5](#_Toc9691043)

[四、 文件规范： 7](#_Toc9691044)

[五、 关于性能 7](#_Toc9691045)

[JS性能优化原则 7](#_Toc9691046)

[1. 大循环体性能优化： 7](#_Toc9691047)

[2.方法绑定 7](#_Toc9691048)

[3.js面向对象开发： 8](#_Toc9691049)

[4.避免使用全局变量以及在DOM文档中重复对相同节点查找 9](#_Toc9691050)

参考资料

1. Vue.js 风格指南

<https://cn.vuejs.org/v2/style-guide/index.html>

1. JavaScript 代码规范

<https://www.runoob.com/js/js-conventions.html>

1. HTML(5) 代码规范

<https://www.runoob.com/html/html5-syntax.html>

1. CSS编码规范

<https://github.com/fex-team/styleguide/blob/master/css.md>

# 命名规范：

1. 变量定义可以使用var或let（包括全局）；
2. 多个变量定义用逗号分开，每个另起一行，逗号在最后
3. 函数定义采用function开头定义（待定）
4. 变量命名方式采用驼峰式，例如：nowPageNum
5. 尽量避免使用全局变量，如需要使用，带双下划线前缀，首页字母小写，例如：\_\_cacheList
6. 全局模块、单例变量、公共组件等首写字母大写，例如：PageController
7. 模块内全局变量，带下划线前缀，例如：\_cacheList
8. 模块内dom对象，带$符号前缀，例如：$dom
9. 函数形参与内部私有变量，驼峰式就可以，例如：pageIndex
10. 常量采用全大写，用\_分割，例如：MAX\_PAGE
11. 使用简洁的格式载入 JavaScript 文件 ( type 属性不是必须的)
12. 不使用中文命名。

# 格式规范：

1. 代码注释采用jsdoc方式，需注明代码的依赖
2. 代码缩进采用4个空格方式
3. 操作符前后保留1个空格，例如：1 + 1 = 2;
4. 每行代码结束添加分号
5. 程序块要采用缩进风格编写，缩进的空格以统一的开发工具为准。函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格。
6. 为了便于阅读每行字符建议小于数 80 个，如果一个 JavaScript 语句超过了 80 个字符，建议在 运算符或者逗号后换行；
7. 同功能代码写在一个区域，变量定义统一在顶部，例如

var \_mod = 1,

\_mod2 = 3;

function getList(){}

function getPage(){}

各种类型定义使用，如下：

var num = 1;

var str = 'test';

var array = [];

var json = {};

if else语句使用，if只有一句代码也需要{}

if(条件1 || 条件2){

//执行1

}

else {

//执行条件2

}

for语句使用

for(var i = 0,l = list.length;i < l;i++){

//执行语句，list[i]

}

for(var key in jsonObj){

//执行语句，jsonObj[key]

}

while语句使用

var i = 0,

l = list.length;

while(i < l){

//执行语句

i++;

//注意死循环

}

do while语句，尽量避免使用，采用for或者while代替 switch语句使用，当if else达到3个层级以上用swtich

switch(index){

case 1:

//do something

break;

case 2:

//do something

break;

default:break;

}

try catch语句，与if else类似 尽量避免使用continue、with、goto、eval、new Function等前端业界冷门或者有诟病语句

# 代码规范：

1. setTimeout，setInterval方法需要传方法引用，禁止传字符串 计时器时间间隔，需要大于100ms（做动画除外），小于100ms，用户无法感知，以及耗资源 多元操作符使用

//容错、赋默认值，注意容错值是否可以false，0，''等值

var event = window.event || e;

//函数执行

isName && Test();

//与下面语句意思一样，建议使用以下这种

if(isName){

Test();

}

//三元操作符

var val = isName ? 'name' : 'error';

//不建议三元层级关系太多，容易混乱以及可读性差，例如

var val = isName ? 'name' : isChines ? 'china' : 'error';

1. 对于超过1次使用的dom或者bom对象，需要用变量缓存起来。例如：

var dom = document.getElemntById("test");

dom.style.left = '1px';

dom.style.top = '2px';

1. 回调中包含另外回调（匿名函数），尽量避免，影响可读性，例如：

$.ajax(function(){

$.post(function(){

//do something

});

});

1. 多重循环，尽量避免，例如：

for(var i = 0,l = 10;i < l;i++){

for(var j = 0;k = 10;j < k;j++){

//do something

}

}

1. 禁止直接用onxxx绑定事件，例如：

document.onkeydown = function(e){

//do something

}

1. 全局情况下，判断变量是否为存在，例如：

//错误的使用

if(foo)return true;

//正确的使用

if(window.foo)return true;

if(typeof foo !== "undefined")return true;

# 文件规范：

1. 文件名与文件内的模块名保持一致，字母全小写，用-分割，例如：mod-index.js 同页面或者同大模块的文件，放在一个独立文件夹中，例如： index文件夹 |--mod-index.js |--db-index.js wxss文件放head中

# 关于性能

## JS性能优化原则

1. 减少改变dom结构的次数
2. 减少选择器的层数，尽量使用id,或带id的选择器，如使用$("#ul\_test li")而不要使用$(".ul\_test li")
3. 注意内存回收，特别是将dom节点或dom节点绑定的方法赋值给某个变量，要注意内容回收
4. 使用面向对象开发模式，减少变量污染
5. 减少绑定方法的次数
6. 资源文件载入最小化，如选择合适的图片格式，减少图片大小，对js及css进行压缩等。

## 1. 大循环体性能优化：

1. for循环体要先将循环次数放到一个变量中，避名大循环体下多次取值影响性能：

//错误的写法

for(var i = 0; i < m.length; i++){

}

//正确的写法

for(var i=0,len=m.length; i<len; i++){

}

1. 如果循环体每一次循环都需要多次对dom结构添加元素，先使用变量保存所有要添加的临时DOM节点，在循环完后一次性修改dom结构，避免每次修改dom结构时浏览器对整个dom结构进行重新解析，从而影响性能

## 2.方法绑定

使用on()或delegate()代替bind()来为节点绑定方法

使用bind绑定方法经常会面临几个常见问题：

1. 当dom节点发生变化时，绑定事件失效，
2. 每次修改dom节点都要重新绑定事件
3. 使用ajax之类添加的节点，只能在ajax回调之后绑定，当重复调用ajax时会面对上一个问题

使用on()或delegate()的实现事件代理 使用 delegate()和on()方法的事件处理程序适用于当前或未来的元素（比如由脚本创建的新元素）,当元素节点发生变化时无需重新绑定

$(document).on("click","#ul\_list li",function(e){

});

$("#test").delegate("li","click",,function(e){

});

## 3.js面向对象开发：

在开发过程中尽量使用面向对象开发模式：

1. 单例模示：对各个子功能进行模块封装，实现模块化开发，减少全局变量的使用，方便代码管理及模块修改，使用闭包灵活改变公有或私有变量的生命周期。
2. 类：对可重用插件进行封装,方便拓展及重用；

类结构示例（采用单例和原型链结合），减少实例化时使用new 实例化对象

var Mobile = function(){ //定义单例

var \_styleLink=null;//所有对象实例公有参数

var setParam=function(options){//所有对象实例公有函数

}

var \_class = function(options){//内部类

var classParam="test";//对象私有变量

this.options=$.extend({

param1:null,

param2:null,

callback:null

},options)

this.init();

}

\_class.prototype = {//对象私有方法

init : function(){//对象初始化

this.addEvent();

},

addEvent:function(){//事件绑定

}

}

return {

setting:function(options){//提供给外部调用的方法，修改所有实例的公有参数

setParam(options);

},

create : function(options){//提供给外部调用的方法，创建实例

return new \_class(options);

}

}

}();

//调用实例

Mobile.setting({//设置对象所有实例的公有参数

param1:"1111",

param2:"2222"

})

var mb = Mobile.create({//创建实例1

param1:"aaa",

param2:"bbb",

callback:function(){}

});

var mb2= Mobile.create({//创建实例2

param1:"ccc",

param2:"ddd",

callback:function(){}

});

## 4.避免使用全局变量以及在DOM文档中重复对相同节点查找

1. 使用全局变量的开销比使用局部变量的开销大。
2. 重复对同一个节点进行dom文档搜索对浏览器消耗也很大，应在第一次查找时将结果保存在一个变量后，后面重复使用时调用变量；

var $li\_tags=$("#ul\_test li");

for(var i= 0; len = li\_tags.length; i < len ;i++){

}

$li\_tags.find("a").remove();