第8章 框架开发

课程提要

- 封装自执行函数
- 暴露统一的接口、设置严格模式
- 封装核心部分函数
- 添加拓展方法
- 扩展工具方法
- 选择元素
- 扩展事件

8.1 封装自执行函数

在开发的时候，我们通常使用大量的外部框架，也会同时创建很多JS文件，这时候我们的变量名很容易重名，造成项目变量的混乱。

为了防止这一情况发生，我们使用自执行函数，防止污染外部变量。

```
(function (globol) {

})(typeof window!==“undefined"?window:this);
```

同时传入一个window对象，防止在IE情况下，window被修改。

以后再使用window就可以使用globol来代替window了。

8.2 暴露统一的接口

暴露统一的接口，目的是为了方便用户的记忆，减少污染外部变量的几率，简化操作。

这个统一的接口，又可以叫做命名空间，给一个封闭的代码空间命名，就可以通过这个名字来找到里面的内容。

```
globol.frameWork = globol.h = frameWork;
```

在书写代码的时候，最早写JS的习惯是可以不添加 var 或者 let 这些关键字来修饰变量名的，但是这种方式会默认把没有添加var或者let的变量，设置为window上的变量，成为全局变量，很容易不注意就污染其他变量。

我们可以添加严格模式，避免这种情况的发生。

```
"use strict”;
```

8.3 封装核心部分函数

如果我们希望核心部分函数为可以通过new关键字来创建的话，那很容易。

```
function frameWork() {

}
```

这样就搞定了。在使用的时候使用new FrameWork();即可为了简化用户的操作，我们将设置无new的创建方式。

核心思想是： 在用户调用FrameWork()这个函数的时候 我们就已经帮他创建好一个对象。

```
frameWork = function (selector) {
    return {};
};
```

但是这种方式，咱们在扩展方法的时候很不容易扩展。我们换一种方式。

单独定义一个构造函数，在用户调用FrameWork的时候，初始化咱们定义的构造函数。

为了后面好操作FrameWork的原型对象，我们使用core来代替下它的原型对象，设置它的构造函数及版本号。

```
FrameWork.core = FrameWork.prototype = {
    version:version,
    constructor:FrameWork,
    FrameWork:FrameWork
};
```

接下来我们创建真正用来创建FrameWork对象的构造函数。

```
var init = FrameWork.core.init = function (selector) {
    return this;
};
```

并让他们有共同的原型对象。

```
init.prototype = FrameWork.core;
```

当我们创建FrameWork对象的时候 就直接初始化init。

```
FrameWork = function (selector) {
    return new FrameWork.core.init(selector);
};
```

使用方法如下：

```
h(“div”);
```

8.4 添加拓展方法

为了统一扩展FrameWork的方法，我们添加了extend来方便我们后续的添加。

```
FrameWork.extend = FrameWork.core.extend = function (obj){
    for(var key in obj){
        this[key] = obj[key];
    }
};
```

主要是通过调用extend这个函数，传入一个JSON对象，我们把JSON对象中的键值对添加到FrameWork对象上面，这样我们就可以扩展FrameWork对象了。

8.5 扩展工具方法

8.5.1 遍历类数组的对象

我们经常会用到遍历类数组的对象，如果使用JS原生的forEach是不支持的，我们就来自己扩展一下这个方法。

如果我们要扩展这个方法的话，需要把类数组的所有元素一个个读取出来，把每一次读取出来的值，都传递给使用函数的地方，那咱们就可以使用回调函数来操作了。

```
each:function (callback) {
    for (var i=0;i<this.length;i++){
        callback(this[i],i,this);
    }
},
```

我们把这个函数添加到扩展函数中，需要调用extend传入这个JSON对象。

```
FrameWork*.core.extend({
    each:function (callback) {
        for (var i=0;i<this.length;i++){
            callback(this[i],i,this);
        }
    }
});
```

8.5.2 处理字符串的操作方法

在操作DOM的时候，我们是通过传入字符串的第一个符号来区分他是id类型还是class类型，还是元素类型。

我们来封装一个去掉字符串第一个字符的方法。也同样添加到extend里面。

```
FrameWork.core.extend({
    removeFirstChat:function (string){
        return string.slice(1,string.length)
    }
});
```

8.6 选择元素

8.6.1 分区字符串类型

我们来封装一个基础版的DOM查询功能。这些操作是不需要对外公开的操作。

首先我们需要区分选择器的类型。

先区分是否是标签类型：

```
selector[0] === "<" && selector[ selector.length - 1 ] === ">" && selector.length >= 3
```

区分是否是id类型：

```
/^#/.test(selector)
```

区分是否是class类型：

```
/^\./.test(selector)
```

剩余就是元素类型。

8.6.2 判断类型

标签类型：

```
if(selector[ 0 ] === "<" && selector[ selector.length - 1 ] === ">" && selector.length
>= 3){
    target[0] = selector;
    target.length = 1;
}
```

id类型：

```
if (/^#/.test(selector)){
    var dom = document.getElementById(target.removeFirstChat(selector));
    target[0] = dom;
    target.length = 1;
    return target;
}
```

元素类型：

```
var doms = document.getElementsByTagName(selector);
var i = 0;
for (;i<doms.length;i++){
    target[i] = doms[i];
}
target.length = doms.length;
```

8.6.3 使用方式

使用的时候 就可以直接通过h()函数来调用了。

```
var a = h("<p>213</p>");
console.log(a);
console.log(h(".box"));
console.log(h("#header"));
console.log(h("div"));
console.log($(".div"));
var box = h(".box");
```

8.7 扩展事件

DOM元素的事件，也是常用的API，在封装事件函数的时候，**需要注意：**

用户调用时候的函数，使我们封装的函数，里面的实际事件函数需要我们手动去调用。

```
FrameWork.core.extend({
    click:function (callback) {
        console.log("click",this);
        this.each(function (item) {
            console.log("click",item);
            item.onclick = function (event) {
                callback(event);
            }
        })
    }
});
```

使用方法：

```
var box = h(".box");
box.click(function () {
    alert();
})
```

8.8 课程总结

- 框架开发的注意事项
- 自执行函数的作用
- 拓展函数的设计

8.9 实训

1. jQuery框架开发为什么要封装自执行函数?
2. 为什么要暴露统一的接口?
3. 完成上面所讲框架的开发。