1. SeaJS的使用：

引用sea-debug.js和sea.js

在Print.js中实现这样一个方法：

define(function (require, exports, module) {

exports.printWelcome = function () {

alert(exports.message);

};

});

调用的方法有两种：

在sea.js的上一级目录

<script type="text/javascript" src="SeaJS/sea.js"></script>

<script type="text/javascript">

// seajs.use('http://localhost/web/SeaJS/Print.js', function (e) {

// e.printWelcome();

//);

seajs.use('Print', function (e) {

e.message = "欢迎";

e.printWelcome();

});

</script>

在sea.js的同一级目录

<script type="text/javascript" src="./sea.js"></script>

<script type="text/javascript">

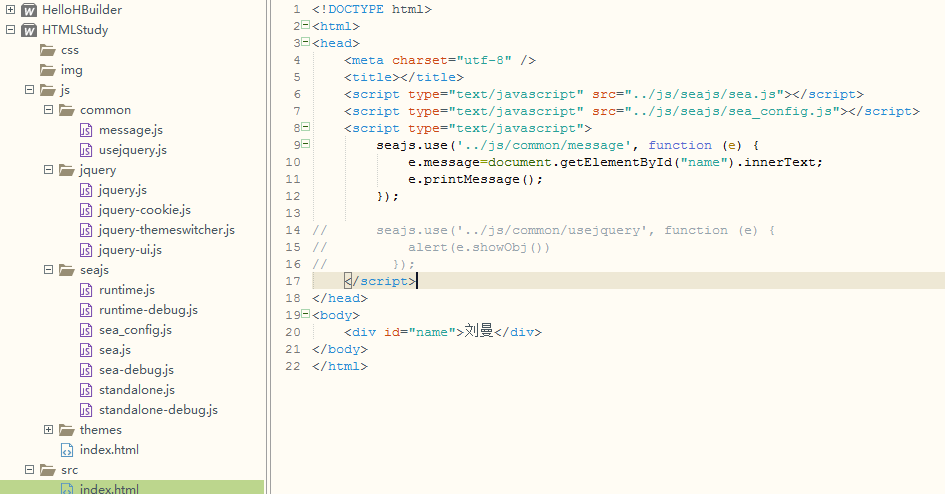
seajs.use('./Print', function (e) {

e.printWelcome();

});

</script>

跨目录的例子：



该页面为src下的index.html。

# 2. CMD 模块定义规范:require 是一个方法，接受 [模块标识](https://github.com/seajs/seajs/issues/258) 作为唯一参数，用来获取其他模块提供的接口。

define(function(require, exports) {

// 获取模块 a 的接口

var a = require('./a');

// 调用模块 a 的方法

a.doSomething();

});

### require.async require.async(id, callback?)

require.async 方法用来在模块内部异步加载模块，并在加载完成后执行指定回调。callback 参数可选。

define(function(require, exports, module) {

// 异步加载一个模块，在加载完成时，执行回调

require.async('./b', function(b) {

b.doSomething();

});

// 异步加载多个模块，在加载完成时，执行回调

require.async(['./c', './d'], function(c, d) {

c.doSomething();

d.doSomething();

});

});

**注意**：require 是同步往下执行，require.async 则是异步回调执行。require.async 一般用来加载可延迟异步加载的模块。

3. **CMD 模块定义规范:exports 是一个对象，用来向外提供模块接口。**

define(function(require, exports) {

// 对外提供 foo 属性

exports.foo = 'bar';

// 对外提供 doSomething 方法

exports.doSomething = function() {};

});

除了给 exports 对象增加成员，还可以使用 return 直接向外提供接口。

define(function(require) {

// 通过 return 直接提供接口

return {

foo: 'bar',

doSomething: function() {}

};

});

如果 return 语句是模块中的唯一代码，还可简化为：

define({

foo: 'bar',

doSomething: function() {}

});

**特别注意**：下面这种写法是错误的！

define(function(require, exports) {

// 错误用法！！!

exports = {

foo: 'bar',

doSomething: function() {}

};

});

正确的写法是用 return 或者给 module.exports 赋值：

define(function(require, exports, module) {

// 正确写法

module.exports = {

foo: 'bar',

doSomething: function() {}

};

});

**提示**：exports 仅仅是 module.exports 的一个引用。在 factory 内部给 exports 重新赋值时，并不会改变 module.exports 的值。因此给 exports 赋值是无效的，不能用来更改模块接口。

4. **CMD 模块定义规范:module 是一个对象，上面存储了与当前模块相关联的一些属性和方法。**

这就是 CMD 模块定义规范的所有内容。经常使用的 API 只有 define, require, require.async,exports, module.exports 这五个。其他 API 有个印象就好，在需要时再来查文档，不用刻意去记。

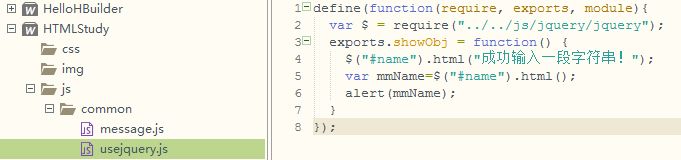
与 RequireJS 的 AMD 规范相比，CMD 规范尽量保持简单，并与 CommonJS 和 Node.js 的 Modules 规范保持了很大的兼容性。通过 CMD 规范书写的模块，可以很容易在 Node.js 中运行，后续会介绍。

**5.使用jquery。**

简单使用的例子：

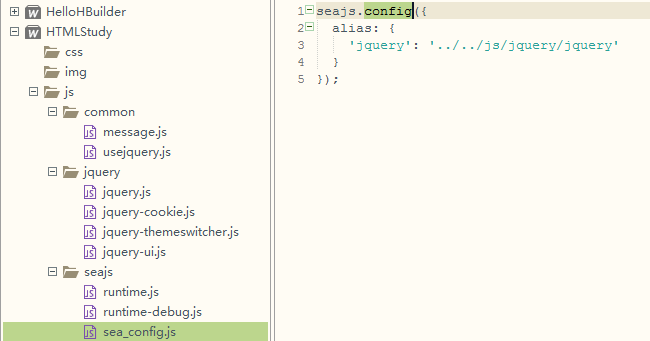
****

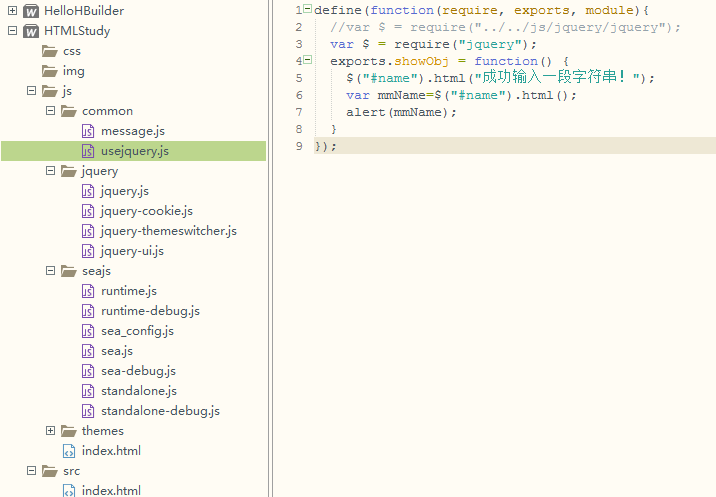
复杂一些，在外部js中使用：

****

****

也可以在config中进行配置，简化每次引用的写法：

****

****