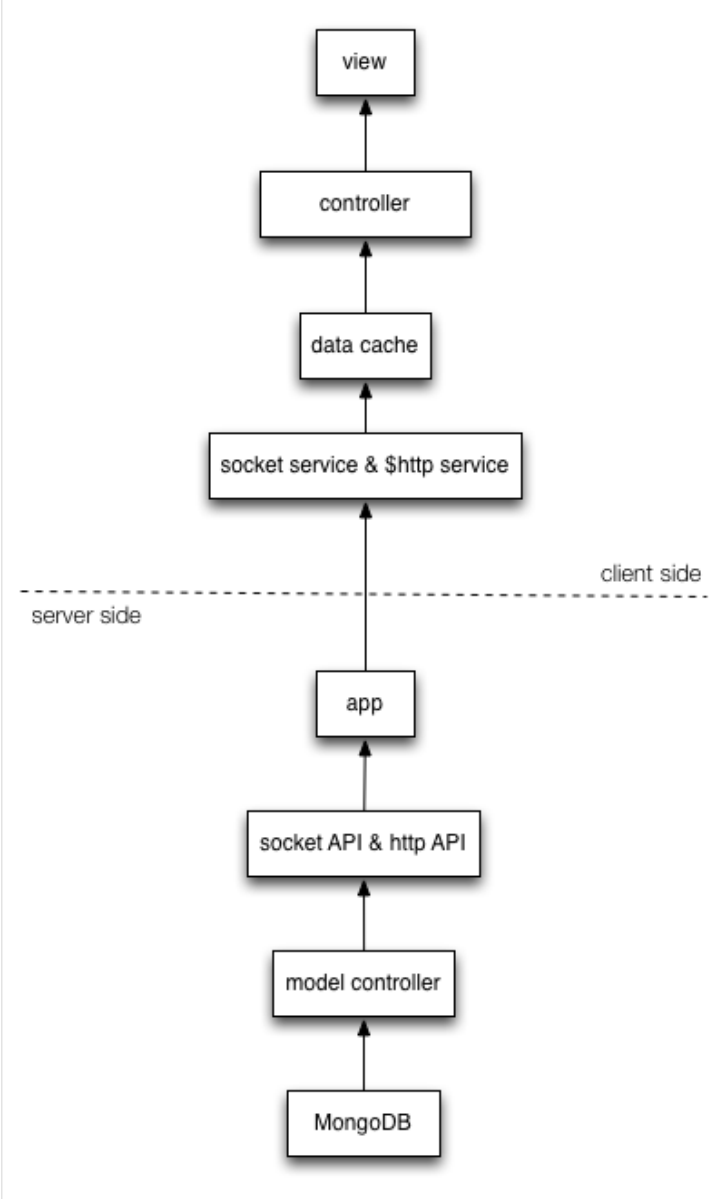
金股棒开发文档

金股棒架构分析与发布

在本文中，我们将一步步编写出来聊天室PC端和APP端重新做一次梳理，以及进行一些优化，让整个项目更容易理解和扩展。其次我们还会介绍一些前端流行的工具，帮助我们构建项目，便于发布。

项目结构:

目前PC端和APP端项目的架构大致相似，如下图:



箭头代表了读取数据的流向，服务端和客户端基本上都分为三层:

- 将服务端逻辑从app.js分拆为http和socket两个服务中;
- 在客户端提供一个统一的数据接口层，向上为controller提供数据服务，向下和服务

端通信，同步数据。

分拆http和socket服务

首先简化app.js：

```
// ...

var api = require('./services/api')var socketApi = require('./services/socketApi')

// ...



app.post('/api/login', api.login)

app.get('/api/logout', api.logout)

app.get('/api/validate', api.validate)

// ...



io.sockets.on('connection', function(socket) {



  socketApi.connect(socket)



  socket.on('disconnect', function() {

    socketApi.disconnect(socket)

  })



  socket.on('technode', function(request) {

    socketApi[request.action](request.data, socket, io)

  })

})

// ...
```

我们把http和socket的回调分别放到api.js和socketApi.js中，在socket通信方面做了简化，使用**technode**作为统一的事件名，而需要调用的接口名，则由请求数据中的**action**来决定。每个socket请求都会变成下面这样：

客户端的请求：

```
socket.emit('technode', {
  action: 'getRoom'
})
```

下面是服务端的返回：

```
socket.emit('technode', {
  "action": "getRoom",
  "data": [{
    "name": "Socket.IO",
    "_id": "52b0e5dd0a5e66fa26000001",
    "__v": 0,
    "createAt": "2013-12-18T00:01:33.528Z",
    "users": [],
    "messages": []
  }]
})
```

客户端则根据action，进行不同的处理：

```
socket.on('technode', function (data) {
  switch (data.action) {
      // ...
  }
})
```

客户端缓存

为什么需要客户端缓存？有两点原因：

1. 在第三章的实现中，在房间列表和房间切换时，controller都会通过socket从服务端重新获取房间列表或房间；
2. 在第三章的实现中，我们无法在controller之间共享数据，比如在LoginCtrl中，用户登录后，我们需要更新$rootScope的用户信息，采用了scope事件机制来实现。

我们需要一个缓存数据和共享数据的组件，这个组件将服务端请求来的数据缓存下来，避免重复的从服务端请求相同的数据，其次是对所有的controller提供接口，让controller间可以共享（读取、修改）同一份数据。
我们把这个组件命名为server，与服务端通信完全通过这个组件，数据缓存到这个组件之中，controller直接与它通信，不必关心真正的服务器是什么样的。

```
angular.module('techNodeApp').factory('server', ['$cacheFactory', '$q', '$http', 'socket',
function($cacheFactory, $q, $http, socket) {

 var cache = window.cache = $cacheFactory('technode')

 socket.on('technode', function(data) {

  switch (data.action) {

   case 'getRoom':

    if (data._roomId) {

     angular.extend(cache.get(data._roomId), data.data)

    } else {

     data.data.forEach(function (room) {

      cache.get('rooms').push(room)

     })

    }

    break

   // case something else

   // handle for socket events

  }

 })



 socket.on('err', function (data) {

  // handle server err

 })
```

```
  return {
    validate: function() {
      var deferred = $q.defer()
      $http({
        url: '/api/validate',
        method: 'GET'
      }).success(function(user) {
        angular.extend(cache.get('user'), user)
        deferred.resolve()
      }).error(function(data) {
        deferred.reject()
      })
      return deferred.promise
    }
    // more API
  }
}])
```

在server中，我们使用了两个Angular提供的组件，**$q**和**$cacheFactory**。

$q

$q是Angular对JavaScript异步编程模式Promise的实现，参考了 https://github.com/kriskowal/q 。在TechNode对它的用法相对比较简单，仅仅是将Ajax请求隐藏起来。以server.validate为例：

```
validate: function() {
  var deferred = $q.defer()
  $http({
    url: '/api/validate',
    method: 'GET'
```

```
  }).success(function(user) {
    angular.extend(cache.get('user'), user)
    deferred.resolve()
  }).error(function(data) {
    deferred.reject()
  })
  return deferred.promise
}
```

$q.defer()获取一个differed（推迟）对象，然后return deferred.promise先返回promise（承诺），在服务器端成功返回后，resolve（兑现）承诺，或者遇到问题，reject（拒绝）兑现。

在金股棒technode.js中我们可以这样使用：

```
server.validate().then(function() {
  if ($location.path() === '/login') {
    $location.path('/rooms')
  }
}, function() {
  $location.path('/login')
})
```

server.validate()获取promise（承诺）对象，then(resolvedCallback, rejectCallack)（然后）根据承诺的兑现情况进行不同的处理。

换句话说，technode.js中的techNodeApp问server，用户是不是登录了，server必须调用服务端接口进行验证，因此server给techNodeApp许诺，techNodeApp则只需要针对许诺是否兑现进行处理就好了。

所有与http请求相关的接口，我们都做了相似的处理。

$cacheFactory

$cacheFactory是Angular提供的缓存组件，该组件直接将数据存放在内存中。

```
var cache = window.cache = $cacheFactory('technode')



// ...

cache.put('rooms', [])



// ...

cache.get('rooms') && cache.get('rooms').forEach(function(room) {

  if (room._id === _roomId) {

    room.users = room.users.filter(function(user) {

      return user._id !== _userId

    })

  }

})
```

直接调用$cacheFactory，传入cacheId，Angular就为我构造出一块缓存区域，我们就可以通过get、put等等方法来存储或者获取缓存数据了。

$cacheFactory还提供了一种TechNode中未使用的特性，即这块缓存可以是LRU的，什么是LRU？即这块缓存是有大小的（避免缓存开销过大，影响网易性能），并且这块缓存使用LRU算法来淘汰长时间未使用的数据。

controller与server

基于server组件修改后的RoomCtrl：

```
angular.module('techNodeApp').controller('RoomCtrl', ['$scope', '$routeParams', '$scope', 'server',
function($scope, $routeParams, $scope, server) {



  $scope.room = server.getRoom($routeParams._roomId)



  // ...

}])
```

基于server的controller可以发现如下的变化：

- RoomCtrl不再直接与服务端通信读取当前的房间信息；
- 无需监听用户进入、离开或者新消息的事件。

RoomCtrl只需调用server.getRoom，传入房间的id即可。那房间信息不是需要到服务端读取么？这是怎么实现的？

这完全得益于Angular数据绑定特性，即数据变化，视图也会跟着变化：

```
getRoom: function(_roomId) {
  if (!cache.get(_roomId)) {
    cache.put(_roomId, {
      users: [],
      messages: []
    })
    socket.emit('technode', {
      action: 'getRoom',
      data: {
        _roomId: _roomId
      }
    })
  }
  return cache.get(_roomId)
}
```

这里的处理方式与**promise**有异曲同工之妙。**getRoom**方法，如果在缓存中没有找到房间的数据，就先新建一个房间对象，不过里面的数据都是空的（此时，RoomCtrl渲染出来的是一个空的房间视图），然后通过socket向服务端请求房间数据；如果找到就直接返回从缓存中获取的房间数据，RoomCtrl就可以渲染出来一个正常的房间视图。

而在服务端返回房间信息后，

```
case 'getRoom':
  if (data._roomId) {
    angular.extend(cache.get(data._roomId), data.data)
```

```
} else {
  data.data.forEach(function (room) {
    cache.get('rooms').push(room)
  })
}
```

我们使用服务端的数据填充到空房间即可，Angular即根据数据的变化，渲染出新的房间视图。

我们必须保证更新的房间对象必须是视图绑定的对象，因此我们一开始就返回一个房间对象，后面只是修改这个对象的属性。

金股棒PC端模块

沙龙房间模块：

我们将为TechNode添加沙龙房间的功能，借助socket.io的room功能可以很快地实现，我们开始吧。

新增pages/rooms.html页面：

```
<div class="main-rgt room">
  <h2>在线沙龙</h2>
  <div class="room-top">
    <a ng-if="me.level=='66'" class="rooma1" ng-href="/createRoom">创建房间<img src="../images/room.png"/></a>
    <a class="rooma1" ng-click="salon_default()" style="{{ a1_css }}">所有<img src="../images/all.png"/></a>
    <a class="rooma1" ng-click="searchCollect()" style="{{ a2_css }}">收藏<img src=" ../images/collection.png"/></a>
    <a class="rooma1" ng-click="searchHistory()" style="{{ a3_css }}">历史<img src="../images/hstr.png"/></a>
  </div>
  <div class="room-ctr" ng-Scrollbar-home scrollbar-x="false" scrollbar-y="true" scrollbar-config="{show: true}">
    <dl class="rlx" ng-repeat="room in filteredRooms|orderBy:'-users.length'" ng-click="enterRoom(room)">
      <dt>
```

```
        <h3>{{room.name}}</h3><img src="{{room.creator_avatarUrl}}"/><a ng-if="room.online==true"
                                                    style="background:url(../images/room-pic03.png) no-repeat left
center;">在线</a><a
            ng-if="room.online==false"
            style="background:url(../images/room-pic-11.png) no-repeat left center;">离线</a></dt>
        <dd>
            <p>{{room.intro}}</p>
            <span class="room-span1">{{room.status}}</span><span
                class="room-span2">{{room.users.length}}</span><span
                    class="room-span3">LV.1</span>
            <img ng-model="img_model" ng-show="display(room._id)" src="../images/room-pic01.png"/>
        </dd>
      </dl>
    </div>
</div>
```

添加房间API:

修改数据模型

因为我们加入了房间的概念，所以我们有必要修改一下数据模型:
首先添加房间的Schema，在models中添加room.js文件，添加如下代码:

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema
ObjectId = Schema.ObjectId


var Room = new Schema({
    name: String,
```

```
    creator_id: ObjectId,
    creator_name: String,
    creator_avatarUrl: String,
    intro: String,
    status: String,
    online: Boolean,
    createAt: {type: Date, default: Date.now}
});


module.exports = Room
```

修改消息的的Schema：

```
var mongoose = require('mongoose')
var Schema = mongoose.Schema,
  ObjectId = Schema.ObjectId

var Message = new Schema({
  content: String,
  creator: {
    _id: ObjectId,
    email: String,
    name: String,
    level:String,
    level_name:String,
    avatarUrl: String,
    nick_name:String
  },
```

```
  _roomId: ObjectId,

  createAt:{type: Date, default: Date.now}

})



module.exports = Message
```

添加了一个_roomId的作为指向房间的外间，让消息与房间对应起来。
同理为用户模型也添加一个_roomId

实现rooms的controller

我们需要两个接口：

- 获取所有的房间；
- 创建新的房间。

```
exports.create = function (room, callback) {

    var r = new db.Room()

    r.name = room.name

    r.status = room.status

    r.creator_id = room.creator_id

    r.creator_name = room.creator_name

    r.creator_avatarUrl = room.creator_avatarUrl

    r.online = room.online

    r.intro = room.intro

    // r.level=room.creator.level

    // r.level_name=room.creator.level_name

    r.save(callback)

}



exports.read = function (callback) {

    db.Room.find({}, function (err, rooms) {
```

```
if (!err) {
    var roomsData = []
    async.each(rooms, function (room, done) {
        var roomData = room.toObject()
        async.parallel([

                function (done) {
                    db.User.find({
                        _roomId: roomData._id,
                        online: true
                    }, function (err, users) {
                        done(err, users)
                    })
                },
                    function (done) {
                    db.Message.find({
                        _roomId: roomData._id
                    }, null, {
                        sort: {
                            'create_at': -1
                        },
                        limit: 10
                    }, function (err, messages) {
                        done(err, messages.reverse())
                    })
```

```
                }
            ],
            function (err, results) {
                if (err) {
                    done(err)
                } else {
                    roomData.users = results[0]
                    roomData.messages = results[1]
                        roomsData.push(roomData)
                        done()
                    }
            });
        }, function (err) {
            callback(err, roomsData)
        })
    }
  })
}
```

提供socket的房间API

修改socket.js的socket部分，为客户端提供两个接口——新建房间和读取所有房间列表：

```
exports.getRoom = function (data, socket) {
    console.log('获取房间信息')
    if (data && data._roomId) {
        Controllers.Room.getById(data._roomId, function (err, room) {
            if (err) {
```

```
            socket.emit('err', {
                msg: err
            })
        } else {
            console.log(global.onlineUsers)

            room.allUser = global.onlineUsers
            socket.emit('technode', {
                action: 'getRoom',
                _roomId: data._roomId,
                data: room

            })
        }
    })
} else {
    Controllers.Room.read(function (err, rooms) {
        if (err) {
            socket.emit('err', {
                msg: err
            })
        } else {
            socket.emit('technode', {
                action: 'getRoom',
                data: rooms
```

```
            })
        }
    })
  }
}
```

```
exports.createRoom = function (room, socket, io) {
  Controllers.Room.create(room, function (err, room) {
    if (err) {
      socket.emit('err', {
        msg: err
      })
    } else {
      room = room.toObject()
      room.users = []
      io.sockets.emit('technode', {
        action: 'createRoom',
        data: room
      })
    }
  })
}
```

登录后跳转至房间列表

用户登录成功后，我们直接跳转至沙龙房间页面，但是现在我们有了房间的功能，因此当用户登录成功后，首先跳转至聊天室列表。在router.js中添加上房间列表的路由/rooms：

```
angular.module('techNodeApp').config(['$routeProvider', '$locationProvider', function ($routeProvider, $locationProvider) {
    $locationProvider.html5Mode({
        enabled: true,
        requireBase: false
    });
    $routeProvider.when('/rooms', {
        templateUrl: '/pages/rooms.html',
        controller: 'RoomsCtrl'
    }).when('/register/:status', {
            templateUrl: '/pages/register.html',
            controller: 'registerCtrl'
        }
    ).otherwise({
        redirectTo: '/login'
    })
}])
```

在static/controllers下添加rooms.js，实现房间列表控制器RoomsCtrl

```
angular.module('techNodeApp').controller('RoomsCtrl', function($scope) {
  // Nothing
})
```

房间列表控制器RoomsCtrl

- 所有房间，默认按房间人气排序

```
    /**
 * 获取全部房间
 * $rootScope.room          获取所有收藏,所有浏览
```

```
 */

$scope.filteredRooms = $scope.rooms = server.getRooms()
```

- 历史收藏

```
/**

 * 点击历史收藏

 * historyArr 数组

 */

$scope.searchCollect = function () {

    if ($rootScope.me._id) {

        var historyArr = []

        $rootScope.room[0].collect.forEach(function (history) {

            var temple = $scope.rooms.filter(function (room) {

                return room._id.indexOf(history.room_id) > -1

            })

            historyArr = historyArr.concat(temple)

        })

        $scope.filteredRooms = historyArr

    } else {

        $location.path('/login')

    }

}
```

- 历史浏览

```
/**

 * 点击历史浏览

 * historyArr 数组
```

```
 */
$scope.searchHistory = function () {
    if ($rootScope.me._id) {
        var historyArr = []
        $rootScope.room[0].history.forEach(function (history) {
            var temple = $scope.rooms.filter(function (room) {
                return room._id.indexOf(history._id) > -1
            })
            historyArr = historyArr.concat(temple)
        })
        $scope.filteredRooms = historyArr
    } else {
        $location.path('/login')
    }
}
```

- 进入房间

在房间列表中可以点击单个房间，触发进入房间事件enterRoon

```
/**
 * 进入房间
 * room    房间信息
 * collect  更新收藏记录
 * @param room
 */
$scope.enterRoom = function (room) {
    $location.path('/rooms/' + room._id)
}
```

进入房间以后，可以关注和取消关注分析师，收藏及取消收藏房间

我的理财师模块

用户登录成功后，先判断他是客户还是分析师（理财师）

- 客户

在technode.js中添加如下代码:

```
/**
 * 我的客户
 * @param id
 */
$rootScope.enterOnline = function (id) {
    $rootScope.eject3_css = {
        "display": "block"
    }
    var index_id = id + $rootScope.me._id
    var data = [id, $rootScope.me._id, index_id];
    $rootScope.customerRoom = server.getAnalystRoom(data)
    //监听模型变化,不断获取最新高度
    setInterval(function () {
        $rootScope.$watch('customerRoom', function (newValue, oldValue, scope) {
        if (newValue == oldValue) {
            setTimeout(function () {
                var scroll = $("#eject_4")[0].scrollHeight
                $("#eject_4").scrollTop(scroll);
                }, 100)
            }
        })
```

```
    }, 1000)

    $rootScope.value = id

  }
```

登陆之后客户的理财师就全部加载成功，全部是隐藏状态。客户在点击我的理财师的时候eject3_css取消隐藏，同理在分析师和理财师登陆的状态下，我的客户的信息也是全部加载成功。

- 分析师，理财师

在technode.js中添加如下代码:

```
/**

 * 我的理财师

 * 显示和隐藏聊天记录

 * 点击所有理财师显示

 */

$rootScope.rgtClick = function () {

    if (!$rootScope.me._id) {

        $location.path('/login')

        ngDialog.open(

            {

                template: '<h1 style="text-align: center" >请登陆</h1>',

                plain: true,

                className: 'ngdialog-theme-default',

                height: '200px',

                width: '180px'

            });

    } else {

        $rootScope.dpdn_css = {'display': 'block'}

        if ($rootScope.me.level == '0') {
```

```
if (!$rootScope.me.belong_metal_id && !$rootScope.me.belong_card_id) {
    $location.path('/bindingCP')

} else {
    // $scope.users=server.getBindUser($scope.me._id)
    /**
     * customerRoom  根目录
     * metal        贵金属
     * card         邮币卡
     * analyst      分析师
     */

    $rootScope.user = $rootScope.me
    $rootScope.enterMetal = function (me) {
        if (me.belong_metal_id) {
            $rootScope.eject_css = {
                "display": "block"
            }

            $rootScope.users = me.belong_metal_id
            var index_id = $rootScope.me._id + me.belong_metal_id
            var data = [me.belong_metal_id, $rootScope.me._id, index_id];
            $rootScope.room = server.getAnalystRoom(data)

            //监听模型变化,不断获取最新高度
```

```
            setInterval(function () {
                $rootScope.$watch('room', function (newValue, oldValue, scope) {
                    if (newValue == oldValue) {
                        setTimeout(function () {
                            var scroll = $("#eject_1")[0].scrollHeight
                            $("#eject_1").scrollTop(scroll);
                        }, 100)
                    }

                })
            }, 1000)
        } else {
            alert('您还没有贵金属理财师,点击前往绑定理财师')
            $location.path('/bindingCP/')
        }
    }
        if (newValue == oldValue) {
                setTimeout(function () {
                    var scroll = $("#eject_1")[0].scrollHeight
                    $("#eject_1").scrollTop(scroll);
                }, 100)
            }

        })
    }, 1000)
```

```
          }
         }
        }
      }
    }
}
```

金股棒后台

目前分为六个模块:

- 房间管理
- 客户管理
- 日志管理
- 分析师管理
- 理财师管理
- 沙龙聊天信息管理

技术栈选取:

- ng-grid http://angular-ui.github.io/ui-grid/
- material design https://material.angularjs.org/latest/

金股棒APP

APP端架构同PC端类似，不同点在于我的理财师模块，用户在登录成功以后，识别用户的身份以后，在router中添加/analystRoom,/customerRoom，对应不同controller。其他模块基本一样。

接下来，我们将学习如何将前端程序打包，发布！

使用Grunt打包金股棒

开发时，为了解耦和便于维护，我们把代码拆成单独的文件，JavaScript代码、CSS代码和HTML都是单独的。在生产环境中，为了提高性能，我们需要把这些分开的文件合并到一起。如果你的网站使用CDN的化，我们还需要给每个版本的文件，添加上唯一的标识，便于维护CDN的缓存。

Grunt是目前JavaScript最流行的项目自动化构建工具。Grunt官方提供了很多插件，也有大量的第三方插件。我们可以轻松地使用Grunt检查、压缩合并代码，甚至发布应

用程序。我们将基于grunt-usemin等几个流行的Grunt插件来构建TechNode项目。

首先我们需要做一些准备，安装Grunt命令行和运行时，在TechNode根目录新建Gruntfile.js。

```
npm install -g grunt-cli && npm install grunt --save-dev && touch Gruntfile.js
```

为了使用grunt-usemin来压缩我们的代码，我们需要在index.html添加一些特殊的注释来来帮助grunt-usemin找到需要合并的文件：

```
<!-- build:css /css/technode.css -->

<link rel="stylesheet" href="/components/bootstrap/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="/styles/style.css">

<link rel="stylesheet" href="/styles/login.css">

<link rel="stylesheet" href="/styles/rooms.css">

<link rel="stylesheet" href="/styles/room.css">

<!-- endbuild -->



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

<!-- build:js /script/technode.js -->

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

<script type="text/javascript" src="/components/bootstrap/dist/js/bootstrap.min.js"></script>

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

<script type="text/javascript" src="/components/angular-route/angular-route.js"></script>

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

<script type="text/javascript" src="/components/angular-moment/angular-moment.js"></script>

<script type="text/javascript" src="/components/moment/lang/zh-cn.js"></script>

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

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

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

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

```
<script type="text/javascript" src="/directives/auto-scroll-to-bottom.js"></script>

<script type="text/javascript" src="/directives/ctrl-enter-break-line.js"></script>

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

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

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

<script type="text/javascript" src="/controllers/message-creator.js"></script>

<!-- endbuild -->
```

我们分别在css和javascript的引用周围加上了注释，**<!-- build:css /css/technode.css -->**标明我们需要把下面这些css都合并到technode.css这个文件中，javascript全都合并到technode.js中。

注意，socket.io.js这个文件并没有包含进来，因为它是socket.io自己输出的，并没有在我们的自己的源码中。当然，我们甚至可以把这个文件保存到源码中，自己引用也是可以的。

首先使用grunt-contrib-copy将不需要打包压缩的文件拷贝到build目录中，修改Gruntfile.js

```
module.exports = function (grunt) {

 grunt.initConfig({

   copy: {

     main: {

       files: [

         {expand: true, cwd: 'static/components/bootstrap/dist/fonts/', src: ['**'], dest: 'build/fonts'},

         {'build/index.html': 'static/index.html'},

         {'build/favicon.ico': 'static/favicon.ico'}

       ]

     }

   }

 })
```

```
  grunt.loadNpmTasks('grunt-contrib-copy')



  grunt.registerTask('default', [
    'copy'
  ])
}
```

grunt-usemin为我们提供了一个useminPrepare的task，这个task就是基于我们在index.html文件中的配置，自动生成合并和压缩代码的配置：

```
module.exports = function (grunt) {
  grunt.initConfig({
    copy: {
      main: {
        files: [
          {expand: true, cwd: 'static/components/bootstrap/dist/fonts/', src: ['**'], dest: 'build/fonts'},
          {'build/index.html': 'static/index.html'},
          {'build/favicon.ico': 'static/favicon.ico'}
        ]
      }
    },
    useminPrepare: {
      html: 'static/index.html',
      options: {
        dest: 'build'
      }
    }
```

```
  })

  grunt.loadNpmTasks('grunt-usemin')

  grunt.loadNpmTasks('grunt-contrib-copy')



  grunt.registerTask('default', [

    'copy',

    'useminPrepare'

  ])

}
```

npm install grunt-usemin --save-dev，运行grunt试试看：

```
Running "useminPrepare:html" (useminPrepare) task

Going through static/index.html to update the config

Looking for build script HTML comment blocks



Configuration is now:



  concat:

  { generated:

   { files:

      [ { dest: '.tmp/concat/css/technode.css',

          src:

           [ 'static/components/bootstrap/dist/css/bootstrap.min.css',

             'static/styles/style.css',

             'static/styles/login.css',
```

```
            'static/styles/rooms.css',
            'static/styles/room.css' ] },
      { dest: '.tmp/concat/script/technode.js',
        src:
         [ 'static/components/jquery/jquery.js',
           'static/components/bootstrap/dist/js/bootstrap.min.js',
           'static/components/angular/angular.js',
           'static/components/angular-route/angular-route.js',
           'static/components/moment/moment.js',
           'static/components/angular-moment/angular-moment.js',
           'static/components/moment/lang/zh-cn.js',
           'static/technode.js',
           'static/services/socket.js',
           'static/services/server.js',
           'static/router.js',
           'static/directives/auto-scroll-to-bottom.js',
           'static/directives/ctrl-enter-break-line.js',
           'static/controllers/login.js',
           'static/controllers/rooms.js',
           'static/controllers/room.js',
           'static/controllers/message-creator.js' ] } ] } }


uglify:
{ generated:
 { files:
```

```
    [ { dest: 'build/script/technode.js',
        src: [ '.tmp/concat/script/technode.js' ] } ] } }


 cssmin:
 { generated:
  { files:
    [ { dest: 'build/css/technode.css',
        src: [ '.tmp/concat/css/technode.css' ] } ] } }
```

它为我们生成了本来需要手动编写的其他task的配置，接下来，安装其他几个需要的grunt task，继续修改Gruntfile.js：

```
module.exports = function (grunt) {
  grunt.initConfig({
    copy: {
      main: {
        files: [
          {expand: true, cwd: 'static/components/bootstrap/dist/fonts/', src: ['**'], dest: 'build/fonts'},
          {'build/index.html': 'static/index.html'},
          {'build/favicon.ico': 'static/favicon.ico'}
        ]
      }
    },
    useminPrepare: {
      html: 'static/index.html',
      options: {
        dest: 'build'
      }
```

```
    }
  })
  grunt.loadNpmTasks('grunt-usemin')
  grunt.loadNpmTasks('grunt-contrib-copy')
  grunt.loadNpmTasks('grunt-contrib-concat')
  grunt.loadNpmTasks('grunt-contrib-uglify')
  grunt.loadNpmTasks('grunt-contrib-cssmin')

  grunt.registerTask('default', [
    'copy',
    'useminPrepare',
    'concat',
    'uglify',
    'cssmin'
  ])
}
```

安装好新的依赖，再运行grunt试试看。首先concat根据useminPrepare生成的配置，将css和js分别合并到.tmp/concat/css/technode.css和.tmp/concat/script/technode.js中；然后uglify和cssmin分别将这两个文件压缩成了build/css/technode.css和build/script/technode.js，我们的css文件和js文件就打包压缩好了。

于是我们整个构建的过程结束了，所有文件都按照我们想要的方式处理好了。

我们再来回顾一下打包的过程，开始那么多的js，首先被concat到了tmp/concat/technode.js中，然后aglify压缩到build/script/tecnhode.js中，接着rev根据文件内容为其生成了唯一的标示7add9650.technode.js，最后，usemin再把build/index.html中的js区块换成了<script src="/script/7add9650.technode.js"></script>。这就是我们采用的整个打包压缩过程。同理css也是如此。