[1. 搭建Pomelo 3](#_Toc430269726)

[1、安装 python 3](#_Toc430269727)

[2、安装 vs2010 3](#_Toc430269728)

[3、安装 Node.js 3](#_Toc430269729)

[4、安装 npm 3](#_Toc430269730)

[5、安装 pomelo 3](#_Toc430269731)

[6、安装 node-gyp 3](#_Toc430269732)

[7、安装 msnodesql 3](#_Toc430269733)

[8、安装 SQL Server Native Client 11.0 4](#_Toc430269734)

[9、安装 MongoDB 4](#_Toc430269735)

[10、安装 Mongoose 4](#_Toc430269736)

[2. 配置数据库 5](#_Toc430269737)

[1、SQL Server 5](#_Toc430269738)

[2、MongoDB 5](#_Toc430269739)

[3. 创建Pomelo 7](#_Toc430269740)

[1、本地新建目录 7](#_Toc430269741)

[2、进入新建目录 7](#_Toc430269742)

[3、初始化项目 7](#_Toc430269743)

[4、安装依赖包 7](#_Toc430269744)

[5、启动game-server服务器 7](#_Toc430269745)

[6、启动web-server服务器 7](#_Toc430269746)

[7、关闭项目 7](#_Toc430269747)

[8、Debug-BAT文件 7](#_Toc430269748)

[9、Release-BAT文件 8](#_Toc430269749)

[4. 添加依赖包 9](#_Toc430269750)

[1、添加同步下载 9](#_Toc430269751)

[2、直接安装 9](#_Toc430269752)

[3、引用包 9](#_Toc430269753)

[5. 节点认识 10](#_Toc430269754)

[1、添加节点 10](#_Toc430269755)

[2、handle节点 12](#_Toc430269756)

[3、remote节点 13](#_Toc430269757)

[4、lifecycle 14](#_Toc430269758)

[5、filter 14](#_Toc430269759)

[6. 主要语法 15](#_Toc430269760)

[1、变量 15](#_Toc430269761)

[2、Json 15](#_Toc430269762)

[3、数组 17](#_Toc430269763)

[4、函数 19](#_Toc430269764)

[7. HTTP服务器 20](#_Toc430269765)

[1、安装express 20](#_Toc430269766)

[2、安装express-generator 20](#_Toc430269767)

[3、方案设计 20](#_Toc430269768)

[8. 定时器任务 21](#_Toc430269769)

[9. 数据库CRUD 23](#_Toc430269770)

[1、SQL Server 23](#_Toc430269771)

[2、MongoDB 25](#_Toc430269772)

[10. 异步处理 28](#_Toc430269773)

[11. 加密解密 29](#_Toc430269774)

[12. 数据压缩 30](#_Toc430269775)

[13. 充值计费 31](#_Toc430269776)

[14. Sublime 32](#_Toc430269777)

[1、插件 32](#_Toc430269778)

[2、快捷键 32](#_Toc430269779)

[15. WebStorm 33](#_Toc430269780)

# 搭建Pomelo

## 1、安装 python

python 2.0 < version < 3.0

下载地址：http://www.python.org/download/

## 2、安装 vs2010

编译环境 visual studio 2010，或者也可以直接找VS2008以上的runtime安装

## 3、安装 Node.js

下载地址：http://nodejs.org/download/

在 CMD 输入 node --version，若安装成功会打印 node 的版本号 “v0.8.21”

## 4、安装 npm

下载地址：https://github.com/npm/npm

在 CMD 输入 node cli.js install -gf 安装

## 5、安装 pomelo

在 CMD 输入 npm install pomelo –g

或者 npm install d:\pomelo-master –g

-g 表示全局安装，即安装到nodejs安装目录下

## 6、安装 node-gyp

在 CMD 输入 npm install node-gyp -g

## 7、安装 msnodesql

下载地址：https://github.com/Azure/node-sqlserver

node-sqlserver-master.zip

在 CMD 输入 node-gyp configure

在 CMD 输入 node-gyp build

将 .\build\Release\sqlserver.node 拷贝到 .\lib

将 node-sqlserver-master 重命名 msnodesql

将 msnodesql 拷贝到 .\game-server\node\_modules

.\build .\scripts .\src .\test 均可删

## 8、安装 SQL Server Native Client 11.0

下载地址：http://download.microsoft.com/download/F/E/D/FEDB200F-DE2A-46D8-B661-D019DFE9D470/ENU/x86/sqlncli.msi

sqlncli.msi

## 9、安装 MongoDB

下载地址：https://fastdl.mongodb.org/win32/mongodb-win32-i386-2.6.8.zip

mongodb-win32-i386-2.6.8.zip

## 10、安装 Mongoose

下载地址：https://codeload.github.com/LearnBoost/mongoose/zip/master

在 CMD 输入 npm install mongoose

将下载编译好的mongoose拷贝到 .\game-server\node\_modules

# 配置数据库

## 1、SQL Server

安装数据库：

运行 SQLEXPR\_2005企业管理工具.exe 安装包

多次运行可以安装多个实例

安装时可以启用sa用户

实例配置：

开始菜单\所有程序\Microsoft SQL Server 2005\配置工具\SQL Server Configuration Manager

展开 SQL Server 2005 网络配置

点击协议 TCP/IP 启用

双击 TCP/IP 切换 IP地址 选项卡

IPALL TCP端口 填入 1435

开始菜单\所有程序\Microsoft SQL Server 2005\配置工具\SQL Server 外围应用配置器

服务和连接的外围应用配置器

选择数据库 Database Engine 中的远程连接

选择 同时使用TCP/IP和named pipes

选择 SQL Server Browser 服务 启动类型 改为 自动 并启动

重启 SQL服务 或者 重启电脑

## 2、MongoDB

安装数据库：

将mongodb-win32-i386-2.6.8.zip中的bin文件夹解压到E:\Program Files\MongoDB文件夹下

创建文件夹

E:\Program Files\MongoDB\hunter\db

E:\Program Files\MongoDB\hunter\log

在log文件夹下创建一个日志文件hunter.log

E:\Program Files\MongoDB\hunter\log\hunter.log

一个文件夹可以看作一个实例

实例启动：

a.自带的服务启动，需要有日志文件：

cd E:\Program Files\MongoDB\bin\

mongod.exe -port 28071 -dbpath "E:\Program Files\MongoDB\hunter\db" -logpath "E:\Program Files\MongoDB\hunter\log\hunter.log" -install -serviceName "MongoDB280711"

删除服务

mongod.exe -port 28071 -dbpath "E:\Program Files\MongoDB\hunter\db" -remove -

serviceName "MongoDB28071"

b.创建一个不需要日志文件的服务

sc create MongoDB28071 binpath= """"E:\Program Files\MongoDB\bin\mongod.exe""" -port 28071 -dbpath """E:\Program Files\MongoDB\hunter\db""" -service" displayname= "MongoDB28071" start= auto

sc start MongoDB28071

停止服务

sc stop MongoDB28071

删除服务

sc delete MongoDB28071

err: Unclean shutdown detected.

"E:\Program Files\MongoDB\bin\mongod.exe" -repair -dbpath "e:\Program Files\MongoDB\hunter\db"

# 创建Pomelo

## 1、本地新建目录

在 CMD 输入 mkdir helloWorld

## 2、进入新建目录

在 CMD 输入 cd helloWorld

## 3、初始化项目

在 CMD 输入 pomelo init

选择 1

## 4、安装依赖包

在 CMD 输入 npm-install.bat

(如果是Linux环境直接运行：sh npm-install.sh )

## 5、启动game-server服务器

在 CMD 输入 cd game-server && pomelo start

默认启动为 development 环境，如需启动production环境，添加参数 –e production

pomelo start -e production

development环境可以使用ctrl+c或者关闭cmd结束pomelo

production环境关闭需要用stop、kill命令，或者在任务管理器结束所有node.exe

## 6、启动web-server服务器

在 CMD 输入 cd web-server && node app

## 7、关闭项目

在 CMD 输入 pomelo stop [id]

在 CMD 输入 pomelo kill [--force]

## 8、Debug-BAT文件

D:

cd workspace\hunterPomelo\game-server

pomelo start > > ..\..\1.txt

## 9、Release-BAT文件

D:

cd workspace\hunterPomelo\game-server

pomelo start

# 添加依赖包

## 1、添加同步下载

对应文件 \game-server\package.json

{

"name": "helloWorld",

"version": "0.0.1",

"private": false,

"dependencies":

{

"pomelo": "1.0.3",

"async": ">=0.0.1",

"pomelo-schedule": ">=0.0.1"

}

}

版本号有多种方式，可以给定版本号，也可以使用最新版本

添加后执行 \npm-install.bat 可以下载依赖包

package.json：存储着工程的信息及模块依赖，当在 dependencies 中添加依赖的模块时，运行 npm install，npm 会检查当前目录下的 package.json，并自动安装所有指定的模块

## 2、直接安装

也可以直接将依赖包放到 \game-server\node\_modules 目录下

## 3、引用包

var async = require('async');

# 节点认识

## 1、添加节点

对应目录 \game-server\app\server

创建文件夹

\game-server\app\server\helloApp\handler

\game-server\app\server\helloApp\remote

\handler 目录下的js文件属于 handler

代码示例:

module.exports = function(app)

{

return new helloAppHandler(app);

};

var helloAppHandler = function(app)

{

this.app = app;

this.channelService = app.get('channelService');

this.sessionService = app.get('sessionService');

};

var pro = helloAppHandler.prototype;

pro. testFunc = function(msg, session, next){next()}

\remote 目录下的js文件属于 remote

代码示例:

module.exports = function(app)

{

return new helloAppRemote(app);

};

var helloAppRemote = function(app)

{

this.app = app;

this.channelService = app.get('channelService');

this.sessionService = app.get('sessionService');

};

var pro = helloAppRemote.prototype;

pro. testFunc = function(cb){cb()}

\chat 目录下的js文件不属于APP

配置对应

对应文件 \game-server\config\servers.json

代码示例:

{

"development": {

"helloApp": [

{

"id": "helloApp-server-1",

"host": "192.168.0.138",

"port": 3100,

"clientPort": 3050,

"frontend": true,

"sercerId" : 1,

"name" : "测试服1区",

"database" : "DB\_HUNTER\_GAME",

"status" : 1

},

{

"id": "helloApp-server-2",

"host": "192.168.0.138",

"port": 3101,

"clientPort": 3051,

"frontend": true,

"sercerId" : 2,

"name" : "测试服2区",

"database" : "DB\_HUNTER\_GAME",

"status" : 3

}

]

}

}

development 表示环境

helloApp 与文件夹名字对应

id 必须且唯一，表示：服务标示

host 必须，表示：服务ip

port 必须且唯一，表示：服务内部通讯端口，remote接口

frontend 非必须 true，表示：服务对外可以通讯

clientPort 非必须唯一，表示：服务对外通讯端口，需要frontend为true才有效，handle接口

对应文件 \game-server\config\adminServer.json

代码示例:

[

{

"type": "helloApp",

"token": "agarxhqb98rpajloaxn34ga8xrunpagkjwlaw3ruxnpaagl29w4rxn"

}

]

如果未添加则出现

[ERROR] console - Caught exception: SyntaxError: Unexpected token

错误提示

代码有错误,没有出现对应的{},()则出现

[ERROR] console - Caught exception: SyntaxError: Unexpected identifier

添加配置 \game-server\app.js

var routeUtil = require('./app/util/routeUtil');

app.configure('production|development', helloApp|connector|gate', function() {

app.route('helloApp', routeUtil.helloApp);

app.loadConfig('SQLServer', app.getBase() + '/config/sqlServer.json');

var dbclient = require('./app/database/sqlserver/sqlfunc').init(app);

app.set('dbclient', dbclient);

});

#routeUtil.js

var exp = module.exports;

exp.helloApp = function(session, msg, app, cb){

var helloAppServers = app.getServersByType('helloApp');

if (!helloAppServers || helloAppServers.length === 0)

{

cb(new Error('can not find helloApp servers.'));

return;

}

var res = helloAppServers[0];

cb(null, res.id);

};

app.route('helloApp', routeUtil.helloApp); 用来确定remote节点

loadConfig函数，需要使用绝对路径，使用示例：var sqlConfig = app.get('SQLServer');

dbclient 使用示例：pomelo.app.get('dbclient')

## 2、handle节点

客户端可连

可以与remote通过rpc互通数据

链接方式：

self.app.rpc.helloApp.helloAppRemote.testFunc(session, null);

session为app.route设定的函数的第一个参数.

null表示函数的回调为cb();

handle无法链接别的handle

handle只有客户端链接的时候才会初始化，但是只创建一次

[WARN] pomelo - [D:\workspace\hunterPomelo\game-server\node\_modules\pomelo\lib\filters\handler\timeout.js] request "connector.entryHandler.logonUserIDByDemo" timeout.

代码编译错误

Ps:代码编译错误没有提示，直接程序不再执行

error1：console.error(res["data"].length);

由于res["data"]不是数组，无法使用length属性，则程序不再执行

error2：resCode = CODE.DB\_GAMECONFIGVER.DB\_GET\_GAMECONFIGVER\_ERROR\_CODE

由于CODE中没有定义DB\_GAMECONFIGVER，导致无法取值，则程序不再执行

[2014-09-24 13:41:28.442] [INFO] console - [2014-09-24 13:41:28.441] [WARN] pome

lo-rpc - [D:\workspace\hunterPomelo\game-server\node\_modules\pomelo\node\_modules

\pomelo-rpc\lib\rpc-client\mailboxes\ws-mailbox.js] timer is not exsits, id: 0,

host: 192.168.0.138, port: 4100

timer is not exsits 是cb回调了两次以上引起，请检查有回调的地方

## 3、remote节点

客户端不可连

可以与别的remote通过rpc互通数据

链接方式：

self.app.rpc.helloApp.helloAppRemote.testFunc(session, null);

session为app.route设定的函数的第一个参数.

null表示函数的回调为cb();

当remote调用自己内部的函数时

链接方式：

pro.leaveRoom = function(uid, sid, cb){cb();}

pro.leaveRoomOffLine = function(uid, sid, cb)

{

self.leaveRoom(uid, sid, function(){});

cb();

};

不可以用null

remote在启动服务器的时候被创建，因此可以用于初始化数据，存储数据大部分也放在remote端

rpc传递参数的时候，如果mg:{uid:undefined}，则uid这个key会被删除

## 4、lifecycle

## 5、filter

# 主要语法

## 1、变量

var a = 1;

console.error(a);

console.error(!a);

console.error(!!a);

a = 0;

console.error(a);

console.error(!a);

console.error(!!a);

[ERROR] console - 1

[ERROR] console - false

[ERROR] console - true

[ERROR] console - 0

[ERROR] console - true

[ERROR] console - false

## 2、Json

格式

appRoomInfo = {

"r1":{"room":"room\_id\_Date.now()","copyid":1,"hostuid":1001,"start":false,"curnum":4,

"seat":{"0":1001,"1":1002,"2":1003,"3":1004}},

"r2":{"room":"room\_id\_Date.now()","copyid":1,"hostuid":2001,"start":false,"curnum":4,

"seat":{"0":2001,"1":2002,"2":2003,"3":2004}},

"r3":{"room":"room\_id\_Date.now()","copyid":1,"hostuid":3001,"start":false,"curnum":4,

"seat":{"0":3001,"1":3002,"2":3003,"3":3004}}

}

定义

var appRoomInfo = {}

数据添加

appRoomInfo.[“r4”] = {}

appRoomInfo.r4 = {}

不能多层创建

如果没有r4，则不能进行如下操作

appRoomInfo.r4.room = “123”

数据访问

tmpInfoR1 = appRoomInfo[“r1”]

tmpInfoR1 = appRoomInfo.r1

tmpInfoR2 = appRoomInfo.r1.r2

如果没有appRoomInfo没有字段r1

tmpInfoR1返回undefined

tmpInfoR2报错TypeError: Cannot read property 'r2' of undefined

tmpInfo = {1:1}

tmpInfo[“1”]

tmpInfo.1 报错：SyntaxError: Unexpected number

数据修改

obj是引用不是拷贝

tmpJson = appRoomInfo[“r1”]

tmpJson[“room”] = “changeRoom”

则appRoomInfo[“r1”][“room”] === “changeRoom”

data是拷贝

tmpData = appRoomInfo[“r1”][“room”]

tmpData = “changeRoom”

但appRoomInfo[“r1”][“room”] !== “changeRoom”

数据遍历

for (key in appRoomInfo)

{

selfVal = appRoomInfo[key];

}

字符串转换

var value = 1;

var stringData = '{"test":value}';

jsonDataE = eval("(" + stringData + ")");

jsonDataP = JSON.parse(stringData);

结果

jsonDataE console - { test: 1 }

jsonDataP console - Caught exception: SyntaxError: Unexpected token v

at Object.parse (native)

var stringData = '{"test": value}';

jsonDataE = eval("(" + stringData + ")");

jsonDataP = JSON.parse(stringData);

结果

jsonDataE console - Caught exception: ReferenceError: f is not defined

jsonDataP console - Caught exception: SyntaxError: Unexpected token v

at Object.parse (native)

var stringData ="{'test':'value'}";

jsonDataE = eval("(" + stringData + ")");

jsonDataP = JSON.parse(stringData);

结果

jsonDataE console - { test: 'value' }

jsonDataP console - Caught exception: SyntaxError: Unexpected token '

at Object.parse (native)

var stringData = '{"test": "value"}';

jsonDataE = eval("(" + stringData + ")");

jsonDataP = JSON.parse(stringData);

结果

jsonDataE console - { test: 'value' }

jsonDataP console - { test: 'value' }

eval在解析字符串时，会执行该字符串中的代码

警告：关于JSON和eval需要注意的是：在代码中使用eval是很危险的，特别是用它执行第三方的JSON数据（其中可能包含恶意代码）时，尽可能使用JSON.parse()方法解析字符串本身。该方法可以捕捉JSON中的语法错误，并允许你传入一个函数，用来过滤或转换解析结果。此方法以被Firfox 3.5 、IE8 及 Safari 4 原生支持。大多数javascript类库包含的JSON解析代码会直接调用原生版本，如果没有原生支持的话，会调用一个略微不那么强大的非原生版本来处理。

## 3、数组

pkgVer.push(pkgVerTemp);

push的是引用,不是拷贝

http://www.cnblogs.com/qiantuwuliang/archive/2011/01/08/1930499.html

pkgVer[0] = pkgVerTemp;

也可以

数组链接

actData = actData.concat(targetRes["data"]["npcTargetChange"]);

数组删除

actDeleteData = actData.splice(i, 1);

actDeleteData为删除的项

actData为结果项

var a = [0, 1];

console.error(a[0]);

console.error(!a[0]);

console.error(!!a[0]);

console.error("-------------");

console.error(a[1]);

console.error(!a[1]);

console.error(!!a[1]);

console.error("-------------");

console.error(a[4]);

console.error(!a[4]);

console.error(!!a[4]);

[ERROR] console - 0

[ERROR] console – true

[ERROR] console - false

[ERROR] console - -------------

[ERROR] console - 1

[ERROR] console - false

[ERROR] console - true

[ERROR] console - -------------

[ERROR] console - undefined

[ERROR] console - true

[ERROR] console - false

//遍历1

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

alert(values[i]);

}

//遍历2

for(var ele in values){

alert(values[ele]);//下标

}

## 4、函数

if 是从左往右进行判断

# HTTP服务器

## 1、安装express

在 CMD 输入 npm install express -g

## 2、安装express-generator

在 CMD 输入 npm install express-generator -g

express 4.0 之后需要安装 generator 来使用命令

https://github.com/nswbmw/N-blog

## 3、方案设计

需求来源：

游戏数据统计，比如在线人数，提供修改接口，比如版本信息。

游戏充值接口为http服务，需要接受支付方的支付信息，并给予玩家发放道具以及充值信息。

a、在game-server中集成http服务，将其中的一个节点作为webconnector，好处是实时性高，坏处是游戏宕机，则服务延迟或者无法接收，造成充值信息无法通知到位。

b、在web-server中启动nodejs的express库来开启http服务，好处是game和web分离，且在web-server中有pomeloclient.js，可以直接连接game，做到实时更新。坏处是文档少，无法隐藏js源码，所有html均会暴露js文件，而js文件直接连接game，将game接口也暴露出去，需要做网络物理隔绝，或者信息来源验证。

c、使用Apache + php方式，好处是技术成熟，库很多，如thinkPHP，不会暴露源码。坏处是暂时不知道能不能连接game。如果无法连接game，当有充值信息的时候，则需要game服务器做轮询，浪费资源。

d、使用Apache + php方式，在game-server中集成http服务，php接收充值信息，通过game-server通知用户。

# 定时器任务

1、安装pomelo-schedule

下载地址：https://github.com/NetEase/pomelo-scheduler

在package.json中添加 "pomelo-schedule": ">=0.0.1"

运行npm-install.bat

2、示例代码：

var schedule = require('pomelo-schedule');

module.exports = function(app)

{

return new scheduleRemote(app);

};

var scheduleRemote = function(app)

{

this.app = app;

this.playerSchedule = schedule.scheduleJob({

start: Date.now(),

period: 3000,

count: 10

}, this.getGameVer, {

name: 'getGameVer'

});

};

var pro = scheduleRemote.prototype;

pro.getGameVer = function(data)

{

console.log("data");

}

//CancelJob

schedule.cancelJob(this.playerSchedule);

//Fire 10000ms after now, and run 10 times with a 1000ms interval.

var trigger1 = {

start : Date.now() + 10000, //Start time, use the time in date object

period : 1000, //Fire interval, the precision is millisecond

count : 10 //Fire times, in this case the trigger will fire 10 times.

}

//Fire right now, and run 10 times with 1000ms interval.

var trigger2 = {

period : 1000,

count : 10

}

//Fire right now, and run for ever with 1000ms interval.

var trigger3 = {

period : 1000

}

//Fire 3000ms after right now, run only once.

var trigger4 = {

start : Date.now() + 3000;

}

//The job will fire right now, run only once.

var trigger5 = {

}

//Illegal! The 'count' attribute cannot used alone without 'period'.

var trigger6 = {

count : 10;

}

# 数据库CRUD

## 1、SQL Server

连接方式：

var sql = require('msnodesql');

var sqlConn;

var sqlConfig = {

"driver" : "SQL Server Native Client 11.0",

"host" : "192.168.0.138",

"port" : "1435",

"database" : "master",

"user" : "sa",

"pwd" : "sa",

"useTrustedConnection" : true,

"userDB" : "DB\_HUNTER\_USERS",

"platformDB" : "DB\_HUNTER\_PLATFORM",

"gameDB" : "DB\_HUNTER\_GAME"

}

var driver = sqlConfig.driver;

var server = sqlConfig.host + ',' + sqlConfig.port;

var user = sqlConfig.user;

var pwd = sqlConfig.pwd;

var database = sqlConfig.database;

var useTrustedConnection = sqlConfig.useTrustedConnection;

var conn\_str = "Driver={" + driver + "}; Server={" + server + "}; " + (useTrustedConnection == true ? "Trusted\_Connection={Yes};" : "UID=" + user + "; PWD=" + pwd + "; ") + "Database={" + database + "};";

sql.open(config.sqlConnStr(app), function(err, conn)

{

if (err)

{

console.error(err);

return;

}

sqlConn = conn;

});

执行查询：

for (var arg in args)

{

procArgs = procArgs + arg + " = '" + args[arg] + "', ";

}

if ((!!procArgs.length) != false)

{

procArgs = procArgs.substring(0, (procArgs.length - 2));

}

sqlString = "DECLARE @dbReturn INT USE " + dbName + " EXEC @dbReturn = " + procName + " " + procArgs + " SELECT @dbReturn AS dbReturn";

sqlConn.query(sqlString,

function(err, results, more)

{

if (err)

{

cb(err, null);

return;

}

if ((!!more) == true)

{

if (results.length != 0)

{

procRes["data"] = results;

}

else

{

procRes["data"] = [];

}

}

else

{

procRes["dbReturn"] = results[0]["dbReturn"];

cb(null, procRes);

}

});

调用方式：

var procName = "SP\_GP\_GetPlayerHomeBulidSelf";

var args = {};

args["@dwUserId"] = uid;

pomelo.app.get('dbclient').gameQuery(procName, args, function(err, res)

{

if (err !== null)

{

cb(err, null);

}

else

{

if (res["dbReturn"] == 0)

{

resCode = CODE.COMMON.OK;

resData = {

"buildings": buildings

};

}

else

{

resCode = CODE.DB\_PLAYERHOME.DB\_GET\_HOMEBUILD\_ERROR\_CODE + res["dbReturn"];

resData = {};

}

resMsg = {

"code": resCode,

"data": resData

};

cb(null, resMsg);

}

});

PS：存储过程必须有返回值。

## 2、MongoDB

连接方式：

var mongoose = require('mongoose');

mongoose.connect("mongodb://user:pass@localhost:port/database" , function(err)

{

if (err)

{

console.error(err);

}

else

{

mongoose.connection.on("connected", function(){ mongoConnFlag = true; });

mongoose.connection.on("disconnected", function(){ mongoConnFlag = false; });

}

});

增删改查：

var db = mongoose.connection.db;

db.collectionNames(function(err, collections)

{

if (err !== null)

{

console.error("creatCollections : " + err);

}

else

{

var collectionsObj = {};

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

{

collectionsObj[collections[i]["name"]] = 0;

};

if ((!!collectionsObj["players"]) === false)

{

db.createCollection("players", function(err, playerCol)

{

db.createIndex("players", "uid", {"unique":true}, function(){})

});

}

}

});

如果不纯在名为players的collection时，创建它，并创建了一个名为uid的索引，且uid不能重复。ps：可以是有一条且仅有一条数据没有uid。

var playerSchema = mongoose.Schema({

uid : Number, hunter: Number

});

Schema架构是预先设定的一个字段集，之后save，update操作的所有数据，都不能超过这个预设的Schema，超过的部分，不会对数据库进行改写。upsert为true也不能，如需要增加字段，需要先改写Schema。

var playerModel = mongoose.model("playerInfo", playerSchema, "players");

model对应mongodb里的collection子集，如果不填写最后一个参数players，则默认对应mongodb里名为"playerInfos"的collection。

var ins = new playerModel({uid:1});

ins.save(function(err)

{

if (err) return console.error(err);

console.error("saved");

console.error(ins);

});

new playerModel({uid:1})中字段不能超过Schema的字段集。

playerModel.find(function (err, players)

{

if (err) return console.error(err);

console.error("find")

console.error(players)

players[0].update({$set:{hunter:888}}, function(err)

{

console.error("update");

});

});

查询model中的数据。并对model中数据进行修改或添加，修改或添加的字段不能超过Schema的字段集。

playerModel.update({uid:1}, {$set: {name: 3}}, function(err, res)

{

if (err) return console.error(err);

console.error(res);

});

修改model中的数据。

mongoose.connection.db.eval("add(12,3)", function(err, res){console.error(res);});

执行mongodb里的存储过程

# 异步处理

var async = require("async");

async.series([

function(cb) { cb(null, 1);},

function(cb) { cb(null, 2);},

function(cb) { cb(null, 3);},

function(cb) { cb(null, 4);},

function(cb) { cb(null, 5);},

], function(err, results) {

console.error('series err: ', err);

console.error('series results: ', results);

});

series err: undefined

series results: [ 1, 2, 3, 4, 5 ]

async.waterfall([

function(cb) { cb(null, 1);},

function(res, cb) { cb(null, 2);},

function(res, cb) { cb(null, 3);},

function(res, cb) { cb(null, 4);},

function(res, cb) { cb(null, 5);}

], function(err, results) {

console.error('waterfall err: ', err);

console.error('waterfall results: ', results);

});

waterfall err: null

waterfall results: 5

主要需要注意的是series err: undefined返回的是undefined。

# 加密解密

<https://github.com/NetEase/pomelo/wiki/pomelo-1.0%E6%96%B0%E7%89%B9%E6%80%A7>

考虑到安全性方面的问题，pomelo 1.0版本中增加了对tls及wss协议的支持

# 数据压缩

<https://github.com/NetEase/pomelo/wiki/%E6%B6%88%E6%81%AF%E5%8E%8B%E7%BC%A9>

# 充值计费

渠道充值信息先发送到游戏充值接口，如http://x.x.x.x/payAction.php，将数据先写入sqlserver数据库，然后通过POST等方式将数据传给pomelo服务器的http接口，得到通知的pomelo服务器根据信息添加玩家的购买结果，如果玩家在线，则推送给玩家充值成功的信息。

如果pomelo服务器未启动，则需要在第一次启动的时候检测未成功处理的玩家购买信息。



# Sublime

## 1、插件

按下Ctrl+Shift+P调出命令面板。

输入install 选择Package contral: Install Package，然后在列表中选中要安装的插件。

JsFormat：格式化js代码

AutoFileName：输入”/”即可看到相对于本文件夹的其他文件

Nodejs：node代码提示

## 2、快捷键

Preferences->Package Settings->JsFormat->Setting-Default

"brace\_style": "expand" 参数有四种"collapse", "expand", "end-expand", "expand-strict",用来决定格式化后花括号的位置

鼠标中键可以竖选。

PS：sublime没有检测代码的功能，可以使用webStorm或者Visual Studio Code先进行静态检测代码有无错误。

# WebStorm

open->选择文件夹

webStorm->run->edit configurations->+号->node.js

name->hunterPomelo

working directory-> D:\workspace\hunterPomelo\game-server

javascript file-> app.js

application parameters-> env-development

在servers.js中，room配置中添加 "args": "--debug=55556"

webStorm->run->edit configurations->+号->node.js remote debug

name->roomDebug

host->127.0.0.1

port->55556

single instance only