

Pomelo 启动流程

1. create app 创建应用 // application.js
2. Init app 初始化 //application.js
3. defaultConfiguration 配置// apputil.js
4. setupEnv //配置环境 apputil.js
5. Load master //加载master apputil.js loadConfigBaseApp //application.js
6. Load servers //加载 servers apputil.js
7. processArgs // 进程参数 apputil.js
8. Configlogger //配置日志 apputil.js

在执行完上面的操作之后，app进入到INITED状态，同时pomelo的createApp返回。

1. 创建应用 create app

*Application.set = function (setting, val, attach) {  
 if (arguments.length === 1) {  
 return this.settings[setting];  
 }  
 this.settings[setting] = val;  
 if(attach) {  
 this[setting] = val;  
 }  
 return this;  
};*

*Application.get = function (setting) {  
 return this.settings[setting];  
};*

1. 开启应用程序 app.start(); app.js

*Application.start = function(cb) {  
 //开始时间*

*this.startTime = Date.now();  
 if(this.state > STATE\_INITED) {  
 utils.invokeCallback(cb, new Error('application has already start.'));  
 return;  
 }  
 var self = this;  
 appUtil.startByType(self, function() {  
 appUtil.loadDefaultComponents(self);  
 var startUp = function() {* *appUtil.optComponents(self.loaded, Constants.RESERVED.START, function(err) {  
 self.state = STATE\_START;  
 if(err) {  
 utils.invokeCallback(cb, err);  
 } else {  
 logger.info('%j enter after start...', self.getServerId());  
 self.afterStart(cb);  
 }  
 });  
 };  
 var beforeFun = self.lifecycleCbs[Constants.LIFECYCLE.BEFORE\_STARTUP];  
 if(!!beforeFun) {  
 beforeFun.call(null, self, startUp);  
 } else {  
 startUp();  
 }  
 });*};

加载默认组件 apputil.js

appUtil.loadDefaultComponents(self); //加载默认组件 apputil.js

module.exports.loadDefaultComponents = function(app) {  
 var pomelo = require('../pomelo');  
 // load system default components  
 if (app.serverType === Constants.RESERVED.MASTER) {  
 // console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_++'+pomelo.master +'+++++\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_');  
 app.load(pomelo.master, app.get('masterConfig'));

// *This.setting[masterconfig]=function{new master;};pomelo.master='\_\_master\_\_'*

*Pomelo.components[master]=new master; this.loaded={new master }* } else {  
 app.load(pomelo.proxy, app.get('proxyConfig'));

*This.setting[*proxyConfig*]=function{new* proxy*;};pomelo.*proxy*='\_\_*proxy*\_\_'*

*Pomelo.components[*proxy*]=new* proxy*; this.loaded={new* proxy*}*  
 if (app.getCurServer().port) {  
 app.load(pomelo.remote, app.get('remoteConfig'));

*This.setting[*remoteConfig*]=function{new* remote*;};pomelo.*remote*='\_\_*remote*\_\_'*

*Pomelo.components[*remote*]=new* remote*; this.loaded={new* remote*}*  
 }  
 if (app.isFrontend()) {  
 app.load(pomelo.connection, app.get('connectionConfig'));  
 app.load(pomelo.connector, app.get('connectorConfig'));  
 app.load(pomelo.session, app.get('sessionConfig'));  
 // compatible for schedulerConfig  
 if(app.get('schedulerConfig')) {  
 app.load(pomelo.pushScheduler, app.get('schedulerConfig'));  
 } else {  
 app.load(pomelo.pushScheduler, app.get('pushSchedulerConfig'));  
 }  
 }  
 app.load(pomelo.backendSession, app.get('backendSessionConfig'));

*//This.setting[backendseession]=function{new backendseession;};pomelo.master='\_\_backendseession\_\_'*

*Pomelo.components[backendseession]=new backendseession; this.loaded={new master,new backendseession }*

app.load(pomelo.channel, app.get('channelConfig'));

*//This.setting[channelConfig]=function{new channel;};pomelo.channel='\_\_channel\_\_'*

*Pomelo.components[channel]=new channel; this.loaded={new master,new backendseession,new channel, }*

app.load(pomelo.server, app.get('serverConfig'));

*This.setting[serverconfig]=function{new server;};pomelo.channel='\_\_server\_\_'*

*Pomelo.components[server]=new server; this.loaded={new master,new backendseession,new channel,new server }*

}  
 app.load(pomelo.monitor, app.get('monitorConfig'));

*This.setting[monitorconfig]=function{new monitor;};pomelo.channel='\_\_monitor\_\_'*

*Pomelo.components[monitor]=new monitor;* *this.loaded={new master,new backendseession,new channel,new server ,new monitor}*

3.load 函数

Application.load = function(name, component, opts) {  
 if(typeof name !== 'string') {  
 opts = component;  
 component = name;  
 name = null;  
 if(typeof component.name === 'string') {  
 name = component.name;  
 }  
 }  
 if(typeof component === 'function') {  
 component = component(this, opts);  
 }  
 if(!name && typeof component.name === 'string') {  
 name = component.name;  
 }  
 if(name && this.components[name]) {  
 // ignore duplicat component  
 logger.warn('ignore duplicate component: %j', name);  
 return;  
 }  
 this.loaded.push(component);  
 if(name) {  
 // components with a name would get by name throught app.components later.  
 this.components[name] = component;  
 }  
 return this;

Pomelo get 追加属性 pomelo.js

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.backendSession* *\_\_\_\_\_\_\_\_\_\_\_load:module.exports = new backendseesion*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.channel \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new channel*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.connection \_\_\_\_\_\_\_\_\_\_\_\_load:module.exports = new connection*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.connector \_\_\_\_\_\_\_\_\_\_\_\_load:module.exports = new connector*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.**dictionary\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new dictionary*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.master\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new master*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.monitor\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new monitor*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.protobuf\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new protobuf*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.proxy\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new proxy*

*\_\_\_\_\_\_\_\_\_\_\_\_ pomelo.pushScheduler\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new pushscheduler*

*\_\_\_\_ \_\_\_\_\_ \_\_pomelo.remote\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new remote*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.server\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new server*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.session\_\_\_\_\_\_\_\_\_\_\_\_ load:module.exports = new session*

*fs.readdirSync(\_\_dirname + '/components').forEach(function (filename) {  
 if (!/\.js$/.test(filename)) {  
 return;  
 }*

*var name = path.basename(filename, '.js');  
 var \_load = load.bind(null, './components/', name);  
 Pomelo.components.\_\_defineGetter\_\_(name, \_load);  
 Pomelo.\_\_defineGetter\_\_(name, \_load);  
});**\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.serial \_\_\_\_\_\_\_\_\_\_\_load:module.exports =* *new Filter*

*\_\_\_\_\_\_\_\_\_\_\_\_ pomelo.time \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new Filter*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.timeout \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new Filter*

*\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.toobusy \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new Filter*

*fs.readdirSync(\_\_dirname + '/filters/handler').forEach(function (filename) {  
 if (!/\.js$/.test(filename)) {  
 return;  
 }  
 var name = path.basename(filename, '.js');  
 var \_load = load.bind(null, './filters/handler/', name);  
 Pomelo.filters.\_\_defineGetter\_\_(name, \_load);  
 Pomelo.\_\_defineGetter\_\_(name, \_load);  
});  
\_\_\_\_\_\_\_\_\_\_\_\_\_pomelo.rpcLog \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new Filter*

*\_\_\_\_\_\_\_\_\_\_\_\_ pomelo.toobusy \_\_\_\_\_\_\_\_\_\_\_load:module.exports = new Filter*

*fs.readdirSync(\_\_dirname + '/filters/rpc').forEach(function (filename) {  
 if (!/\.js$/.test(filename)) {  
 return;  
 }  
 var name = path.basename(filename, '.js');  
 var \_load = load.bind(null, './filters/rpc/', name);  
 Pomelo.rpcFilters.\_\_defineGetter\_\_(name, \_load);  
});  
//var \_load = load.bind(null, './components/', name);  
function load(path, name) {  
 if (name) {  
 return require(path + name);  
 }  
 return require(path);*

Start 启动

Application.start = function(cb) {  
 //开始时间  
 this.startTime = Date.now();  
 if(this.state > STATE\_INITED) {  
 utils.invokeCallback(cb, new Error('application has already start.'));  
 return;  
 }  
   
 var self = this;  
 appUtil.startByType(self, function() {  
 appUtil.loadDefaultComponents(self);  
 var startUp = function() {  
 /\*  
 \*\* param.load  
 \* loaded={new master,new backendseession,new channel,new server ,new monitor}  
 \* param.start  
 \* /  
 \*/  
 appUtil.optComponents(self.loaded, Constants.RESERVED.START, function(err) {  
 self.state = STATE\_START;  
 if(err) {  
 utils.invokeCallback(cb, err);  
 } else {  
 logger.info('%j enter after start...', self.getServerId());  
 //console.log('\_\_\_\_\_\_\_\_\_\_'+self.getServerId());  
 self.afterStart(cb);  
 }  
 });  
 };  
 var beforeFun = self.lifecycleCbs[Constants.LIFECYCLE.BEFORE\_STARTUP];  
 if(!!beforeFun) {  
 beforeFun.call(null, self, startUp);  
 } else {  
 startUp();  
 }  
 });  
};

apputil.js

*//appUtil.optComponents(self.loaded, Constants.RESERVED.START, function(err) {*

module.exports.optComponents = function(comps, method, cb) {  
 var i = 0;

//async.eachSeries = function (arr, iterator, callback) {  
 callback = callback || function () {};  
 if (!arr.length) {  
 return callback();  
 }  
 var completed = 0;  
 var iterate = function () {  
 var sync = true;  
 iterator(arr[completed], function (err) {  
 if (err) {  
 callback(err);  
 callback = function () {};  
 }  
 else {  
 completed += 1;  
 if (completed >= arr.length) {  
 callback(null);  
 }  
 else {  
 if (sync) {  
 async.nextTick(iterate);  
 }  
 else {  
 iterate();  
 }  
 }  
 }  
 });  
 sync = false;  
 };  
 iterate();  
};  
async.forEachSeries = async.eachSeries;

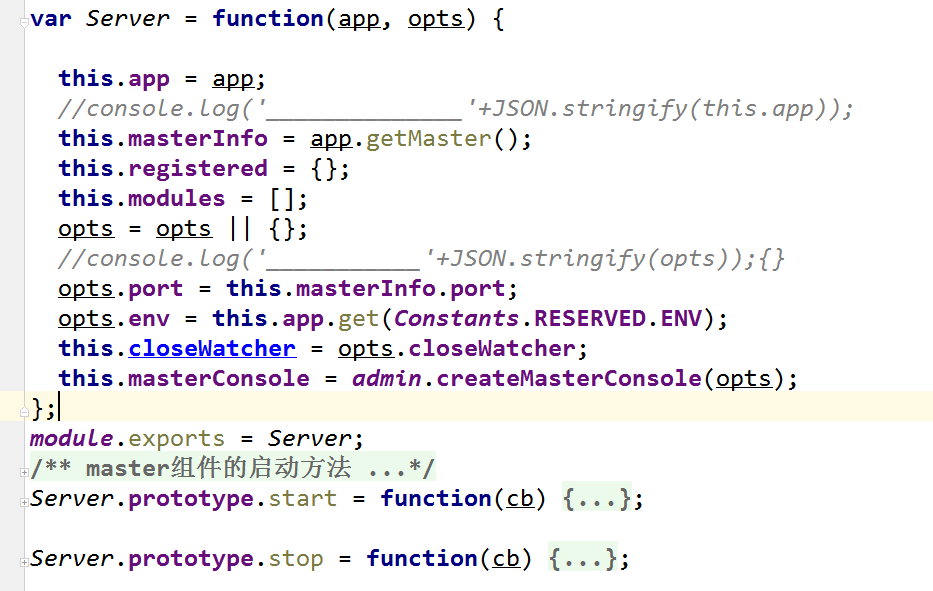
async.forEachSeries(comps, function(comp, done) {

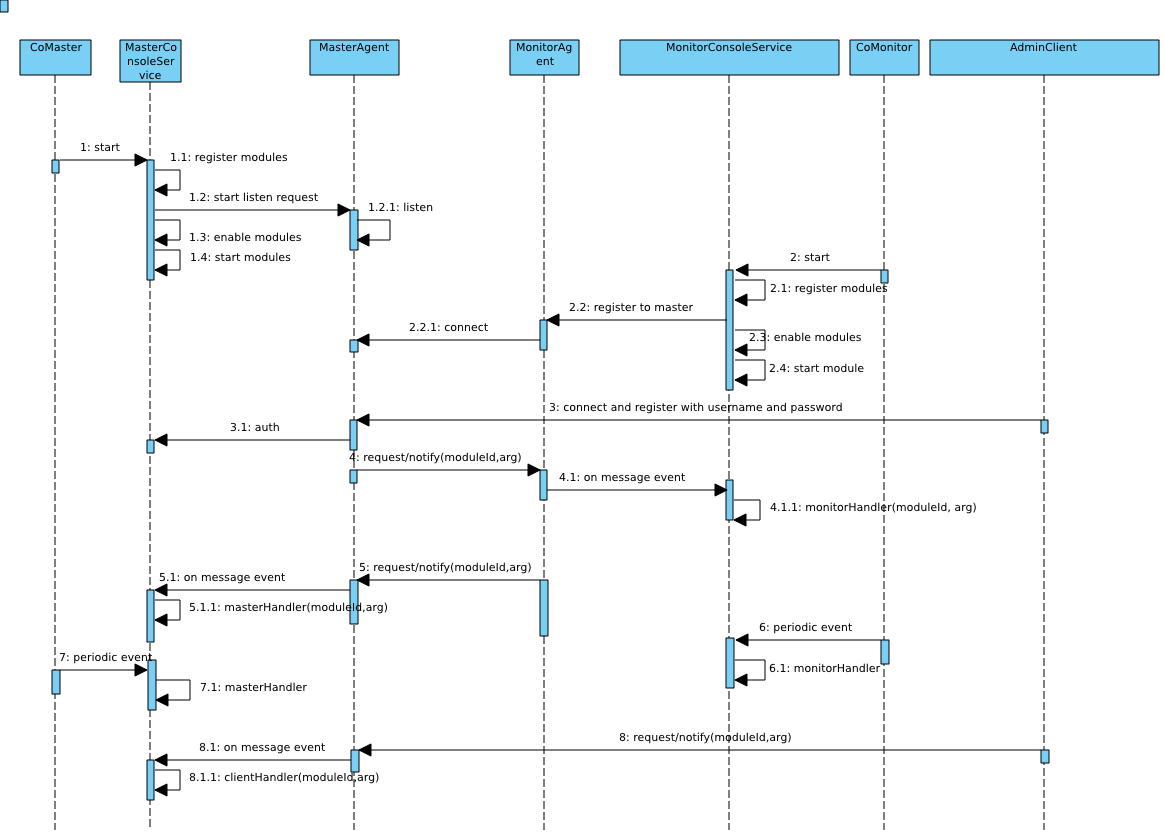
i++;

if (typeof comp[method] === 'function') {

// comp.start() master.start= server.start  
 comp[method](done);  
 } else {  
 done();  
 }  
 }, function(err) {  
 if (err) {  
 if(typeof err === 'string') {  
 logger.error('fail to operate component, method: %s, err: %j', method, err);  
 } else {  
 logger.error('fail to operate component, method: %s, err: %j', method, err.stack);  
 }  
 }  
 utils.invokeCallback(cb, err);  
 });  
};

MASTER.JS 组件组成部分





Master.server 启动

*Server.prototype.start = function(cb) {  
 moduleUtil.registerDefaultModules(true, this.app, this.closeWatcher);  
 moduleUtil.loadModules(this, this.masterConsole);*

*[2017-06-19 07:58:11.113] [INFO] console - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"\_\_masterwatcher\_\_"*

*[2017-06-19 07:58:11.115] [INFO] console - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"watchServer"*

*[2017-06-19 07:58:11.115] [INFO] console - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"\_\_console\_\_"  
 var self = this;  
 // start master console  
 this.masterConsole.start(function(err) {  
 if(err) {  
 process.exit(0);  
 }  
 moduleUtil.startModules(self.modules, function(err) {  
 if(err) {  
 utils.invokeCallback(cb, err);  
 return;  
 }  
 if(self.app.get(Constants.RESERVED.MODE) !== Constants.RESERVED.STAND\_ALONE) {  
 starter.runServers(self.app);  
 }  
 utils.invokeCallback(cb);  
 });  
 });*

*registerDefaultModules 注册*

*Application.registerAdmin = function(moduleId, module, opts) {  
 var modules = this.get(Constants.KEYWORDS.MODULE);//null  
 if(!modules) {  
 modules = {};  
 this.set(Constants.KEYWORDS.MODULE, modules);  
 // this.setting[Constants.KEYWORDS.MODULE]=null  
 }*

*if(typeof moduleId !== 'string') {  
 opts = module;  
 module = moduleId;  
 if(module) {  
 moduleId = module.moduleId;  
 }  
 }  
 if(!moduleId){  
 return;  
 }  
 modules[moduleId] = {  
 moduleId: moduleId,  
 module: module,  
 opts: opts  
 };  
 //console.log('\_\_\_\_\_\_\_\_\_\_'++'\_\_\_\_\_\_\_\_\_\_\_');  
};*

*loadModules 加载模块*

*var pro = module.exports;*

*pro.loadModules = function(self, consoleService) {  
 // load app register modules  
 var \_modules = self.app.get(Constants.KEYWORDS.MODULE);  
 if(!\_modules) {  
 return;  
 }  
 var modules = [];  
 for(var m in \_modules){  
 //console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_'+m);  
 modules.push(\_modules[m]);  
 }  
 var record, moduleId, module;  
 for(var i=0, l=modules.length; i<l; i++) {  
 record = modules[i];  
 if(typeof record.module === 'function') {  
 module = record.module(record.opts, consoleService);  
 } else {  
 module = record.module;  
 }  
 moduleId = record.moduleId || module.moduleId;  
 if(!moduleId) {  
 logger.warn('ignore an unknown module.');  
 continue;  
 }  
 consoleService.register(moduleId, module);   
 self.modules.push(module);*

*this.modules = [];  
  
};*

*}};*

*this.masterConsole.start*

*this.masterConsole.start(function(err) {  
 if(err) {  
 process.exit(0);  
 }  
 moduleUtil.startModules(self.modules, function(err) {  
 if(err) {  
 utils.invokeCallback(cb, err);  
 return;  
 }*

*//Moudleutil.js 递归启动模块start moudleutil.js文件 实现缺省注册、 加载模块、 开启模块组 、 功能*

*pro.startModules = function(modules, cb) {  
 // invoke the start lifecycle method of modules  
 if(!modules) {  
 return;  
 }  
 startModule(null, modules, 0, cb);  
};*

*var startModule = function(err, modules, index, cb) {  
 if(err || index >= modules.length) {  
 utils.invokeCallback(cb, err);  
 return;}  
 var module = modules[index];  
 if(module && typeof module.start === 'function') {  
 module.start(function(err) {  
 startModule(err, modules, index + 1, cb);  
 });  
 } else {  
 startModule(err, modules, index + 1, cb);  
 }};  
 if(self.app.get(Constants.RESERVED.MODE) !== Constants.RESERVED.STAND\_ALONE) {  
 starter.runServers(self.app); 开启服务器  
 }  
 utils.invokeCallback(cb);  
 });  
});*

*MASTERCONSOLE 监听事件*

**this**.**masterConsole**.on(**'error'**, **function**(err) {  
 **if**(!!err) {  
 ***logger***.error(**'masterConsole encounters with error: '** + err.**stack**);  
 **return**;  
 }});  
**this**.**masterConsole**.on(**'reconnect'**, **function**(info){  
 self.**app**.addServers([info]);});  
*// monitor servers disconnect event***this**.**masterConsole**.on(**'disconnect'**, **function**(id, type, info, reason) {  
 ***crashLogger***.info(***util***.format(**'[%s],[%s],[%s],[%s]'**, type, id, Date.now(), reason || **'disconnect'**));  
 **var** count = 0;  
 **var** time = 0;  
 **var** pingTimer = **null**;  
 **var** server = self.**app**.getServerById(id);  
 **var** stopFlags = self.**app**.get(***Constants***.**RESERVED**.**STOP\_SERVERS**) || [];  
 **if**(!!server && (server[***Constants***.**RESERVED**.**AUTO\_RESTART**] === **'true'** || server[***Constants***.**RESERVED**.**RESTART\_FORCE**] === **'true'**) && stopFlags.indexOf(id) < 0) {  
 **var** *setTimer* = **function**(time) {  
 pingTimer = setTimeout(**function**() {  
 ***utils***.ping(server.**host**, **function**(flag) {  
 **if**(flag) {  
 *handle*();  
 } **else** {  
 count++;  
 **if**(count > 3) {  
 time = ***Constants***.**TIME**.**TIME\_WAIT\_MAX\_PING**;  
 } **else** {  
 time = ***Constants***.**TIME**.**TIME\_WAIT\_PING** \* count;  
 }  
 *setTimer*(time);  
 }});}, time);};  
 *setTimer*(time);  
 **var** *handle* = **function**() {  
 ***clearTimeout***(pingTimer);  
 ***utils***.checkPort(server, **function**(status) {  
 **if**(status === **'error'**) {  
 ***utils***.invokeCallback(cb, **new** Error(**'Check port command executed with error.'**));  
 **return**;  
 } **else if**(status === **'busy'**) {  
 **if**(!!server[***Constants***.**RESERVED**.**RESTART\_FORCE**]) {  
 ***starter***.kill([info.pid], [server]);  
 } **else** {  
 ***utils***.invokeCallback(cb, **new** Error(**'Port occupied already, check your server to add.'**));  
 **return**;  
 }}  
 setTimeout(**function**() {  
 ***starter***.run(self.**app**, server, **null**);  
 }, ***Constants***.**TIME**.**TIME\_WAIT\_STOP**);  
 });  
 };  
 }});**this**.**masterConsole**.on(**'register'**, **function**(record) {***starter***.bindCpu(record.**id**, record.**pid**, record.**host**);});  
**this**.**masterConsole**.on(**'admin-log'**, **function**(log, error) {  
 **if**(error) {  
 ***adminLogger***.error(**JSON**.stringify(log));  
 } **else** {  
 ***adminLogger***.info(**JSON**.stringify(log));}});

Master stop

Server.prototype.stop = function(cb) {  
 this.masterConsole.stop();  
 process.nextTick(cb);  
};

MASTER CONSOLE 服务分析

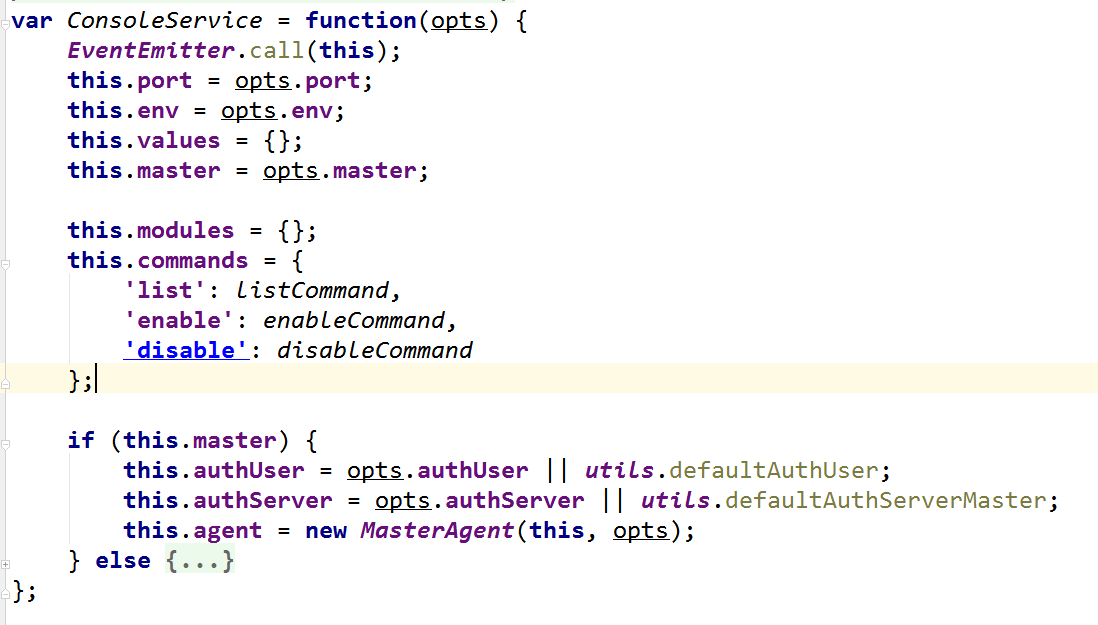
**this**.**masterConsole** = ***admin***.**createMasterConsole**(opts);//master.js

Pomelo\_admin index.js

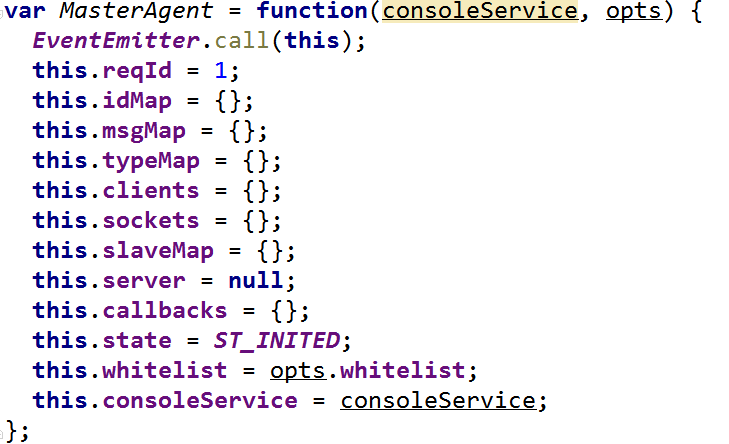


**masterConsole来自consoleservice 的工厂模式 consoleservice服务是一个封装很全面的模块**

*Consoleservice 类的属性*

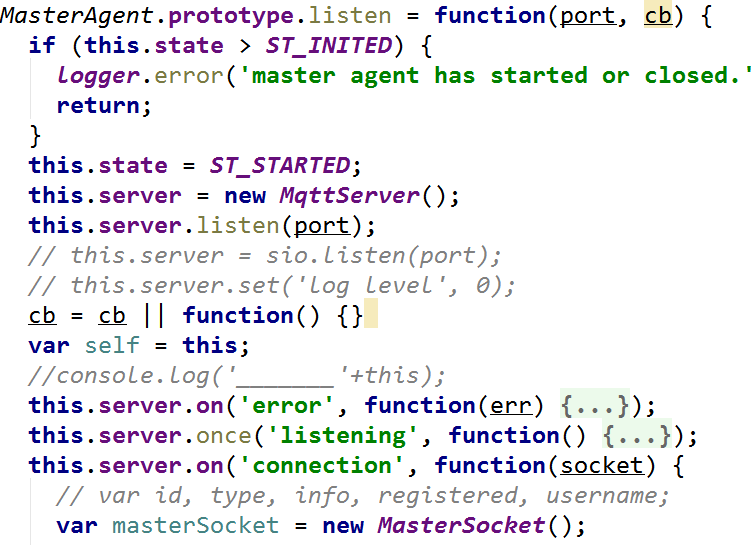
**

*MasterAgent 代理类*

**

*mtqqserver 包装类*

*这里我们关注一下 masteragent listen方法*

**

*MASTERagent 类封装了 mtqq类 mastersocket 包装了 sock*

**我们看看 master masterconsole masteragent mastersock 四大类 这里我们关注一下内部是如何运作的。**

*这里我们关注一下* **masterconsole 的构造start**

**ConsoleService.prototype.start = function(cb) {  
 if (this.master) {  
 var self = this;  
 this.agent.listen(this.port, function(err) {**

**// masterconsole的启动调用了masteragent 的listen 在函数里 创建了mastersock  
 if (!!err) {  
 utils.invokeCallback(cb, err);  
 return;  
 }**

**// 在listen 回调中导出 触发 master 注册 重连 断开 事件  
 exportEvent(self, self.agent, 'register');   
 exportEvent(self, self.agent, 'disconnect');  
 exportEvent(self, self.agent, 'reconnect');  
 process.nextTick(function() {  
 utils.invokeCallback(cb);  
 });  
 });  
 } else {  
 logger.info('try to connect master: %j, %j, %j', this.type, this.host, this.port);  
 this.agent.connect(this.port, this.host, cb);  
 exportEvent(this, this.agent, 'close');  
 }  
 exportEvent(this, this.agent, 'error');  
  
 for (var mid in this.modules) {  
 this.enable(mid);  
 }  
};**

**这里关注一下 这段代码的精髓 是如何触发事件**

**exportEvent(self, self.agent, 'register');**

**var exportEvent = function(outer, inner, event) {**

**// this.agent.on(‘register’,function{**

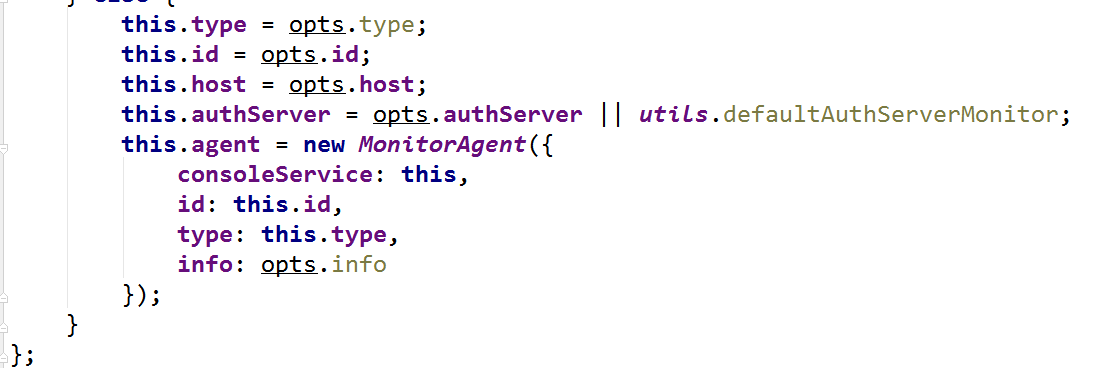
**Var args=arguments[0-args.length]; })  
 inner.on(event, function() {  
   
 var args = Array.prototype.slice.call(arguments, 0);  
   
 args.unshift(event);**

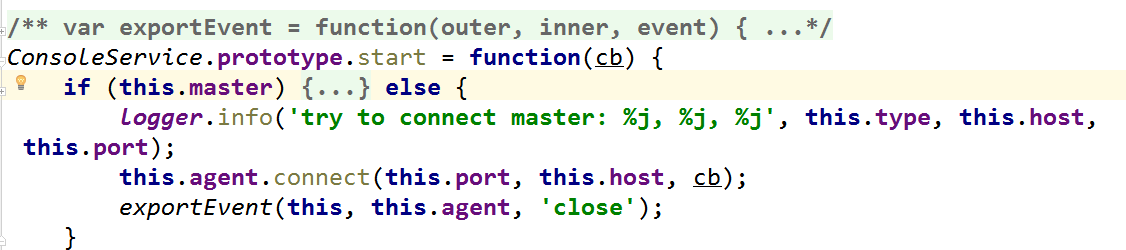
**// 这里将 event 添加到数组args[0]中；**

**outer.emit.apply(outer, args); // this.emit .apply(this, args[0], ...); 这里将函数作用域this,将arg[0]作为 event 触发 那么这里调用的方法就是 this.masterConsole.on('register', function(record) 来自master.js**

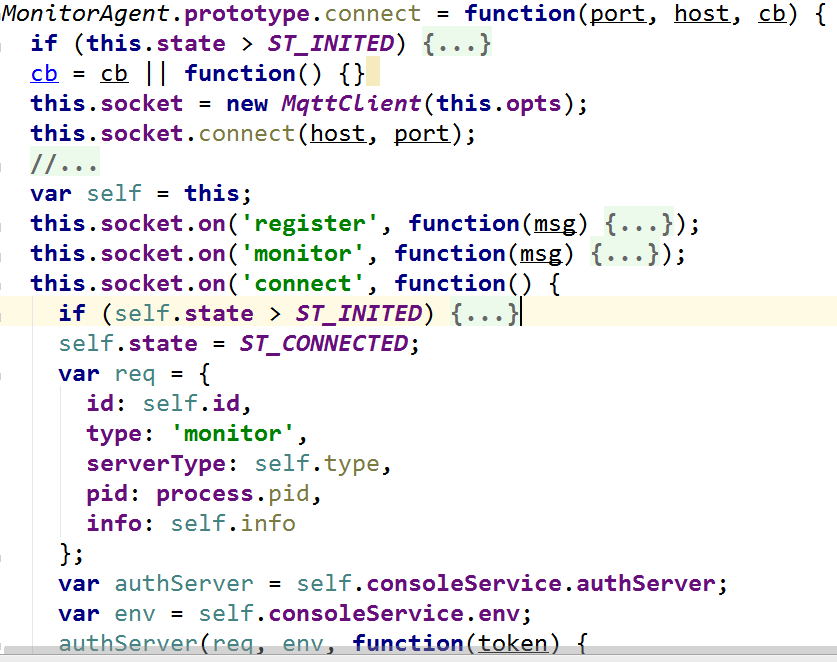
**});  
};**

**接下来我们看看 monitor 服务是如何运作的**

****

****

**与master 类似，monitor agent 类似客户端 connect master agent**

****

**这里 构造了一个 mqttclient对象 req 是发送的请求 authserver 认证服务方法 将抛出一个发送事件，这个事件 这里简单的说下authserver的出处**

**来自consoleserver.js**

**this**.**authServer** = opts.**authServer** || ***utils***.defaultAuthServerMonitor;

***utils***.defaultAuthServerMonitor**来自 pomelo-admin util-utils 文档 参数最终来自config-adminserver.JSON**

**//monitoragent.js**

**authServer(req, env, function(token) {  
 req['token'] = token;  
 self.doSend('register', req);  
});**

**MonitorAgent.prototype.doSend = function(topic, msg) {  
 this.socket.send(topic, msg);  
}**

**MqttClient.prototype.send = function(topic, msg) { // this.socket 来自mqttcon  
 this.socket.publish({  
 topic: topic,  
 payload: JSON.stringify(msg)})}**

**//mqttcon.js**

**['connect',  
 'connack',  
 'publish',  
 'puback',  
 'pubrec',  
 'pubrel',  
 'pubcomp',  
 'subscribe',  
 'suback',  
 'unsubscribe',  
 'unsuback',  
 'pingreq',  
 'pingresp',  
 'disconnect'].forEach(function(cmd) {  
 Connection.prototype[cmd] = function(opts, cb) {  
 opts = opts || {}  
 opts.cmd = cmd;  
 this.write(opts)  
 if (cb)  
 setImmediate(cb)  
 }  
 });**

**//Mtqqserver.js**

**socket.on('publish', function(pkg) {  
 var topic = pkg.topic;  
 var msg = pkg.payload.toString();  
 msg = JSON.parse(msg);  
 //console.log('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'+topic);  
 //logger.debug('[MqttServer] publish %s %j', topic, msg);  
 socket.emit(topic, msg);  
});**

**接下来观看一下 masteragent 的socket的注册事件**

**//masteragent.js**

**socket.on('register', function(msg) {  
 // register a new connection  
 //  
 masterSocket.onRegister(msg);  
});**

**从这里可以看出 monitor 通过调用 sock.publish 方法触发 publish事件后触发register事件 从而达到注册。 一个基本的master 与monitor运行流程 就如UMI 分析图一样。**