Table of Contents

- Introduction 1.1
- 开发约定 1.2
  - PHP编码规范 1.2.1
  - OMS编码规范 1.2.2
- 安装部署 1.3
  - 运行环境要求 1.3.1
  - linux单机部署(>= V3.5.0) 1.3.2
  - 集群部署 1.3.3
  - 日志服务 1.3.4
  - 任务管理 1.3.5
- 框架机制 1.4
  - APP目录结构及说明 1.4.1
  - 缓存机制 1.4.2
  - 导出机制 1.4.3
- 核心模块设计与说明 1.5
  - 订单预处理机制 1.5.1
  - 规则筛选器 1.5.2
  - 自动审单插件 1.5.3
  - 系统对接 1.5.4
    - 前端销售平台 1.5.4.1
    - 后端仓库 1.5.4.2
    - 电子发票 1.5.4.3
  - 跨境申报 1.5.5
  - 电子面单 1.5.6
  - 库存管理 1.5.7
    - 库存深度 1.5.7.1
    - 库存冻结 1.5.7.2
  - 商品 1.5.8
  - 仓库 1.5.9
  - 采购 1.5.10
  - 人工分派 1.5.11
  - 人工审单 1.5.12
  - 单据打印 1.5.13
  - 校验 1.5.14
  - 发货 1.5.15
  - 订单分组 1.5.16
  - 自动分派 1.5.17
  - 自动审单 1.5.18
  - 自动分仓 1.5.19
- 常见问题FAQ 1.6

安装部署 1.6.1
操作使用 1.6.2
开发问题 1.6.3
系统联通 1.6.4
对接BBC配置 1.6.4.1
对接FTP WMS 1.6.4.2
ec联通oms绑定步骤 1.6.4.3

开发手册

开发约定

PHP编码规范

OMS目前已针对PHP5.6版本做兼容，在写法上面需要注意下面几个点：

引用的使用

实例化类的对象时底层已经使用引用方式，所以不用再在类的实例化的时候增加引用符号

错误的写法：

```
$orderObj = &app::get('ome')->model('orders');
```

正确的写法：

```
$orderObj = app::get('ome')->model('orders');
```

方法定义的参数为引用方式，实际调用方法时参数还写着引用符号

错误的写法：

```
function test(&$a){

}

$this->test(&$a);
```

正确的写法：

```
function test(&$a){

}

$this->test($a);
```

参数的调用

参数名使用全局变量将会导致一个致命错误

错误的写法：

```
function foo($_GET, $_POST){

}
```

方法的调用

定义的方法是一个共有方法，调用的时候不能使用静态方法的方式调用

错误的写法：

```
function test(){

}

self::test();
```

正确的写法：

```
static public function test(){

}

self::test();
```

类的继承，方法重写

子类继承父类的方法，方法命名的参数与保持一致

错误的写法：

```
class a{
    function test($param1,&$param2){

    }
}

class b extends class a{
    function test(){

    }
}
```

正确的写法：

```
class a{
    function test($param1,&$param2){

    }
}

class b extends class a{
    function test($param1,&$param2){

    }
}
```

方法套方法

写代码一时的便捷，方法套方法的使用

错误的写法：

```
kernel::single('base_xml')->xml2array(file_get_contents(ROOT_DIR.'/config/deploy.xml'),'base_deploy');
```

正确的写法：

```
$deploy = file_get_contents(ROOT_DIR.'/config/deploy.xml');
kernel::single('base_xml')->xml2array($deploy,'base_deploy');
```

更加详细的**PHP**版本升迁变更请查看：

1. 从 PHP 5.3.X 迁移到 PHP 5.4.X
2. 从 PHP 5.4.x 迁移到 PHP 5.5.x
3. 从PHP 5.5.x 移植到 PHP 5.6.x

OMS编码规范

安装部署

运行环境要求

最低配置

单机 4CPU 8G内存 5M带宽

主流客户使用推荐

推荐商家直接购买阿里云产品，减轻商家相关服务器硬件的运维成本，并且可以按需弹性升级(方便、快捷)

1.云主机ECS(web) 8cpu 16G内存

2.云数据库RDS(myqsl db) 中型

注意事项

项目交付上，一定要告知商家/客户在服务器硬件投入上的利弊，严禁将多套软件产品(不仅仅指商派体系的软件产品)部署在一台单机上运行，以免因小失大！

linux单机部署(>= V3.5.0)

基于商派封装的源进行快速简易的安装

这是一个使用yum源安装shopex OMS的标准化教程，可以大大减轻搭建环境的工作量。

我们希望提供给部署运维人员的不仅仅是一个教程，也是一个思路，而不是一个傻瓜式的部署步骤(ctrl+c加ctrl+v)，在实际安装部署过程中，由于操作系统或环境的差异，出现依赖库冲突等问题，可自行调整或咨询官方后进行解决。

约定

1. 操作系统为Centos 6.5 x64 或6系列的以上版本（centos 6.x 或者el6）
2. 系统基于最小化安装即可

PS : 7.0以上 是不支持的,注意系统必须是64位

关闭**selinux**

```
#命令行临时关闭 SeLinux
setenforce 0
#修改SeLinux配置，下次启动则默认关闭
修改  /etc/selinux/config  文件
将  SELINUX=enforcing  改为  SELINUX=disabled
```

初始化**yum**源

将shopex-lnmp源加入到系统中，在线地址查看有哪些软件 http://mirrors.shopex.cn/shopex/shopex-lnmp/ 可以安装

```
yum install wget -y
cd /etc/yum.repos.d/
wget http://mirrors.shopex.cn/shopex/shopex-lnmp/shopex-lnmp.repo
```

安装**epel**扩展源

```
yum install epel-release -y
```

安装**OMS**环境，主要是**php apache mysql zend redis**以及任务管理器**(taskmgr)**的运行**php(**支持线程**)**

```
apache+php+mysql51
yum install -y httpd php-pdo php-xml php-pecl-imagick php-soap php php-fpm php-bcmath php-pecl-memcached php-pecl-igbinary php
-common php-mysql php-mcrypt php-mbstring php-pecl-memcache php-pecl-redis php-gd php-cli php-xmlrpc php-pecl-mongo php-ZendGu
ardLoader mysql51 redis

安装 php-pthreads
yum install php-pthreads -y
```

配置文件以及相关目录说明

```
站点根目录 建议/data/httpd/oms.xxx.com(xxx为具体客户的真实域名)

apache:
配置文件目录 /etc/httpd/conf/httpd.conf

php:
配置文件  /etc/php.ini
php 扩展配置文件目录 /etc/php.d/
zend 配置文件   /etc/php.d/Zend.ini

mysql5.1:
basedir /usr/local/mysql51
datadir /data/mysql/3306
配置文件 /usr/local/mysql51/my.cnf

可以使用 mysqladmin password 'xxxxxx' 设置mysql的root密码

php-pthreads:
basedir /usr/local/php-pthreads

memcache:
配置文件 /etc/sysconfig/memcached

redis :
配置文件 /etc/redis.conf

关于启动命令
apache
service httpd start
mysql51
service mysql51 start
memcache
service memcached start
redis
service redis start

关于加入开机启动项
chkconfig xxx on
xxx 为服务名
```

php配置

```
修改 /etc/php.ini
date.timezone = "Asia/Shanghai" //调整时区

修改 /etc/php.d/zend.ini
zend_loader.license_path = '/data/httpd/oms.xxx.com/config/developer.zl' //请根据实际情况进行配置
```

1) 复制代码到安装目录

```
/data/httpd/oms.xxx.com/
```

2) 对 **data,config,public** 目录赋 **777**

```
chmod -R 777 /data/httpd/oms.xxx.com/data
chmod -R 777 /data/httpd/oms.xxx.com/config
chmod -R 777 /data/httpd/oms.xxx.com/public
```

3) 配置**apache**

```
#ServerName www.example.com:80
修改为
ServerName oms.xxx.com:80

DocumentRoot "/var/www/html"
修改为
DocumentRoot "/data/httpd/oms.xxx.com/"

<Directory "/var/www/html">
修改为
<Directory "/data/httpd/oms.xxx.com/">

当然如果apache服务下设置多个站点话，我们也可以设置vhost：

<VirtualHost *:80>
    DocumentRoot "/data/httpd/oms.xxxx.com"
    ServerName oms.xxxx.com
    <Location /script/>
        Order deny,allow
        Deny from all
    </Location>
    <Location /config/>
        Order deny,allow
        Deny from all
    </Location>
    <Directory "/data/httpd/oms.xxxx.com">
        Options FollowSymLinks
        AllowOverride None
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>

使用/etc/init.d/httpd configtest进行配置文件检测，出现 Syntax OK 表示检查没有问题，使用 /etc/init.d/httpd start 启动web服务
```

4) 开始安装应用

```
直接访问OMS访问地址，一步步执行下去即可。
```

5) taskmgr app 配置(详细说明请查看"任务管理"章节说明)

```
App/taskmgr/config/config.php
一般默认安装后只要修改以下两个参数
//设置定时任务的请求域名，建议直接走内网127.0.0.1(或内网ip)，执行脚本服务器能通过127.0.0.1(或内网ip)访问OMS站点即可，如果设置真实域名还要从本机跳到外网再绕一圈再进来，有点多余。
define('DOMAIN', '127.0.0.1');

//定义内部任务请求的token
define('REQ_TOKEN', 'SzjJ7VF9w2Dy3yt4K9vcV3LEwFc5pX3cXHu');
```

6) crontab 部署

```
部署前，检查下redis队列有没有起来，没有的话执行/etc/init.d/redis start

//任务管理器守护进程
* * * * * /bin/sh /alidata1/httpd/oms.xxx.com/app/taskmgr/check.sh

//历史任务处理日志清理
0 0 * * * /bin/sh /alidata1/httpd/oms.xxx.com/app/taskmgr/cleanlogs.sh
```

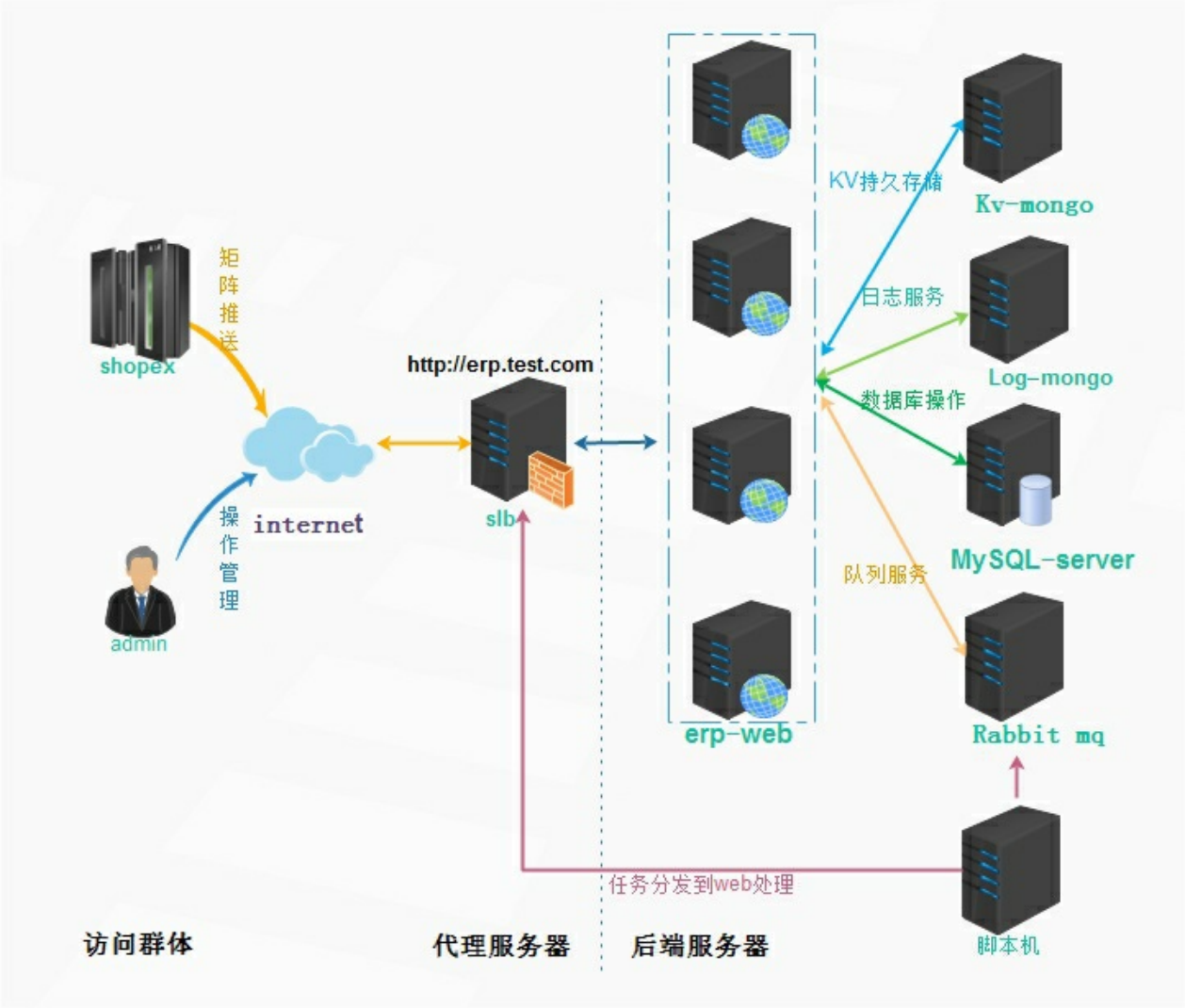
完成以上步骤后，OMS系统即可正常访问使用。

集群部署

集群部署方案说明

详尽的说明集群部署情况下，各系统的分布分工和承担业务的职责。

推荐集群部署示意图



系统组成说明

Slb负载均衡服务

针对访问请求做负载均衡，分发到不同Web服务上进行处理。

Web服务

主要存放OMS系统的程序代码、Log日志服务的服务端代码

Web机配置的Web Server建议是Apache

在Web机上我们还可部署相应memcache缓存服务，提供给OMS系统应用调用(支持memcache集群组的配置)

Kv服务

推荐基于mongodb方式的存储引擎，kv在OMS系统中主要涉及的是一些后台系统配置的信息，数据量小，在特定的处理时会调用到这些配置信息做逻辑处理。

Log日志服务

开发手册中有单独一个章节说明Log服务，这里就不废话了。

日志服务主要还是将系统的基于Rpc的一些请求和响应的日志存放到基于capped类型的mongodb存储引擎中，减轻MySql数据库的读写压力。

MySql服务

数据库存储服务，这块地球人都懂不说了。

Mq队列任务服务

同日志服务，有单独章节说明。

web机的一些操作可能会产生任务，比如后台导出、批量发货校验，这些复杂的任务会存储到队列服务中。

脚本服务

主要将Mq队列任务服务中的具体任务工作分发到web机进行最终的处理。

日志服务

给**ERP**部署独立的日志服务

将日志服务从数据库存储剥离开，独立部署基于MONGODB的存储服务，提升系统整体性能及扩展性的架构。

Step 1

安装mongodb服务

Step 2

修改logsystem里的配置信息文件config.php

```
define('STORAGE_ENGINE','mongo');#日志存储的介质
define('SYS_API_LOGS',FALSE);#是否写文件记录接收的日志参数
define('SYS_ERROR_LOGS',FALSE);#是否写文件记录接收的错误日志参数
define('SYS_ERROR_REPORTING',E_ALL & ~E_NOTICE);#php报错级别 该配置不动即可
define('SYS_SETTIMEZONEAPICOUNTLOGS_TIME',300);#单位秒 该配置不动即可

define('SYS_LOGDATABASE_NAME','logdb');#数据库名称
define('SYS_LOGDATABASE_NUMS',1);#数据库数量

define('SYS_LOG_DOMAIN','api.log.taoex.com');#日志服务系统的域名
define('SYS_LOG_INTOQUEUE',false);#请求进来的日志内容是否进队列走一层

#具体mq链接配置信息
$GLOBALS['_MQ_LOG_CONFIG'] =  array(
    'host' => '127.0.0.1',
    'port' => '5672',
    'login' => 'erp',
    'password' => 'erp_3315',
    'vhost' => 'erp_callback',
    'routerkey' => 'tg.sys.*.*',
);
```

日志MQ机制可以无视，SYS_LOG_INTOQUEUE参数默认false不用去动，其他参数默认也可直接使用，非技术深究就保持默认即可。

修改logsystem里mongo服务的配置信息文件 server.inc.php 修改具体的mongo服务的 ip 和端口号，默认：

```
return array(
    '127.0.0.1' => array(
        '0' => '27017',
    ),
);
```

Step 3

修改rockmongo(mongo日志管理工具)里的配置文件config.php

```
$i = 0;

$MONGO["servers"][$i]["mongo_name"] = "ERP_LOG:27017";//mongo server name
$MONGO["servers"][$i]["mongo_host"] = "127.0.0.1";//mongo host
$MONGO["servers"][$i]["mongo_port"] = "27017";//mongo port
$MONGO["servers"][$i]["mongo_timeout"] = 0;//mongo connection timeout
$MONGO["servers"][$i]["mongo_auth"] = false;//enable mongo authentication?
$MONGO["servers"][$i]["control_auth"] = true;//enable control users, works only if mongo_auth=false
$MONGO["servers"][$i]["control_users"]["erp"] = "erp_3315";//one of control users ["USERNAME"]=PASSWORD, works only if mongo_a
uth=false
$MONGO["servers"][$i]["ui_only_dbs"] = "";//databases to display
$MONGO["servers"][$i]["ui_hide_dbs"] = "";//databases to hide
$MONGO["servers"][$i]["ui_hide_collections"] = "";//collections to hide
$MONGO["servers"][$i]["ui_hide_system_collections"] = false;//if hide the system collections

$i++;
```

主要配置name host port users 4个参数即可。配置好后，域名访问配置好，能正常登陆即可。

Step 4

将logsystem和rockmongo配置web访问，用二级域名。

logsystem日志服务的访问域名可以是内网，rockmongo管理地址最好是外网不然不方便管理。

两个代码可以和OMS程序代码放在一个目录同级。

Step 5

php命令行执行logsystem里script\init.php的代码，自动创建日志服务的库和表。

执行命令后使用rockmongo管理工具登陆查看检查是否创建成功。

默认的库名叫logdb0,表名为tg_apilog和tg_apilog_detail。

Step 6

使用rockmongo修改两个表的属性。

参数为:

is capped 是否固定大小 勾选。

size 最大占用存储空间 300M-500M 具体看服务器资源，按实际情况调节。

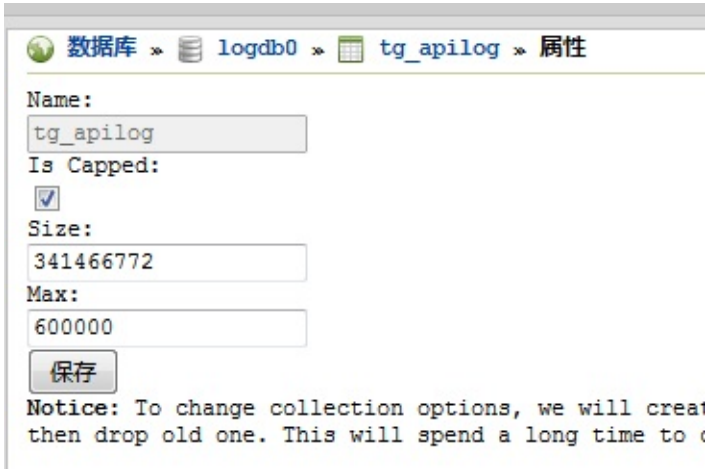
max 最多记录行数 60w 具体看服务器资源，按实际情况调节。

在右侧顶部点击更多里的属性，编辑is capped size max三个参数，设置完后，可以同样点表里的统计查看是否设置成功。

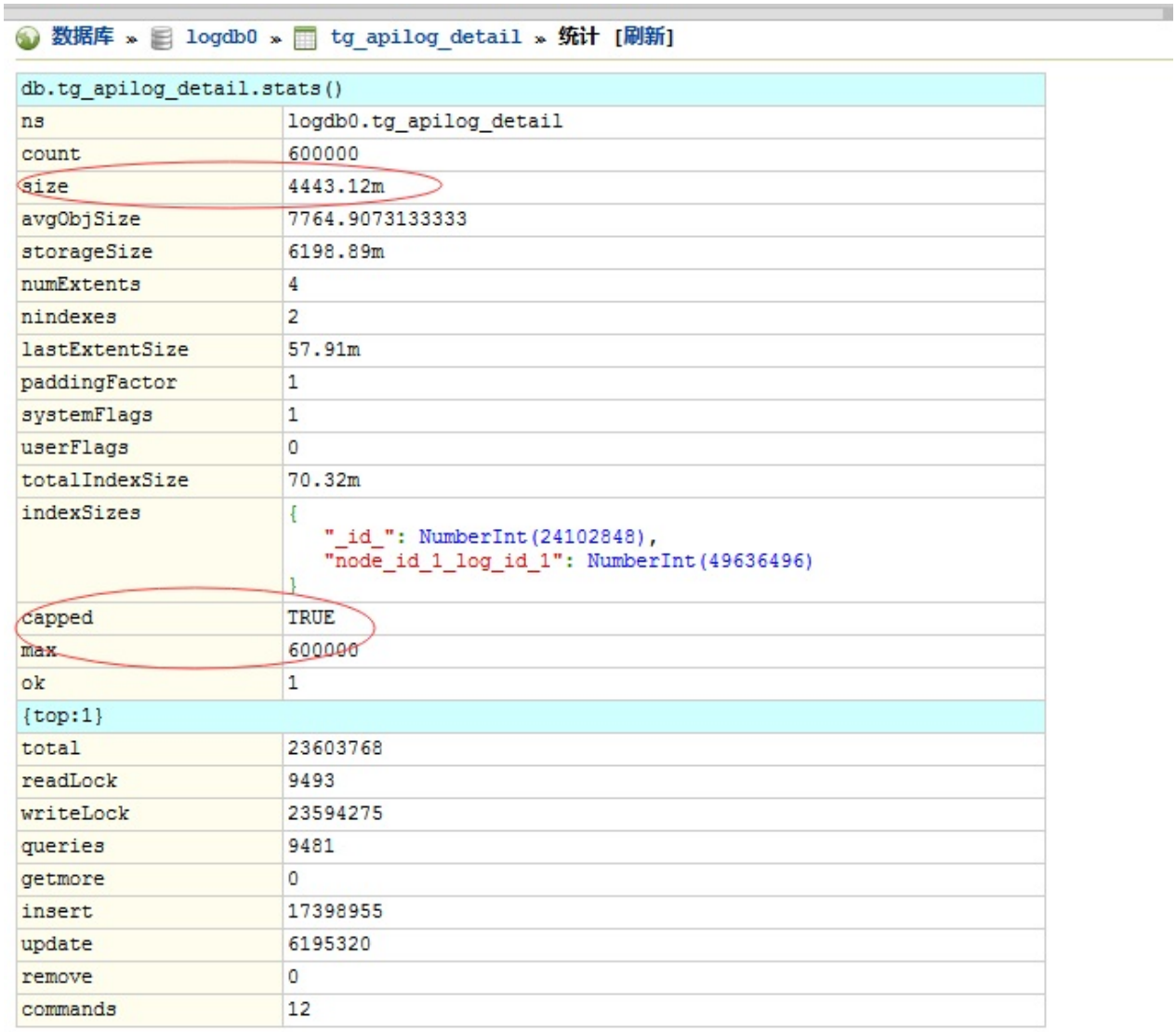
截图一


数据库 » logdb0 » tg_apilog
查询[Array|JSON] | 刷新 | 插入 | 清除 | 新字段 | 统计 | 导出 | 导入 | 更多 »
属性
索引
更名
复制
克隆
校验
删除
DESC
ASC
ASC
ASC
ields(0) ▼ | Hints(0) ▼ | 条数: 0 | Rows:10 | 动作: findAll
查询
分析
清除条件
Cost 0.002782s
1 2 3 4 5 6 7 8 9 10 11 下一页 » (10/600000)
600000 更新 | 删除 | 新字段 | 复制 | 刷新 | 文本 | Expand
{
"_id": ObjectId("55d2f497cacd368410b87a1b"),

截图二


数据库 » logdb0 » tg_apilog » 属性
Name:
tg_apilog
Is Capped:
Size:
341466772
Max:
600000
保存
Notice: To change collection options, we will crea
then drop old one. This will spend a long time to

截图三



Step 7

最后，OMS程序代码config.php加两行代码日志服务器的存储方式和访问域名

(前面的部署步骤都做完，再做这步，尤其第六步修改类型，读写磁盘很厉害，如果OMS日志先链接到mongodb进来再改属性会很卡)

```
define('APILOG_SWITCH','api');
define('APILOG_URL','http://api.log.taoex.com/index.php');
```

api.log.taoex.com可以定义随意url地址，内网绑host即可，需保证内网访问通畅。

最后上面设置完，接着就可以到OMS里查看同步日志里是否空了，只出现了新的日志记录内容，如果是这样就设置ok了。

任务管理服务

版本修订

作者	时间	内容
kami	long long ago	初版
kami	2016-04-28	任务管理优化、接管原来的后台伪队列、定时任务。如果要扩展定时单次或者循环任务可以在timer任务类型里增加
kami	2016-06-30	针对使用redis作为队列服务部署在某些服务器上负载飙升的问题解决,在配置项中增加wait_time参数，识别队列为空时短暂休眠

基于**3.5.0+**版本新增的任务管理服务**(TASKMGR APP)**

任务管理的设计和研发主要是为了解决OMS中一些定时任务或者标化的业务处理任务的执行效率、性能以及并发处理问题。

我们大致将**OMS**中的任务分别五类：

为了方便管理，我们对软件的安装做了一些简单的约定

1.定时任务**timer**

原先OMS后台的一些队列任务，按分钟、小时、天、月触发的任务以及报表等

2.一般标化处理任务**task**

管理员操作触发的任务，批量发货、校验

3.导出任务**export**

数据导出后台任务

4.初始化定时任务**init**

将定时任务加入队列，定时任务依赖这个初始化任务，比如每半小时出发一个任务或者固定时间点14点执行一个任务，就会在这里初始化丢进队列

5.请求响应及回调队列任务**rpc**

实现将OMS相关的请求回调及响应内容走队列，并做流速可控

注：由于定时任务没有相应的参数而和时间有关，所以任务生成到队列中需要借助taskmgr任务管理中的另外一个init(初始化定时任务)任务去实现。

Taskmgr APP的核心程序文件及路径**:**

```
Taskmgr/taskDaemon.php 任务启动脚本
Taskmgr/lib/controller 不同类型任务的命名
Taksmgr/lib/task 具体线程任务，按任务类型区分文件夹task、export、init、rpc、timer
Taskmgr/lib/rpc 任务请求接收入口及签名算法
Taskmgr/lib/connecter 任务的存储介质
Taskmgr/lib/cache 导出分片任务的临时存储介质
Taskmgr/lib/storage 导出任务最终存储介质
Taskmgr/lib/third php socket直连amqp方式，不走amqp php C扩展
Taskmgr/lib/thread 线程操作的lib封装
Taskmgr/logs 任务执行的日志目录，按每天一个文件夹
```

配置：

```
error_reporting(E_ALL & ~E_NOTICE & ~E_DEPRECATED);

//定义数据提供者队列
//define('__CONNECTER_MODE','rabbitmq');

//define('__RABBITMQ_INTERFACE__','pecl');

//mq配置定义
/*
$GLOBALS['__RABBITMQ_CONFIG'] = array(

    'HOST'      => '127.0.0.1',
    'PORT'      => '5672',
    'USER'      => 'task',
    'PASSWD'    => 'task123',
    'VHOST'     => 'erp_task',
    'ROUTER'    => 'erp.task.%s.*',
    'QUEUE_PREFIX' => 'ERP',
);
*/

define('__CONNECTER_MODE','redis');

$GLOBALS['__REDIS_CONFIG'] = array(
    'HOST'      => '127.0.0.1',
    'PORT'      => '6379',
    //'PASSWD'    => 'erp3310',
    'QUEUE_PREFIX' => 'ERP',
    'WAIT_TIME' => 1,
);

//缓存存储介质提供者
//define('__CACHE_MODE','memcache');

//define('__MEMCACHE_CONFIG','127.0.0.1:11211,127.0.0.1:11212');

define('__CACHE_MODE','filesystem');

//文件存储介质提供者
//define('__STORAGE_MODE','ftp');

/*
$GLOBALS['__STORAGE_CONFIG'] = array(
    'HOST'    => '127.0.0.1',
    'PORT'    => '21',
    'USER'    => 'test',
    'PASSWD'  => 'test',
    'TIMEOUT' => '30',
    'PASV'    => false,
);
*/

define('__STORAGE_MODE','local');

//设置为真实的域名
define('DOMAIN', '127.0.0.1');

//定义内部任务请求的token
define('REQ_TOKEN', 'SzjJ7VF9w2Dy3yt4K9vcV3LEwFc5pX3cXHu');
```

核心配置的选项

1. 队列任务存储的介质redis/rabbitmq
2. 临时文件的暂存介质memcache/filesystem
3. 最终文件生成方式local/ftp
4. 任务请求的域名真实地址(DOMAIN)以及请求时候的验签名token(REQ_TOKEN)

PS：

针对队列的配置除了账号基本信息外，我们还可以自定义队列名的前缀

2和3选项主要是针对导出任务的信息存储

任务的启动与暂停：

默认情况下，我们只要在crontab里部署check.sh守护进行即可启动所有类型的任务,命令行如下：

```
* * * * * /bin/bash /data/httpd/56/app/taskmgr/check.sh
```

当然我们也可以针对单个任务类型进行启用与暂停，这里需要注意当我们修改了程序代码后，需要重启任务不然是无法直接看到效果的。

```
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php start timer
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php start task
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php start export
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php start init
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php start rpc

/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php stop timer
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php stop task
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php stop export
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php stop init
/usr/local/php-pthreads/bin/php /data/taskmgr/taskDaemon.php stop rpc
```

任务日志的清理：

也是通过部署crontab任务，具体触发时间自己定义凌晨跑即可，清理两周前的日志

```
0 1 * * * /bin/bash /data/httpd/56/app/taskmgr/cleanlogs.sh
```

如何在一个类型任务中添加一种处理任务，以下我们用批量校验为例：

Step 1 增加任务白名单

找到taskmgr/lib/whitelist.php , 在对应的任务列表里添加相应的代码：

```php
<?php

class taskmgr_whitelist{

    //进队列业务逻辑处理任务
    static public function task_list(){
        return $_tasks = array(
            'autochk'      => array('method'=>'wms_autotask_task_check', 'threadNum'=>5),
            'autodly'      => array('method'=>'wms_autotask_task_consign', 'threadNum'=>5),
            'autorder'     => array('method'=>'ome_autotask_task_combine', 'threadNum'=>5),#自动审单 ExBOY
            'autoretryapi' => array('method'=>'erpapi_autotask_task_retryapi', 'threadNum'=>5),
        );
    }

    //定时任务，线程数不允许修改
    static public function timer_list(){
        return $_tasks = array(
            //'bgqueue' => array('method'=>'ome_autotask_timer_bgqueue', 'threadNum'=>1),
            //'misctask' => array('method'=>'ome_autotask_timer_misctask', 'threadNum'=>1),
            'inventorydepth' => array('method'=>'ome_autotask_timer_inventorydepth', 'threadNum'=>1),
            'financecronjob' => array('method'=>'ome_autotask_timer_financecronjob', 'threadNum'=>1),
            'batchfill' => array('method'=>'ome_autotask_timer_batchfill', 'threadNum'=>1),
            'cleandata' => array('method'=>'ome_autotask_timer_cleandata', 'threadNum'=>1),
            'hour' => array('method'=>'ome_autotask_timer_hour', 'threadNum'=>1),
            'cancelorder' => array('method'=>'ome_autotask_timer_cancelorder', 'threadNum'=>1),
            'ordersprice' => array('method'=>'ome_autotask_timer_ordersprice', 'threadNum'=>1),
            'orderstime' => array('method'=>'ome_autotask_timer_orderstime', 'threadNum'=>1),
            'rmatype' => array('method'=>'ome_autotask_timer_rmatype', 'threadNum'=>1),
            'sale' => array('method'=>'ome_autotask_timer_sale', 'threadNum'=>1),
            //'catsalestatis' => array('method'=>'ome_autotask_timer_catsalestatis', 'threadNum'=>1),//两个统计报表已废弃
            //'productsalerank' => array('method'=>'ome_autotask_timer_productsalerank', 'threadNum'=>1),
            'storestatus' => array('method'=>'ome_autotask_timer_storestatus', 'threadNum'=>1),
            'stockcost' => array('method'=>'ome_autotask_timer_stockcost', 'threadNum'=>1),
            'logistestimate' => array('method'=>'ome_autotask_timer_logistestimate', 'threadNum'=>1),
            'queue' => array('method'=>'ome_autotask_timer_queue', 'threadNum'=>1),
```

1. 数组中的Key(比如:autochk)是具体任务的标识
2. method是具体任务标识对应的处理方式(比如autochk对应的是wms/lib/autotask/task下的check.php)
3. threadNum是具体当前队列任务处理的最大启用线程数，一般就按默认的设置为准(高能预警，非战斗人员请勿随意修改该参数)

Step 2 添加任务的线程处理程序

Taskmgr/lib/task/task/autochk.php 常规的业务处理继承abstract.php 大致程序处理逻辑为：

在线程中连接mq队列，堵塞模式（类似死循环不停的取要处理的任务）取数据，打http请求。

当返回成功的时候，告诉mq队列 ack任务已处理（失败的话里面加了三次重试，但有的任务可能不需要重试，这里要注意）。

当你在白名单中添加了一个具体任务类型的任务后，重启这个任务。该任务就变成运行状态。

PS:后续重试机制会改成按单个任务可配置，未完待续。。。。

Step 3 添加具体的处理任务处理Lib

批量校验的任务处理是wms/lib/autotask/task/check.php,这个就是具体任务具体怎么处理了。

方法名固定为process，params为传入参数,error_msg为处理时遇到的错误可以返回，处理方法最终执行成功的话，需要返回true,失败则为false

Step 4 如何添加一个任务到队列

Taskmgr提供了一个添加任务的lib方法，在taskmgr/lib/interface/connecter.php

```php
//校验任务加队列
$push_params = array(
    'data' => array(
        'log_text' => $bldata['log_text'],
        'log_id' => $bldata['log_id'],
        'task_type' => 'autochk'
    ),
    'url' => kernel::openapi_url('openapi.autotask','service')
);
kernel::single('taskmgr_interface_connecter')->push($push_params);
```

url和task_type是必要参数,url是目标客户OMS的自动任务接收的openapi域名地址,task_type就是任务的标示，也就是上面白名单中的数组key，其他参数就是具体任务处理需要用到的应用级参数。

注：由于taskmgr新版本已经接管了原来的ecos框架misctask任务，所以建议二次开发的时候，不要再把定时任务放在misctask下的分钟、小时、天、周、月的函数里

框架机制

APP目录结构及说明

APP名	说明
apibusiness	与前端销售平台，订单、发货、退款、售后相关业务的核心通讯接口系统
archive	历史数据归档系统，主要是归档订单、发货单数据
base	框架基础app,基本的MVC框架。带有队列，key-value存储接口。类smarty的模版系统
channel	系统集成渠道管理框架
console	控制台，主要包含采购、调拨、其他出入库等业务申请管理
crm	联通CRM系统，实现订单促销，主要针对订单送赠品业务
customs	实现宁波跨境电商在线申报系统在线对接
dbeav	数据动态扩展，使用Entity-Attribute-Value模型动态扩展数据列，并提供数据回收站机制
desktop	设置管理员账户，提供一个带有权限和工作流的操作环境。
eccommon	电商通用组件
entermembercenter	企业用户中心管理
erpapi	统一仓储WMS所有请求、响应的接口管理
image	图片存储系统
inventorydepth	库存深度管理，主要管理前端销售平台商品的库存回写可自定义规则(目前支持淘宝、拍拍、京东)
invoice	发票管理
iostock	出入库相关事务管理
logistics	物流公司选择规则管理
logisticsaccounts	物流对账管理
logisticsmanager	电子面单、快递模板等信息管理
middleware	外部仓储WMS对接适配器
monitor	
ome	OMS核心应用订单相关全流程业务处理管理
omeanalysts	统一报表管理
omeauto	自动审单配置及业务处理系统
omecsv	导入导出管理
omedlyexport	发货单导出特殊格式管理
omepkg	捆绑商品管理
omevirtualwms	模拟仓储物流接口回打系统
openapi	开放数据接口系统,提供相关数据给第三方系统调用
pam	用户认证系统
purchase	采购业务相关处理管理
rpc	老第三方仓储通信控件，入口已被erpapi接管
sales	销售单相关处理业务管理
setup	系统安装核心管理

siso	出入库流水控件
taoexlib	基础库，主要涉及数据导出、短信业务的处理
taoguanallocate	调拨单相关处理业务管理
taoguaninventory	盘点单相关处理业务管理
taoguaniostockorder	出入库单相关处理业务管理
taskmgr	任务管理器，主要包含导出任务，后台队列任务，定时任务，自动审单、批量发货、校验等任务调度和触发
tgkpi	拣货绩效管理
tgstockcost	库存成本计算管理
wms	自建仓储全流程处理管理
wmsmgr	仓储添加、绑定管理
wmsvirtual	仓储物流结果回传被动获取管理,比如京东发货出库，入库等操作需被动获取，没有主动推送。

缓存机制

缓存机制

区别与其他系统，OMS都是一些流程化的订单发货单处理，不涉及过多的页面展示和呈现，所以缓存一般使用在一些不怎么经常发生变化的数据上，或一些允许延迟呈现的数据。

目录结构说明

路径：

app\base\lib\static

文件：

cachecore.php 缓存封装的Lib类(直接引用系统框架使用的缓存存储引擎，具体支持的缓存模式见app\base\lib\cache文件夹下，一般使用memcache)

调用用例

```
读取缓存数据
cachecore::fetch($_inner_key);

保存缓存数据
cachecore::store($_inner_key, $this->user_data, 60*15);
```

$_inner_key为具体保存的缓存键注意唯一性，store方法第二个参数是具体缓存的数据，store方法第三个参数是具体缓存的时效时间

二开使用的场景

针对一些开发场景下，有的数据变动可能性比较小，但是被调用使用的几率比较大，这种时候我们可以使用缓存存储

可能有人会说kv呢，原先系统也有kv方式的持久化存储方式，该存储引擎一般建议只是存储一些简单的、数据量小的配置设置信息

导出机制

数据导出机制

数据导出一直是企业比较关心的问题，原先OMS支持两种数据导出方式，一种是数据量少于500条的时候直接渲染数据输出csv文件，一种是通过系统脚本上部署crontab脚本循环执行一个个导出任务。

原先的方式最大的问题就是导出数据处理的效率，以及导出数据数据量的瓶颈。

新版导出(基于3.5.0新增的任务管理服务)，更大程序上的优化了这块。

新老版本导出比较

老版导出：

预分配5个进程，循环所有用户列表，取用户的一条任务进行执行，进程设定超时时间为30分钟，有些任务数据10w多就可能无法完成中断，或者导出任务多、或者大数据的时候导致任务处理比较慢，而且进程开的多对脚本机的负载大，有的时候进程会僵死。

新版导出：

大体上采用任务队列+多线程的技术架构。导出任务支持指定主内容导出的字段自定义以及先后顺序，并且可以选择是否需要导出明细内容，处理效率提升，处理导出任务数增加，处理的数据量增加，更加稳定高效。

目录结构说明

app\taoexlib\lib\ietask.php 增加系统识别的走后台导出的任务定义

app\ome\lib\export\whitelist.php 使用新版导出的相关数据定义

路径： app/ome/lib/autotask

exportsplit.php 任务切片

dataquerybysheet.php 常规分片任务查询

dataquerybyquciksheet.php 快速分片任务查询

dataquerybywhole.php 无法切片的全局查询

createfile.php 导出数据组装合并生成文件

逻辑梳理说明

1. 将数据量大于一定数量满足条件的，支持后台导出的数据内容导出任务塞进队列
2. 读取导出任务进行拆分，可拆分的根据查询条件找到记录可以唯一定位的主键数组，或者没有主键定位的用导出的总数量，根据上述两种数据进行任务的切分，一般200个切为1个小任务比较合理高效，然后根据预先判断的大于1000的进常规分片查询任务队列，小于1000的进快速分片查询任务队列，而有些不能切分的任务复杂的查询导出，走直接全局查询任务队列
3. 读取分片任务(快/慢)以及全局查询任务队列的任务，进行结果内容的查询，并将任务保存在缓存中，当所有分片任务或整个全局查询任务结束时判断，是否都已经完成，完成的添加创建导出文件的队列任务
4. 读取创建导出文件的组装任务，根据分片数(全局任务结果查询后也会分片保存在缓存中)组装数据在远程ftp里生成csv导出文件

二开及代码解读

增加一个导出数据模型支持后台导出

找到app\taoexlib\lib\ietask.php 文件，在$support_model变量中进行追加

```
var $support_model = array('omeanalysts_mdl_ome_shop','omeanalysts_mdl_ome_sales','ome_mdl_orders','omeanalysts_mdl_ome_delivery','sales_mdl_sales','wms_mdl_inventory','inventorydepth_mdl_shop_frame','omeanalysts_mdl_ome_products','omeanalysts_mdl_ome_goodsale','omeanalysts_mdl_ome_storeStatus','omeanalysts_mdl_ome_goodsrank','omeanalysts_ome_shop','omeanalysts_mdl_ome_branchdelivery','omeanalysts_mdl_ome_aftersale','ome_mdl_goods','finance_mdl_bill_order','finance_mdl_ar_statistics','omedlyexport_mdl_ome_delivery','iostock_mdl_iostocksearch','omeanalysts_mdl_ome_income','drm_mdl_distributor_product_sku','finance_mdl_analysis_bills','finance_mdl_analysis_book_bills','ome_mdl_branch_product','tgstockcost_mdl_costselect','tgstockcost_mdl_branch_product','console_mdl_branch_product','wms_mdl_branch_product','ome_mdl_reship','omeanalysts_mdl_ome_cod','invoice_mdl_order','wms_mdl_delivery','taoguaniostockorder_mdl_iso','wms_mdl_delivery_outerlogi','ome_mdl_reship_refuse','console_mdl_inventory_apply');
```

定义导出任务的处理方式和配置参数

找到app\ome\lib\export\whitelist.php文件

```
<?PHP
/**
 * 导出白名单
 *
 * @author kamisama.xia@gmail.com
 * @version 0.1
 */

class ome_export_whitelist
{

    static public function allowed_lists($source=''){
        $data_source = array(
            'ome_mdl_orders' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'order_id', 'structure'=>'multi'),
            'sales_mdl_sales' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'sale_id', 'structure'=>'multi'),
            'ome_mdl_goods' => array('cansplit'=>0, 'splitnums'=>200),
            'iostock_mdl_iostocksearch' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'iostock_id', 'structure'=>
'single'),
            'omedlyexport_mdl_ome_delivery' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'delivery_id', 'structu
re'=>'spec'),
            'wms_mdl_inventory' => array('cansplit'=>0, 'splitnums'=>200),
            'omeanalysts_mdl_ome_goodsale' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'item_id', 'structure'=>
'single'),
            'omeanalysts_mdl_ome_products' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_sales' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_aftersale' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'item_id', 'structure'=
>'single'),
            'omeanalysts_mdl_ome_shop' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_income' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_cod' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'delivery_id', 'structure'=>'
single'),
            'omeanalysts_mdl_ome_branchdelivery' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'tgstockcost_mdl_costselect' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'tgstockcost_mdl_branch_product' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_goodsrank' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_storeStatus' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'omeanalysts_mdl_ome_delivery' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'finance_mdl_bill_order' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'finance_mdl_ar_statistics' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'finance_mdl_analysis_bills' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'finance_mdl_analysis_book_bills' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'console_mdl_branch_product' => array('cansplit'=>0, 'splitnums'=>200),
            'wms_mdl_branch_product' => array('cansplit'=>0, 'splitnums'=>200),
            'drm_mdl_distributor_product_sku' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'inventorydepth_mdl_shop_frame' => array('cansplit'=>1, 'splitnums'=>200, 'structure'=>'single'),
            'ome_mdl_reship' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'reship_id', 'structure'=>'multi'),
            'invoice_mdl_order' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'id', 'structure'=>'single'),
            'wms_mdl_delivery'=>array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'delivery_id', 'structure'=>'spec'),
            'wms_mdl_delivery_outerlogi'=>array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'delivery_id', 'structure'=>
'spec'),
            'taoguaniostockorder_mdl_iso'=>array('cansplit'=>1, 'splitnums'=>200, 'primary_key'=>'iso_id', 'structure'=>'multi
'),
            'ome_mdl_reship_refuse' => array('cansplit'=>1, 'splitnums'=>200, 'primary_key' => 'reship_id', 'structure'=>'mult
i'),
        );

        if(!empty($source)){
            return isset($data_source[$source]) ? $data_source[$source] : '';
        }else{
            return $data_source;
        }
    }
}
```

参数说明：

cansplit代表这个导出任务是否可以分片，绝大多数90%的都是可以的

splitnums 分片任务的切分数量，一般为200，合理高效

primary_key 任务分片的主键，有的话就根据主键数量切分，没的话根据count出来的结果数量切分

structure 导出数据的结构：single 单层、multi 双层、spec 特殊(发货单是比较特殊的明细和主内容在一行)

任务是否可拆分分片

具体根据业务数据是否有主键，按主键维度可以分片查询，比如有的数据订单，销售单，每个单据都有唯一的主键，这样的数据就可以定义为'cansplit'=>1。

这里要注意如果分片的主键无法用getList框架方法获取，需要在model层中定义方法getPrimaryIdsByCustom

有些数据因为数据是临时从多张表拼接组成或者带group by出来的，所以无法拆分，所以只能走非拆分流程

可拆分分片任务查询数据

一种标准的finder查询就是和列表页查询出来的内容一模一样，比如标化的订单，销售单等数据

```
$exportLib = kernel::single('desktop_finder_export');
$data = $exportLib->work($date_source,$params);
```

还有种就是报表用的比较多，直接自定义方法，在具体调用的数据model层中定义getExportDataByCustom方法

```
public function getExportDataByCustom($fields, $filter, $has_detail, $curr_sheet, $start, $end){
    /*具体获取分片数据的处理代码*/
}
```

扩展导出内容

如果要扩展一个标准finder查询的主内容，可以查看订单导出中的，优惠方案的导出扩展方式，查找关键字export_extra

```
    /**
     * 订单导出列表扩展字段
     */
    function export_extra_cols(){
        return array(
            'column_discount_plan' => array('label'=>'优惠方案','width'=>'100','func_suffix'=>'discount_plan'),
            'column_mark_type_colour' => array('label'=>'订单备注图标颜色','width'=>'100','func_suffix'=>'mark_type_colour'),
        );
    }

    /**
     * 买家备注扩展字段格式化
     */
    function export_extra_discount_plan($rows){
        return kernel::single('ome_exportextracolumn_order_discountplan')->process($rows);
    }
    /**
     * 订单备注图标颜色扩展字段格式化
     */
    function export_extra_mark_type_colour($rows){
        return kernel::single('ome_exportextracolumn_order_marktypecolour')->process($rows);
    }
```

1). export_extra_cols 方法定义要扩展的字段

2). export_extra加func_suffix后缀方法，定义根据结果内容追加扩展的结果内容到结果数组中

完成以上两步即可

自定义的方法要扩展一般本身列表项就是通过get_schema自定义的，所以只要在该方面里增加字段，并且在上面的getexportdatabycustom方法里增加处理即可。

双层结构的涵明细的怎么扩展，基本就是找到查询方法自定义，finder类的标准明细内容查询是调用getexportdetail方法

核心模块设计与说明

订单预处理机制

订单预处理扩展机制

针对前端获取下来的订单，在从暂存区获取出来的时候，做一些额外附加的数据处理，比如：获取发票信息，追加赠品等等。

目录结构说明

路径：

app\ome\lib\preprocess

文件：

1). const.php 预处理需要打标的状态码定义

2). crm.php Crm赠品处理插件

3). entrance.php 预处理入口Lib类

4). invoice.php 一号店发票补全处理插件

5). outstorage.php Vjia出库失败订单处理插件

6). tbgift.php 淘宝赠品处理插件

逻辑梳理说明

引用预处理的节点

埋点具体在订单进入OMS系统后，可处理正常订单会在订单暂存区，在人工分派或自动审单分派订单的时候出发，具体可查看ome_preprocess_entrance的调用点

ome_preprocess_entrance入口Lib

调用该处理方式的时候入参是具体的订单号，然后根据白名单定义的预处理内容，依次进行处理。

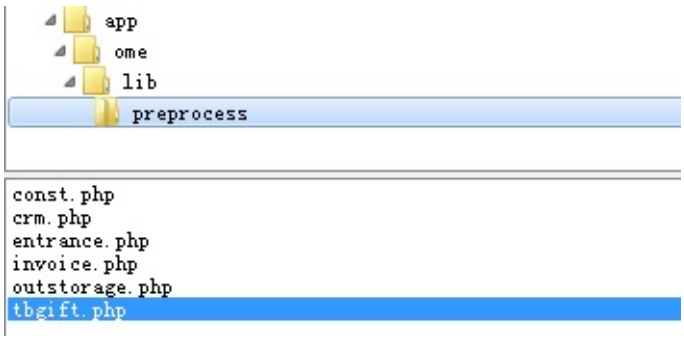
二次开发扩展

Step 1 扩展预处理方法

找到ome_preprocess_entrance Lib文件，在$__methods_list变量中扩展处理插件的名称，代码如下：

```
static private $__methods_list = array('tbgift','invoice','2'=>'crm','outstorage','定义新的插件名');
```

Step 2 在ome_preprocess_entrance Lib文件同级路径增加对应插件名称的Lib 文件



Step 3 新插件文件的逻辑编写，定义必要处理方法**process**

```
public function process($order_id,&$msg){
    /*具体逻辑处理代码*/
}
```

规则筛选器

规则筛选器

为了更加灵活扩展的设置一些规则定义，以及方便管理，主要体现在订单分组、分仓的规则设置中。

目录结构说明

路径：

app\omeauto\lib\auto\type 规则插件目录

文件：

abstract.php 插件抽象类，归整插件的展示渲染

interface.php 插件接口类，定义插件必须定义的方法

address.php 省级地区检查订单是否匹配插件

cod.php 识别是款到发货还是货到付款的订单插件

itemnum.php 识别订单内容中的商品总数量插件

money.php 识别订单总金额插件

platform.php 识别是否是来源平台的订单插件，比如：淘宝，京东，当当等等

shop.php 识别是否是具体店铺的订单插件

sku.php 识别订单是否是在指定时间范围内包含指定商品插件

skunum.php 识别订单内容中的商品种类数量插件

二次开发扩展

规则筛选器主要运用在订单分组，仓库分组的规则设置中

具体界面操作调用控制器app\omeauto\controller\order\type.php中的addrole方法

Step 1

扩展的时候，我们需要在app\omeauto\view\order\type\addrole.html页面中增加一个插件的定义

```
<tr>
    <th width="60"><label for="dom_el_bc">筛选条件：</label> </th>
    <td >
        <select name="type_id" id="type_id">
            <option value="address" <{if $init.role=='address'}>selected<{/if}>>收货地址</option>
            <option value="platform" <{if $init.role=='platform'}>selected<{/if}>>来源平台</option>
            <option value="shop" <{if $init.role=='shop'}>selected<{/if}>>前端店铺</option>
            <option value="cod" <{if $init.role=='cod'}>selected<{/if}>>付款方式</option>
            <option value="money" <{if $init.role=='money'}>selected<{/if}>>订单金额</option>
            <option value="sku" <{if $init.role=='sku'}>selected<{/if}>>活动订单</option>
            <option value="skunum" <{if $init.role=='skunum'}>selected<{/if}>>商品种类数</option>
            <option value="itemnum" <{if $init.role=='itemnum'}>selected<{/if}>>商品总数量</option>
            <option value="weight" <{if $init.role=='weight'}>selected<{/if}>>商品总重量</option>
        </select>
    </td>
</tr>
```

Step 2

在app\omeauto\lib\auto\type目录下，添加具体的插件文件，定义如下方法：

checkParams 检查插件的配置参数是否合法

roleToString 将配置参数转换成UI可展示的文本形式展示内容

setRole 设置已经创建好的配置内容

vaild 具体插件检查数据的调用方法

自动审单插件

自动审单插件

自动审单过程中，OMS会调用系统定义的一系列插件进行处理，判定订单是否可以自动审下去，并给订单打上状态标记。

目录结构说明

路径：

app\omeauto\lib\auto\plugin 审单插件目录

文件：

abstract.php 插件抽象类，定义了一些插件的获取名称、状态码的通用方法

interface.php 插件接口类，定义插件必须定义的方法

abnormal.php 检查订单是否暂停、异常插件

arrived.php 检查运费配送能不能到达插件

branch.php 检查自动确认仓储插件

crm.php 检查淘宝订单是否有优惠赠品插件

flag.php 检查备注和旗标插件

logi.php 检查自动分配的物流公司(物流优选)插件

member.php 检查是否同一个用户有多订单插件

ordermulti.php 检查是否有其他订单可以合并插件

oversold.php 检查订单是否是超卖插件(目前淘宝订单有这个逻辑标识)

pay.php 检查合并订单是否有未支付插件

shopcombine.php 检查是否有疑似合并的订单插件

store.php 检查订单货品仓库库存是否足够插件

tax.php 检查是否开发票插件

tbgift.php 检查淘宝订单是否有优惠赠品插件

product.php 检查订单明细中货号信息是否完整存在插件

二次开发扩展

Step 1

优先在路径app\omeauto\lib\auto\plugin下面定义新的插件处理文件

一般需要定义以下方法及参数：

```
protected $__STATE_CODE = omeauto_auto_const::__STORE_CODE; 定义是否要打标，打的状态码

//检查处理方法
public function process(& $group, &$confirmRoles) {
}

//插件标题定义方法
public function getTitle() {
}

//获取提示信息方法
public function getAlertMsg(& $order) {
}
```

状态码定义文件app\omeauto\lib\const.php 16进制

```
<?php

class omeauto_auto_const {

    //有未付订单
    const __PAY_CODE        = 0x00000001;
    //备注和留言
    const __FLAG_CODE       = 0x00000002;
    //物流公司标记
    const __LOGI_CODE       = 0x00000004;
    //产品不匹配
    const __PRODUCT_CODE    = 0x00000008;
    //用户多订单
    const __MEMBER_CODE     = 0x00000010;
    //乡村物流标记
    const __LOGI_LITE_CODE  = 0x00000020;
    //单订单
    const __SINGLE_CODE     = 0x00000040;
    //多订单
    const __MUTI_CODE       = 0x00000080;
    //仓库
    const __BRANCH_CODE     = 0x00000100;
    //库存
    const __STORE_CODE      = 0x00000200;
    //异常
    const __ABNORMAL_CODE   = 0x00000400;
    //单订单且有备注
    const __EXAMINE_CODE    = 0x00000800;
    //超卖订单
    const __OVERSOLD_CODE   = 0x00001000;
    //淘宝订单优惠中有赠品信息
    const __PMTGIFT_CODE    = 0x00002000;
    const __COMBINE_CODE    = 0x00004000;
    //CRM赠品信息
    const __CRMGIFT_CODE    = 0x00008000;
    //检测订单是否开发票
    const __TAX_CODE        = 0x00010000;
    //检查物流到不到
    const _LOGIST_ARRIVED = 0x00040000;
}
```

Step 2

找到自动审单插件调用的程序代码app\omeauto\lib\auto\group.php

方法 getPluginNames 定义审单调用的插件

```
private function getPluginNames($cfg) {
    $combine_select = app::get('ome')->getConf('ome.combine.select');
    if ($combine_select=='1') {
             $plugins = array(
            'branch', //先选择仓库
            'store', //在判断库存
            'flag', //备注和留言
            'pay', //有未付订单
            'logi', //判定物流
            'abnormal', //数据字段异常订单
            'oversold',//超卖订单
            'tbgift',//淘宝订单有赠品
             'crm',//crm赠品
             'tax',//开发票
             'arrived',//物流到不到
        );
    } else{
        $plugins = array(
            'branch', //先选择仓库
            'store', //在判断库存
            'flag', //备注和留言
            'pay', //有未付订单
            'logi', //判定物流
            'member', //用户多地址
            'ordermulti', // 是否多单合
            'abnormal', //数据字段异常订单
            'oversold',//超卖订单
            'tbgift',//淘宝订单有赠品
            'shopcombine',
             'crm',//crm赠品
             'tax',//开发票
             'arrived',//物流到不到
        );
    }

    return $plugins;
}
```

找到自动审单插件调用的程序代码app\omeauto\lib\auto\combine.php

构造 __construct 定义获取提示信息的插件清单

```
public function __construct() {
    $combine_select = app::get('ome')->getConf('ome.combine.select');
    if ($combine_select == '1') {
        $this->_plugins = array('pay', 'flag', 'logi',  'branch', 'store', 'abnormal','oversold','tbgift','crm','tax','arrived
');
    }else{
        $this->_plugins = array('pay', 'flag', 'logi', 'member', 'ordermulti', 'ordersingle', 'branch', 'store', 'abnormal','o
versold','tbgift','shopcombine','crm','tax','arrived');
    }

    if (self::getCnf('chkProduct') == 'Y') {
        $this->_plugins[] = 'product';
    }
}
```

系统对接

前端销售平台

ECStore对接

OMS请求**ECStore**相关业务接口

1. store.trade.status.update 订单状态更新
2. store.trade.memo.update 更新订单备注
3. store.trade.payment.add 添加支付单
4. store.trade.refund.add 添加退款单
5. store.trade.reship.add 添加退货单
6. store.trade.shipping.add 添加发货单
7. store.trade.shipping.update 更新发货单物流信息
8. store.trade.shipping.status.update 更新发货单状态
9. store.items.quantity.list.update 批量库存回写
10. store.trade.aftersale.status.update 更新售后申请的状态
11. store.trade.buyer_message.add 添加买家留言
12. store.shop.payment_type.list.get 获取支付方式
13. iframe.tradeEdit 订单iframe编辑
14. store.trade.fullinfo.get 单拉订单详情
15. store.trades.sold.get 获取时间段的订单列表
16. store.trade.update 订单更新
17. store.trade.ship_status.update 更新顶大发货状态
18. store.trade.pay_status.update 更新订单支付状态
19. store.trade.memo.add 添加订单备注
20. store.trade.shippingaddress.update 更新交易收货人信息
21. store.trade.payment.status.update 更改支付状态
22. store.trade.refund.status.update 更改退货单状态
23. store.trade.reship.status.update 更改退货单状态
24. store.trade.item.freezstore.update 更改预占
25. store.trade.aftersale.add 添加售后申请单

OMS响应**ECStore**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.payment.add 付款单创建
3. ome.payment.status_update 付款单状态更新
4. ome.refund.add 退款申请单创建

BBC对接

OMS请求**BBC**相关业务接口

1. store.trade.refund.add 添加退款单
2. store.items.quantity.list.update 更新库存
3. store.item.get 获取单个商品
4. store.items.all.get 获取前端商品
5. store.item.sku.get 获取前端SKU
6. store.items.list.get 通过ID获取前端商品
7. store.trade.refund.add 添加退款单
8. store.trade.status.update 订单状态更新
9. store.refund.refuse 拒绝退款
10. store.trade.aftersale.status.update 更新售后申请状态
11. store.logistics.offline.send 添加发货单

BBC请求**OMS**相关业务接口

1. ome.order.add 订单创建
2. ome.refund.add 退款申请单
3. ome.aftersale.add 售后申请单
4. ome.aftersale.logistics_update 售后物流信息更新

唯品会（**vop**）对接

OMS请求唯品会（**vop**）相关业务接口

1. store.logistics.offline.send 发货

OMS响应**vop**相关业务接口

1. ome.order.add 订单创建 和更新

SHOPEX B2B 对接

OMS请求**B2B**分销王相关业务接口

1. store.trade.update 订单更新
2. store.trade.shippingaddress.update 更新交易收货人信息
3. store.trade.memo.add 添加订单备注
4. store.trade.memo.update 更新订单备注
5. store.trade.buyer_message.add 添加买家留言
6. store.trade.status.update 订单状态更新
7. store.trade.shipping.add 添加发货单
8. store.trade.reship.add 添加退货单
9. store.trade.reship.status.update 更改退货单状态
10. store.trade.refund.add 添加退款单
11. store.trade.payment.add 添加支付单
12. store.trade.aftersale.add 添加售后申请单

13. store.trade.aftersale.status.update 更新售后申请状态
14. store.items.quantity.list.update 更新库存
15. store.shop.payment_type.list.get 获取支付方式

OMS响应**B2B**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.payment.add 付款单创建
3. ome.payment.status_update 付款单状态更新
4. ome.refund.add 退款申请单创建
5. ome.aftersale.add 售后申请

SHOPEX485 对接

OMS请求**485**相关业务接口

1. store.trade.update 订单更新
2. store.trade.shippingaddress.update 更新交易收货人信息
3. store.trade.memo.add 添加订单备注
4. store.trade.memo.update 更新订单备注
5. store.trade.buyer_message.add 添加买家留言
6. store.trade.status.update 订单状态更新
7. store.trade.ship_status.update 更新订单发货状态
8. store.trade.shipping.add 添加发货单
9. store.trade.shipping.update 更新物流信息
10. store.trade.shipping.status.update 更新发货单状态
11. store.trade.reship.add 添加退货单
12. store.trade.refund.add 添加退款单
13. store.trade.payment.add 添加支付单
14. store.items.quantity.list.update 更新库存
15. store.shop.payment_type.list.get 获取支付方式
16. store.trade.aftersale.status.update 更新售后申请的状态
17. store.trade.aftersale.add 添加售后申请单

OMS响应**SHOPEX485**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.payment.add 付款单创建
3. ome.payment.status_update 付款单状态更新
4. ome.refund.add 退款申请单创建
5. ome.aftersale.add 售后退款退货申请

店掌柜 对接

OMS请求店掌柜相关业务接口

1. store.trade.update 订单更新
2. store.trade.status.update 订单状态更新
3. store.trade.ship_status.update 更新订单发货状态
4. store.trade.pay_status.update 更新订单支付状态
5. store.trade.memo.add 添加订单备注
6. store.trade.memo.update 更新订单备注
7. store.trade.shippingaddress.update 更新交易收货人信息
8. store.trade.payment.add 添加支付单
9. store.trade.payment.status.update 更改支付状态
10. store.trade.refund.add 添加退款单
11. store.trade.refund.status.update 更新退款单状态
12. store.trade.reship.add 添加退货单
13. store.trade.reship.status.update 更改退货单状态
14. store.trade.shipping.add 添加发货单
15. store.trade.shipping.update 更新物流信息
16. store.trade.shipping.status.update 更新发货单状态
17. store.items.quantity.list.update 更新库存
18. store.trade.item.freezstore.update 更新预占
19. store.trade.aftersale.status.update 更新售后申请状态
20. store.trade.aftersale.add 添加售后申请单
21. store.trade.buyer_message.add 添加买家留言
22. store.shop.payment_type.list.get 获取支付方式

OMS响应店掌柜相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.payment.add 付款单创建
3. ome.payment.status_update 付款单状态更新
4. ome.refund.add 退款申请单创建
5. ome.aftersale.add 售后退款退货申请

淘宝 对接

OMS请求淘宝相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.fenxiao.items.quantity.list.update 更新库存（分销）
3. store.logistics.offline.send 线下物流
4. store.logistics.online.send 线上物流

5. store.items.all.get 获取前端商品
6. store.items.list.get 通过ID获取前端商品
7. store.item.approve_status.update 单个商品上下架
8. store.item.approve_status_list.update 批量上下架
9. store.item.sku.get 获取前端SKU
10. store.item.get 获取单个商品
11. store.fenxiao.products.get 分销商品
12. store.fenxiao.product.update 更新分销商品
13. store.refund.message.get 退款凭证获取
14. store.refund.refuse 拒绝退款
15. store.refund.message.add 回写留言和凭证
16. store.refund.good.return.agree 同意退货
17. store.tmall.refund.good.return.refuse
18. store.tmall.refund.message.get
19. store.tmall.refund.refuse
20. store.tmall.trade.refund.examine 同意退款
21. store.tmall.refund.good.return.agree 天猫同意退货申请
22. store.logistics.address.search 卖家地址库
23. store.eai.order.refund.good.return.check 回填退货物流信息
24. store.waybillallocation.requestwaybillnum 获取电子面单
25. store.trade.shippingaddress.update 回写淘宝收货地址
26. store.eai.order.refund.get 单拉退款单信息
27. store.tmc.message.produce 淘宝全链路接口

-- 天猫2015年新的售后接口

1. store.eai.order.refund.i.get 单拉退款单信息
2. store.tmall.refund.i.good.return.agree 同意退货
3. store.tmall.refund.i.good.return.refuse 拒绝退货
4. store.tmall.refund.i.message.get 获取凭证
5. store.tmall.refund.i.refuse 获取拒绝退款
6. store.tmall.trade.refund.i.examine 同意退款
7. store.wlb.order.jzpartner.query 家装服务商
8. store.wlb.order.jzwithins.consign 家装发货

OMS响应淘宝相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

拍拍 对接

OMS请求拍拍相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.trade.delivery.send 拍拍发货接口
3. store.logistics.offline.send 线下物流
4. store.items.all.get 获取前端商品
5. store.item.approve_status.update 单个商品上下架
6. store.item.approve_status_list.update 批量商品上下架
7. store.item.sku.get 获取前端SKU
8. store.item.get 获取单个商品

OMS响应拍拍相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.refund.add 退款申请单创建
3. ome.aftersalev2.add 退款申请

京东 对接

OMS请求京东相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流
3. store.items.all.get 获取前端SKU
4. store.items.list.get 通过ID获取前端商品
5. store.item.get 获取单个商品
6. store.item.sku.get 获取前端SKU
7. store.item.approve_status.update 单个商品上下架
8. store.item.approve_status_list.update 批量上下架
9. store.refund.good.return.check 确认退货

OMS响应京东相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

一号店 对接

OMS请求一号店相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流
3. store.trade.invoice.get 获取发票抬头
4. store.refund.good.return.agree 同意退货

5. store.refund.good.return.refuse 拒绝退货
6. store.refund.good.return.check 确认退货

OMS响应一号店相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

QQ网购 对接

OMS请求**QQ**网购相关业务接口

1. store.trade.delivery.send 拍拍发货接口
2. store.logistics.offline.send 线下物流

OMS响应**QQ**网购相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 退款申请

当当对接

OMS请求当当相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流

OMS响应当当相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 退款申请

亚马逊对接

OMS请求亚马逊相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应亚马逊相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 退款申请

V家商城 对接

OMS请求**V**家商城相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流
3. store.trade.invoice.get 获取发票抬头
4. store.trade.outstorage 发货出库
5. store.logistics.resend.confirm 发货出库确认

6. store.logistics.consign.resend 修改配送信息

OMS响应**v**家相关业务接口

1. ome.order.add 订单创建 和更新

阿里巴巴 对接

OMS请求阿里巴巴相关业务接口

1. store.item.get 获取单个商品
2. store.logistics.offline.send 线下物流

OMS响应阿里巴巴相关业务接口

1. ome.order.add 订单创建 和更新

苏宁 对接

OMS请求苏宁相关业务接口

1. store.item.get 获取单个商品
2. store.logistics.offline.send 线下物流
3. store.items.quantity.list.update 更新库存（直销）
4. store.items.custom.get 获取商品信息

OMS响应苏宁相关业务接口

1. ome.order.add 订单创建 和更新

银泰 对接

OMS请求银泰相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.custom.get 获取商品信息
3. store.items.quantity.list.update 更新库存（直销）

OMS响应银泰相关业务接口

1. ome.order.add 订单创建 和更新

工行对接

OMS请求工行相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应工行相关业务接口

1. ome.order.add 订单创建 和更新

蘑菇街对接

OMS请求蘑菇街相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应蘑菇街相关业务接口

1. ome.order.add 订单创建 和更新

国美对接

OMS请求国美相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应国美相关业务接口

1. ome.order.add 订单创建 和更新

微信对接

OMS请求微信相关业务接口

1. store.item.get 获取单个商品
2. store.logistics.offline.send 线下物流
3. store.items.quantity.list.update 更新库存（直销）

OMS响应微信相关业务接口

1. ome.order.add 订单创建 和更新

建设银行对接

OMS请求建设银行相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应建设银行相关业务接口

1. ome.order.add 订单创建 和更新

美丽说 对接

OMS请求美丽说相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）
3. store.refund.good.return.agree 美丽说同意退款、退货（退款退货用的同一个接口）

OMS响应美丽说相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

飞牛 对接

OMS请求飞牛相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应飞牛相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售前退款

有赞 对接

OMS请求有赞相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应有赞相关业务接口

1. ome.order.add 订单创建 和更新

卷皮 对接

OMS请求卷皮相关业务接口

1. store.logistics.offline.send 线下物流

OMS响应卷皮相关业务接口

1. ome.order.add 订单创建 和更新

蜜芽宝贝 对接

OMS请求蜜芽宝贝相关业务接口

1. store.logistics.offline.send 线下物流
2. store.items.quantity.list.update 更新库存（直销）

OMS响应蜜芽宝贝相关业务接口

1. ome.order.add 订单创建 和更新

public b2c 对接

OMS请求**public b2c**相关业务接口

1. store.trade.update 订单更新
2. store.trade.status.update 订单状态更新
3. store.trade.ship_status.update 更新订单发货状态
4. store.trade.pay_status.update 更新订单支付状态
5. store.trade.memo.add 添加订单备注
6. store.trade.memo.update 更新订单备注
7. store.trade.shippingaddress.update 更新交易收货人信息
8. store.trade.payment.add 添加支付单

9. store.trade.payment.status.update 更改支付状态
10. store.trade.refund.add 添加退款单
11. store.trade.refund.status.update 更改退款单状态
12. store.trade.reship.add 添加退货单
13. store.trade.reship.status.update 更改退货单状态
14. store.trade.shipping.add 添加发货单
15. store.trade.shipping.update 更新物流信息
16. store.trade.shipping.status.update 更新发货单状态
17. store.items.quantity.list.update 更新库存（直销）
18. store.trade.item.freezstore.update 更新预占
19. store.trade.aftersale.status.update 更新售后申请的状态
20. store.trade.aftersale.add 添加售后申请单
21. store.trade.buyer_message.add 添加买家留言
22. store.shop.payment_type.list.get 获取支付方式

OMS响应**public b2c**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.payment.add 付款单创建
3. ome.payment.status_update 付款单状态更新
4. ome.refund.add 退款申请单创建
5. ome.aftersale.add 售后申请

beibei 对接

OMS请求**beibei**相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流

OMS响应**beibei**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

有量 对接

OMS请求有量相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流

OMS响应有量相关业务接口

1. ome.order.add 订单创建 和更新

萌店 对接

OMS请求萌店相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流

OSM响应萌店相关业务接口

1. ome.order.add 订单创建 和更新

折**800** 对接

OMS请求折**800**相关业务接口

1. store.items.quantity.list.update 更新库存（直销）
2. store.logistics.offline.send 线下物流

OSM响应折**800**相关业务接口

1. ome.order.add 订单创建 和更新
2. ome.aftersalev2.add 售后申请

后端仓储

顺丰仓储

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知
11. store.vendors.get 向WMS推送创建供应商列表

响应

1. wms.stockout.status_update 出库单状态更新通知(包含销售出库)
2. wms.stockin.status_update 入库单状态更新通知(包含退货入库)
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

京东仓储

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.item.update 向WMS推送更新物料通知
3. store.wms.saleorder.create 向WMS推送创建发货单通知
4. store.wms.saleorder.cancel 向WMS推送取消发货单通知
5. store.wms.saleorder.get 向WMS推送获取发货单通知
6. store.wms.returnorder.create 向WMS推送添加退换单通知
7. store.wms.returnorder.get 向WMS推送获取退换单通知
8. store.wms.transferorder.create 向WMS推送创建转储单通知
9. store.wms.transferorder.cancel 向WMS推送取消转储单通知
10. store.wms.inorder.create 向WMS推送创建入库单通知
11. store.wms.inorder.cancel 向WMS推送取消入库单通知
12. store.wms.inorder.get 向WMS推送获取入库单通知
13. store.wms.outorder.create 向WMS推送创建出库单通知

14. store.wms.outorder.cancel 向WMS推送取消出库单通知
15. store.wms.outorder.get 向WMS推送获取出库单通知
16. store.wms.warehouse.list.get 向WMS推送获取仓库列表
17. store.wms.logistics.companies.get 向WMS推送获取物流公司列表
18. store.vendors.get 向WMS推送创建供应商列表

响应

京东没有主动推响应 需要ERP通过相关.get接口去查询获取结果

富勒**WMS**

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

欧唯特

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知

7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

发网物流

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

科捷

请求

1. store.wms.item.add 向WMS推送添加物料通知

2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

伊藤忠物流

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

11. wms.stockout.status_update 出库单状态更新通知
12. wms.stockin.status_update 入库单状态更新通知
13. wms.inventory.add 盘点申请单
14. wms.stock.status_update 库存对账
15. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

酷武物流

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

百世物流

请求

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.saleorder.cancel 向WMS推送取消发货单通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.transferorder.cancel 向WMS推送取消转储单通知
7. store.wms.inorder.create 向WMS推送创建入库单通知
8. store.wms.inorder.cancel 向WMS推送取消入库单通知
9. store.wms.outorder.create 向WMS推送创建出库单通知
10. store.wms.outorder.cancel 向WMS推送取消出库单通知

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知

6.wms.reship.status_update 退货单状态更新通知

1. wms.delivery.status_update 发货单状态更新通知

奇门仓储

请求

目前商派**ERP**已经可以支持奇门版本为**2.2 ,2.3, 2.4**三个版本

1. store.wms.item.add 向WMS推送添加物料通知
2. store.wms.saleorder.create 向WMS推送创建发货单通知
3. store.wms.item.update 向WMS推送更新物料通知
4. store.wms.returnorder.create 向WMS推送添加退换单通知
5. store.wms.transferorder.create 向WMS推送创建转储单通知
6. store.wms.inorder.create 向WMS推送创建入库单通知
7. store.wms.outorder.create 向WMS推送创建出库单通知

1. store.wms.order.cancel 单据取消

响应

1. wms.stockout.status_update 出库单状态更新通知
2. wms.stockin.status_update 入库单状态更新通知
3. wms.inventory.add 盘点申请单
4. wms.stock.status_update 库存对账
5. wms.stockdump.status_update 转储单状态更新通知
6. wms.reship.status_update 退货单状态更新通知
7. wms.delivery.status_update 发货单状态更新通知

电子发票

1.store.einvoice.createreq 电子发票开票（开蓝和冲红）

请求参数

business_type（默认：0。对于商家对个人开具，为0;对于商家对企业开具，为1;），platform（电商平台代码），tid（电商平台对应的订单号），serial_no(开票流水号)，payee_address（开票方地址），payee_name（开票方名称），payee_operator（开票人），invoice_amount（开票金额），invoice_time（开票日期），invoice_type（发票(开票)类型，蓝票blue,红票red，默认blue），payee_register_no（收款方税务登记证号），payer_name（付款方名称, 对应发票台头），sum_price（合计金额），sum_tax（合计税额），invoice_items（电子发票明细），provider_appkey（开票服务商的APPKEY），proxy_appkey（商家自己申请的放在开票代理客户端的appkey），normal_invoice_code（原发票代码开红票传入），normal_invoice_no（原发票号码开红票传入）erp_tid（erp中唯一单据 可选），payee_bankaccount（开票方银行及帐号 可选），payer_register_no（付款方税务登记证号。对企业开具电子发票时必填），invoice_memo（发票备注 可选），payer_address（消费者地址 可选），payer_bankaccount（付款方开票开户银行及账号 可选），payer_email（消费者电子邮箱 可选），payer_phone（消费者联系电话 可选），payee_checker（复核人 可选），payee_receiver（收款人 可选），payee_phone（收款方电话 可选）。

2.store.einvoice.create.result.get 电子发票开票后获取开票结果

请求参数

platform（电商平台代码），serial_no（开票流水号），tid（电商平台对应的订单号），payee_register_no（收款方税务登记证号）。

3.store.einvoice.url.get 获取电子发票url地址

请求参数

node_id（店铺节点号），platform（平台），invoice_no（发票号码），tid（订单号），expire（地址有效期）。

4.store.einvoice.invoice.get 查询已回传淘宝的电子发票

请求参数

tid（订单号）。

5.store.einvoice.invoice.prepare 电子发票天猫状态更新

请求参数

tid（订单号），invoice_action_type（开票操作类型 1 交易成功 2 退货、退款成功 3 电子换纸质 4 换发票内容），invoice_type（发票类型，1:蓝票，2:红票 ），serial_no(开票流水号)，invoice_title（发票抬头）。

6.store.einvoice.detail.upload 电子发票回流天猫

请求参数

tid（订单号），invoice_code（发票代码），invoice_no（发票号码），invoice_file_data（发票文件内容 目前只支持jpg,png,bmp,pdf格式），invoice_type（发票类型 1 蓝票 2 红票），invoice_items（电子发票明细），electronic_invoice_no（电子发票号），serial_no（开票流水号），payee_register_no（收款方税务登记号），invoice_amount（开票金额），invoice_time（开票日期），payee_name（开票方名称），tid（主订单

id），invoice_title（发票抬头），image_type（图片类型），qr_code（二维码 可选），anti_fake_code（防伪码 可选）。

跨境申报

APP应用目录

app/customs/

OMS请求矩阵相关业务接口

1. store.cnec.jh.order 获取申报单号
2. store.cnec.jh.pay 获取进口支付单
3. store.cnec.jh.lgs 获取进口运单
4. store.cnec.jh.cancel 撤消跨境订单
5. store.cnec.jh.query (按时间)申报状态查询
6. store.cnec.jh.decl.byorder (按订单)申报状态查询

电子面单

APP应用目录

app/logisticsmanager/

OMS请求矩阵相关业务接口

京东电子面单

1. store.etms.waybillcode.get 获取面单号

   直连,即打印时直接返回运单号.

   方法组织调用路径 logisticsmanager/lib/service/360buy.php

   public function get_waybill_number($params) {}

2. store.etms.waybill.send 京东电子面单物流回传

   订单发货回写至前端前调用此接口,此接口回传成功，运单号才可以发货回写京东成功

   方法组织调用路径 logisticsmanager/lib/service/360buy.php

   public function delivery($delivery_id) {}

EMS电子面单

1. store.waybillprintdata.get 获取面单号

   获取面单是批量获取的，一次打印请求获取100,先获取保存在sdb_logisticsmanager_waybill表中,再从表里取未使用单号.

   方法组织调用路径 logisticsmanager/lib/rpc/request/ems.php

   public function get_waybill_number($data) {}

2. store.print.data.create 电子面单物流回传

   订单发货回写至前端前调用此接口

   方法组织调用路径 logisticsmanager/lib/service/ems.php

   public function delivery($delivery_id) {}

SF电子面单

1. store.sf.orderservice 获取面单号

   直连,即打印时直接返回运单号.

   方法组织调用路径 logisticsmanager/lib/service/sf.php

   public function get_waybill_number($params) {}

2. store.sf.ordersearchservice 搜索顺丰面单

   当获取运单号返回失败 res='8016'时,调用此方法，根据对应订单号和发货单号返回，如果此接口返回成功，返回订单号对应的运单号

   方法组织调用路径 logisticsmanager/lib/service/sf.php

   public function search_waybill_number($params) {}

STO电子面单

1. store.waybill.mailno.get 批量获取面单号

申通目前获取面单是批量获取的，一次打印请求,先获取保存在sdb_logisticsmanager_waybill表中,再从表里取未使用单号

方法组织调用路径 logisticsmanager/lib/rpc/request/sto.php

public function get_waybill_number($data) {}

2. store.waybill.data.add 电子面单物流回传

申通通过此接口打印时调用此接口，此接口会返回大头笔

方法组织调用路径 logisticsmanager/lib/service/sto.php

public function delivery($delivery) {}

淘宝云栈电子面单

1. store.wlb.waybill.i.get 获取面单号

直连,即打印时直接返回运单号.

方法组织调用路径 logisticsmanager/lib/service/taobao.php

调用方法:public function get_waybill_number($params) {}

1. store.wlb.waybill.cancel 淘宝云栈电子面单取消

当撤销发货单或打回时会调用此方法 方法组织调用路径 logisticsmanager/lib/rpc/request/taobao.php

调用方法: public function cancel_billno($data){}

1. store.wlb.waybill.print 淘宝云栈官方电子面单打印确认

订单发货回写至前端前调用此接口

方法组织调用路径 logisticsmanager/lib/service/taobao.php

public function delivery($delivery_id){}

韵达电子面单

1. store.yd.orderservice 获取面单号 直连,即打印时直接返回运单号.

方法组织调用路径 logisticsmanager/lib/service/yunda.php

public function get_waybill_number($params) {}

1. store.yd.searchorderservice 搜索韵达面单

当调用获取运单号接口失败，并返回err_msg =‘更新订单请使用更新接口‘ 时调用此方法，通过订单信息，查询订单对应的运单号

方法组织调用路径 logisticsmanager/lib/service/yunda.php

public function search_waybill_number($params) {}

库存管理

库存深度

库存深度

提供OMS货品在往前端销售平台回写库存的时候，需要不同的回写机制或者规则回写库存，并且可以细到管理到每一个sku的库存回写

名词解释

可售库存

可售库存 = 该货品供货的仓库库存store之和 - 全局预占库存冻结 - 仓库预占库存冻结

如果是捆绑货品的可售库存，我们一般是算出捆绑下关联的每个可售库存，然后取最小值进行回写

发布库存

发布库存一般使用在第一次回写库存，或者某一个货品库存想直接更新成一个指定的库存数的时候用到

全局预占

所有涉及的店铺预占冻结之和

店铺预占

具体货品在当前店铺下订单还没有拆分完，冻结还在店铺这层上的没到仓库的，预占冻结数

目录结构说明

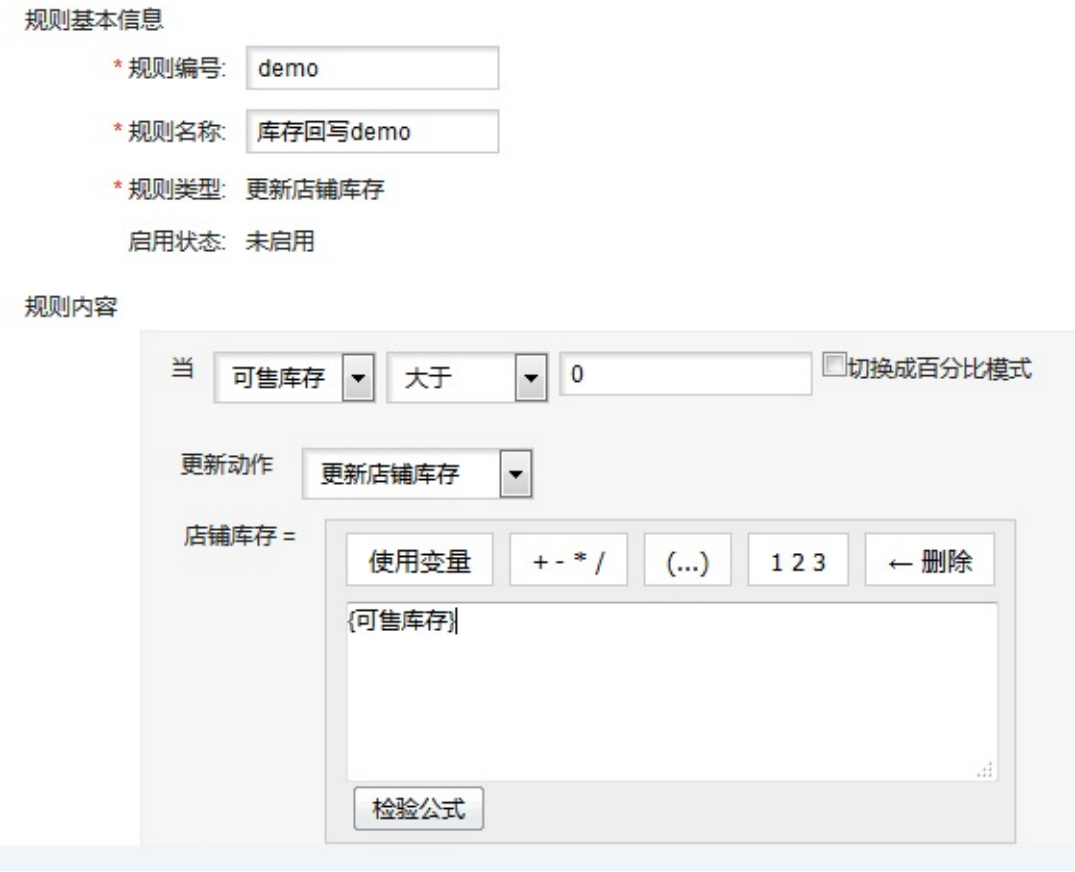
库存深度是一个相对独立的模块，都在app/inventorydepth下

库存回写的触发点是在 app/ome/lib/autotask/inventorydepth.php 调用Lib inventorydepth_logic_stock start方法

回写逻辑说明

1. 获取距离当前时间5分钟内，库存发生变化的并且前端货号与OMS内的货号关联的普通/捆绑货品信息
2. 根据回写货品获取响应的回写规则
3. 根据规则计算货品当前的可售库存，冻结信息
4. 调用库存回写接口回写库存信息

规则应该如何设置



一般情况下，一个店铺里的货品，如果只在一个仓库有，而且这个仓库只供应这个店铺（比如：淘宝），那么我们按照上图的方式回写库存就可以了。

但是如果我们是一个仓库下的货品要供货1个以上店铺（比如：淘宝、京东、拍拍都卖），那么在这种共享库存的情况下，我们建议可以按照一定的比例（比如每个店都回写可售库存的30%，留1成做备用）按比例回写库存，类似这样填写：{可售库存}*0.3，并且针对一个阶段的销售情况调整这个比例。

当然也有这种情况，有的客户我仓库库存就是只有1000件货，但是店铺上就是要铺一个放大的库存数，那我们也可以设置规则为类似这样：{可售库存}*3

库存冻结

一、货品总冻结

货品总冻结是订单从前端店铺（例：淘宝、京东、Ecstore等平台）进入ERP后，货品所预占的冻结数量，直到订单完成发货后才会释放货品的冻结数量；

库存深度计算货品总冻结公式为：

货品总冻结 = 货品（所有）店铺冻结 + 货品（所有）仓库冻结

二、货品仓库冻结

货品仓库冻结是订单上的货品审核生成发货单后，所对应仓库预占的冻结数量；

增加仓库冻结的节点：

1). 调拨出库

2). 采购退货

3). 新建发货单

释放仓库冻结的节点：

1). 发货单完成发货

5). 取消、打回、撤销发货单

三、店铺冻结

店铺冻结是货品从前端店铺（例：淘宝、京东、Ecstore等平台）进入ERP后，对应店铺所预占的冻结数量，当订单审核生成发货单后，货品对应的店铺冻结数量会被释放；

增加店铺冻结的节点：

1). 创建订单

2). 订单更新或编辑

3). 订单暂停

4). 失败订单

5). 撤销或取消发货单

释放店铺冻结的节点：

1). 取消订单

2). 订单更新或编辑

3). 新建发货单

4). 余单撤消

商品(物料商品结构)

说明

新版物料版本商品结构是重构的，将原来的商品、货品的概念变更为销售物料、和基础物料

销售物料就是前台销售平台定义的商品，他可以是普通、促销(捆绑)、赠品，对应订单结构中的object层，基础物料是实际销售物料给到仓库时候，仓库实际发的货品，对应订单结构中的item层

这样就可以实现不同平台的销售物料，实际对仓库来说发的是同一个基础物料 并且将原来商品类型、品牌、规则的强关联去掉，因为对仓库来说只关注我要发什么

数据更新逻辑：

新增一个基础物料

sdb_ome_basic_material表新增一条记录

sdb_ome_basic_material_ext扩展信息新增一条记录(重量、金额、包装等)

sdb_ome_basic_material_conf 基础配置信息新增一条记录(保质期)

sdb_ome_basic_material_conf_special 特殊扫码配置新增一条记录

新增一个销售物料

sdb_ome_sales_material表新增一条记录

sdb_ome_sales_material_ext扩展信息新增一条记录(价格、包装)

sdb_ome_sales_basic_material 销售物料基础物料关联表

sdb_ome_sales_material_shop_freeze 初始化销售物料店铺冻结记录

仓库

说明

仓库核心的配置：

仓库类型，对应的仓储服务端(自有|第三方绑定)

关联店铺，回写库存的仓库供货关系

仓库属性，线上、线下(线上仓库货品库存会回写前端)

发货属性，发货仓、备货仓(备货仓不能作为有效仓库进行发货)

仓库属性，自建仓库、第三方仓库(第三方仓库只是在发货流程上简化，导出导入方式)

数据更新逻辑：

新增一个仓库

sdb_ome_branch新增一条记录

wms_id对应sdb_channel_channel表的主键channel_id识别出是自有仓储还是第三方走接口的仓储

采购

说明

采购是出入库的一种类型

比较常用的出入库还有调拨、直接出入库，只是生成的业务单据、流程差异，但最终的结果都是生成相应的出入库明细内容、仓库库存发生变化

举例采购入库：

app\console\controller\admin\purchase.php 新建入口控制器文件

app\wms\controller\admin\eo.php 自有仓入库控制器文件

核心函数：

doSave 新建采购入库单函数

save_eo_confirm 采购入库处理函数

核心逻辑：

1. 新建采购入库单审核后，增加在途数量
2. 审核后取消，减少在途数量
3. 采购入库生成出入库明细、仓库可售库存增加、在途数量扣减

数据更新逻辑：

新建采购入库单审核后，sdb_ome_branch_product

arrive_store 增加相应数量

审核后取消或最终取消，arrive_store 减少相应数量

采购最终入库， arrive_store 减少相应数量，实际库存store增加相应数量

人工分派

核心代码说明：

app\ome\controller\admin\order.php 人工分派控制器入口文件

核心函数：

do_dispatch 订单分配处理函数

核心处理：

1. 分派前订单预处理，自动分派前也会做同样的处理
2. 将订单分派指定到相应的订单确认小组和客服上

数据更新逻辑：

分派成功后，sdb_ome_orders表字段更新

group_id 订单确认小组ID

op_id 订单小组管理员ID(客服)

dispatch_time 订单分派时间点

人工审单

核心代码说明：

app\wms\controller\admin\order.php 人工审单控制器入口文件

核心函数：

finish_combine 订单审核处理函数

核心逻辑：

1. 匹配仓库(审单界面人工选择)
2. 匹配物流公司(审单界面人工选择)
3. 检查订单状态、库存是否满足
4. 生成相应的发货单、释放店铺预占、预占仓库库存、第三方仓储需接口推送

数据更新逻辑：

审单成功后，sdb_ome_orders表字段更新confirm 更新为 Y，process_status 更新为 splited

sdb_ome_delivery、sdb_ome_delivery_items、sdb_ome_delivery_items_detail新增发货通知单主表、明细、以及订单明细关联发货明细记录

sdb_ome_delivery_order 发货单ID与订单号ID关联表新增记录

自有仓储wms_delivery表生成记录，第三方直接接口推送发货通知单

单据打印

打印触发控制器入口文件

app\wms\controller\admin\receipts\print.php

核心函数：

html风格与控件风格区别只是展示方式不同，**html**风格是抛出**php**数据数组展示，控件风格抛出**json**数据给打印控件解析展示

1. toPrintStock html风格 备货单打印函数
2. toPrintMerge html风格 发货单打印函数
3. toPrintStockNew 控件风格 备货单打印函数
4. toPrintMergeNew 控件风格 发货单打印函数
5. toPrintExpre 快递单打印函数

打印所需原始数据封装类

app\wms\lib\delivery\print.php

核心函数：

getPrintDatas 获取打印单据原始数据

核心处理内容：

1. 打印有效性检查(参数、订单、发货单状态等)
2. 发货单排序、打印批次生成处理
3. 原始数据的组织
4. 打印模式标记(单品打印、多品打印、还是普通打印)

原始发货单数据有了，不同的单据需要格式化不同的展示内容

1. app\wms\lib\delivery\print\stock.php 备货单数据格式化类
2. app\wms\lib\delivery\print\delivery.php html风格发货单数据格式化类
3. app\wms\lib\delivery\print\newdelivery.php 控件风格发货单数据格式化类
4. app\wms\lib\delivery\print\ship.php 快递单数据格式化类

核心函数：

format 几个单据实现的格式化函数名一样

数据更新逻辑：

备货单打印,sdb_wms_delivery表字段print_status | 1(按位或)

发货单打印,字段print_status | 2

快递单打印,字段print_status | 4

所以如果一个发货单三个单子都打印了，字段print_status的值是7

根据系统-发货配置中的设置识别到底要打印哪几个单据，都打印了，process_status更新为1，发货单已打印

校验

校验触发控制器入口文件

app\wms\controller\admin\check.php

核心函数：

1. index 逐单/整单校验(根据传入type类型参数识别 barcode 逐单 all 整单)
2. group_check 分组校验(根据订单审单分配的所属分组批量校验)
3. batchIndex 批量校验(提交一批运单，后台执行批量校验)

核心逻辑：

已打印的发货单，检查拣货的内容是否与发货单上的内容对应，通过条形码进行校验计数，当发货单上货品以及数量都核对成功后，发货单标记为已校验

数据更新逻辑：

1. 校验成功一件货品，对应的sdb_wms_delivery_items表货品明细记录字段verify_num加1
2. 某一个货品需校验number与已校验verify_num相同时，字段verify为true
3. 整个发货单校验完成,sdb_wms_delivery表字段process_status变更为3

发货

发货触发控制器入口文件

app\wms\controller\admin\consign.php

核心函数：

1. index 逐单发货(一单单发货)
2. group_consign 分组发货(根据订单审单分配的所属分组批量发货)
3. batch 批量发货(提交一批运单，后台执行批量校验)

核心逻辑：

已校验发货单打包出库，仓储发货单标记为已发货，并且状态回传给OMS

数据更新逻辑：

1. 发货成功后，sdb_wms_delivery表字段status变更为3，process_status变更为7
2. status字段 0 默认 1 取消 2 暂停 3 发货

上面我们主要说的是自有仓储的仓库发货处理流程，针对ONex OMS来说除了自有仓储外，绝大多数都是在使用第三方仓储(新建、取消、发货回传等业务对接)。

OMS在订单审单后，会生成发货通知单，对应sdb_oms_delivery主表及相关扩展表，然后组织接口数据，请求第三方仓储创建发货通知单。

第三方仓储发货回传也会打相应的发货回传接口，以发货单号delivery_bn为映射的唯一单据号。

发货通知单信息推送仓储核心文件

app\ome\lib\delivery\notice.php

核心函数：

1. create 发货通知单创建(审单生成发货通知单时出发)
2. cancel 发货通知单取消(撤销发货单时候触发)
3. pause 发货通知单暂停(自有仓储才会使用、订单暂停第三方仓储直接走取消)
4. renew 发货通知单恢复(自有仓储才会使用)
5. search 发货通知单查询 (京东仓储才会使用，比较特殊，状态什么需要根据关联单据号被动人工获取)

发货通知单响应处理类

app\ome\lib\event\receive\delivery.php

核心函数：

1. consign 发货回传接收处理(适用于所有仓储类型)
2. setPrint 打印回传接收处理(仅自有仓储，第三方仅提供取消和发货两种业务)
3. setCheck 校验回传接收处理(仅自有仓储)
4. rebackDly 取消回传接收处理(仅自有仓储、科捷)
5. updateDetail 更新详情(重量、物流信息)回传接收处理(仅自有仓储)

发货回传指的是仓储根据推送给他的发货信息发货后，将发货结果告诉OMS(无论是自有仓储还是第三方仓储)。

核心处理内容：

1. OMS发货通知单的变更
2. 库存、冻结的变化
3. 订单状态变更
4. 出入库明细生成
5. 销售单生成
6. 短信、订单状态回传前端销售平台等等

订单分组

说明：

自动审单流程中，订单分组是必要条件。 后续的处理流程都是基于订单所属的订单分组进行的。

数据更新逻辑：

新建订单分组后，sdb_omeauto_order_type表新增一条记录，字段group_type值为order类型

字段说明：

1. oid：自动审单规则的关联id
2. did：自动分派规则的关联id

自动分派

自动分派封装类

app\omeauto\lib\auto\dispatch.php

核心函数：

getAutoDispatchUser 获取自动分派的订单分派确认小组和人

核心逻辑：

根据传入订单识别出具体属于的订单分组，根据订单分组关联的分派规则获取到具体分派的确认小组和人

数据更新逻辑：

1. 新增分派规则后,sdb_omeauto_autodispatch表新增一条记录
2. 根据设置时选中的订单分组，更新sdb_omeauto_order_type表具体订单分组的did字段的值

自动审单

自动审单入口文件：

app\omeauto\lib\auto\combine.php 自动审单入口文件

核心函数：

process

核心逻辑：

根据传入的订单ids数组，识别出属于哪个订单分组，然后根据订单分组执行相应的分派、审单规则、物流优选、分仓逻辑

关联的类文件：

1. app\omeauto\lib\auto\group.php 分组订单的封装处理类(审单插件处理)
2. app\omeauto\lib\auto\group\item.php 具体订单审核处理(根据插件处理结果)

数据更新逻辑：

1. 新增审单规则后, sdb_omeauto_autoconfirm表新增一条记录
2. 根据设置时选中的订单分组，更新sdb_omeauto_order_type表具体订单分组的oid字段的值

自动分仓

核心代码说明：

1. app\omeauto\lib\auto\plugin\branch.php 自动选择仓库封装类
2. app\omeauto\lib\auto\plugin\store.php 库存检查封装类

核心函数：

process

核心逻辑：

根据自动仓库规则识别当前订单满足条件的仓库列表，再根据仓库库存实际情况是否满足，选定一个权重高的仓库

数据更新逻辑：

1. 新增分派规则后, sdb_omeauto_order_type表新增一条记录,group_type字段值为branch
2. 在规则中设置具体哪些仓库启用以及权重保存后，sdb_omeauto_autobranch表(按仓库规则id与仓库id一对多的关系)增加多条记录

常见问题**FAQ**

安装部署

任务不执行

任务不执行、休眠的问题

1. 控制面板-其他-队列管理中有一些任务一直休眠中，然后手工启动就可以了。
2. 系统-导出任务中任务一直是请求中，不动

遇到上述两类问题，请按以下步骤进行检查：

1. TASKMGR里config.php文件是否配置好了，并且进程脚本是否正确的stop start命令重启过。
2. 检查TASKMGR下logs里的日志，每个任务都会有相应的日志文件，查看里面的请求内容，尤其是域名是否正确，并且根据域名访问下，是否可以正常访问，比如因为开启了防火墙，或者内网机器不能访问外网域名等等(日志里请求状态200为正常状态 500基本是php代码报错 100不通)。

基本上做了上面两步操作问题即可解决。

PS：有的时候服务器禁止了本机通过域名访问自己，任务日志里看到请求的状态是0，这个时候可以直接在etc/hosts里指定下当前oms的域名是127.0.0.1即可

多套**OMS**服务器部署策略

真实项目需求

客户购买了多套OMS，实现不同区域的业务分开管理(大陆、国外)，服务器应该如何部署

情况一

有钱任性，前期就考虑不要因为资源共享而导致后续资源或者服务紧张，最终导致业务服务的性能瓶颈问题

建议：按照具体项目方案部署两套完全独立的OMS系统，并且相关的硬件、服务也独立

情况二

节约成本，在量不大的时候可以共用一部分硬件、服务或资源

建议：

1. web服务，代码部署两套在相同的web机上指定不同的目录路径以及域名访问地址
2. 缓存服务，比如memcache 按具体系统分不同端口，可在同一个服务器上
3. KV服务，比如redis 按具体系统分不同端口，可在同一个服务器上
4. 队列服务，这个比较特殊，一般轻量级的会用redis(注意这里的redis队伍服务绝对不可以和kv服务用同一个，至少是不同的端口)，中大型的用rabbitmq。

PS：

taskmgr任务管理中，队列名前缀我们可以自定义(脚本机和web机上的代码app\taskmgr\config\config.php都需要修改,基于v3.7.6版本)，所以可以部署两套脚本，不同的队列名，用同一个队列服务。

并且两套脚本要部署在一台服务上，需要调整守护进程app\taskmgr\check.sh里的进程是否存在判断，可以根据脚本所在不同路径加grep的过滤，并且app\taskmgr\taskDaemon.php中需要修改下后缀为.pid文件的文件，区分成不同的文件名原来是erp-前缀

oms订单状态没有回传至ecstore，在oms中状态回写一直显示 “发货中”是什么原因？

如果是待发货状态，说明没有正常发起发货请求，可以查看下队列里是否存在回写的任务没执行，

如果没有正常执行，请检查定时任务taskmgr的配置

如果是发货中查看日志中是否有请求，查看是否正常返回，如果超时可以重试

配置了日志服务，查询功能无法正常使用可以在哪里查询？

检查日志服务的存储文档定义是否添加了索引

当然日志服务的管理工具rockmongo也提供单独的界面做查询

oms库存，自动回写天猫店铺功能失效是为什么？

1.确认一下店铺回写库存有没关闭

2.检查定时任务taskmgr配置 http://club.ec-os.net/doc/oms-dev/500.faq/2.cron.md

报表-》商品销售情况 列表中的销售金额有小数，是如何计算的？

因存在订单优惠金额，系统在计算商品销售金额时，
会将订单优惠金额依据商品价格分摊到各商品上，所以才会出现小数。

比如说订单优惠金额为254，则会将这254分摊到各商品上，分摊比例为商品金额占总金额的比，
就是（订单优惠金额 / 订单实际总金额）*商品实际金额

导出任务列表一直显示运行中，无法下载，是为什么？

如果是写本地文件的方式，检查路径/data/export是否有写权限

批量单据校验，及批量发货时，状态一直显示等待中，怎么办？

检查定时任务taskmgr配置

http://club.ec-os.net/doc/oms-dev/500.faq/2.cron.md

oms批量同步会进入缓存队列中，稍后执行，在队列中看到这次同步是休眠的状态，是否是自动启用的？

检查定时任务taskmgr配置

http://club.ec-os.net/doc/oms-dev/500.faq/2.cron.md

加入缓存队列中稍后会自动执行，但是进入休眠状态说明计划任务，

没有配置正确，需确定crontab -l 下的计划任务是否全部开启

oms的计划任务是怎么做的？目前oms自动派单是由计划任务定时执行吗，还是订单进入系统后立即分派的？

```
http://club.ec-os.net/doc/oms-dev/200.install-deploy/4.taskmgr-server.md
```

关于**oms**安装的参考文档在哪里看？

```
http://club.ec-os.net/doc/oms-dev/200.install-deploy/1.yum3-5-0.md
```

oms的授权文件要放在哪里，安装**oms**，选择**“**正式环境**”**，输入**shopexID**，直接可以进入**oms**后台，而且显示是正式授权，但是使用的时候有并发限制是怎么回事？

oms授权文件配置在/etc/php.d/Zend.ini中配置，配置的是oms代码包自带的授权文件，

所以有并发限制，需配置正式环境的授权文件路径。

怎么切换**oms**到沙箱的调试环境，奇门仓储调试又要做哪些配置和注意些什么**?**

第一步，安装后修改配置信息文件config/config.php在底部增加如下代码：

```
define('LICENSE_CENTER_INFO','http://service.ex-sandbox.com/');
define('SHOP_USER_ENTERPRISE','http://passport.ex-sandbox.com/index.php');
define('SHOP_USER_ENTERPRISE_API','http://passport.ex-sandbox.com/api.php');
define('OPENID_URL','http://openid.ex-sandbox.com/redirect.php');
define('LICENSE_CENTER','http://service.ex-sandbox.com/openapi/api.php');
define('LICENSE_CENTER_V','http://service.ex-sandbox.com');
define('MATRIX_URL','http://rpc.ex-sandbox.com/async');
define('MATRIX_REALTIME_URL','http://rpc.ex-sandbox.com/sync');
define('MATRIX_RELATION_URL','http://sws.ex-sandbox.com');
```

第二步，修改数据库内sdb_base_network表内的node_url字段：

```
http://matrix.ecos.shopex.cn 替换成 http://rpc.ex-sandbox.com
```

第三步，调用框架函数，获取沙箱环境下的OMS证书以及节点

修改根目录下script/update/script/的脚本，可以拿updateOme.php改一下，主要目的执行一下base_certificate::register();

这里要注意，如果这里获取证书和节点号不成功，可能是两个情况，一个是你的测试服务器部署在外网，外网访问授权服务的时候要添加外部访问的白名单才可以正常请求，另外种情况，你可以开启系统其他配置的系统内部日志，查看data下的请求log是超时还是其他问题

以上三步都执行成功了，oms环境便切换成了沙箱环境

基于上述处理步骤后，奇门仓储配置步骤

第一步，奇门会对对接的仓储提供测试环境的cutomerId appkey获取到这个信息

第二步，联系这边的技术进行奇门测试环境的绑定

第三步，oms仓储管理里进行绑定

完成以上三步，绑定即完成

操作使用

降级开关设置

内置的降级开关设置

OMS区别与其他系统，对于数据量承载的要求会相当高，尤其在搞大促的时候，系统原有存储着大量数据，在做一些操作的时候会出现加载慢的问题，在这种时候针对业务需要和性能需要之间，我们往往更加需要的系统的操作使用流畅性，所以这里我们提供了一些降级的开关设置，减少一些可忽略的业务数据计算统计和展示。

逐单校验与发货

设置是否在逐单校验、逐单发货的页面展示还有多少单据要处理。

列表展示**TAB**的计数

设置是否展示相关数据，比如订单、发货单列表上TAB分栏的计数

PS:目前这个开关选项只在订单、发货比较核心的数据列表TAB展示起作用，具体如果其他地方需要的话，可找到具体列表展示的控制器中，找到_views方法里TAB的ADDON键定义的count方法，将count改为viewcount方法即可。

比如仓储中心里的发货单列表TAB

```
#原来:
$sub_menu[$k]['addon'] = $oDelivery->count($v['filter']);
#改为:
$sub_menu[$k]['addon'] = $oDelivery->viewcount($v['filter']);
```

请求响应及日志

设置请求响应内容是否进队列(防止被第三方打死)，系统内部日志是否记录

是否进队列依附于TASKMGR任务管理服务，这里就不多说了。

系统内部日志指的是原来系统针对一些APP更新操作日志，非OMS系统操作日志(ECOS框架机制),具体日志会生成在OMS安装部署根目录下data\logs下，日积月累这块的LOG日志也会很大，OMS这块默认开关设置不起开启，建议关闭(排查问题基本也不会用这块，实在想开启的，记得定时清理一下)。

发货单撤销失败

真实项目需求

客户使用的WMS是第三方如科捷,伊藤忠

情况一

在ERP里订单已经生成了发货单,但是由于退款或者其它情况，需要暂停订单，或者撤销发货单 这个时候，由于ERP需要向WMS发送撤销发货单，成功时，才能做暂停订单的操作

WMS接收发货单成功，撤销成功

情况二

WMS由于超 时或者其它情况，没有接收成功，会返回失败，返回发货单不存在类似的错误，可以查看发货单日志里面有返回失败的原因。

解决：

1. 菜单位置：仓储中心-》第三方发货单撤销失败 待发货tab里。有一个“发货单取消” 可以勾选发货单，进行操作，就会模拟第三方仓储向ERP发送取消发货单操作

手工将订单设置为异常订单，此订单不会在我的异常订单中显示？（订单）

我的异常订单并不是显示的异常订单，复审未通过的才进入我的异常订单。

订单模块---查看订单中，【历史订单】和【当前订单】分别列出的是什么类型的订单？

“当前订单”是指款到发货订单未完成发货作业、

货到付款订单未完成付款作业等未处理完成的活动订单的信息列表。

“历史订单”主要管理系统已完成订单历史查看。

历史订单不包括失败订单等异常的订单。订单归档后，从历史订单移到归档订单。

系统中的校验方式有逐单校验、分组校验、整单校验，各种校验方式有什么区别，都是怎么校验的？（发货）

逐单校验：首先输入或者扫描快递单号，将会显示该快递单的所有货品信息，然后通过扫描每个商品的条码，

进行单个商品校验，校验通过所有商品之后进行提交。

分组校验：首先得先在分组规则里设定组别，可以安装订单来自的店铺，校验时选择相应的组别，

并且各组别会显示需要校验订单的数量。点击校验之后将会批量将该组别的订单都校验完成。

整单校验：扫描快递单号之后，显示整单的货品信息，

点击通过完成整个快递单的校验，不会单独检查商品具体信息。

淘宝前端设置了满就送的插件，oms中应该会自动获取到该笔订单的赠品信息，但为什么实际接收中没有赠品？（系统）

需要在oms的系统设置中配置淘宝赠品

订单发票管理-发票列表，未开票状态如何变为已开票？（系统集成）

oms只是列出需开票的信息内容，并且提供openapi给商家调用，

商家可以通过openapi更新发票状态

系统集成--基础设置

里面点击添加一个api，把发票操作都打上勾，这个api会和你的税控机系统对接，

进行信息传输之后打印发票，再把这个信息传送回来到oms系统，从而改变状态。

有OMS的数据库资料和开发开放接口文档资料可以看吗？

OMS所有数据库文档：http://club.shopex.cn/doc/oms-database_dictionary/3.7.2.md

OMS开放接口文档：http://club.shopex.cn/doc/oms-apis/README.md

OMS在绑定亚马逊的时候，密钥和**key**怎么获取？

在客户亚马逊网店设置，

参考http://zhidao.baidu.com/link?url=OVN4ujZJN2HmEt7DFuxmTwWMD2UqJcBLt8wPCkqoppnSwt0QOFbaIbVlk2KT1xUxTpeIxCvjB5XvBvwjeQSL_gwh7OoXU8uNIniYgwAXhuG

京东云仓的物流信息，**OMS**只能人工手动点击【获取发货结果】按钮才能获取到，无法自动获取是为什么？

京东不会主动返回处理信息给到OMS，所以只能oms被动去拉，是京东仓储机制问题。

使用激活码激活后，一级域名和现在的**OMS**的网址两个域名是否都可以使用？

不可以，一个激活码只能绑定一个域名。

开发问题

订单管理这边在待处理里面，点击获取按钮后，为何要等很久才会出界面？

和暂存区的订单比较多，并且可合并订单比较多有关，会做相应的计算的，所以需要等待

OMS的菜单项太多，很多功能不需要,如何进行多余菜单的屏蔽？

设置后台操作习惯,菜单定制,或者新开账户分配对应的权限

OMS http 500错误如何调试？

参考文档 http://support.shopex.cn/helper-helper-73.html

未分派订单--获取订单功能 是从 订单暂存区 自动获取订单，

但有时候 【订单暂存区】有订单， 【未分派订单--获取订单 】取不到，

提示【当前订单暂存区中没有可处理的订单，请稍后再试……】，

但过了一会儿（大概几分钟，中间没有任何人为操作），又能取到了，是什么原因?

这个设置有订单缓冲限制,根据 系统-》系统设置-》订单配置-》订单缓冲时间，

如果是10分钟，那么只能获取在订单暂存区至少10分钟缓冲的订单

标准oms里面是在哪一步触发ecstore发短信机制的？这个功能的具体代码位置在哪里？

oms本身有对订单发货订单的实现发短信机制，

并没有触发ecstore 的发短信的机制，在菜单栏，系统-》短信配置

OMS中有导入导出csv文件的框架， 如果自定义开发功能中，

用到导入导出，能否复用oms中现有的导入导出框架， 如何使用?

导入这块可以查看ecos的框架说明文档

http://club.ec-os.net/doc/ecos/framework-ecos/advance/desktop/export-import.html

oms中导出已经替换了原来ecos的机制，可以查看文档

http://club.ec-os.net/doc/oms-dev/300.framework/3.export.md

系统联通

对接BBC配置

真实项目需求

因BBC和OMS对接,一期走的是矩阵,二期走的是prism 联通方式

情况一

已对接了bbc一期,不想重新安装，想直接切换到BBC二期

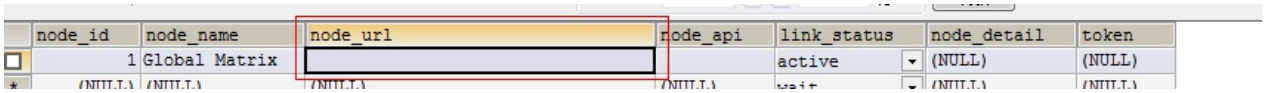
里加上如下图的代码

```
    'IMAGE_MAX_SIZE' => 1024*1024,
    'KV_PREFIX' => 'defalut',

    'LANG' => 'zh-cn',
    'APP_SOURCE' => '897807',
    'APP_TOKEN' => 'd9e3c6eb01737f39b805d35cd6795e60',
);
$constants_ext = array();
$file = ROOT_DIR.'/config/defined_ext.php';
if(!file_exists($file)){
    $fp=fopen($file,"w");
}
include_once $file;
$constants = array_merge($constants,$constants_ext);

foreach($constants as $k=>$v){
    if(!defined($k))define($k,$v);
}
```

根据目录config/下新建一个defined_ext.php的文件

```
<?php $constants_ext=array (
  'LICENSE_CENTER_INFO' =>
  'LICENSE_CENTER' => 'http:/
  'LICENSE_CENTER_V' => 'htt
  'MATRIX_URL' => 'http://f3s
  'MATRIX_REALTIME_URL' =>
  'MATRIX_RELATION_URL' =>
);
```

sdb_base_network 表里node_url 更新为prism 的MATRIX_URL地址



node_id	node_name	node_url	node_api	link_status	node_detail	token
1	Global Matrix			active	(NULL)	(NULL)

然后重新获取证书即可

情况二

直接安装最新版本OMS

OMS安装时联通方式分两种,默认是走公有矩阵

全内网即prism方式，即BBC二期联通方式，附图说明

像图示一样。www.abc.com是示例prism网址，需替换成对应prism网址

剩下的下一步照日常安装即可

新建采购退货单时审核后会触发第三方仓库生成采购单，但是有时候失败了，系统有重试功能吗？

直接在采购列表点击推送到wms按钮

打印发货单的批次号的生成规则，及哪些发货单是属于同一批次的？发货单，备货单各起什么作用？

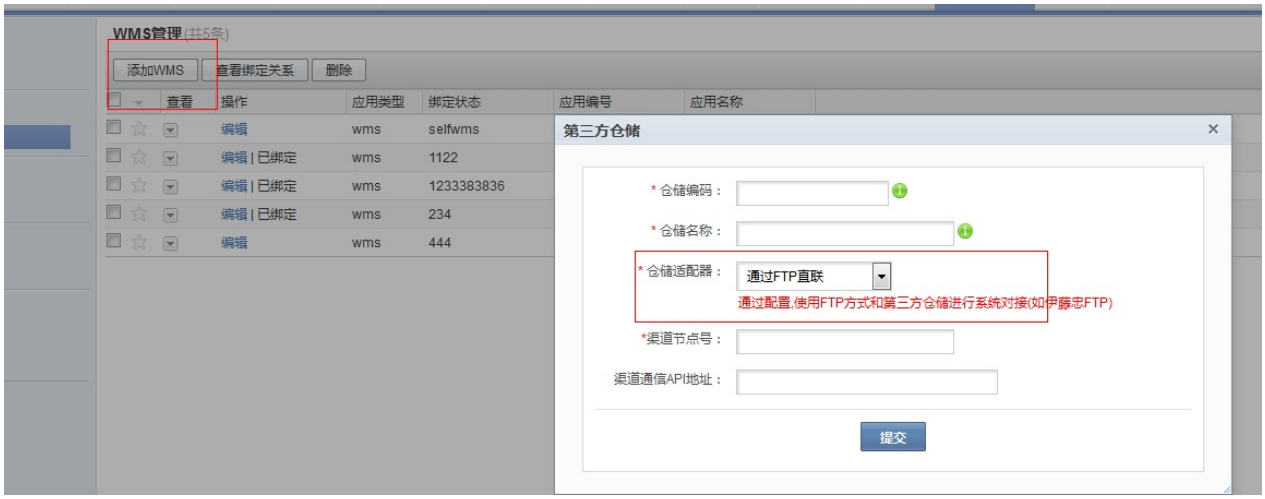
生成规则：操作员id-年月日-批次号-该批次中的排序

同一批次：前三段相同，例如：1-41124-0003_01、1-41124-0003 _02、1-41124-0003 _03 ；

备货单：是备货人员根据备货单明细执行备货的根据 ；

发货单：是指仓库内流转的单据，可以知道什么货物、多少数量需要出库，以及收货人信息

对接**FTP WMS**



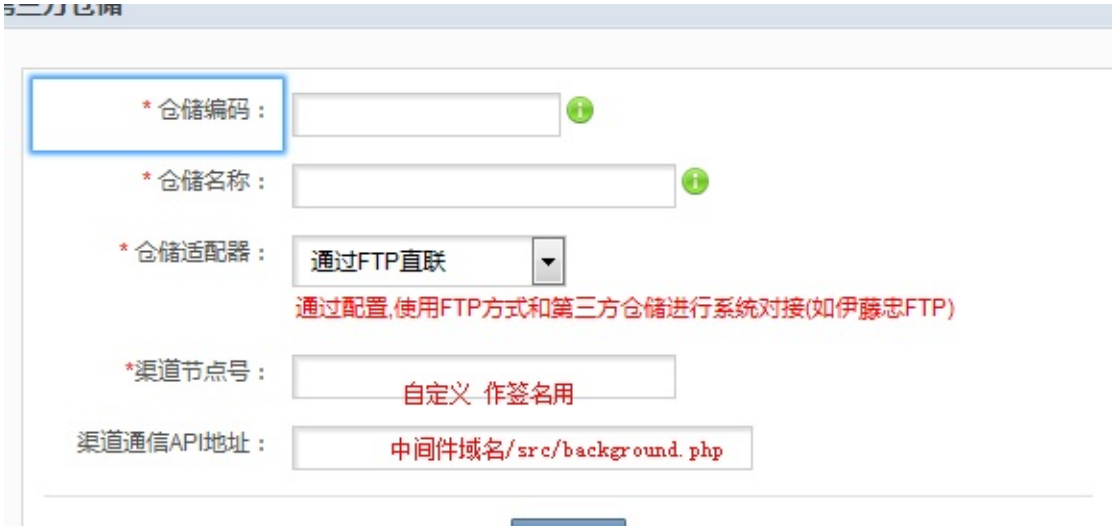
如图示,新建WMS选择了FTP 直联

ftp中间件代码需要申请

拿到中间件代码后 根目录里 config.php 修改如图

```
<?php
define('SELF_DIR', realpath(dirname(__FILE__)));#当前目录


define('OCS_OPENAPI_URL', 'http://erp域名/index.php/api');

#FTP配置
define('FTP_SERVER', '');#FTP服务器地址
define('FTP_UID', '');#FTP用户名i
define('FTP_PWD', '');
define('FTP_PASV', true);#FTP连接模式:false主动,true被动
define('FTP_UPLOAD_DIR', SELF_DIR.'/upload/');#FTP上传目录
define('FTP_UPLOAD_DIR_BACKUP', SELF_DIR.'/upload_copy/');#FTP上传备份
define('FTP_DOWNLOAD_DIR', SELF_DIR.'/download/');#FTP下载目录
define('FTP_DOWNLOAD_DIR_BACKUP', SELF_DIR.'/download_copy/');#FTP
define('FTP_ERROR_DIR', SELF_DIR.'/error_info/');#通信错误数据

#日志
define('LOG_FILE',SELF_DIR.'/logs/{date}/{ip}.log');#日志文件名路径
```

需要注意 中间件目录下download download_copy logs upload upload_copy需要有可读写权限



FTP中间件需要布crontab 脚本 示例如下 其中路径和中间件位置此示例只作参考:

30 /bin/bash /data/httpd/中间件域名/cronjob/crontab_down.sh /usr/local/php/bin/php >/dev/null 2>&1

0 /bin/bash /data/httpd/中间件域名/cronjob/crontab_up.sh /usr/local/php/bin/php >/dev/null 2>&1

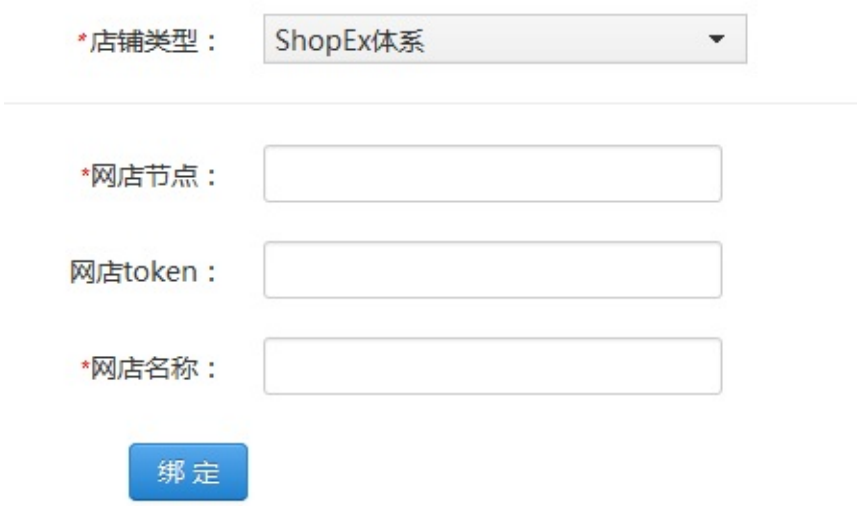
0 /usr/local/php/bin/php /data/httpd/中间件域名/cronjob/crontab_retry.sh >/dev/null 2>&1

ECSTORE联通OMS绑定步骤说明：

1.首先进入控制面板 **->** 数据互联页面。

2.数据互联页面：点击**“**新建绑定关系**”**，输入相关商店系统的名称，再点击**“**申请绑定**”**进入到申请绑定页面。

3.申请绑定页面：输入相关**OMS**的绑定信息进行系统绑定。



4.绑定成功后可点击按钮查看绑定情况，确认系统两边都已绑定完成。

5.绑定后在**ec**后端订单的相关操作不可用，所有的订单操作将在**ec**前端或**oms**中完成。