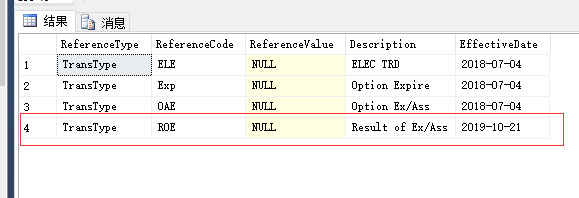
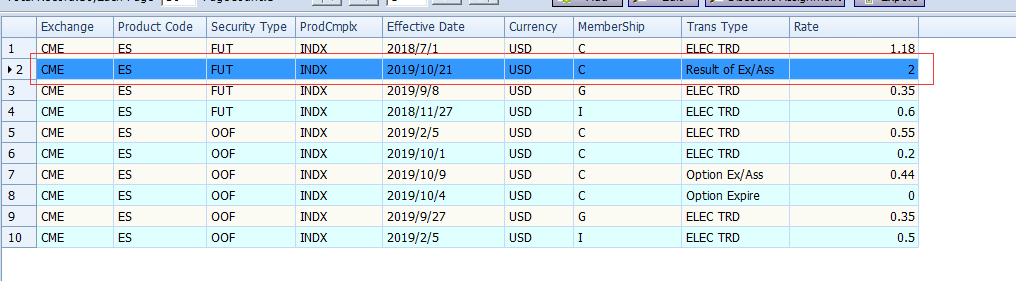
因为期权行权插入一笔期货成交，TrdTyp为18，你查出来的1.18是ELEC TRD,值为0的费率，并不是反向插入的期货成交的费用类型，需要再TReferenceData表中插入  
  
，并且再ExchangFee中增加所有产品这个类型的费用，如  


庄家：I am calling for someone to buy my stock shares. 对于买走这个权利的人，他是看涨的（认购）。

庄家：I put the invitation to buy the shares at a convenient price. 对于买走这个权利的人，他是看跌的（认沽）。

看涨期权（call option），看涨期权又称买进期权，买方期权,买权，延买期权，或“敲进”，是指期权的购买者拥有在期权合约有效期内按执行价格买进一定数量标的物的权利。看涨期权是这样一种合约：它给合约持有者（即买方）按照约定的价格从对手手中购买特定数量之特定交易标的物的权利。

　　看跌期权（put option）

　　如果未来基础资产的市场价格下跌至低于期权约定的价格（执行价格），看跌期权的买方就可以以执行价格（高于当时市场价格的价格）卖出基础资产而获利，所以叫做看跌期权。如果未来基础资产的市场价格上涨超过该期权约定的价格，期权的买方就可以放弃权利。按期权履约的方式可以分欧式期权与美式期权。欧式期权必须持有至到期，是不能提前执行的。美式期权可以看做附有提前执行权利的欧式期权，即可以在到期日前的任一交易日行权。

var allData = from p in dtExchangeFee.AsEnumerable() select p;

var groupData = from p in dtExchangeFee.AsEnumerable()

group p by new { segType = p.Field<string>("SegType"), cy = p.Field<string>("FeeCurrency"), regCode = p.Field<int>("RegCode") } into g

select new

{

key = g.Key,

Data = g

};

var groupData = allData.GroupBy(p => new { segType = p.Field<string>("SegType"), cy = p.Field<string>("FeeCurrency"), regCode = p.Field<int>("RegCode") }).Select(group => new { key = group.Key, Data = group });

平赢、浮盈计算调用RemoveNum5函数

if (trdCapRpt.Tag50Type == "D")//Dealer 更新该笔TtrdCapRpt记录为‘U'-待分配

else//不存在Tag50,将新的Tag50保存到Tag50表,类型为undecided.

首先来说下非泛型转换，这种转换方式就是通过遍历DataTable，然后向Model中的相同字段中赋值即可。

//使用Linq的ToList方法，非泛型

public static List<Entity> ConvertTo(DataTable dt)

{

if (dt == null) return null;

if (dt.Rows.Count <= 0) return null;

List<Entity> list = new List<Entity>();

list = (from DataRow dr in dt.Rows

select new Entity

{

PID = Int32.Parse(dr["PID"].ToString()),

PName = dr["PName"].ToString(),

PPass = dr["PPass"].ToString(),

PAddr = dr["PAddr"].ToString()

}).ToList();

return list;

}

代码很简单，就是通过循环赋值而已。这里我用的Model测试类如下：

FROM TTradeDetail a

LEFT JOIN TTrdCapRpt b ON ( CASE OpenSide

WHEN 1 THEN LongRptID

ELSE ShortRptID

END ) = b.RptID

WHERE CloseFlag = 'Y'

AND LastUpdateDT = CONVERT(VARCHAR(10), GETDATE(), 23)

LEFT JOIN TTrdCapRpt d ON ISNULL(LongRptID, ShortRptID) = d.RptID

WHERE CloseFlag = 'N'

AND a.MQMSecType = 'FUT'

状态为18，表示期权行权反向插入的期货成交，状态为4和5表示期权行权后的状态

TradeType状态为0的为正常成交，3为移仓，为-1的是手动插入，2未EFP,A0为交割，rpttype为A0为交割，都不收费，但在30.7的交割可能要收交易所费用。但在listingreport中需要展示，平赢浮盈模块需要展示

VemuType 为E电子盘，为D交割，为M手动插入,为X移仓，为O行权插入的状态为4 或5的期权成交（Y），或者是状态为18的期货成交（N）,EFP为X

新增一个反向事件合约的类型，TrdTyp=4并且rpttyp=Event

平仓前会先调用交割的方法，这个合约，这个产品的结算日期等于今天的日期的期货，就需要添加反向持仓去平掉TTradeDetail的仓位进行交割，交割必须有结算价，否则跳过不交割。平仓的时候结算价还没出来，柜台交割比清算晚一天，但是经查发现ExpirationCheck的最后交割日是取得LastTradeDate，而自动交割取得是SettlementDate？，这两个字段大部分数据都意思相同，但有部分行相差几天。有的确实settlement date会晚几天，但如果超过last trade day还不平仓，结果还是现货交割，那就麻烦了

//593比对

CMEReport.DataCompare(config);系统先通过ftp从cme下载download Trade Register Report file，然后处理后入库到TTrdRegRpt表，然后根据tybalance的数据与TTrdRegRpt表的数据比对算出一个差值写到TTrdRegDiffenence报表以后生成cmepointbalance.pdf报表使用

DetailOpenPositionReport

UNREALIZED浮盈只针对期货来的说法，期权没有浮盈的说法，OptionValue：期权价值，只针对期权的说法，期货没有期权价值的说法，市值MarketValue，只有期货有市值的说法，期权没有。

计算公式如下：

UNREALIZED：

开多仓：( SettlementPx - TradePrice ) \* Multiplier\*qty

开空仓：( TradePrice - SettlementPx )\* Multiplier\*qty

LongOptionValue：开多仓：SettlementPx \* Qty \* Multiplier，开空仓：0

ShortOptionValue：开空仓-SettlementPx \* Qty \* Multiplier，开多仓0

OptionValue：LongOptionValue- ShortOptionValue

MarketValue：SettlementPx \* Multiplier \* ( Long + Short ) AS MarketValue

数据：

SELECT \* FROM dbo.TyBalance where AccountNo='DAH0010001' and MQMExchangeCode='CBT' and ClearProductCode='C' order by BizDate desc

期货：

本交易日InitialBalance：等于上个交易日的EndingBalance

本交易日的EndingBalance：等于本交易日的InitialBalance-上个交易日的EquityBalance+平盈（CloseProfit）

本交易日InitialCashBalance：等于上个交易日的EndingCashBalance

本交易日EndingCashBalance：等于本交易日的InitialCashBalance+平盈（CloseProfit）

本交易日的EquityBalance：等于本交易日的EndingBalance+OTE(浮盈)

期权：

本交易日InitialBalance：等于上个交易日的EndingBalance

本交易日的EndingBalance：等于本交易日的InitialBalance-上个交易日的EquityBalance+ Premium

本交易日InitialCashBalance：等于上个交易日的EndingCashBalance

本交易日EndingCashBalance：等于本交易日的InitialCashBalance+ Premium

本交易日的EquityBalance：等于本交易日的OptionValue（LongValue - ShortValue）

var tempEntity = new

{

TMFCode = string.Empty,

TAG50 = string.Empty,

TraderCode = string.Empty,

Tag50Type = string.Empty,

TraderName = string.Empty,

ClientCode = string.Empty,

AccountNo = string.Empty

};

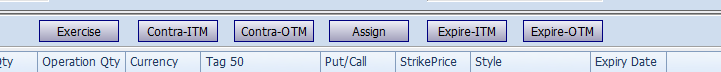
tempEntity = JsonHelper.DeserializeAnonymousType(info.Conditions, tempEntity);

var productCondition = productCode.Aggregate("", (current, s) => current + string.Format("'{0}',", s));

+ sqlWhere.AppendFormat(" AND product.JDECode in ({0})", productCondition.Trim(','));

期权有两种欧式期权和美式期权，欧式期权只能在到期日行权，美式期权则是从开仓之日起至到期日都可以行权。

Excerised:

1. Only long option position can be excerised!
2. The number of excerise rights cannot be greater than the number of positions
3. European options can only be exercised on the expiration date!
4. 

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Side** | **Put/Call** | **Operation** | **Result** | **TrdTyp** |
| Long | Call | Expire-ITM | 到期日价内行权，期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |
| Long | Call | Expire-OTM | 到期日价外不会行权，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Long | Call | Contra-ITM | 到期日价内但放弃行权，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Long | Call | Contra-OTM | 到期日价外但行权，期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Call | Expire-ITM | 到期日价内被assign，期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Call | Expire-OTM | 到期日没有被assign，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Short | Call | Contra-ITM | 到期日价内但没有被assign，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Short | Call | Contra-OTM | 到期日价外但被Assign，期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Long | Put | Expire-ITM | 到期日价内行权，期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Long | Put | Expire-OTM | 到期价外不会行权，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Long | Put | Contra-ITM | 到期日价内但放弃行权，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Long | Put | Contra-OTM | 到期日价外但行权，期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Put | Expire-ITM | 到期日价内被assign，期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Put | Expire-OTM | 到期日没有被assign，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Short | Put | Contra-ITM | 到期日价内但没有被assign，期权插入平仓，期货没有仓位插入 | 期权平仓成交：4 |
| Short | Put | Contra-OTM | 到期日价外但被Assign，期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |
| Long | Call | Exercise | 期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |
| Long | Put | Exercise | 期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Call | Assign | 期权插入平仓，期货插入对应Short头寸 | 期货成交 ：18 期权平仓成交：5 |
| Short | Put | Assign | 期权插入平仓，期货插入对应Long头寸 | 期货成交 ：18 期权平仓成交：5 |

Func<Product, bool> filter = delegate(Product p) { return p.UnitPrice > 10m; };

var query = context.Products

.Where(filter)

.Select(p => new { p.ProductID, p.ProductName });

//需求：打印出整型集合list的元素

List<int> list = new List<int>() { 1, 2, 3, 4, 5 };

//总结匿名方法和lambda常用来初始化委托

//将匿名方法分配给 Action<T> 委托实例

Action< int > concat1 =  delegate ( int  i) { Console.WriteLine(i); };

   list.ForEach(concat1);

}

//将 lambda 表达式分配给 Action<T> 委托实例

Action< int > concat2 = (i => Console.WriteLine(i));

   list.ForEach(concat2);

   Console.ReadKey();

//需求：查找整型集合list中大于3的所有元素组成的新集合，并打印出集合元素

List<int> list = new List<int>() { 1, 2, 3, 4, 5 };

//将匿名方法分配给 Func<T,TResult> 委托实例

Func< int ,bool > concat1 =  delegate ( int i) { return i > 3; };

var newlist1 = list.Where(concat1).ToList();

//将 Lambda 表达式分配给 Func<T,TResult> 委托实例

Func< int ,bool > concat2 = i => i > 3;

   var newlist2 = list.Where(concat2).ToList();

7. ToLookup

　　ToLookup操作符基于一个特定的主键，将返回的元素放置到一个Lookup(TKey TElement)对象中。Lookup是主键的集合，其中每个主键都映射到一个或多个值上。Lookup对象是一个一对多的字典。

var expr = contex.Categories

.ToLookup(c => c.CategoryID,c=> c.CategoryName );

foreach (var item in expr)

{

Console.WriteLine(item.Key);

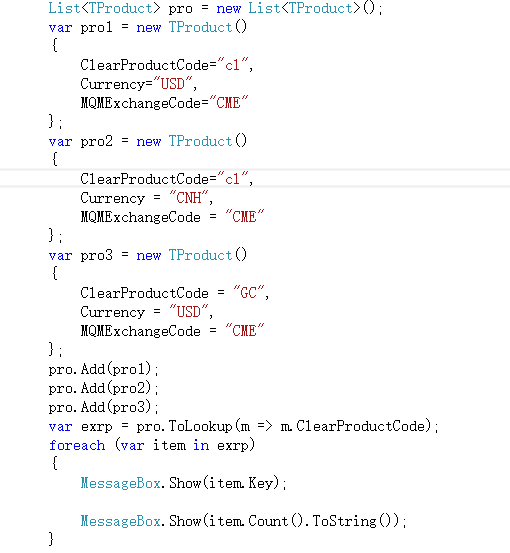
foreach (var i in item)

{

Console.WriteLine(i);

}

}



DataTable查询

var rows = products.AsEnumerable()

.Where(p => p.Field<decimal>("UnitPrice") > 10m)

.Select(p => new

{

ProductID = p.Field<int>("ProductID"),

ProductName = p.Field<string>("ProductName"),

UnitPrice = p.Field<decimal>("UnitPrice")

});

多个DataTable查询

var query = from p in ds.Tables["Product"].AsEnumerable()

from c in ds.Tables["Category"].AsEnumerable()

where p.Field<int>("CategoryID") == c.Field<int>("CategoryID")

&& p.Field<decimal>("UnitPrice") > 10m

select new

{

ProductID = p.Field<int>("ProductID"),

ProductName = p.Field<string>("ProductName"),

CategoryName = c.Field<string>("CategoryName")

};

DataTable数据排序

var rows = products.AsEnumerable()

.Where(p => p.Field<decimal>("UnitPrice") > 10m)

.OrderBy(p => p.Field<int>("SortOrder"))

.Select(p => new

{

ProductID = p.Field<int>("ProductID"),

ProductName = p.Field<string>("ProductName"),

UnitPrice = p.Field<decimal>("UnitPrice")

});

多列分组

var expr = from p in context.Products

group p by new

{

p.CategoryID,

p.Discontinued

}

into g

select new

{

g.Key,

ProductNumber = g.Count()

};

## Trade Register Report对账

CME的仓位系统维护着每个仓位账户的全部交易标的未平仓头寸的数据。CME根据上报的PCS文件生成Trade Register Report文件，盘中和盘后各生成一次，文件包含仓位信息和成交信息，以FIXML的格式传输。如果持有未平仓位的交易标的当日没有交易，文件中就不包含相关的成交信息。CMF可以利用该文件在与后台系统的仓位，金额和成交信息进行对账，并找出不一致的仓位或成交信息后进行调整，保持清算系统的数据与CME的数据保持一致。

仓位信息包括：仓位账户信息，交易标的信息，仓位数量，仓位变动导致的金额变化。Trade Register report的格式说明请参考2.10。清算对账系统首先需要解析Trade Register Report, 按照PosRpt 中的Party role=1 对应的TMF ID, Party Role =38 对应的SubID，type，business date, Instrument ID, Security Type和MMY ,得到对应的 Qty Typ=FIN的long和short对应的仓位数量，与同样的business date, Party role=1 对应的TMF ID, Party Role =38 对应的SubID，type， Instrument ID, Security Type和MMY下的long 仓位进行比较，如果一致，则比较下一个交易合约的仓位；如果不一致，则提取该合约对应的成交明细(ReqID)，与本地的成交明细(ReqID)进行比较，找出不一致的成交信息。可能会发生CME有成交，但是清算公司没有该笔成交的情况。

比对结果生成文件提供给清算人员进行后续处理。若信息不一致，比对结果记录：交易日, 合约，成交记录编号，比对结果等信息，供清算公司详查。比对过程由后台程序自动完成，当Trade Register Report文件下载后就触发比对过程。用户可以在清算系统前台进行查询和手动触发比对过程。

清算系统中缺失的成交记录需要尽快补录进清算系统。流程如下：

CME接收到CMF上传的PCS后，会生成Trade Register Report供CMF下载，内容包含每个合约的最终仓位，当日的盯市盈亏，以及对应的当日的成交明细及市值变动。

17点收盘后BizDate会变成下一日

每天16点收盘，不再有成交进来，中途休息一小时后17点开盘，有新的成交进来

在未平仓前如果进来的成交的Tag50在成交表里不存在，则会提示Tag50 is unknow，如果再平仓，因为关联不上，会把不存在的Tag50插入到Tag50表，Type为U，再次等待下次平仓来决定是否需要平

else if (Tag50Type == "U" || Tag50Type == "L")//U--Undecided、L--非法成交

{

continue;

}

SELECT temp.Tag50 ,

temp.Tag50Type

FROM ( SELECT a.Tag50 ,

'' AS Tag50Type ,

b.TAG50 AS joinTag50

FROM TTrdCapRpt a WITH ( NOLOCK )

LEFT JOIN dbo.TTAG50 b ON a.Tag50 = b.TAG50

) temp

WHERE temp.joinTag50 IS NULL

UNION

SELECT TAG50 ,

'' AS Tag50Type

FROM TTAG50

WHERE Tag50Type = 'U'

UNION

SELECT DISTINCT

a.Tag50 ,

'D' AS Tag50Type

FROM TTrdCapRpt a WITH ( NOLOCK )

LEFT JOIN TTAG50 b ON a.Tag50 = b.TAG50

WHERE Tag50Type = 'D';

现在的移历史成交存在的问题是可能有些成交的部分明细在历史，部分明细在当前detail

**1. Mvvm定义**MVVM是Model-View-ViewModel的简写。即模型-视图-视图模型。【模型】指的是后端传递的数据。【视图】指的是所看到的页面。【视图模型】mvvm模式的核心，它是连接view和model的桥梁。它有两个方向：一是将【模型】转化成【视图】，即将后端传递的数据转化成所看到的页面。实现的方式是：数据绑定。二是将【视图】转化成【模型】，即将所看到的页面转化成后端的数据。实现的方式是：DOM 事件监听。这两个方向都实现的，我们称之为数据的双向绑定。总结：在MVVM的框架下视图和模型是不能直接通信的。它们通过ViewModel来通信，ViewModel通常要实现一个observer观察者，当数据发生变化，ViewModel能够监听到数据的这种变化，然后通知到对应的视图做自动更新，而当用户操作视图，ViewModel也能监听到视图的变化，然后通知数据做改动，这实际上就实现了数据的双向绑定。并且MVVM中的View 和 ViewModel可以互相通信。MVVM流程图如下：



2、查询TyTradeDetail，string sql = string.Format("select MAX(InsertDate) from dbo.TyTradeDetail");

DateTime.TryParse(SqlHelper.ExecuteScalar(SqlHelper.GetConnSting(), CommandType.Text, sql).ToString(), out obj);

string timeInsert = obj.ToString("yyyy-MM-dd");, 1=1 and InsertDate > '{0}' and Status = 'N'，能查出当天的持仓，如Position Transfer的记录

4、Trade Enquiry，当前的成交，查询的是TTrdCapRpt表

5、Closeout Enquiry查询的时候TTradeDetail left TTrdCapRpt UNION ALL TTradeDetailHistory LEFT TTrdCapRptHistory

6、Position Enqiury查当前的持仓也可以直接查TTradeDetail为N的记录

7、Trade History Entry查询的是 TTrdCapRpt union all TTrdCapRptHistory

8、Option Execution 查询的时候TTradeDetail表CloseFlag = 'N' and a.MQMSecType = 'OOF'的记录

9、Execution History查询的是TTrdExecutionHistory的记录

10、Allocation查询的是TTrdCapRpt union all TTrdCapRptHistory

发送email依赖于StatementReport的状态，StatementReport状态为2的时候才会发送，而StatementReport的状态改为2的时机是在生成结单转换成PDF的时候

where BizDt=CONVERT(varchar(10), GETDATE(), 23) and Position=1 and CloseOut='Y' and SettlePrice=1 and ClientMargin=1 and Delivery=1，系统最后一次自动触发结单生成是4：50，此时要依赖于交割状态，所以交割定在4:40交割改变交割状态

1. GR的FTP传送和Dropbox的文件，虽然两者是同样的，但是应该分为两条线，分别传送。Dropbox里面的文件备份是为了当FTP没法正常传输的时候作为备份而准备的，所以两者应该互不影响，这样当FTP因为什么原因停止运行的时候，风控部门还是可以去Dropbox里面取文件。这个麻烦您看一下把。
2. Dropbox里面的四个开盘文件加入时间：Account，Position，Price，和Rates；几点几分就好了， 啦； 20200109\_170237\_TRADE; 年月日，钟分秒。

AccountMoneyReport里涉及到保证金

hi guys, 今天QM的5月合约到期，发现系统交割的逻辑问题：

1. 在4点收盘前，系统就自动插入交割平仓单（transType=D），把仓位降下去了。这样生成的PCS, CGM等报表，是交割以后的仓位了。但实际我们需要报的是交割之前的仓位。所以交割逻辑运行应该是收盘结算之后，然后报表生成之前。产生这个问题的原因是以前平仓的时候调交割，用的是日中结算价，16:00前一定会把仓位降下去。所以自动交割从来就没跑过。目前已经改为轮询等到日末结算价进行交割

Sgx发布避免在21点以后，因为有可能会过了触发计算保证金的时间

Apex的成交是通过mq推过来的，sgx因为还不是清算会员，所以上手辉里推到ftp，然后人工去ftp拿到后导入到系统

SELECT \* FROM dbo.TTrdCapRpt A LEFT JOIN dbo.TTAG50 B ON A.Tag50=B.TAG50 WHERE A.BizDt='2020-05-15' AND B.Tag50Type IS NULL 查询的是通过tag50没有关联上的记录，也就是相当于tag50在TTrdCapRpt表存在，但在TTAG50表不存在。没有必要

SELECT \* FROM(SELECT b.\* FROM dbo.TTrdCapRpt A LEFT JOIN dbo.TTAG50 B ON A.Tag50=B.TAG50 WHERE A.BizDt='2020-05-15') t WHERE t.TAG50 IS NULL

一个tag50在那边其实对应就是一个账户登录账号, 香港是这么认为设定的，这样倒是也简单了，tag50=登录账号

但其实大部分公司的tag50和客户登录账号完全没关系。一个人拥有多个账号

Sgx的合约是通过接口推过来的

A50指数期货合约

结算价和汇率在ftp服务端只更新了apex的状态。

服务度从官网下载Sgx结算价动作完成后更新下载是否成功状态

sgx汇率的状态目前未更新，都用apex，sgx的汇率是通过客户单导入

APEX合约是下载的，sgx的合约还是人工导入

作业执行时间：

UpSGXDate：16:01

UpAPEXDate：10:00

APEXHolidayImportData：18:45

DownSettlePriceFromSGXWebSite方法里最后调用apex报表、sgx报表生成方法

1. 对于选中的原单据AlloType肯定要改
2. 插入到历史的AlloType也要改，OrignRtpID也要改。
3. 产生的新单据在当前表OperationType=NULL, OrignRtpID要改
4. 对于Dealer Allo，OrignRtpID和OperationType都改

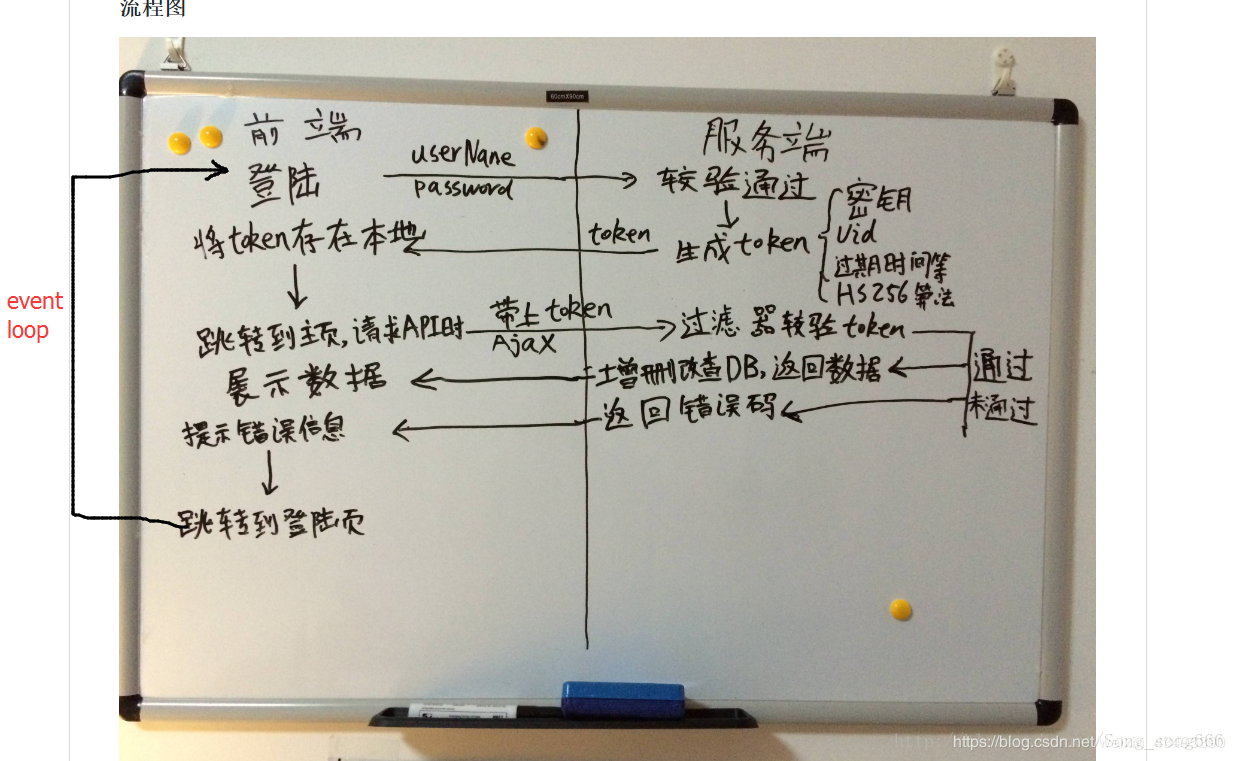
首先聊聊session和cookie，session对象存储在服务器端节点内存中，cookie存储在客户端浏览器中。一般是客户端请求服务器，服务器端生成session对象，将session对象存储在jvm内存中，并且在响应头中放入sessionId响应给客户端，客户端收到响应后，将sessionid存储在本地。当浏览器第二次请求时会将本地cookie中存储的seesionId通过请求头的方式传递给服务器，这样服务器和客户端就能保持会话信息啦！如下图

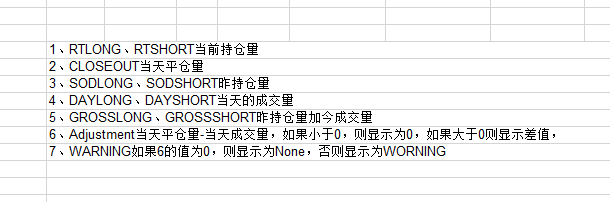
那么为什么会出现分布式session问题呢，为了提高服务器端的负载能力，后台一般将服务器节点做集群，通过ngnix通过轮询的方式转发到目标服务器。打个比方，当浏览器首次访问A服务器生成session对象，然后在访问生成的session对象，如果正好被ngnix转发到了A服务器，那么没问题可以获取到session对象，如果不巧请求被转发到B服务器，由于之前生成的session对象在A服务器，B服务器根本没有生成session对象，很自然访问不到session对象。

知道了问题的存在，那么我们如何解决了，可以通过token令牌代替session解决，或者采用spring-session解决，这里我们选择后一种方式，其实解决思路都是一样的。分布式session不能共享是由于session对象存储在jvm内存中，那么如果共享，答案是将session放入redis中存在，这样不管有多少台应用服务器节点，都能共享redis中存储的session对象。思路很简单，下面给出实现

**解决方案：**

* 使用cookie来完成（很明显这种不安全的操作并不可靠）
* 使用Nginx中的ip绑定策略，同一个ip只能在指定的同一个机器访问（不支持负载均衡）
* 利用数据库同步session（效率不高）
* 使用tomcat内置的session同步（同步可能会产生延迟）
* 使用token代替session





//这个是整个表格都不可编辑

gridView1.OptionsBehavior.Editable = false;

//这个是某列不可编辑，gridColumn1你不想让编辑的列可以是gridColumn1，gridColumn2，gridColumn3 自己根据情况设置就好了

gridColumn1.OptionsColumn.AllowEdit = false;

dev 操作汇总网址: <https://www.cnblogs.com/wordgao/p/4517011.html>

form load之前要先初始化所有Control包括comboBox，而comboBox一初始化就会引发SelectedIndexChanged事件。

InitializeComponent在构造函数里运行的，先有对象，再有对象的LOAD事件，将代码行：  
this.comboBox1.SelectedIndexChanged += new System.EventHandler(this.comboBox1\_SelectedIndexChanged);  
放在LOAD事件处理函数里

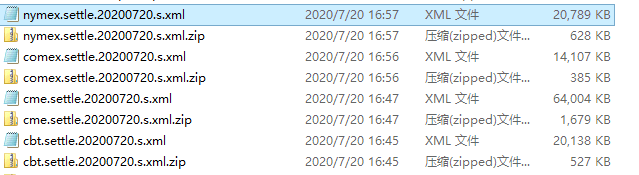
尝试下载cme和NYMEX成交的时间是20:00，下载到本地的时间大概是20:30左右

C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\[%WS[`%5SSEEVX~(PW{J$XD.png

尝试下载cme交易所合约的时间18:10:00，下载到本地的时间大概在18:10-18:20之间



尝试下载结算价的时间是16:30，下载到本地的时间大概在16:45到16:57左右



尝试下载593CSV的时间是20:30，下载到本地的时间大概是20:50

C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\4%AI7T$T9QZR4U6~(HBQTGG.png

尝试下载.s.pa2.zip的时间是18:00，下载到本地的时间大概是18:10左右

C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\RVX$[82XYEPIY4(8J9GSU43.png

尝试下载交易所的汇率的时间是13:00，下载到本地的时间是13:00.

C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\$YPN34M906UZOE[%~JLMWOH.png

尝试下载SgxSpan、HKEXSpan、APEXSpan的时间是13:05，下载到本地的时间是13:05

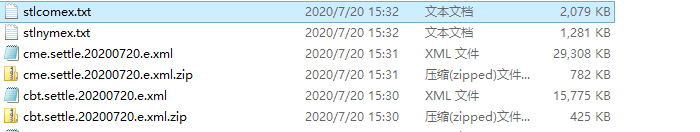


尝试下载LMESpan的时间是16:35，下载到本地的时间是16:35

C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\{__4E8$OLK1_1TED3SBUQR2.png

尝试下载.a.pa2.zip的是时间10:30，下载到本地的时间大概是10:40左右C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\9FFTWT`(01EM2Z1BD7F`FP0.png

尝试下载早期结算价的时间是15:30，下载到本地的时间如下：



每天凌晨3、4点钟生成的C:\Users\david\AppData\Roaming\Tencent\Users\709239679\QQ\WinTemp\RichOle\L[_[HS~AQ~[XCM1T]WYQ8J2.png

是日中有新合约的时候从zip文件中解压出来的

首先是按照这个上手下面的position account进行平仓，如果position account 是speculative account, 就可以自动平仓。 但如果这个position account 是omnibus account（类似于DAHK001),如果我们知道这个position account 下面的business unit, 就可以按照 business unit 进行平仓。如果不知道这个position account 下面的business unit， 那就没办法在这个omnibus position account上进行自动平仓，需要在手动关掉。

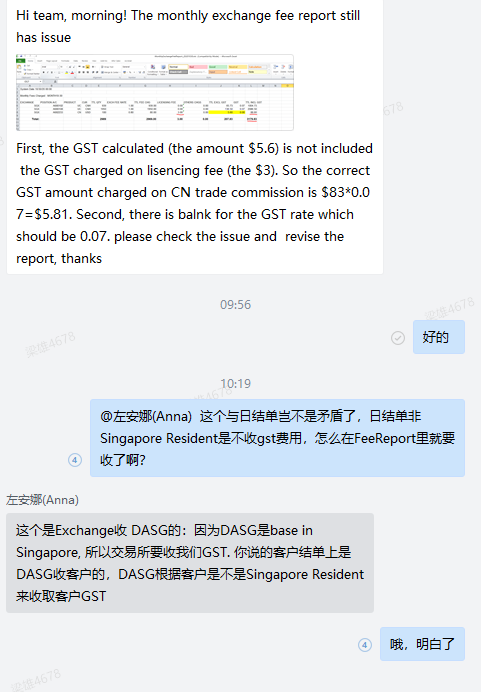
客户日结单费用：broker费用+clientcommission+exchangfee

BrokerReport费用：broker费用+exchangfee+licensefee

费用组里捆绑逻辑

1. 为Y的费用组不能和为N的费用组一起捆绑到Account上
2. 多个为N的费用组里如果有相同的产品，不能一起捆绑到Account上

GC:黄金，CL:原油，NQ：小纳指100，ES:小标普500，HG:铜,SI:白银，YM:小道指



CGM是报告账户所有人的交易标的持仓仓位，即将每个账户实际所有人在当日的每个交易标的上多头与空头成交记录分别累计后进行对冲，再分别加上盘前的多头与空头仓位得最终的持仓信息。

根据2.5 CGM的计算逻辑，在CGM的计算结果基础上，将属于相同TMF的账户所有人的相同交易合约的多空仓位分别进行累计，即得到PCS中Customer账户的计算结果。如果存在House账户下的持仓，直接基于tag50和交易合约的基础上分别计算出多方仓位之和与空方仓位之和，并将记录加入PCS文件。

CME接收到CMF上传的PCS后，会生成Trade Register Report供CMF下载，内容包含每个合约的最终仓位，当日的盯市盈亏，以及对应的当日的成交明细及市值变动。

SMTM Amt = Today’s Settlement Price – yesterday’s Settlement Price

TVAR = Today’s Settlement Price – Open Price

FMTM = SMTM + TVAR

当日的成交明细部分与Trade Capture Report 类似，不同处如下：

MsgEvtSrc = ‘REG’

增加AgrsrInd：Aggressor Indicator (N = Order initiator is passive Y = Order initiator is aggressor)

增加AllocInd：Allocation Indicator (0 = Allocation not required 1 = Allocation required (give-up trade) allocation information not provided (incomplete)

所有在该合约下的成交记录中的TVAR amount相加之和等于PosRpt中的 TVAR Amount.

CMF解析接收到的Trade Register Report，与清算系统的成交信息比对，如果存在差异，则在对账日志中记录不一致的成交信息。清算人员可以在清算系统中查询相关信息，并根据TradeID查询交易系统的成交记录，比对后决定是否导入清算系统。并在第二天开盘前登录CME网站，调整SOD仓位。

清算会员完成基础信息的维护，如下载CME关于活跃合约的文件并保存，维护清算客户提供的账户所有人信息与Tag50之间的对应关系等。

2. 清算会员盘中通过MQ连接实时接收CME传送的成交信息，保存至清算系统中的记账数据表，据此生成CME要求的PCS/CGM文件并上传至CME规定的FTP目录下；

3.清算会员盘后下载CME检查PCS/CGM文件后生成的CGM Margin Summary, Position Reconciliation, trade register report以及结算价文件，完成与清算客户之间的持仓对账，保证金的计算，向清算客户发送对账单，并完成与清算客户和CME之间的保证金划转。

4. 清算会员生成large trader report分别发送给CME和CFTC，CFTC会检查CME和CMF发的report是否一致以便于发现交易问题。Note:在清算系统开发的第一阶段不考虑期权的行权。

CGM只上传Customer 账户的仓位信息。Tag50为Dealer但是在清算时仍没有分配仓位的成交记录，在CGM中按照交易者的Tag50上报,在PCS中将仓位直接累计到对应的交易合约后上报。 Tag50为undecided但是在清算时仍没有确认Tag50类型的成交记录，如果其成交账户是Customer类型的话，在CGM中按照TMF的联系信息上报,仓位信息不可以跟其他的Tag50对冲。在PCS中将仓位直接累计到对应的交易合约后上报。

根据2.5 CGM的计算逻辑，在CGM的计算结果基础上，将属于相同TMF的账户所有人的相同交易合约的多空仓位分别进行累计，即得到PCS中Customer账户的计算结果。如果存在House账户下的持仓，直接基于tag50和交易合约的基础上分别计算出多方仓位之和与空方仓位之和，并将记录加入PCS文件。

如果多方仓位为0，但是当日有成交记录，则最终的仓位为0，需要在PCS中上报，以便于平掉交易所的头寸。

在CGM计算结果生成之后，将每个TMF的同一交易合约的多头仓位累计起来就得到 PCS中的多头仓位信息，从而生成PCS文件并保存在数据库。同时记录生成时间等信息备查。

CME的仓位系统维护着每个仓位账户的全部交易标的未平仓头寸的数据。CME根据上报的PCS文件生成Trade Register Report文件，盘中和盘后各生成一次，文件包含仓位信息和成交信息，以FIXML的格式传输。如果持有未平仓位的交易标的当日没有交易，文件中就不包含相关的成交信息。CMF可以利用该文件在与后台系统的仓位，金额和成交信息进行对账，并找出不一致的仓位或成交信息后进行调整，保持清算系统的数据与CME的数据保持一致。

仓位信息包括：仓位账户信息，交易标的信息，仓位数量，仓位变动导致的金额变化。Trade Register report的格式说明请参考2.10。清算对账系统首先需要解析Trade Register Report, 按照PosRpt 中的Party role=1 对应的TMF ID, Party Role =38 对应的SubID，type，business date, Instrument ID, Security Type和MMY ,得到对应的 Qty Typ=FIN的long和short对应的仓位数量，与同样的business date, Party role=1 对应的TMF ID, Party Role =38 对应的SubID，type， Instrument ID, Security Type和MMY下的long 仓位进行比较，如果一致，则比较下一个交易合约的仓位；如果不一致，则提取该合约对应的成交明细(ReqID)，与本地的成交明细(ReqID)进行比较，找出不一致的成交信息。可能会发生CME有成交，但是清算公司没有该笔成交的情况。

比对结果生成文件提供给清算人员进行后续处理。若信息不一致，比对结果记录：交易日, 合约，成交记录编号，比对结果等信息，供清算公司详查。比对过程由后台程序自动完成，当Trade Register Report文件下载后就触发比对过程。用户可以在清算系统前台进行查询和手动触发比对过程。

GenerateDailyConcentrationReport方法计算保证金生成DailyConcentrationReport.csv文件从来就没有成功过。

风控基础报表持仓sql语句需要优化，union的两个sql语句可以合并成一个

DailyLiquidityRport.csv是生成到d：\\ RiskReport后再拷贝到GRBackup

DailyConcentrationReport.csv是生成到d：\\ RiskReport后再拷贝到GRBackup

保证金计算的时候不需要计算当天要交割的仓位，tytradetail表里settledate为当天的数据过滤掉。因为

带有output的insert语句.

@@identity只能返回当前会话最后生产的标识列.  如果一次性插入多条语句的话. 需要返回这些自动生产的标识列. 那么outpu就派上用场了.

declare @temp table(k int, v nvarchar(200))

insert into t1(datacol)

output inserted.keycol, inserted.datacol

into @temp

select \* from @temp  
  
--------------------

insert into DownLog  
output  
newid(),inserted.id,inserted.ID,inserted.ID, inserted.ID,getdate() into MatchOrderLog  
values('123444','aaaaaa.txt',getdate(),1,1,1)

select  \* from  MatchOrderLog

带有output的delete语句和update语句也大同小异.

delete语句能操作deleted临时表. 而update的本质是delete和insert一次能操作inserted和updated两张表.

update loanee set MatchOrder=@MatchOrder,UpdateTime=getdate()  
  
output  
newid(),deleted.id,deleted.ApplicationID,deleted.MatchOrder, inserted.MatchOrder,getdate() into MatchOrderLog  
where ApplicationID=@ApplicationID and DeleteState=0 and LoaneeType=0

============================

deleted.参数名  是更改前的参数值

inserted.参数名  是更改或者插入后的参数值

使用 格式    ：    insert into  表名      output    inserted.参数名,inserted.参数名2    into  记录表     values(表值,表值2,表值3...)

 　　　　　　　　 update   表名  set  参数名=参数值,参数名2=参数值2 ...     output    inserted.参数名,inserted.参数名2    into  记录表     where  条件...

sqlserver发布和订阅是，如果没有权限有可能是代理账户没有权限，需要到服务里设置权限

ManulaAloccation把昨天已经平掉的的仓位重新开出来的话会存在平盈在当天变成浮盈又出来的问题

TSettlePrice 存放的是盘中的预结算价，然后收盘会会用盘后结算价替换掉盘中结算价。

TQuote是每30秒保存下持仓和成交未N的合约的行情，如果行情中没有，就用TPreSetlPx

的结算价代替，TPreSetlPx保存了盘后当天的所有结算价

传给GR的也是成交为N的和持仓合约从行情中拿，如果拿不到则从TQuote拿

Mq写成交的时候每次都初始化了tradelog，不需要，因为在formload里已经初始化过一次了resetLogger

TAllSettlePrice 变成了盘后当天所有结算价，然后再转点后会把TAllSettlePrice拷贝到TPreSetlPx作为前一天结算价

TAllSettlePrice这个表只在手动添加结算价和下载cme结算价，从span里解析结算价的时候用到了，转点后把TAllSettlePrice这个表的数据全部拷贝到TPreSetlPx这个表作为前结算价

TPreSetlPx这个表在计算CalcOptionsImpliedVolatility IV的时候用到，在计算PayCollect里用到，以及更新Tuote表里用到

Cme 至少有一个为非清算日但不是cme清算日时，时会走30.7盘后结算，此时tybanlance和tcashbalance没有问题，因为没有成交会拿前一天的过来，但是cme因为没有结算价文件，保证金会算不出来，由于cmepointbanlance表会取前一天保证金eod，出错。

我有一个Winform应用程序，重载了方法“WndProc”代码如下：

const int WM\_MOUSEMOVE = 0X0200;

protected override void WndProc(ref Message m)

{

switch (m.Msg)

{

case WM\_MOUSEMOVE:

Debug.Write(string.Format("CurrentDate:{0}", DateTime.Now) + Environment.NewLine);

break;

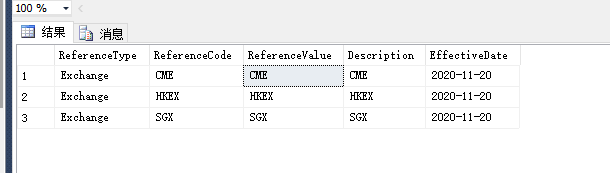
}

base.WndProc(ref m);

}

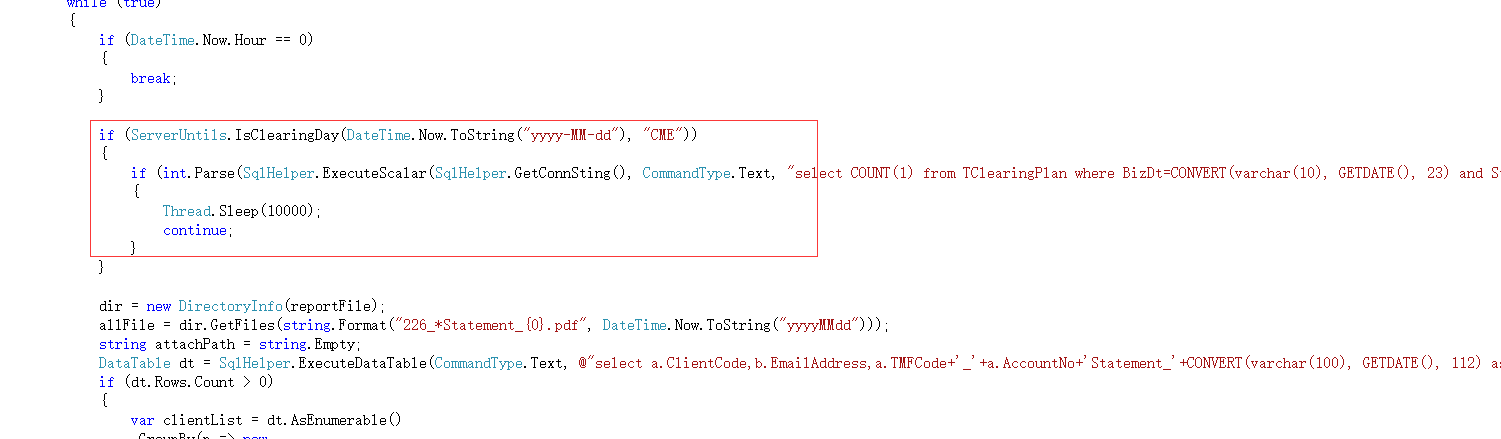
我想捕获鼠标在应用程序上移动时的消息，但是当我把鼠标放置在应用程序上面，然后就不动鼠标，但是在vs.net中的Output中就看到一直在输出信息！太奇怪了。我并没有移动鼠标，为什么会捕获到这个消息了呢？

请教高人，我只想当鼠标移动的时候捕获到这个消息，应该怎么处理？谢谢了



加这个表的目的是为了防止在客户端节假日配置里加一个还没有任何交易的交易所配置影响程序。

收盘之前生成报表不进行pdf 转换

发送结单是cme清算日需要等状态，30.7清算日因为是最后，不需要等

//customer账户下，按Trader加入OnimbusAccountList

//house账户下，相同AccountNo下，不同的Trader累计加入OnimbusAccountList

DealerAllocation操作目前不是很严谨，因为加入做完dealer后操作人员不进行平仓，dealer单据改为正常的tag50，并且状态为N，tradeDetail的仓位被删除，这个时候生成cgm报表会出现dealer的仓位在tytradedetail里，并且在成交表里也能查出状态为N的dealer单据，union后会重复。如果此时要解决这个问题，只需要在DealerAllocation也删除tytradedetail的仓位

**一、什么是期权行权**

期权是交易双方关于未来买卖权利达成的合约。行权是指期权合约权利方要求义务方按照约定时间、价格和方式履行期权约定的义务，期权的义务方则必须在期权权利方行权时提供相应的义务。

如果你手中有一张上汽集团的标的证券的认购期权上汽集团购4月1300，行权日是4月23日。行权价格是13元。就是说，到4月23日这天，你有资格用13元/股的价格买该股票5000股。

如果到了这天，该股的市场价是15元，别人买5000股要花75000元，而王先生这天则可以用65000元就买5000股，假如当初王先生买入期权时期权权利金为1.2元，那么王先生一共花费了71000元，共节省4000元。王先生当然合算了。如果王先生真买，这个行为就叫行权。

但是如果到了这天，该股的市场价是12元，王先生当然不会用15元/股的价格买，那么王先生手里的一张认购期权就是废纸。王先生肯定选择放弃行权了。

需要指出的是，上交所推出的个股期权是欧式期权，意味着在只能在规定日期，即期权到期日（合约到期月份的第四个星期三）以事先约定好的价格向期权义务方行权。

**二、行权交收**

**1.行权日期**

投资者在面临期权行权时，有几个重要的日期需要弄清楚，包括合约最后交易日、到期日、行权日、交收日。

**1）合约最后交易日（T日）**

合约最后交易日是指到期期权合约最后可以进行场内交易的时间。最后交易日设为每个合约到期月份的第四个星期三（遇法定节假日顺延）。

**2）合约到期日（T日）**

合约最后到期日是指合约的存续期截止的日期，也是期权合约必须履行的最后日期。一般情况下，合约到期日就是最后交易日。

**3）合约行权日（T日）**

合约行权日是指投资者可以进行行权申报的日期。上交所将合约行权日与最后交易日设置为同一天。

**4）合约交收日（T+1日）**

合约交收日是指中国结算公司将权利方行权后得来的标的证券过户给投资者的日期，合约交收日为行权日的下一个交易日。投资者只有在合约交收日的下一个工作日（T+2），才能卖出行权所得的标的证券。

在初期方案中，上交所为了考虑到股指期货投资者套利套保的需要，将期权行权日设置为合约到期月份的第三个星期五（T日）。期权是T+1日交割，下周二T+2日才能交易卖出，导致时间跨度过长。由于周末是各类信息发布的集中期，为避免投资者承担不必要的风险，上交所将行权日定为合约到期月份的第四个星期三。这样，投资者可以在星期五卖出行权得来的股票。也就是说投资者最快可以在期权到期月份第四个星期五将行权得来的股票卖掉。

**2.行权时间**

期权到期行权日暨最后交易日，期权行权指令提交时间为上午9：30－11：30，下午13：00－15：30。

相比于股票和期权交易时间，上交所为了使在最后时刻买入期权的投资者可以行权，将期权行权指令提交时间延长30分钟。投资者可以最晚在当天的15点30提交行权申报指令。在此期间，投资者如果想撤销之前提交的行权指令可以随时撤单，或重新申报。同时，投资者也可以重复累次提交行权委托。在行权日下午15点30分后，投资者将无法撤销行权指令。

行权日买入的期权，投资者当日可以行权。

**3.行权指派**

行权指派就是中国结算公司对所有期权权利方的有效行权申报，对义务方进行行权配对。为了保证公平性，交易所是按比例指派给期权义务方交付标的证券或资金。被指派的投资者必须准备足额的标的证券或资金，完成行权交收。

**4.行权交收**

行权交收是指在行权交收日，中国结算公司将被指派义务方的标的证券和资金划付给期权的权利方。

个股期权采用实物交割的方式，即在行权时，双方进行现金与标的证券的交收。对于认购期权，权利方根据行权价将对应的资金交给义务方，义务方将对应标的证券交给权利方；对于认沽期权，权利方将股票交给义务方，义务方根据行权价将对应的资金交给权利方。

**5.关于自动行权**

投资者担心可能在行权日当天未及时行权造成不必要的损失。一般来说，证券公司会提供关于自动行权的服务，帮助投资者对符合条件的实值期权自动行权

SQL Server推荐使用 SET 而不是 SELECT 对变量进行赋值。  
当表达式返回一个值并对一个变量进行赋值时，推荐使用 SET 方法。  
下表列出 SET 与 SELECT 的区别。请特别注意红色部分。

|  |  |  |
| --- | --- | --- |
|  | **set** | **select** |
| 同时对多个变量同时赋值 | 不支持 | 支持 |
| 表达式返回多个值时 | 出错 | 将返回的最后一个值赋给变量 |
| 表达式未返回值 | 变量被赋null值 | 变量保持原值,但若有标量函数返回值，则为null |

下面以具体示例来说明问题：  
create table chinadba1(  
userid int ,  
addr varchar(128)   
)  
go  
insert into chinadba1(userid,addr) values(1,'addr1')  
insert into chinadba1(userid,addr) values(2,'addr2')  
insert into chinadba1(userid,addr) values(3,'addr3')  
go  
  
表达式返回多个值时，使用 SET 赋值   
declare @addr varchar(128)  
set @addr = (select addr from chinadba1)  
/\*  
--出错信息为  
服务器: 消息 512，级别 16，状态 1，行 2  
子查询返回的值多于一个。当子查询跟随在 =、!=、<、<=、>、>= 之后，或子查询用作表达式时，这种情况是不允许的。  
\*/  
go  
表达式返回多个值时，使用 SELECT 赋值 declare @addr varchar(128)  
select @addr = addr from chinadba1  
print @addr --结果集中最后一个 addr 列的值  
--结果: addr3   
go  
  
表达式未返回值时，使用 SET 赋值 declare @addr varchar(128)  
set @addr = '初始值'  
set @addr = (select addr from chinadba1 where userid = 4 )  
print @addr --null值   
go  
  
表达式未返回值时，使用 SELECT 赋值 declare @addr varchar(128)  
set @addr = '初始值'  
select @addr = addr from chinadba1 where userid = 4  
print @addr --保持原值  
go  
  
需要注意的是，SELECT 也可以将标量子查询的值赋给变量，如果标量子查询不返回值，则变量被置为 null 值。  
此时与 使用 SET 赋值是完全相同的  
对标量子查询的概念大家应该都觉得陌生，举个例子就能说明  
declare @addr varchar(128)  
set @addr = '初始值'  
--select addr from chinadba1 where userid = 4 为标量子查询语句  
select @addr = (select addr from chinadba1 where userid = 4)   
print @addr --null值  
go

尝试下载cme和NYMEX成交的时间是20:00，下载到本地的时间大概是20:30左右，会写到TTrdRegRpt，查这个表可以查出当天交易的量，收盘后的持仓，昨持仓等等。

尝试下载593CSV的时间是20:30，下载到本地的时间大概是20:50，会写到TTradeRegistSum

# sql中的不等于<>，NULL需要单独判断

[向着远方奔跑](https://www.jianshu.com/u/c4cbdfd7682d" \t "_blank)关注

2020.05.06 16:05:08字数 96阅读 842

sql中，如果字段类型允许为NULL，则当有字段数据为NULL时，sql语句做不等于筛选判断时要注意

**null只能通过is null或者is not null来判断**

例：  
select \* from bl\_ip\_dt where amount <> 800，这条语句查不出amount等于null 的记录  
select \* from bl\_ip\_dt where amount <> 800 or amount is null 是可以的

PFTP只报今仓，CFTC今仓和昨仓都报，但昨仓以0上报，列出产品就行

for (int i = stuList.Count-1; i>=0; i--)

{

if (stuList[i].Name == "Tang")

stuList.Remove(stuList[i]);

}

匿名类中的属性时read only的

ICE 收到的成交時間比cme系統時間晚5個小時

Ice的交易時間段21:00-14:20,收盤時間比cme早

ice交易時間段：<https://www.theice.com/products/254>

日中和日末的平仓放到了paycollect报表中调用

ModelConvertHelper<TradeDetail>.ConvertToModel(ds.Tables[1]).ToList();

BaseInterfaceLogic.SetComboBoxData(cmbPrice, result.dt, "ValueMember", "DisplayMember");

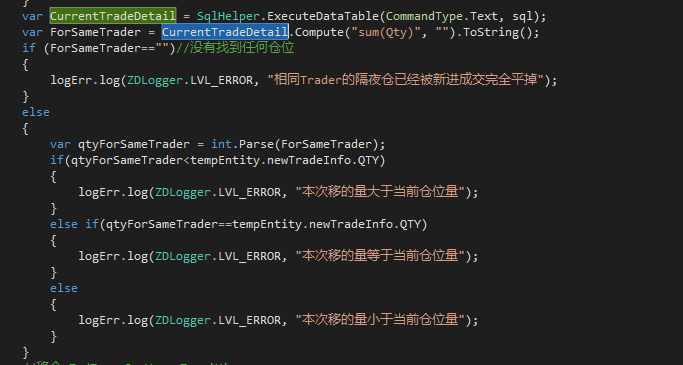
BaseInterfaceLogic.SetComboBoxSelectedValue(cmbPrice, result.dt.Rows[0]["ValueMember"].ToString());

var ListRow = BaseInterfaceLogic.GetSelecteIds(gvMarginCutGroupList, "CreditId", BaseBusinessLogic.SelectedColumn, true);

((Item)combSegType.SelectedItem).value;

发现一个bug，当有移仓进来时，如果该进入的交易为前一个交易日（比如今天6.17，而移仓进来的是06.16的交易），transfer position里显示的前一日收盘仓位没有同步更新，导致找不到移仓进来的仓位，没法进行进一步tag50调整。  
因此希望你们升级一下，仅针对position transfer里的前一日持仓明细，在第二天日中有交易日期为前一日的仓位进入时，根据我们的平仓原则在position transfer里的持仓里进行更新，体现出仓位变动。  
比如：06.16收盘时 tag50 101 GC2107 仓位为 10X5  
06.17 有客户移仓2手GC2107 long进来，tag50也为 101，并且交易日期为06.16. 则此时position transfer里 tag50 101 下的持仓变为12X5。  
这个不影响我们已经出过的06.16的结算报表，06.17结算时仓位也按本来逻辑进行即可。只需要对position transfer里的仓位做出调整

hi梁雄，  
  
还是这样吧，就是再position transfer界面，搜索出来的就还是昨天收盘的持仓记录。  
至于实际仓位和隔夜持仓不符合的问题，我们就在实际操作的时候来解决了。例如我们不全部处理，只按照实际仓位挑着处理



1. 对于所有Filter来说，如果阻止了请求:即对Response进行了赋值，则后续的Filter不再执行。

on where 是指这前后两个表的连接条件只有on后面的一个，然后对连接好的结果，去执行where条件查询，where后面的列可以不是连接相关的列，where后面出现的是某一个列的条件，是对行的筛选条件，不是列与列的连接关系。where是对A或B筛选之后的数据再去按照ON的条件去连接

string siteName = "零度编程"; string welcome = $"{siteName}，欢迎您!";



linq group by 会区分大小写

SQL 去除小数后面的0：

select cast(2.5000000000000 as real)

select cast(2 as real)

select cast(2.00000 as real)

UPDATE a SET a.RadixPoint=b.RadixPoint FROM dbo.TProduct11 b, dbo.TProduct a WHERE a.MQMExchangeCode=b.MQMExchangeCode AND a.ClearProductCode=b.ClearProductCode AND a.MQMSecType=b.MQMSecType

期货的对盘价和挂盘价：

如果你想主动成交 那么成交的价格为对手价 如果你想挂单成交 那么你挂单的价格就为挂单价 可以这么理解 你为了快速成交 那么你就要去对手那里主动交易 这个价格是对手价 而你如果想挂一单试一试 能成交就成交 不能成交就拉倒 那么你挂单的价格就是挂单价

**市价：**委托下单界面中，定单类型选择**市价**，即委托价格为空，以当前能够成交的最优价格成交；市价下不可设置委托价格。

平仓过程中匿名类中属性是readonly，无法进行设值，重新构造一个类替换之

今天单位一ASP.NET网站，里面有个功能是导出数据，发现一导出就报错，报错内容是：SQL  Server 阻止了对组件 'Ad Hoc Distributed Queries' 的  STATEMENT'OpenRowset/OpenDatasource'  的访问，因为此组件已作为此服务器安全配置的一部分而被关闭。系统管理员可以通过使用 sp\_configure 启用 'Ad Hoc  Distributed Queries'。有关启用 'Ad Hoc Distributed Queries' 的详细信息，请参阅 SQL  Server 联机丛书中的 "外围应用配置器"。

看错误提示就知道是因为SQL Server的Ad Hoc Distributed Queries组件被禁用了，这里我用的SQL Server版本是2005，只需要开启Ad Hoc Distributed Queries就可以了，方法如下：

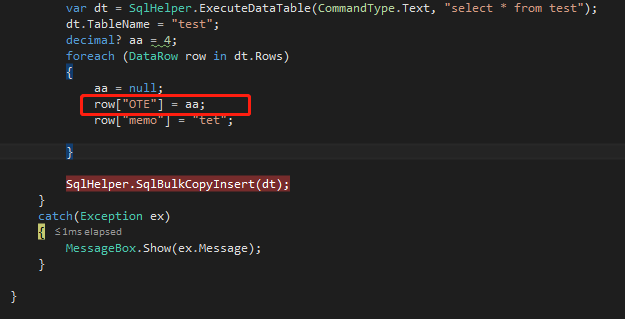
1.开启Ad Hoc Distributed Queries组件，在sql查询编辑器中执行如下语句：

|  |  |
| --- | --- |
| 1  2  3  4 | exec sp\_configure 'show advanced options',1  reconfigure  exec sp\_configure 'Ad Hoc Distributed Queries',1  reconfigure |

2.关闭Ad Hoc Distributed Queries组件，在sql查询编辑器中执行如下语句：

|  |  |
| --- | --- |
| 1  2  3  4 | exec sp\_configure 'Ad Hoc Distributed Queries',0  reconfigure  exec sp\_configure 'show advanced options',0  reconfigure |

var ss= sss.Sum(x => x.Field<decimal?>("OTE"));这种写法是为了防止数据库有NULL值时不会报错，结果要么为0，要么为数值，不可能为null



这种写法会报不能将 Column“OTE”设置为 null。请改用 DBNull。

SELECT MAX(time) FROM test，当test没有行的时候这个语句返回一行，值为null

表TRiskAccountSummary的BeginningEquity相当于结单的NewCashbalance，表TRealTimeRiskDtl

的BeginningEquity相当于结单的Total Equity

TRealTimeRiskDtl表的OTE为公式里今减去昨

Totalpl等于今ote-昨ote+今平盈

计算保证金的方法有15个引用，3个地方调用：

1. 实时计算保证金调用，此时margindetail会计算
2. 计算权益的时候日中和收盘时调用
3. 18:30收盘时调用

TRealTimeRiskDtl表的TotalPL为今OTE-昨OTE加平盈

TRealTimeRiskDtl表的detal值，期货时等于Net，期权是Detal公式计算得出

程序在美国时间1:30移历史成交

作业在美国时间1:40备份TradeDetail

作业在美国时间1:45备份数据库

作业美国时间21:30计算销售人员基础数据

程序在美国时间21:40基于销售人员数据出佣金报表（周五）

作业4:00会删除非cme的结算价

作业在周日1:10进行索引维护

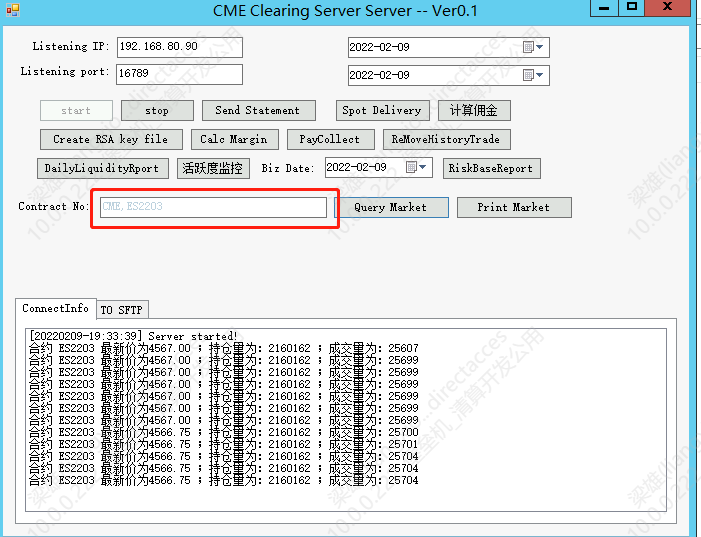
作业在0:0:01秒生成下个清算日期

作业在0:0:05秒把当前所有结算价存入昨结算价表格

select \* from TReferenceData where ReferenceType='WithMQPassMarketData'等于1时不往c:\Users\Administrator\Direct Access USA Dropbox\Risk\GR Backup写文件，否则就会往c:\Users\Administrator\Direct Access USA Dropbox\Risk\GR Backup和feed里写文件

但DailyLiquidityRport.csv 和PayCollect.csv会在 10:35、16:10、21:00GR Backup中BOD生成

获取行情：



linq group by 会区分大小写

Cme旗下的4个交易所的合约和产品都是下载文件解析的

其他交易所的合约和产品需要自己导入

Cme旗下的4个交易所的结算价是通过文件解析的

SGX的结算价是从span里面解析的

HKEX有下载span文件，但是因为不知道解析规则，没有解析结算价，现已从官网通过python下载到

LME结算价是从span里解析的，但解析的最后一位带有币种标识，如AHD,AHY

APEX有下载span文件，但是因为不知道解析规则，没有解析结算价

ICE的span文件是网站下载的，结算价从span中解析

Eurex结算价处于pending状态，保证金通过python调用api已计算

实时查询成交表里新进的合约，如果在合约表里没有，自动从downsftp文件下下找到相应的合约文件解压出来后调用导入合约方法，调用存储过程，有存在就不做任何事，不存在就insert合约和产品（产品存在会更新）

30.7vs seg相关

出入金  
交易所

Tclearclient的R和NR

MarginInfo

Tybalance

Tcashbalance

新客户影响

港交所结算价，什么时候开始就出来了

欧交所结算价

欧交所保证金

tspancode,TLargeTraderFormat

凌晨4点自动删除结算价作业

新客户的上手

新客户的清算日期确定

if (trade[i].StrikePx.HasValue)

dr["StrikePx"] = trade[i].StrikePx;

else

dr["StrikePx"] = DBNull.Value;

港交所'MHI','HSI','HHI','MSI'四个产品的合约和LME的合约，是精确到日的,如AH20220629

保证金double很有可能是product表里的globoxproduct重复了

CAST(a.StrikePx AS REAL) AS StrikePx ,去除小数点后面多余的0

var GroupByRegAndRate = dtMulticurrency.AsEnumerable().GroupBy(o => new { Reg = o.Field<string>("RegCode") }).Select(group => new { key = group.Key, Data = group }).OrderBy(p => p.key.Reg).ToList();

SELECT AccountNo + ClearProductCode + MMY

+ MQMSecType + ISNULL(PutCall, '')

+ ISNULL(dbo.ClearZero(StrikePx), '') AS pk

FROM TyBalance

WHERE BizDate = '{0}'

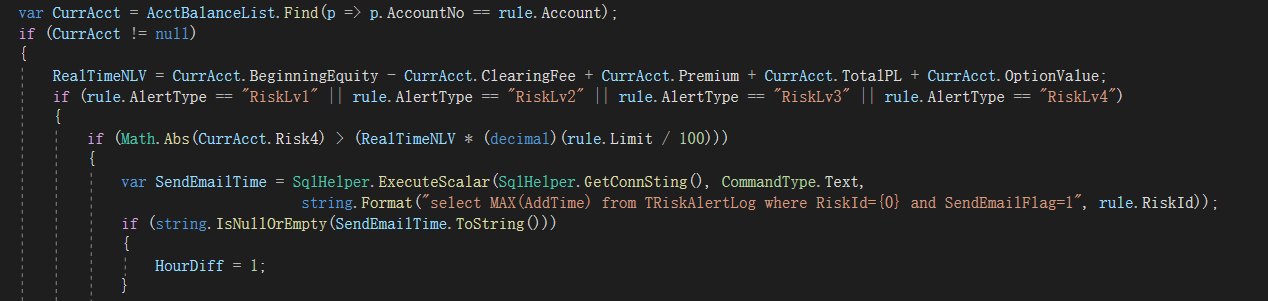
TTradeRegistSum 表的数据来源于593csv，CMEPOINTBALANCE报表用到

TTrdRegRpt表的数据来源于FIXML-226\_EOD\_CME-{0}.xml.zip，内容包含每个合约的最终仓位，当日的盯市盈亏，以及对应的当日的成交明细及市值变动,记录了今成交手数、昨持仓手数、今持仓手数、FMTM(Final Mark-to-Market Amount ),SMTM( Start-ofDay Mark-to-Market Amount ),CMEPOINTBALANCE报表用的到，

TTrdRegDiffenence表的数据来源于tybanlance和TTrdRegRpt对比得到，然后用于生成CMEPOINTBALANCE报表

以上代码按照每一个comfirm的通道维护一个集合，每发送一条数据，集合增加一个元素，每异步响应一条ack或者nack的数据，集合删除一条。SortedSet是一个有序的集合，它的有序是值大小的有序，不是插入时间的有序。JDK中waitForConfirms()方法也是使用了SortedSet集合

RISK4:=SettlePirce\*movement\*Multiplier\*NetQty



Execise行权：

1. 只有Long方向期权的才可以行权
2. 插入期权对应期货的成交（18），状态为N,看涨期权（call），side为1，看跌期权（put），side为2。
3. 成交价格为行权价格
4. 反向插入期权成交（5），状态为Y，成交价格为原期权的开仓价
5. 原期权和反向插入的期权模拟平仓，平盈为0

assign行权：

1. 只有Short方向的期权才可以行权
2. 插入期权对应期货的成交（18），状态为N,看涨期权（call），side为2，看跌期权（put），side为1
3. 成交价格为行权价格
4. 反向插入期权成交（5），状态为Y，成交价格为原期权的开仓价
5. 原期权和反向插入的期权模拟平仓，平盈为0

ExpireITM行权

1. Long和Short方向都可以行权
2. 原仓位Side为long时，看涨期权（call），side为1，看跌期权（put），side为2，原仓Side为short时，看涨期权（call），side为2，看跌期权（put），side为1。插入期权对应期货的成交（18），状态为N。
3. 插入期货成价格为行权价
4. 反向插入期权成交（5），状态为Y，成交价格为原期权的开仓价
5. 原期权和反向插入的期权模拟平仓，平盈为0

ContraITM、ExpireOTM行权

1. Long和Short方向都可以行权
2. 反向插入期权成交（4），状态为Y，成交价格为原期权的开仓价
3. 原期权和反向插入的期权模拟平仓，平盈为0

ContraOTM行权

1. Long和Short方向都可以行权
2. 原仓位Side为long时，看涨期权（call），side为1，看跌期权（put），side为2，原仓Side为short时，看涨期权（call），side为2，看跌期权（put），side为1。插入期权对应期货的成交（18），状态为N。
3. 插入期货成价格为行权价
4. 反向插入期权成交（5），状态为Y，成交价格为原期权的开仓价
5. 原期权和反向插入的期权模拟平仓，平盈为0



调用打包的python .exe

Process po = new Process();

string exepath = @"D:\GitHub\PythonTest\dist\Parsepages.exe";

po.StartInfo.FileName = exepath;

po.StartInfo.UseShellExecute = false;

po.StartInfo.RedirectStandardOutput = true;

po.StartInfo.RedirectStandardInput = true;

po.StartInfo.RedirectStandardError = true;

po.StartInfo.CreateNoWindow = true;

po.Start();

var response = po.StandardOutput.ReadToEnd();

不包含include的索引：

会增加一次通过主键索引的回表，增加逻辑读次数。

包含include的索引：

不需要通过主键索引回表就能找到数据，减少了逻辑读次数，但是增加了索引的空间占用。

建议对select查询字段少的可以用包含include的索引，占用索引空间不大，减少了逻辑读，提升了查询效率。

举例：

create index idx\_A on A(biz\_date) include(name,id)

这个索引中，include包含了name，id 两个字段。

（3）那么这种索引和直接指定 biz\_date，name，id的索引，有什么区别呢？

区别在于 include中的列，是不排序的，一般可以把select中的列，放到include中，好处是索引的开销相对小了。

如果是在where中的列，就不适合放到include子句中，因为这样不利于快速查找过滤数据。

PayCollect：

**Margin Without：**

E3=Customer账户昨收盘Margin\*昨到USD的汇率

E14=House账户昨收盘Margin\*昨到USD的汇率

F3=Customer账户昨收盘后的持仓用span算出的Margin

F14=House昨收盘后的持仓用span算出的Margin

**Option Value：**

E4=Customer账户昨收盘后的OptionValue\*到USD的汇率

E15=House账户昨收盘后的OptionValue\*到USD的 汇率

F4=Customer账户昨收盘加今成交算出的Margin

F15=House账户昨收盘加今成交算出的Margin

**Margin：**

E5=E3-E4

E16=E14-15

**Option Premium**

E8=Customer账户昨收盘后的Premium

E19=House账户昨收盘后的Premium

F8=Customer账户当前的Premium

**Margin（有nl，ns，long，short节点）**

E9=Customer账户昨收盘后的Margin

E20=House账户昨收盘后的Margin

**Unrealized Profit & Loss**

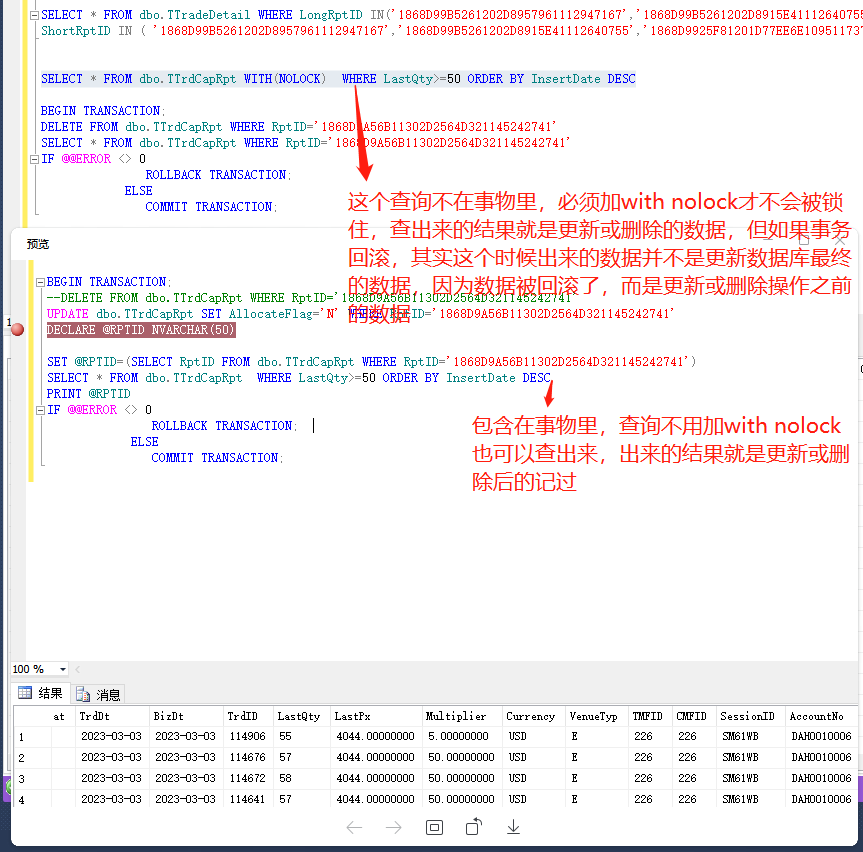
E10=Customer账户昨收盘后的浮盈

E21=House账户昨收盘后的浮盈

**Realized Profit & Loss**

E11=Customer账户昨收盘后的平盈

E22=House账户昨收盘的平盈







目前cme的品种是从文件里解析后然后跟库里比对，如果没有就插入

目前cme的合约是只解析当前成交表里已经有的合约

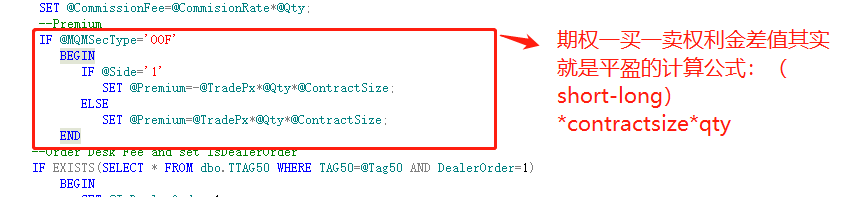
关于MaxRisk的计算：

Master账户是根据合约对冲后计算出来的值再在前面加了负号

子账户是各自算Net的值，Net为负就直接为负数，为正就绝对值后加负号。

所以会发现Master账户的MaxRisk与子账户各自的MaxRisk求和后对不上。

因为当子账户有Net为正数是，写到数据库其实是负数，就这个原因会导致与Master对冲后的数据对不上。



在触发器里如果trdtyp=2，那么就把feetyp设置为1,表示EFP收费类型。

数据库默认FeeTyp是为0

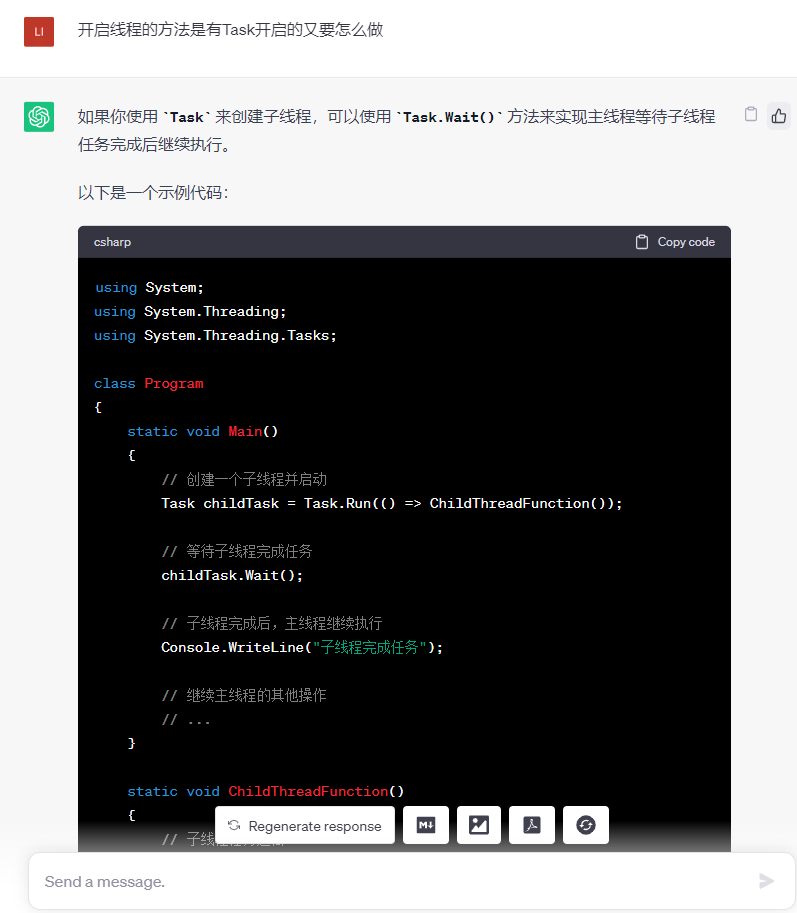
Sql取反操作：SELECT \* FROM dbo.TAccount WHERE NOT (AccountType='N' AND MemberShip='I')



在上述代码中，主线程通过创建一个子线程并调用**Join**方法来等待子线程完成任务。**Join**方法会使主线程暂停，直到子线程执行完毕。

注意，**Join**方法还可以接受一个超时参数，可以设置主线程等待子线程的最大时间。如果超过该时间子线程还没有完成，主线程会继续执行。

希望这可以帮助你实现主线程等待子线程任务完成后继续执行。



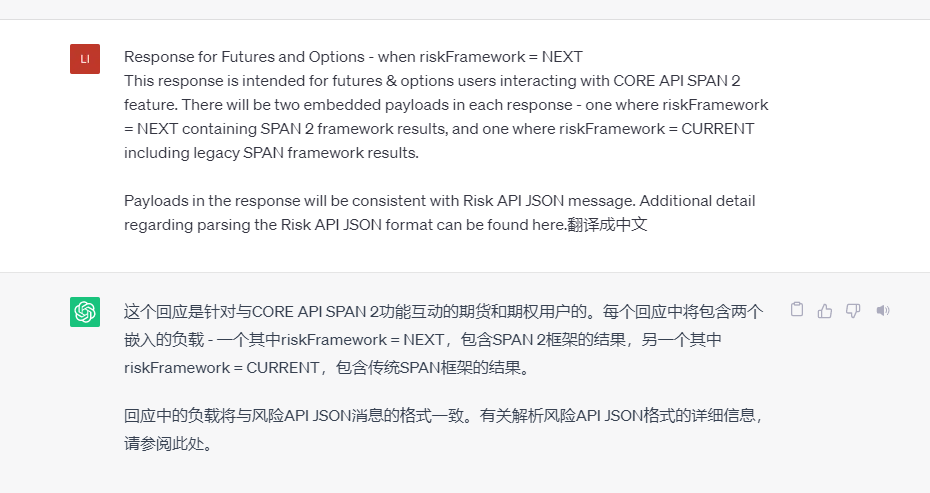
在上述代码中，主线程使用 **Task.Run()** 方法创建并启动子线程。然后，主线程调用 **Wait()** 方法等待子线程完成任务。**Wait()** 方法会使主线程暂停，直到子线程执行完毕。

同样地，**Wait()** 方法也可以接受一个超时参数，可以设置主线程等待子线程的最大时间。如果超过该时间子线程还没有完成，主线程会继续执行。

希望这可以帮助你使用 **Task** 实现主线程等待子线程任务完成后继续执行。

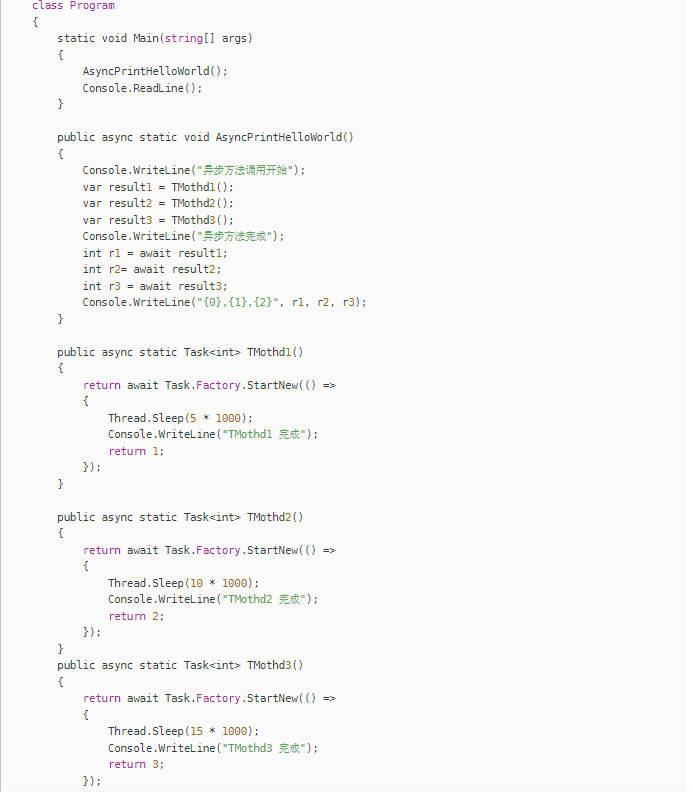
在上述代码中，**Main()** 方法被标记为 **async**，并使用 **await** 来等待子线程的完成。使用 **async/await** 的方式可以让主线程（UI线程）在等待子线程时保持响应，不会阻塞用户界面。

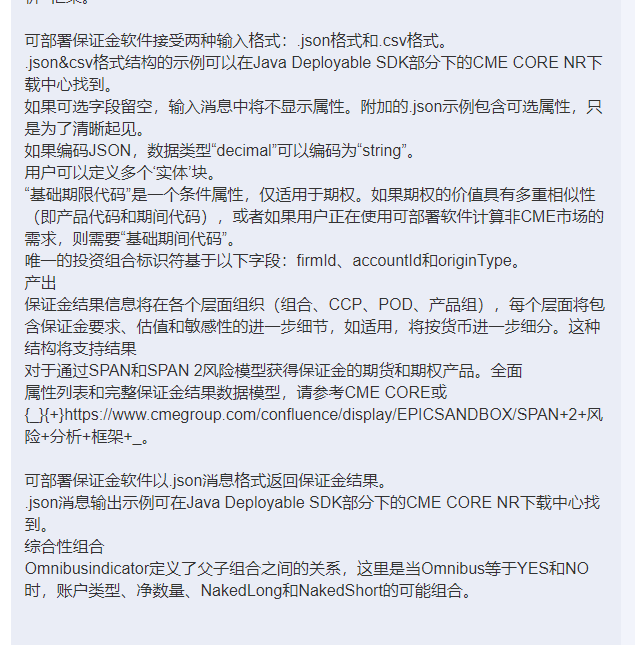
希望这可以解决你的疑问，让你避免UI线程阻塞的问题。

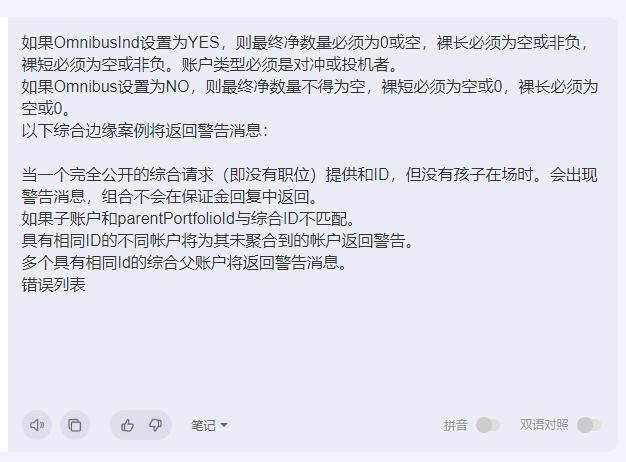


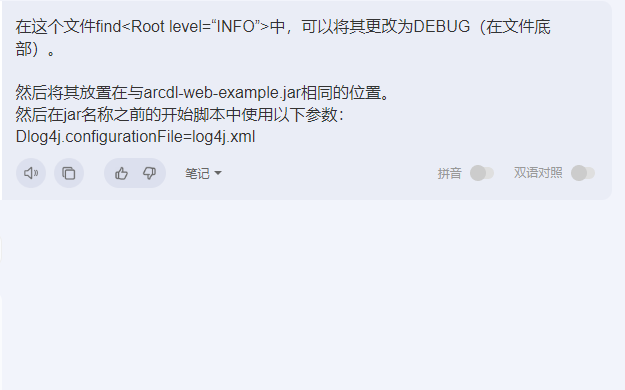


非常经典的三个耗时任务同时并行，最后等待他们的结果

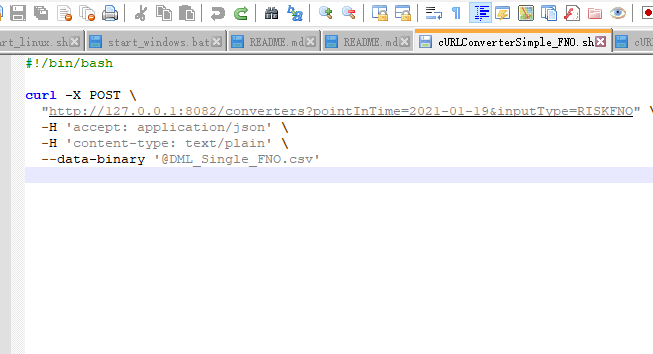
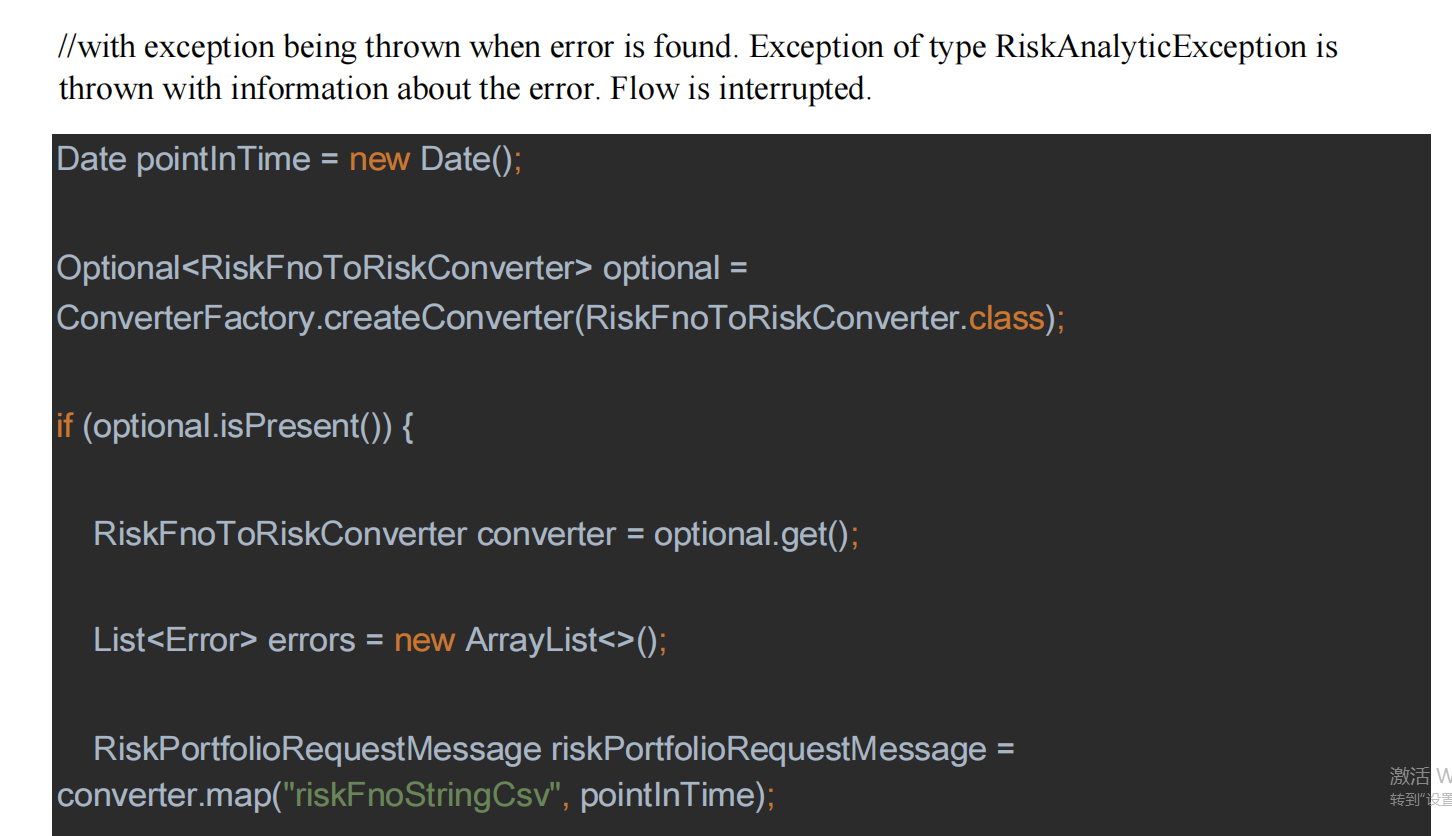
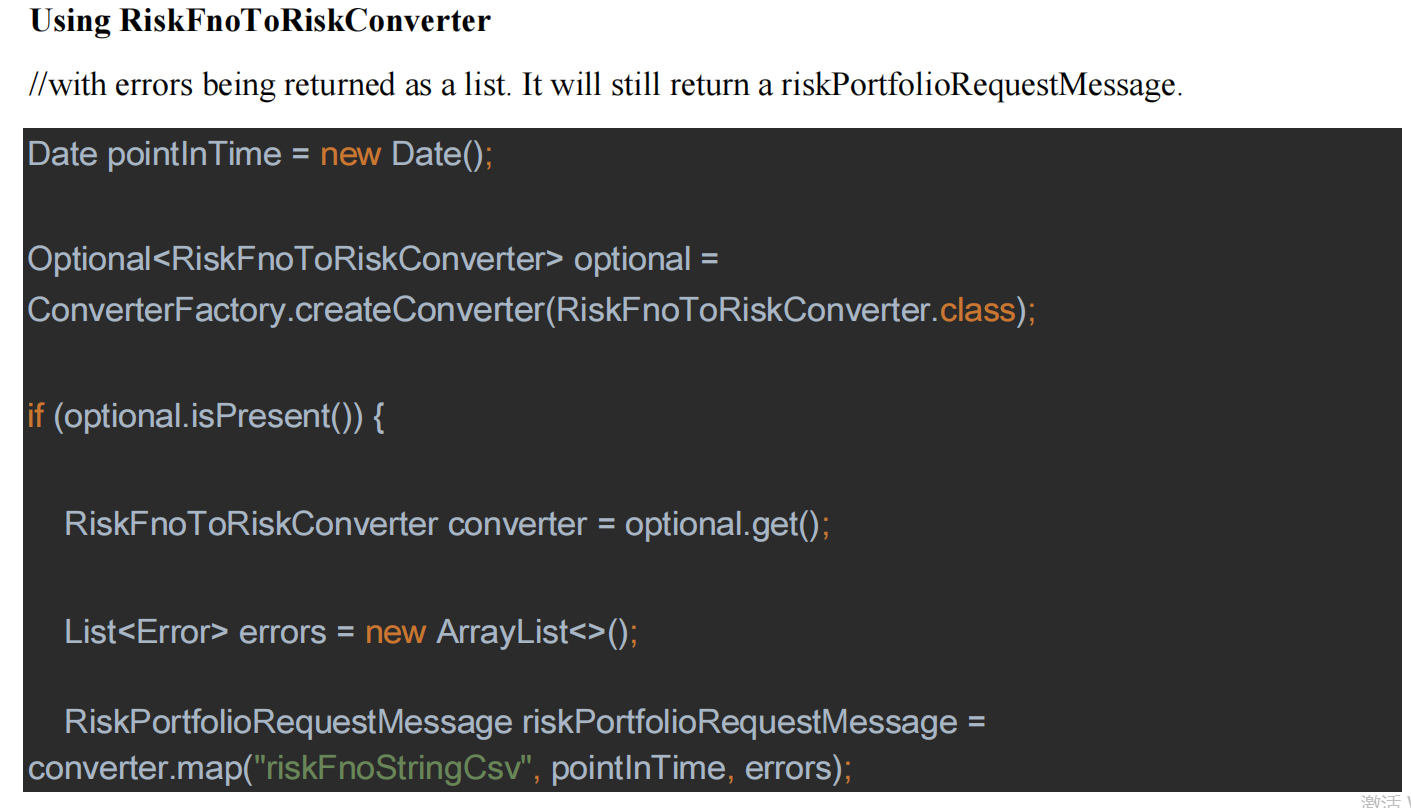
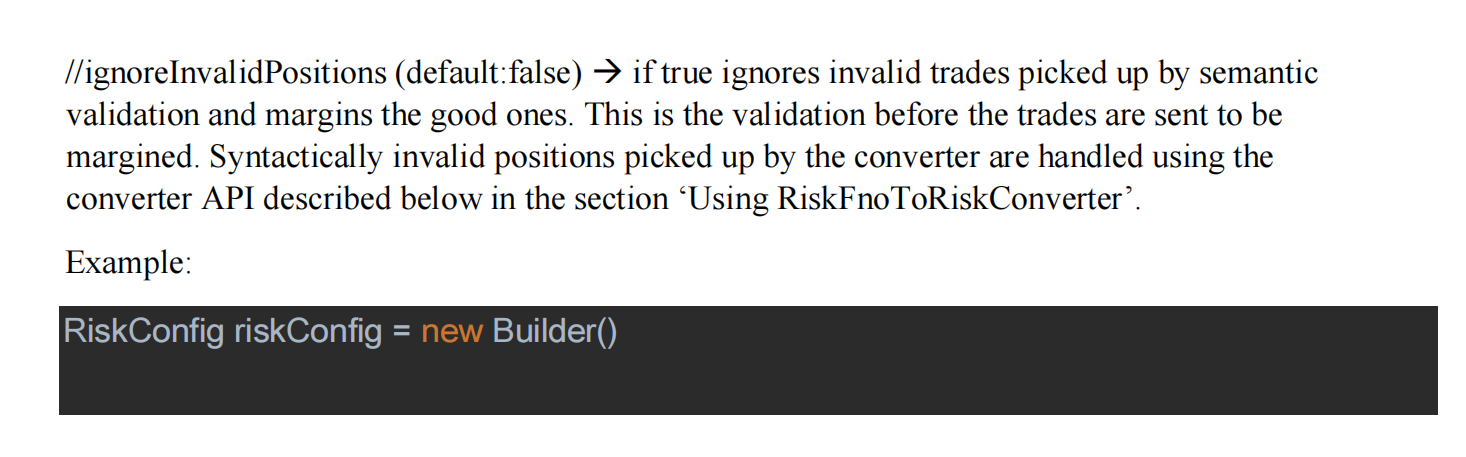




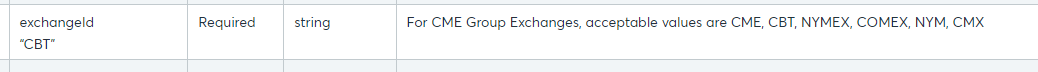


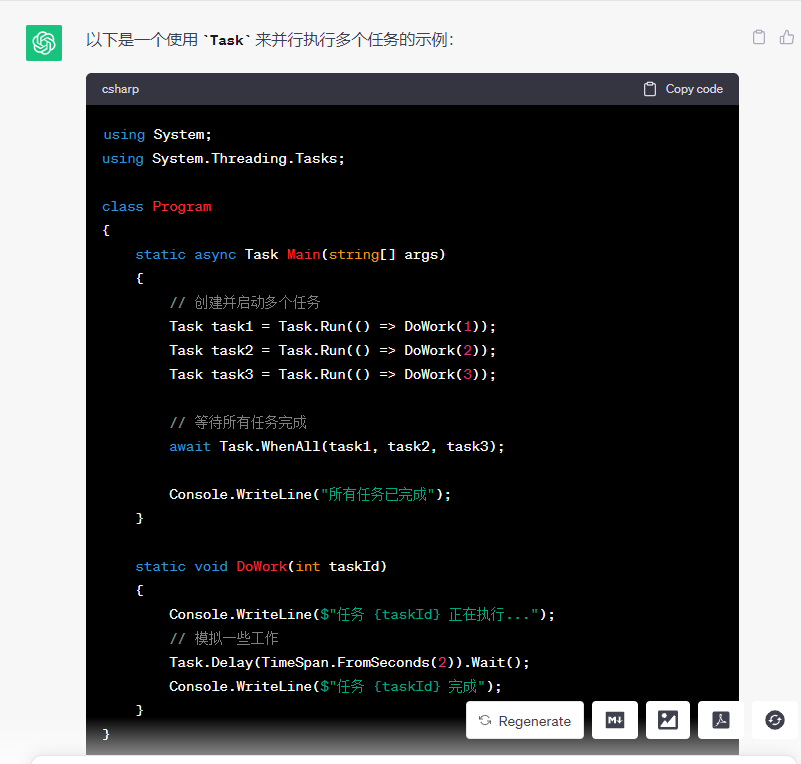


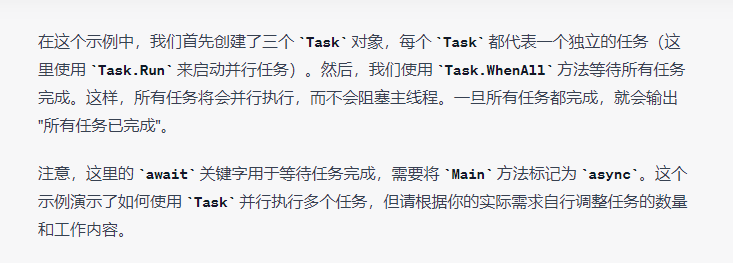
Csv转换成json

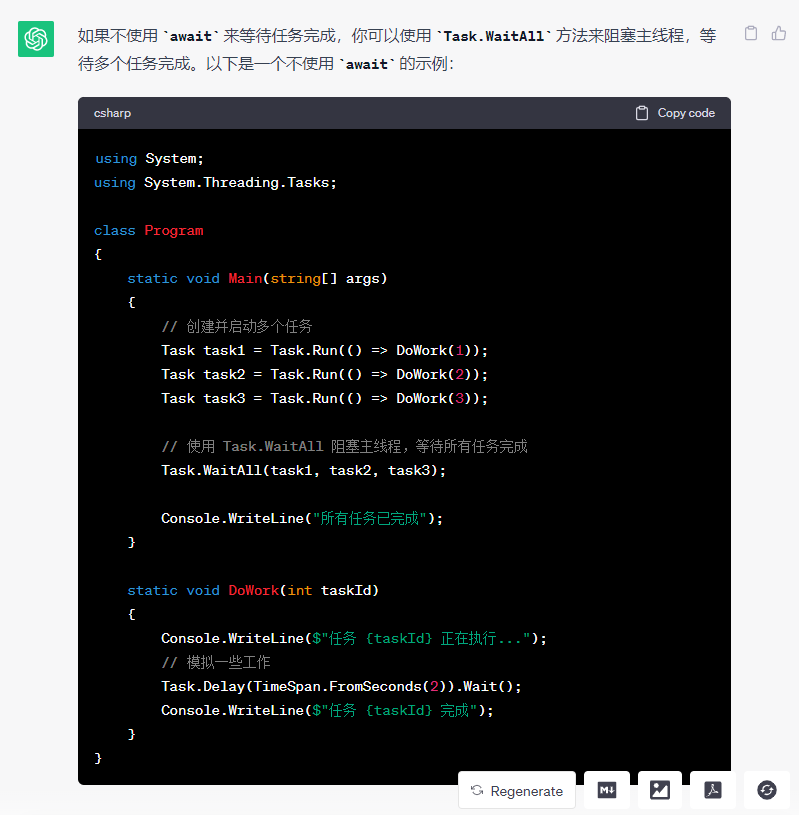


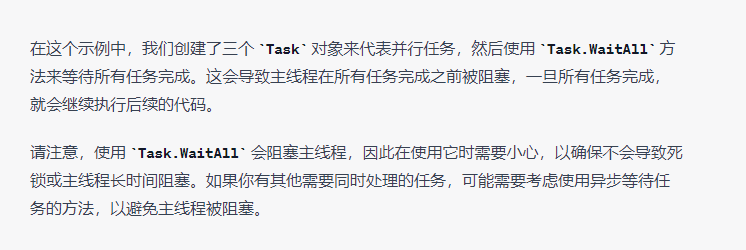
Exchangid可接受的值





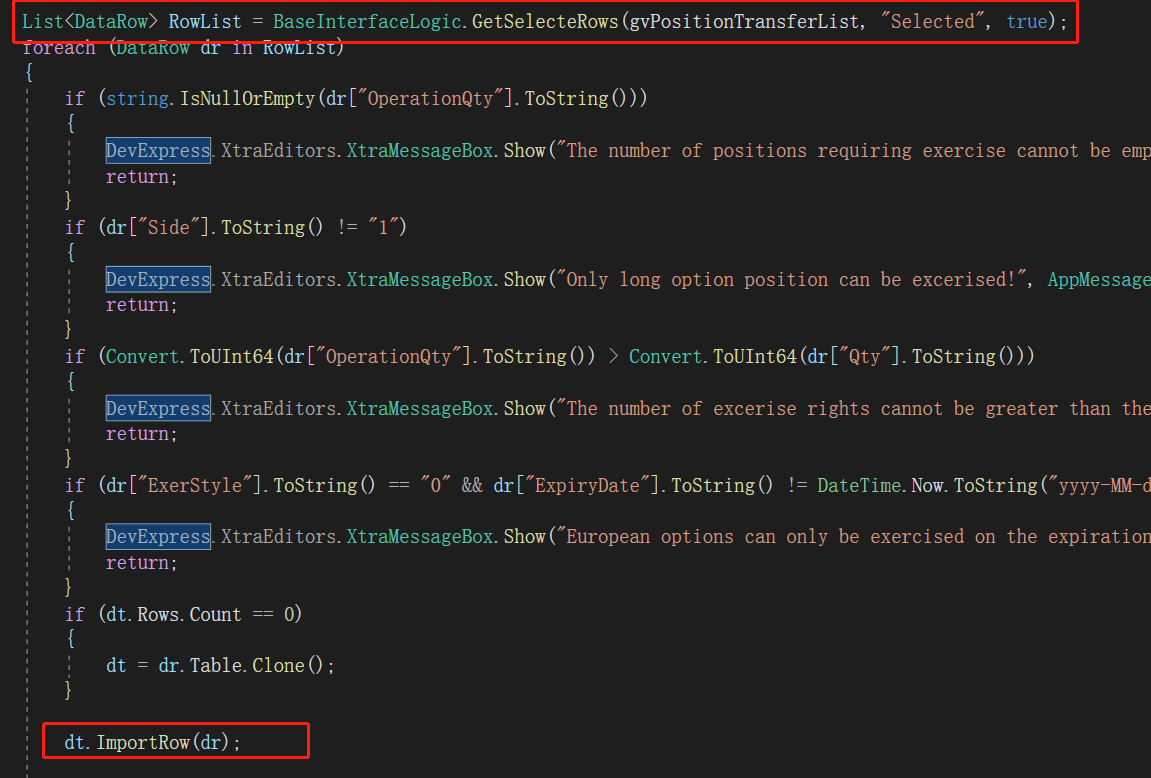


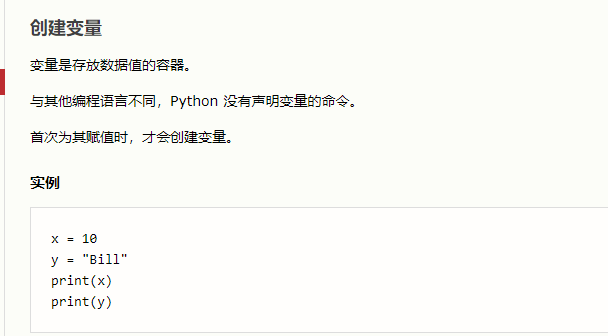


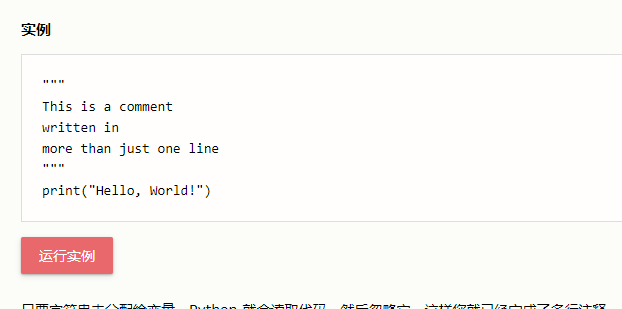
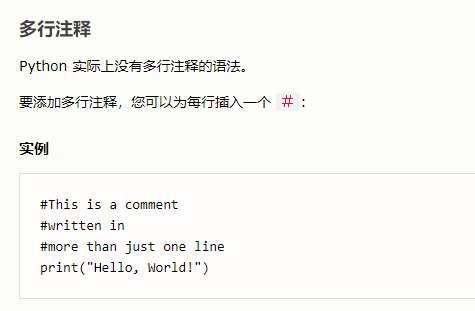
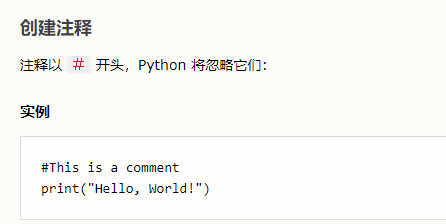


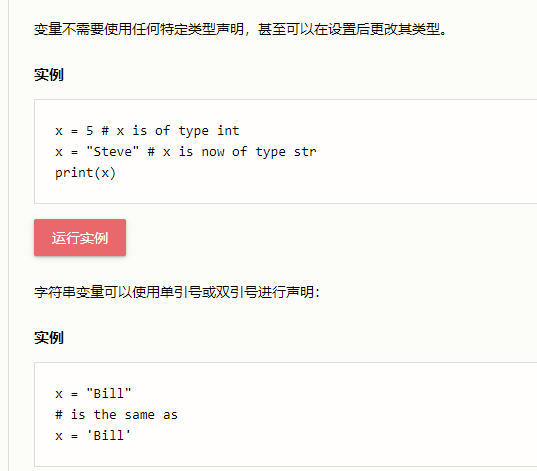








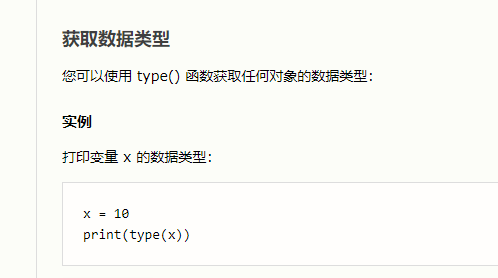
















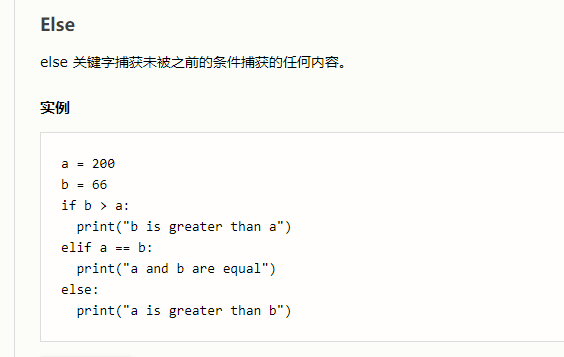


















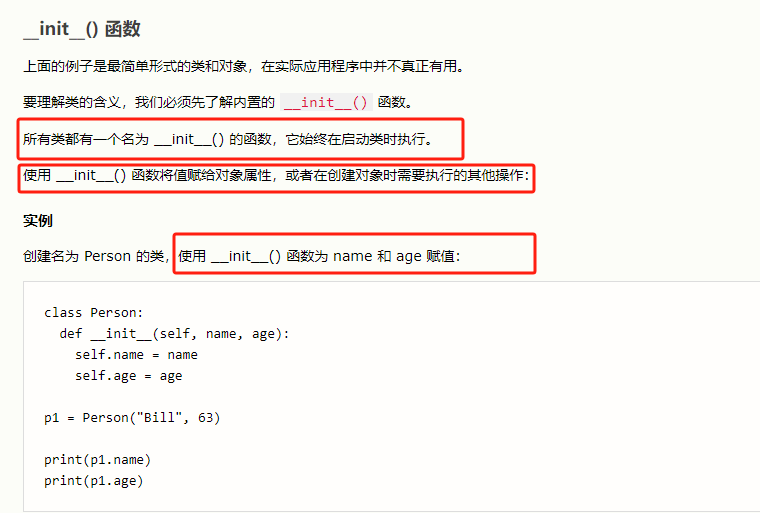














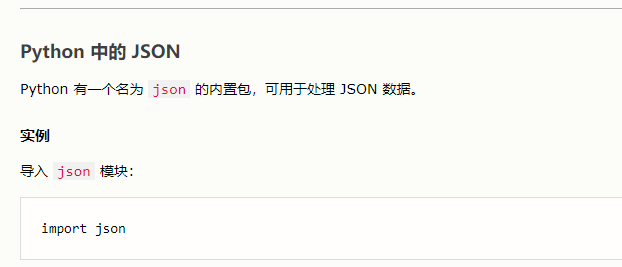






























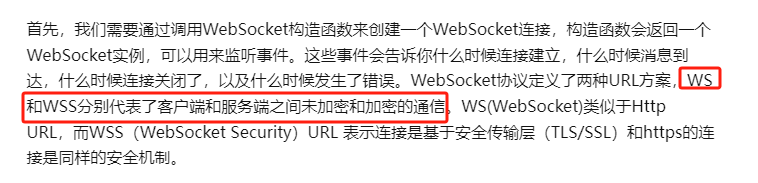


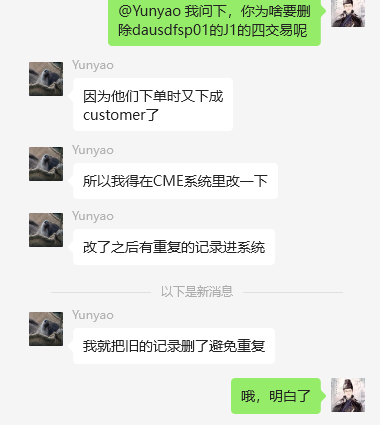


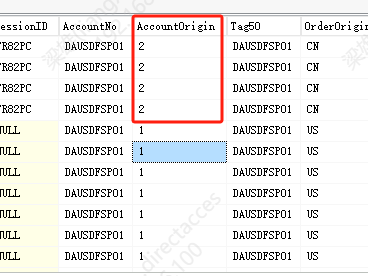


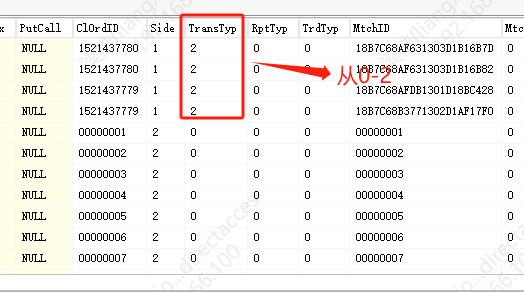


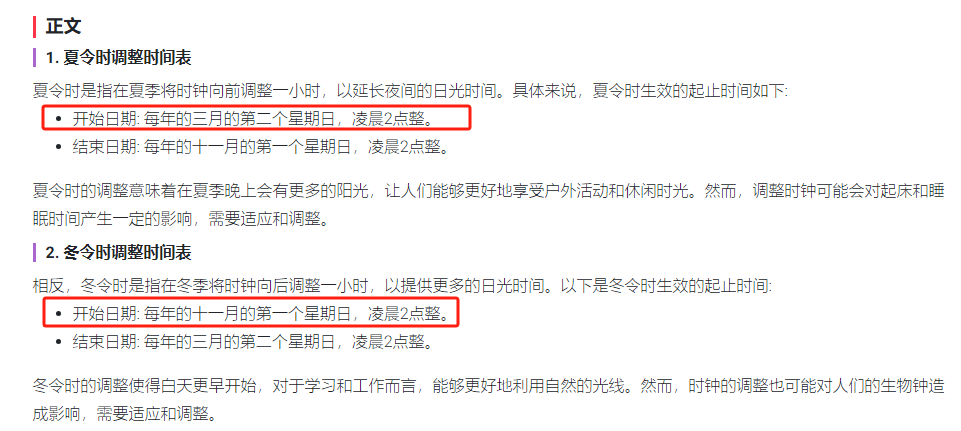


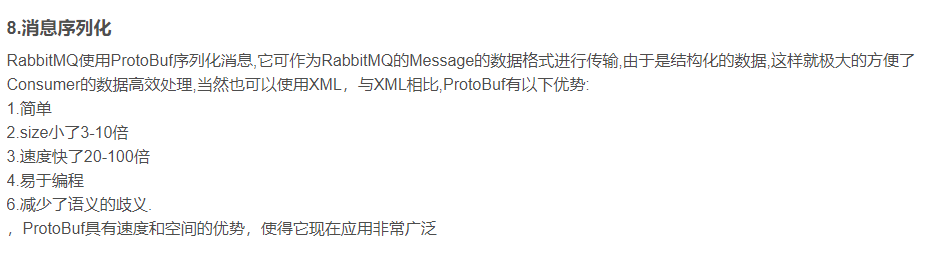












要暂停邮件发送：



是否用rabbitmq传递行情、交易以及开盘四文件的开关

