MP行情数据处理流程

第一章 MP服务程序的层次结构

MP的服务程序架构

MP的服务程序表现为几个可执行文件（.exe），目前分为32位版本的64位版本，开发环境、测试环境和生产环境中的服务程序都为64位版本，运行于Windows Server 2003 R2 64-bit上，各服务程序需要按照一定的层次结构进行搭建，才能服务MP终端用户的请求。

各服务程序可以运行于不同的PC上，也可以运行于同一台PC上，不过，为了考虑服务程序的性能和稳定性，一般会将其分别部署到不同的PC中。有的服务程序在运行过程中需要占用大量的内存或磁盘空间，因此需要单独部署。考虑到成本、性能、吞吐量等，服务程序的物理部署可以是多样的，但是，它们的逻辑部署是一致的。

关于它们的逻辑部署结构图，可以参考MP文档。

MP主要的服务程序

不同层次的服务程序有着不同的职责。下面，将简单的讲述各服务程序的功能：

**（1）Receive Server**

该服务程序接收原始行情数据，并进行解析，接着下发。原始行情数据称为Raw Data，它的格式是由交易所规定的，所以Receive Server必须按照交易所给出的协议文档精确的对Raw Data进行解析，同时，在设计Receive Server时也要保证一定的灵活性，因为交易所规定的原始数据格式可能会发生变化。

Receive Server会将原始数据解析成MP内部通用的数据格式，然后下发给它的下层服务程序: Main Server。注意，Receive Server不会对数据进行任何业务处理，它只负责接收->解析->下发。

**（2）Main Server**

Main Server可以说是MP内部的行情中心，它是Receive Server的下层服务，所有的行情数据经过Receive Server解析之后，都会下发给Main Server，Main Server会根据数据的不同类型进行不同的处理，比如，如果Receive Server下发的是早上的初始化数据（在HKE PRS Spec中对应于DS消息），那么Main Server将会把这些数据拿去进行初始化处理，比如更新Master表等。同时，对于某些行情数据，Main Server会进行一些处理，比如进行业务计算，然后也会把处理之后的结果下发给它的下层服务: Service Server。

**（3）Service Server**

Receive Server和Main Server可以称为MP的高层服务程序，Service Server及其下层服务被称为MP的低层服务程序，之所以称为低层服务，是因为客户端的请求最终只会到达Service Server进行处理，并不会到达Main Server或更上层的服务，因此，Main Server对于客户端请求时不可见的。所有的请求都会交给Service Server统一处理，当然，如果Service Server不能处理，它会将此次请求转发给能进行处理的服务程序，比如，客户端请求历史数据，而Service Server并不能处理此次请求，但是，它知道DB Inquire能处理，因此，它会将此请求转发给DB Inquire，DB Inquire处理完成之后将结果数据返回给Service Serevr,由Service Server再将结果数据下发以返回给客户端。

**（4）DB Inquire**

DB Inquire是用于查询历史数据的服务程序，客户端的历史数据请求就是由Service转发给它进行处理，查询出来的结果数据会返回给Service，并由Service下发给客户端。在MP系统中，数据有两个存放地点，一个是内存数据库MDB，一个是物理数据库SQL Server，内存数据库用于存放访问频繁的数据，物理数据库存放大量的历史数据。DB Inquire会根据不同的请求，去不同的数据库中进行查找。

**（5）MDB**

内存数据库，用于存放可能会被频繁访问的数据，分为MDB\_TICK、MDB\_MINUTELY、MDB\_DAY，MDB\_TICK存放逐笔交易数据，只能存放当天或近期内的交易数据，否则数据量会相当大；MDB\_MINUTELY存放分钟数据，也就是分时图的数据来源，一般也存放近期的数据；MDB\_DAY存放日数据，是日线图的数据来源，几乎存放了从物理数据库有数据以来的所有日数据，也是这三个内存数据库中占用内存最大的，大概1.8G。

**（6）DB Save和MDB Save**

DB Save是连接到Main Server的，它将Main Server下发给它的数据存放到物理数据库中，比如，Main Server接收到股票的交易行情数据，就发送给DB Save，DB Save就把这一条一条的记录存入对应的Contract表中。

同理，MDB Save会将行情数据存入MDB中，以便防止频繁IO访问带来的性能损伤。

**（7）RTS**

RTS职责比较简单，但是相当重要。RTS分为内部RTS和外部RTS，外部RTS就是让客户端直接与之相连，而内部RTS是客户端通过Authentication Server分配得到的。不管是内部RTS还是外部RTS，其主要职责是将客户端的请求转发给Service，同时，还要将Service下发的结果反馈给客户端。

**（8）Authentication Server**

主要负责对客户端的身份进行验证，还有分配内部RTS给客户端进行连接。

第二章 Receive Server处理流程

Receive Server的职责

接收Raw Data，解析，下发。

HKE PRS原始数据的定义

Receive Server的作用是解析原始消息包，不同的交易所，不同的金融品种，其消息包结构都是不同的，因此，要解析原始消息，就必须知道原始消息的具体格式，而这个格式，是在交易所给我们的协议规范中定义的。

我们对Receive Server及其下层所有服务程序的代码分析，都将会以HKE PRS Spec定义的9种不同的消息格式进行讲解。关于PRS原始消息包的具体格式，可以参考《PRS Transmission Spec Version 3.1》。

解析原始消息包

接下来，我们会关注具体的实现代码，来讲解Receive解析原始消息包的流程。

首先，我们要注意两个接口：IPacketAnalyzer和ITransactPacket。

IPacketAnalyzer是框架提供的接口类，其中定义了OnReceiveData方法，任何实现了这个接口的类都必须重写OnReceiveData方法，用于接收原始消息。因此，IPacketAnalyzer实质是用于接收原始行情消息包的接口。注意，IPacketAnalyzer内部有个m\_szPacketData缓冲区（静态数组），用于保存接收到的原始行情消息包。

对于PRS，IPAPRS类实现了IPacketAnalyzer接口，并提供了自己的OnReceiveData方法，用于接收香港的PRS原始行情消息，当接收到了原始行情消息包后，会将行情消息保存于内部缓冲区m\_szPacketData中，接着调用ITransactPacket的DataProcess方法对其进行解析。

ITransactPacket也是框架提供的接口类，其中定义了DataProcess方法，任何实现了这个接口的类都必须重写DataProcess方法，用于解析原始的行情消息，因此，ITransactPacket实质是行情消息解析的接口。

对于PRS，ITPPRS类实现了ITransactPacket接口，并提供了自己的DataProcess方法，专门用于解析香港PRS原始行情消息，同理，ITPSZE类实现了ITransactPacket接口并提供了自己的DataProcess方法，专门用于解析深交所定义的原始行情消息。

ITPPRS内部有一个CCommonData对象的指针m\_pCommonData，指向在堆上的创建的CCommonData对象，该对象内部保存了解析后的数据。

在ITPPRS中，具体的解析任务是由PRSParse对象完成的，HKE PRS Spec定义了9种类型的消息，不同的消息格式是不同的，PRSParse类的职责就是严格按照协议规范解析这9种不同的消息。PRSParse的ParsePacket函数会根据不同的消息类型进行解析，比如如果是DS消息，则会调用ParseDS，如果是TD消息，则调用ParseTD。

以DS消息为例，ParseDS函数内会对原始的DS消息按照协议规范进行解析，并把解析后的数据存储在m\_pCommonData指向的对象中，接着ITP还会将m\_pCommonData指向的数据发往下层服务，即是Main Server。因此，Main Server以及下层所有服务，收到的行情数据都是被放置在CCommonData中已被解析后的数据了。其他原始行情消息的处理过程与此类似。

综上所述，Receive Server的主要两个职责就是：

（1）调用IPAPRS::OnReceiveData接收原始行情消息包

（2） 调用ITPPRS::DataProcess解析消息包，并把解析后的数据下发至Main。

还剩下什么

我们只详细讲解了Receiver Server对HKE PRS的处理流程，对于其他交易所的任何品种，其处理流程都是大致相同的。

同时，关于IPA如何接收数据，ITP如何传输数据，涉及到框架提供的IOCP和线程池。我们只会在业务层面进行探究，关于低层网络通信等代码的具体实现上，可以参考相关书籍自己摸索。

补充说明

Receive Server的业务逻辑代码在ReceivePacket.dll中。

IPAPRS的源码文件路径：

Src\Business\China\ReceivePacket\SocketInterface\IPAPRS.h(IPAPRS.cpp)

ITPPRS的源码文件路径：

Src\Business\China\ReceivePacket\SocketInterface\ITPPTS.h（ITPPRS.cpp）

第三章 Main Server处理流程

Main Server的职责

Main Server是MP所有服务程序的行情中心，是MP内部行情数据的源头。在不同的时段，Main Server有不同的处理逻辑。以香港股票为例，全天交易日可分为四个时段：开市前时段、早市时段、延续早市时段及午市时段。在不同的时段，Main Server会有不同的任务需要处理。

Main Server的所有业务流程的处理逻辑都放在以下两个目录：

1、\Src\Business\China\MChina

2、\Src\Business\China\CommonChina

在下面具体的代码分析中，我们将会以HKE PRS的Future行情为例。

Main Server的具体任务

Main Server的通用业务可以分为：

**（1）CodeLoad**

CodeLoad的作用是加载快照数据，其实就是将数据库某些表的某些常用的字段加载到内存之中。当有新行情到来的时候，快照数据中相应字段的值也会同时被更新，比如ClosePrice，TradingTime等。

对于HKE Future，加载快照数据的处理函数是：LoadHKEPRSFuturesCode。该函数会从HKEFutureMaster和HKEFutureDaily表中将某些常用的字段加载进内存。

Main Server不同时段的处理任务可分为：

**（1）接收初始化数据**

由HKE PRS Spec可知，在每天HKE Future/Option开盘之前，PRS系统会将今天所有要参加交易的品种的初始数据以DS消息推送给我们，Receive Server会接收并解析此消息，然后交给Main Server进行处理，此时Main Server会进行一些初始化操作。

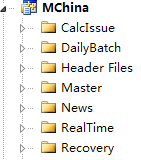
**（2）接收到实时行情**

Receive Server会将实时行情下发给Main Server，Main Server接收到此实时行情后会进行一些简单的计算，并更新Main Server内存中对应的快照数据，然后还会将某些字段推送给Index Server进行计算，Index Server计算完成后会把结果发回给Main Server，同时，Main Server还会将行情数据推送给DB Save、 MDB Save和Service Server。

**（3）Daily Batch**

当一天的交易完全结束之后，Main会做一些日终处理。日终处理基本上是做一些统计数据的计算，然后更新一些表，比如Daily表，Period表等。

下图即是Main Server的业务代码的分类:



由文件夹的名称基本就可以猜到Main Server所要做的任务了。

第四章 Service Server处理流程

Service Server的职责

Service Server是客户请求的最终处理者，客户端的请求最终会交给Service进行处理，如果Service不能处理，它会将请求转发给其他的服务程序（DBInquiry等）进行处理，其他服务程序处理完成后，会将结果返回给Service，再由Service返回给下层服务程序（RTS）和客户端。

Service服务程序的内存中也保存有快照数据，这些快照数据会被实时行情更新。同Main Server类似，当Service启动的时候，会加载快照数据，加载快照数据主要有两种方式：一是从其他服务程序的内存中加载，二是从数据库加载，当然，从数据库加载会相当的耗时。

客户端的请求主要可以分为两类：实时行情的请求和历史数据的请求。

对于客户端实时行情的请求，MP采用的是数据“推送”模式，即一旦客户端订阅了某类行情，当有行情数据到达Service的时候，Service会立即将实时行情推送给订阅者，注意，Service并不知道是哪些客户端订阅了行情数据！Service的订阅者是RTS，因此Service会将实时行情推送给订阅了此类行情的RTS，再由RTS推送给订阅了此类行情的客户端。当客户端取消了对于某些行情数据的订阅（比如客户端关闭了实时行情的报价画面），那么当RTS接收到此类行情时，就不会再推送给该客户端了。同样，Service也不会将RTS没有订阅的行情数据推送给它。

对于客户端历史数据的请求，MP采用的是数据“拉取”模式，即客户端需要一次数据，就要发送一次请求。对于这样历史数据的请求，Service是没有办法处理的，因为历史数据的数据量相当大，是存储在SQL Server数据库或内存数据库中的。Service不可能保存这些数据，因此Service会将请求转发给DB Inquiry，再由DB Inquiry去处理。

MP把历史数据保存在两个地方：1是SQL Server数据库，2是内存数据库。不同的是，MDB保存的是最近一段时间内的历史数据，因为对于这类型的数据，客户端可能会频繁的进行请求（比如绘制分时图），因此将这些近期的历史数据保存与MDB中，将会显著降低系统的IO访问的时间开销。

DB Inquiry会首先试着到MDB中去查询，如果查到了数据就返回给Service，如果没有查到，则再去SQL Server去查询，再将查询返回的结果返回给Service。

Service Server的具体任务

第五章 DB Save和MDB Save处理流程

DB Save Server的职责

DB Save Server为Main Server的下层服务程序，用于把Main推送给它的行情数据保存到数据库中。

DB Save启动的时候也会加载快照数据，这时可以选择是从它的上层服务Main Server的内存中加载（必须保证Main此时内存中存在快照数据），也可以选择从数据库中加载。当交易所定义的原始行情消息推送给Receive Server时，Receive Server会按照协议规定的格式对此行情消息进行解析，然后下发给Main Server，Main Server会进行一些简单的处理（比如计算），然后会下发给DB Save，DB Save接收到Main下发的行情数据，首先会解析此行情数据，确定此行情数据是属于哪个品种（HKE Stock、HKE Future），哪条记录（StandardCode），接着更新自己内存中的对应快照，最后调用相应的存储过程将此条数据存入到数据库中的对应的表中。例如，对于HKE Stock的实时行情数据，DB Save会调用函数SaveHKEContractData,该函数内部首先会更新对应的快照（根据UNDataType在所有快照数据中查找出需要更新的快照），接着调用MPChina数据库中的存储过程usp\_HKEStockContract将此条行情数据存储到数据库中。

下面是一条以HKE Stock实时行情数据调用存储过程usp\_HKEStockContract的SQL语句：

exec usp\_HKEStockContract 3,N'HK7001157000',20110311,111414953,N'01157',456,0,0,0,18.959999,91903625,19.080000,95930781,18.740000,93356251,18.860001,-0.199999,4,-1.05,18.860001,200,3772.000000,3446400,65193874.000000,548,1,3,2,18.916513,99000,1877078.875000,3,1552600,29308143.431396,199,1275800,24114014.813232,311,2200,41808.000244,3,18.860001,18.840000,16800,200,18.880001,18.820000,40000,36600,18.900002,18.799999,77200,56200,18.920002,18.779999,20000,49800,18.940002,18.759998,10000,27600,164000,170400,45.11,50.96,50.96,60.98,0,11,1,2,10,5,11,1,14,1,8

调用此存储过程的参数，绝大部分是Main Server推送给DB Save的行情数据CommonData中的字段，同时，也有少量DB Save中保存的快照数据的某些字段。

下面贴出usp\_HKEStockContract的代码进行简要的分析：

USE [MPChina]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

ALTER Procedure [dbo].[usp\_HKEStockContract]

(

@MarketType tinyint,

@StandardCode nchar(12),

@TradingDate int,

@TradingTime int,

@SecurityID nchar(6),

@TickerKey int,

@TradeChannel tinyint, -- 1: A shares, 2: B shares

@TradeRef int,

@TradeIndex int,

@OpenPrice decimal(9,3),

@OpenTime int,

@HighPrice decimal(9,3),

@HighTime int,

@LowPrice decimal(9,3),

@LowTime int,

@ClosePrice decimal(9,3),

@ChangePrice decimal(9,3),

@ChangeType tinyint,

@ChangeRate decimal(9,2),

@ContractPrice decimal(9,3),

@ContractVolume int,

@ContractAmount decimal(18,3),

@TotalVolume bigint,

@TotalAmount decimal(18,3),

@TickCount int,

@TradingType tinyint,

@SessionType tinyint,

@TradingTrend tinyint,

@AveragePrice decimal(9, 3),

@BeforeSameVolume bigint,

@BeforeSameAmount decimal(18, 3),

@BeforeSameCount int,

@SellVolume bigint,

@SellAmount decimal(18, 3),

@SellCount int,

@BuyVolume bigint,

@BuyAmount decimal(18, 3),

@BuyCount int,

@RotateVolume bigint,

@RotateAmount decimal(18, 3),

@RotateCount int,

@AskPrice1 decimal(9, 3),

@BidPrice1 decimal(9, 3),

@AskSize1 int,

@BidSize1 int,

@AskPrice2 decimal(9, 3),

@BidPrice2 decimal(9, 3),

@AskSize2 int,

@BidSize2 int,

@AskPrice3 decimal(9, 3),

@BidPrice3 decimal(9, 3),

@AskSize3 int,

@BidSize3 int,

@AskPrice4 decimal(9, 3),

@BidPrice4 decimal(9, 3),

@AskSize4 int,

@BidSize4 int,

@AskPrice5 decimal(9, 3),

@BidPrice5 decimal(9, 3),

@AskSize5 int,

@BidSize5 int,

@AskTotalSize bigint,

@BidTotalSize bigint,

@TradingStrength decimal(9, 2),

@ResidualStrength decimal(9, 2),

@MiniResidualStrength decimal(9, 2),

@TickCountStrength decimal(9, 2),

@TradeCount int = 0,

@AskCount1 int = 0,

@BidCount1 int = 0,

@AskCount2 int = 0,

@BidCount2 int = 0,

@AskCount3 int = 0,

@BidCount3 int = 0,

@AskCount4 int = 0,

@BidCount4 int = 0,

@AskCount5 int = 0,

@BidCount5 int = 0,

@OneMonthHighPrice decimal(9,3) = 0,

@OneMonthLowPrice decimal(9,3) = 0,

@ThreeMonthHighPrice decimal(9,3) = 0,

@ThreeMonthLowPrice decimal(9,3) = 0,

@AnnualHighPrice decimal(9,3) = 0,

@AnnualLowPrice decimal(9,3) = 0,

@OneMonthChangeRate decimal(9,3) = 0,

@ThreeMonthChangeRate decimal(9,3) = 0,

@OneYearChangeRate decimal(9,3) = 0

)

AS

if exists ( select StandardCode from HKEStockDaily where TradingDate = @TradingDate and StandardCode = @StandardCode )

begin

update HKEStockDaily

set MarketType = @MarketType,

OpenPrice = @OpenPrice,

OpenTime = @OpenTime,

HighPrice = @HighPrice,

HighTime = @HighTime,

LowPrice = @LowPrice,

LowTime = @LowTime,

ClosePrice = @ClosePrice,

TradingTime = @TradingTime,

ChangePrice = @ChangePrice,

ChangeType = @ChangeType,

ChangeRate = @ChangeRate,

TotalVolume = @TotalVolume,

TotalAmount = @TotalAmount,

TickCount = @TickCount,

AveragePrice = @AveragePrice,

SellVolume = @SellVolume,

SellAmount = @SellAmount,

SellCount = @SellCount,

BuyVolume = @BuyVolume,

BuyAmount = @BuyAmount,

BuyCount = @BuyCount,

RotateVolume = @RotateVolume,

RotateAmount = @RotateAmount,

RotateCount = @RotateCount,

AskPrice1 = @AskPrice1,

BidPrice1 = @BidPrice1,

AskSize1 = @AskSize1,

BidSize1 = @BidSize1,

AskPrice2 = @AskPrice2,

BidPrice2 = @BidPrice2,

AskSize2 = @AskSize2,

BidSize2 = @BidSize2,

AskPrice3 = @AskPrice3,

BidPrice3 = @BidPrice3,

AskSize3 = @AskSize3,

BidSize3 = @BidSize3,

AskPrice4 = @AskPrice4,

BidPrice4 = @BidPrice4,

AskSize4 = @AskSize4,

BidSize4 = @BidSize4,

AskPrice5 = @AskPrice5,

BidPrice5 = @BidPrice5,

AskSize5 = @AskSize5,

BidSize5 = @BidSize5,

AskTotalSize = @AskTotalSize,

BidTotalSize = @BidTotalSize,

TradingStrength = @TradingStrength,

ResidualStrength = @ResidualStrength,

MiniResidualStrength = @MiniResidualStrength,

TickCountStrength = @TickCountStrength,

TradeCount = @TradeCount,

AskCount1 = @AskCount1,

BidCount1 = @BidCount1,

AskCount2 = @AskCount2,

BidCount2 = @BidCount2,

AskCount3 = @AskCount3,

BidCount3 = @BidCount3,

AskCount4 = @AskCount4,

BidCount4 = @BidCount4,

AskCount5 = @AskCount5,

BidCount5 = @BidCount5,

OneMonthHighPrice = @OneMonthHighPrice,

OneMonthLowPrice = @OneMonthLowPrice,

ThreeMonthHighPrice = @ThreeMonthHighPrice,

ThreeMonthLowPrice = @ThreeMonthLowPrice,

AnnualHighPrice = @AnnualHighPrice,

AnnualLowPrice = @AnnualLowPrice,

OneMonthChangeRate = @OneMonthChangeRate,

ThreeMonthChangeRate = @ThreeMonthChangeRate,

OneYearChangeRate = @OneYearChangeRate

where TradingDate = @TradingDate

and StandardCode = @StandardCode;

end

else

begin

insert into HKEStockDaily

(

TradingDate, StandardCode, SecurityID, TradeChannel, MarketType,

OpenPrice, OpenTime, HighPrice, HighTime, LowPrice, LowTime,

ClosePrice, TradingTime, ChangePrice, ChangeType, ChangeRate,

TotalVolume, TotalAmount,

PrevClosePrice,

TickCount, AveragePrice,

SellVolume, SellAmount, SellCount, BuyVolume, BuyAmount, BuyCount,

RotateVolume, RotateAmount, RotateCount,

AskPrice1, AskSize1, BidPrice1, BidSize1,

AskPrice2, AskSize2, BidPrice2, BidSize2,

AskPrice3, AskSize3, BidPrice3, BidSize3,

AskPrice4, AskSize4, BidPrice4, BidSize4,

AskPrice5, AskSize5, BidPrice5, BidSize5,

AskTotalSize, BidTotalSize,

TradingStrength, ResidualStrength, MiniResidualStrength,

TickCountStrength, TradeCount,

AskCount1, BidCount1,

AskCount2, BidCount2,

AskCount3, BidCount3,

AskCount4, BidCount4,

AskCount5, BidCount5,

OneMonthHighPrice,OneMonthLowPrice,

ThreeMonthHighPrice,ThreeMonthLowPrice,

AnnualHighPrice,AnnualLowPrice,

OneMonthChangeRate,ThreeMonthChangeRate,OneYearChangeRate

)

values

(

@TradingDate, @StandardCode, @SecurityID, @TradeChannel, @MarketType,

@OpenPrice, @OpenTime, @HighPrice, @HighTime, @LowPrice, @LowTime,

@ClosePrice, @TradingTime, @ChangePrice, @ChangeType, @ChangeRate,

@TotalVolume, @TotalAmount,

@ClosePrice - @ChangePrice,

@TickCount, @AveragePrice,

@SellVolume, @SellAmount, @SellCount, @BuyVolume, @BuyAmount, @BuyCount,

@RotateVolume,@RotateAmount, @RotateCount,

@AskPrice1, @AskSize1, @BidPrice1, @BidSize1,

@AskPrice2, @AskSize2, @BidPrice2, @BidSize2,

@AskPrice3, @AskSize3, @BidPrice3, @BidSize3,

@AskPrice4, @AskSize4, @BidPrice4, @BidSize4,

@AskPrice5, @AskSize5, @BidPrice5, @BidSize5,

@AskTotalSize, @BidTotalSize,

@TradingStrength, @ResidualStrength, @MiniResidualStrength,

@TickCountStrength, @TradeCount,

@AskCount1, @BidCount1,

@AskCount2, @BidCount2,

@AskCount3, @BidCount3,

@AskCount4, @BidCount4,

@AskCount5, @BidCount5,

@OneMonthHighPrice,@OneMonthLowPrice,

@ThreeMonthHighPrice,@ThreeMonthLowPrice,

@AnnualHighPrice,@AnnualLowPrice

@OneMonthChangeRate,@ThreeMonthChangeRate,@OneYearChangeRate

);

end

if ( 2 != @TradingType )

begin

insert into HKEStockContract

(

StandardCode, TradingDate, TradingTime, SecurityID, MarketType, TickerKey,

TradeChannel, TradeRef, TradeIndex,

OpenPrice, OpenTime,

HighPrice, HighTime,

LowPrice, LowTime,

ClosePrice, ChangePrice, ChangeType, ChangeRate,

ContractPrice, ContractVolume, ContractAmount,

TotalVolume, TotalAmount,

TickCount, TradingType, SessionType, TradingTrend, AveragePrice,

BeforeSameVolume, BeforeSameAmount, BeforeSameCount,

SellVolume, SellAmount, SellCount, BuyVolume, BuyAmount, BuyCount,

RotateVolume, RotateAmount, RotateCount,

AskPrice1, AskSize1, BidPrice1, BidSize1,

AskPrice2, AskSize2, BidPrice2, BidSize2,

AskTotalSize, BidTotalSize,

TradingStrength, ResidualStrength, MiniResidualStrength,

TickCountStrength, TradeCount

)

values

(

@StandardCode, @TradingDate, @TradingTime, @SecurityID, @MarketType, @TickerKey,

@TradeChannel, @TradeRef, @TradeIndex,

@OpenPrice, @OpenTime,

@HighPrice, @HighTime,

@LowPrice, @LowTime,

@ClosePrice, @ChangePrice, @ChangeType, @ChangeRate,

@ContractPrice, @ContractVolume, @ContractAmount,

@TotalVolume, @TotalAmount,

@TickCount, @TradingType, @SessionType, @TradingTrend, @AveragePrice,

@BeforeSameVolume, @BeforeSameAmount, @BeforeSameCount,

@SellVolume, @SellAmount, @SellCount, @BuyVolume, @BuyAmount, @BuyCount,

@RotateVolume, @RotateAmount, @RotateCount,

@AskPrice1, @AskSize1, @BidPrice1, @BidSize1,

@AskPrice2, @AskSize2, @BidPrice2, @BidSize2,

@AskTotalSize, @BidTotalSize,

@TradingStrength, @ResidualStrength, @MiniResidualStrength,

@TickCountStrength, @TradeCount

);

end

insert into HKEStockTodayContract

(

StandardCode, TradingDate, TradingTime, SecurityID, MarketType, TickerKey,

TradeChannel, TradeRef, TradeIndex,

OpenPrice, OpenTime,

HighPrice, HighTime,

LowPrice, LowTime,

ClosePrice, ChangePrice, ChangeType, ChangeRate,

ContractPrice, ContractVolume, ContractAmount,

TotalVolume, TotalAmount,

TickCount, TradingType, SessionType, TradingTrend, AveragePrice,

BeforeSameVolume, BeforeSameAmount, BeforeSameCount,

SellVolume, SellAmount, SellCount, BuyVolume, BuyAmount, BuyCount,

RotateVolume, RotateAmount, RotateCount,

AskPrice1, AskSize1, BidPrice1, BidSize1,

AskPrice2, AskSize2, BidPrice2, BidSize2,

AskTotalSize, BidTotalSize,

TradingStrength, ResidualStrength, MiniResidualStrength,

TickCountStrength, ExpectFlag, TradeCount

)

values

(

@StandardCode, @TradingDate, @TradingTime, @SecurityID, @MarketType, @TickerKey,

@TradeChannel, @TradeRef, @TradeIndex,

@OpenPrice, @OpenTime,

@HighPrice, @HighTime,

@LowPrice, @LowTime,

@ClosePrice, @ChangePrice, @ChangeType, @ChangeRate,

@ContractPrice, @ContractVolume, @ContractAmount,

@TotalVolume, @TotalAmount,

@TickCount, @TradingType, @SessionType, @TradingTrend, @AveragePrice,

@BeforeSameVolume, @BeforeSameAmount, @BeforeSameCount,

@SellVolume, @SellAmount, @SellCount, @BuyVolume, @BuyAmount, @BuyCount,

@RotateVolume, @RotateAmount, @RotateCount,

@AskPrice1, @AskSize1, @BidPrice1, @BidSize1,

@AskPrice2, @AskSize2, @BidPrice2, @BidSize2,

@AskTotalSize, @BidTotalSize,

@TradingStrength, @ResidualStrength, @MiniResidualStrength,

@TickCountStrength, 0, @TradeCount

);

RETURN (0)

MDB Save Server的功能与DB Save Server基本相同，只不过MDB Save是将行情数据存入内存数据库中。

目前MP系统中有三种类型的内存数据库，它们的区别是存储行情数据的周期长短不一样。MDB\_DAY存储日线数据，MDB\_MINUTELY存储分钟数据（一般是7天内），MDB\_TICK存储逐笔数据（当天）。

MDB\_DAY中的数据其实就是从物理数据库DB的Daily表中加载的；MDB\_MINUTELY中的数据是从Minutely表中加载的；MDB\_TICK中的数据是从Contract表中加载的。

当我们启动MDB时，可以选择加载内存数据的方式，一是从其他服务器上的MDB加载，二是从物理数据库的表中加载。如果网络情况正常，推荐采取第一种方式，因为物理数据库中的数据量相当大，因此从数据库加载需要很长的时间。

下面的代码即是从物理数据库加载不同品种的内存数据的SQL语句：

（1）MDB\_MINUTELY: 从SZE加载Stock的Minutely数据

SELECT StandardCode, TradingDate, TradingTime, OpenPrice, HighPrice, LowPrice, ClosePrice, ChangePrice, ChangeType, ChangeRate, ContractVolume, ContractAmount, TotalVolume, TotalAmount, AskPrice1, BidPrice1, AskTotalSize, BidTotalSize FROM **SZEStockMinutely** WHERE **TradingDate >= @Date**

（2）MDB\_TICK： 从HKE加载Stock的Tick数据

SELECT StandardCode, TradingDate, TradingTime, OpenPrice, HighPrice, LowPrice, ClosePrice, ChangePrice, ChangeType, ChangeRate, ContractVolume, TotalVolume, TickCount, TradingType, SessionType, TradingTrend, AskPrice1, BidPrice1, AskTotalSize, BidTotalSize FROM **HKEStockContract** WHERE **TradingDate = @Date**

第六章 DB Inquiry处理流程

DB Inquiry的职责

在第四章对Service Server的讲述中已经提到，客户端的请求大体上可以分为两类：一是实时行情数据的请求，二是历史数据的请求。对于实时行情请求，服务器采用的是“推”模式，这种模式的优点便是客户端不必发送大量的同类请求给服务器，服务器会尽快将行情数据推送给客户端。对于历史数据请求，则是采用“拉”模式，客户端每需要一次历史数据，就要发送一次请求。

当客户端的历史数据查询请求到达Service时，它是无法处理的，因此Service会将此请求转发给DB Inquiry，DB Inquiry的职责即是处理客户端的历史数据请求。DB Inquiry会首先试着在MDB中查询，如果找不到所需数据，则去物理数据库中查询，接着把查询的结果返回给Service，由Service将结果发送给下层服务RTS，RTS会将此结果数据返回给客户端。

对于历史数据查询请求，可以分为两类：由UNDataType构造的请求和号码请求。号码请求是一种特殊的请求，在MP客户端中，如果用户打开某些新闻，资讯画面，则发送的就是号码请求；如果用户打开某些分时图画面或包含历史数据的画面，则发送的是由UNDataType构造的请求。

号码请求的处理函数在：\Src\Business\China\DBInquiry

UNDataType构造的请求处理函数在：Src\Business\Common\Inquiry\HistoryData