09 | 数据传输的质量：金融业务对数据传输有什么要求？

2021-01-11 任杰

分布式金融架构课 进入课程 >

讲述：任杰
时长 20:25 大小 18.70M

你好，我是任杰。这节课我想和你聊一聊怎么做好金融数据的传输。

我们在开篇词提到过，如果你对系统的要求高，通常都会说要按照金融级的标准来设计。所以当我们提到金融数据传输的时候，你可能也会觉得数据传输应该也是要求非常高的，既要速度快，又要流量大，而且还有极强的容灾能力。

没错，在金融行业很多地方会有这样的要求，但是也有一些地方要求并不高。所以这节课我会带你分析一下金融业务在不同场景下对数据传输的要求是什么，以及解决方案都有哪些。

案例分析

按照惯例，我们还是从一个具体的例子开始。和我们之前讲扫码支付一样，这里的例子既要典型，又要简单。所以思考再三，我选择了简化版的券商算法交易平台对接交易所的例子，原因有这些：

1. 涉及场景多。既有事务数据，也有市场数据。

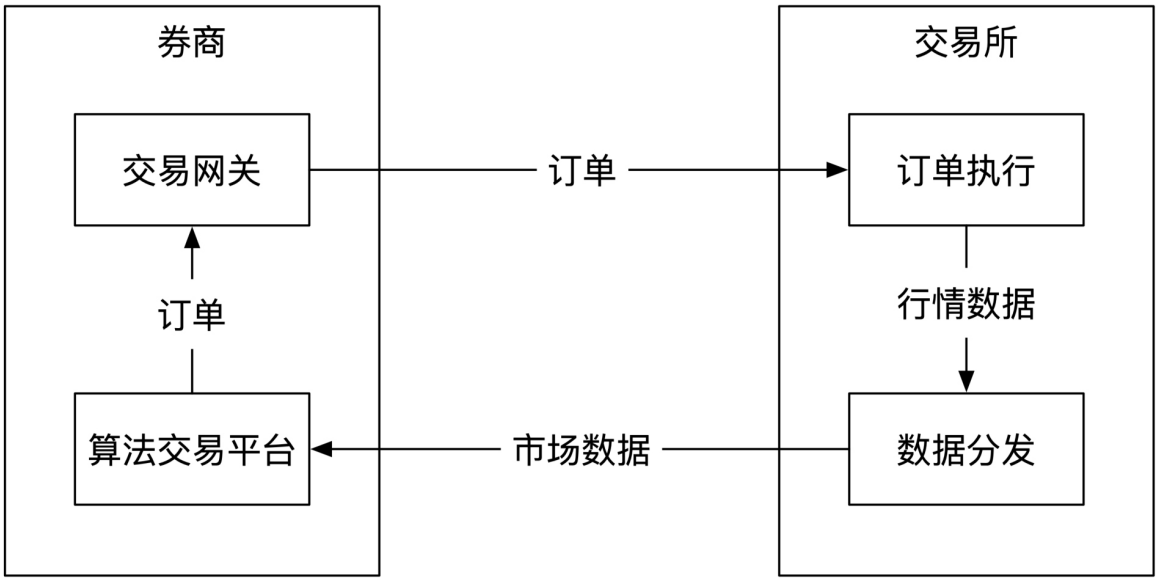
2. 模型简单。只涉及到 2 个主体。

3. 复杂度可选。连接交易所的要求可以很高，也可以很低，具体取决于你愿意出多少钱，有多久的研发时间。

那我们来看看简化的模型是怎样的。

在券商对接交易所的例子里，一共包括两个主体：券商和交易所。券商的任务是将券商提供的市场交易信息发送到自己的算法交易平台，平台分析这些信息之后发送买卖订单给交易所。交易所负责处理收到的订单，在处理完之后将交易信息传递给券商。整个过程都是用软件实现，而不是用硬件。

下图画出了这个简单的交互流程：



交易数据

券商发给交易所的**订单数据属于事务数据**。这里的事务指的是数据库事务（Transaction）。所以交易数据的传输需要满足我们在前一节课提到的**顺序正确性**要求，也就是**既要保证顺序的正确性，也要保证消息处理的一次性**。不了解的同学可以回到上一节课复习一下相关内容，这里就不做复述了。

既然我们这节课讲的是数据传输质量问题，那么我们还是要分析一下可能的异常情况。数据传输已经具有事务性了，还能出什么问题吗？

不知道你有没有在电商促销的时候抢过东西。如果你网页刷新太快，可能会收到远端服务器拒绝访问的消息。这意味着电商服务器觉得你的访问频率过高，临时降低了提供给你的服务质量。

金融行业里的服务器容量也有上限，所以也普遍采取了**限流**这种保护措施。那我们接下来聊聊限流该怎么做？

限流

常用的限流方法有两种，分别是漏桶算法和令牌桶算法。

漏桶算法的原理是消息的生产者将所有请求放到一个容量固定的桶里。消费者会匀速地从桶里消费消息。因为桶的消息满了就丢掉，所以这个桶叫作漏桶。

令牌桶算法和漏桶算法一样，也有一个容量固定的桶，只不过这个桶里装的不是消息，而是令牌。系统会按固定的速度往令牌桶里放令牌，满了之后就会溢出。消费者每处理一个消息都需要消耗一个令牌，所以桶里面的令牌总数决定了处理消息的最快速度，而放令牌的速度决定了处理消息的平均速度。

漏桶算法和令牌桶算法是互联网常见的限流算法，你有兴趣的话可以上网查询相关的实现细节。我们在这里主要看一下这两种算法应该如何选择。

先看看券商应该如何选择。**当两个不同组织之间的金融系统进行对接的时候，接收方一般要假设发送方是恶意的。**因此交易所需要限制券商的消息发送速度，比如一秒钟内最多只能发多少消息。这时候券商可以使用漏桶算法来限制自己对外的消息发送速度。

再来看看交易所应该怎么选择。尽管交易所明文规定了每家券商的速度上限，但是交易所不会相信券商会遵守规则，因此从系统安全的角度考虑，交易所依然会对券商的消息进行限流。这些经过限流的券商流量最终会汇集到一起，再集中处理。

现在又有一个新的问题产生了。虽然对每家的流量都做了限制，但是他们的总流量还是有可能会超出系统承载上限。所以交易所还需要**对总流量做一个限流**。

这时候你就有一个选择，如果你想对总流量做一些微调，就可以选择令牌桶算法，这样就可以通过调整生成令牌的速度来调整处理速度。还有另外一个好处是令牌桶里的总令牌数目代表了系统的峰值处理流量，这样系统还具有一定峰值处理能力。当然你也可以选择安全一点的漏桶算法。

一旦上下游之间做了限流，那么整个系统就需要假设数据会丢失。因此你需要处理好订单发送不出去，或者发送出去后无法被执行这两种情况。

市场数据

交易数据的处理一般具有事务性，所以选择灵活度比较小。市场数据就不一样了，选择面要宽泛很多。

我们先来看看什么是市场数据。这里的市场指的是金融交易市场，**所以市场数据指的是金融市场成交信息。**我们平时关心的股价就是股票买卖双方的成交价格。

交易数据是事务型数据，那**市场数据也是事务型数据吗？这个问题是市场数据处理的最核心问题。**

如果你关心的是自己订单的成交信息，那么这个成交信息是事务类数据。但是在我们的例子里，券商的算法交易平台关心的不是自己的交易信息，而是当前所有人的交易信息，所以算法交易平台并不需要数据有事务的保证，这也意味着**在我们的例子里是允许掉数据的。**

正因为放松了这个假设，我们对市场数据的处理才有了多种不同的选择。因为不同能力的算法交易平台对数据的实效性要求不一样，所以我们先来看看非实时的情况是怎样的。

非实时市场数据

非实时在这里主要指的是那些对延时要求不是特别高的使用场景。这时候消息的传输本着尽量快的原则，稍微慢了几百毫秒或者几秒钟问题不大。绝大多数情况下，你碰到的都是这种非实时的市场数据场景。

我们先看看非实时市场数据的分类，然后再结合前面的例子分析怎么选择数据处理方式。

订阅发布与消息

数据的传输方式分为**订阅发布**（Pub/Sub）和**消息**（Messaging）两大类。在订阅发布的情况下，每个消息的消费者是互相独立的，每个人都需要处理所有消息，并且每个人处理消息的顺序必须是一样的。消息则刚好相反。所有人之间共享所有消息。这意味着每个人处理的只是一部分消息，从他的角度来看消息是断断续续、不连续的。

我们常见的一些数据系统，比如 Apache ActiveMQ、Amazon SQS、IBM WebSphere MQ、 RabbitMQ、RocketMQ，它们在最开始的时候都是按照消息的方式设计的。而 Apache Kafka 和 Google Cloud Pub/Sub 则是按照订阅发布的方式设计的。

请注意这些都只是这些数据系统最开始的设计目标。系统架构在演进过程中可能会同时具有订阅发布和消息的一些能力，比如 Apache Kafka 就是一个典型。

那我们来看看券商需要哪种数据传输方式。如果算法平台只有部分数据的话，可能就会缺失一些重要的历史信号，所以算法交易平台需要所有历史数据。

所以很显然券商需要订阅发布的数据传输方式，通过这个方式从交易所接收数据。这也是为什么现在 Apache Kafka 在金融系统中使用得越来越多的原因。

优化及原理

接下来，我们看看前面例子里的数据传输系统如果想优化，应该怎么做呢？

这里我们需要利用金融数据的一个属性——数据的时效性。**数据的时效性指的是不同时间的数据对你的价值。**

我们先要明确一点的是，对于金融市场数据来说，你永远得不到当前的数据。不管延时有多低，你收到市场数据的时候已经是历史数据了，所以**我们在这里谈论的都是历史数据的**

时效性。

如果所有历史数据对你的价值都是一样高，那么一般来说数据需要尽量完整。相反，如果越接近现在的数据对你的价值越高，那么数据则有可能允许丢失。因为就算丢失了，你只要稍微等一段时间，丢失的数据重要性就会变得很低，这样丢失对你的影响就会很小。

那我们再来看看算法交易平台需要的数据属于哪种类型。算法交易平台属于高频交易类型，它需要根据最近的趋势预测未来的盈利机会。如果数据的时间太久，可能市场上的其他参与者早就利用这些信息赚过了钱，这种时间太久的数据就没剩多少价值了。所以**算法交易平台的数据属于具有时效性的数据，允许部分缺失。**

我在这节课最开始提到过，金融系统不是所有地方的数据要求都很高，这里就是一个例子。所以你一定要结合业务特点来选择合适的系统架构。那知道了这个特点，我们就可以针对性地调整数据传输系统的容灾能力了。

比如说 Apache Kafka 默认带有一定容灾功能。一般要求部署 3 个节点，其中一个是主节点，另外两个是备份节点。

每当 Kafka 收到数据后，会将数据先同步给备份节点，这个备份的过程需要一定时间。备份的节点个数叫作同步数（ISR，In Sync Replica）。我们可以将 Apache 的同步数设为 0，这样我们就能牺牲掉部分不重要的容灾能力，换来更快的处理速度。

当然了，就算数据可以丢失，也要避免这种情况频繁出现。所以，你要建立合理的监控机制，当数据频繁丢失的时候能及时反应。

实时市场数据

金融行业的非实时市场数据的处理和互联网行业的方式比较接近，用的系统架构也比较类似，所以你会有种似曾相识的感觉。接下来我们看看实时市场数据的处理，这和互联网的处理区别比较大。

特殊部署

实时市场数据指的是对延时要求非常高的数据场景。低延时的系统维护对交易所本身是一笔很大的开销，而且因为能支持的用户数有限，所以一般都需要收费，也就是常说的席位

费。

席位费也分不同的等级。当你交完席位费后，一般就能连到交易所，获取一些低延时的数据，同时还有可能得到更详细的数据内容。

一般来说你的服务器和交易所有一定的物理距离。我们都知道光的传播有速度上限，所以这个物理距离会导致一定的延时。比如跨省传播数据的话，延时可能在几毫秒到几十毫秒左右。

如果你觉得这几十毫秒很重要，你可以再用更多的钱来解决这个问题。交易所一般会提供同机房主机服务，你可以将自己的系统部署在交易所的机房内。这样你的系统和交易所系统的物理距离只有几米左右，光速带来的影响基本可以忽略不计。

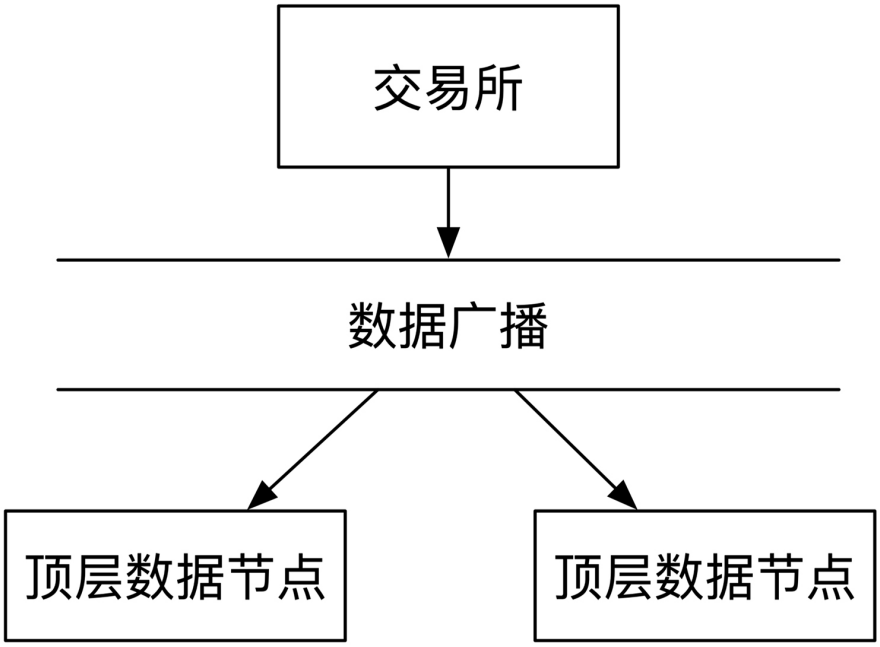
不过，因为交易所机房的物理机器有限，你需要和其他有钱的金融公司竞争这为数不多的位置，可能需要付出天价的运营费用。

数据分发

从前面讲的特殊部署你可以发现，实时市场数据的消费者之间并不是平等关系。谁的钱多，谁的延时就低。这表示**实时数据分发的系统架构是一个层级结构**，越接近上层的人收到的消息越快。

我们先从交易所出发。当交易所生成完数据之后，会将数据传送给顶层的数据节点。这一层会部署多个节点以防单点故障。

假如是按照 Apache Kafka 的模式，交易所数据应该发给一个主节点，主节点负责和备份节点之间通讯。但是这样会有一个网络延时，所以交易所采用的是局域网内广播的方式，所有第一层节点都会同时收到所有数据信息，如下图所示：



顶层数据节点收到数据之后会做两件事情，分别是推送给优先级最高的 VIP 客户，以及推送给下一层的数据分发节点。

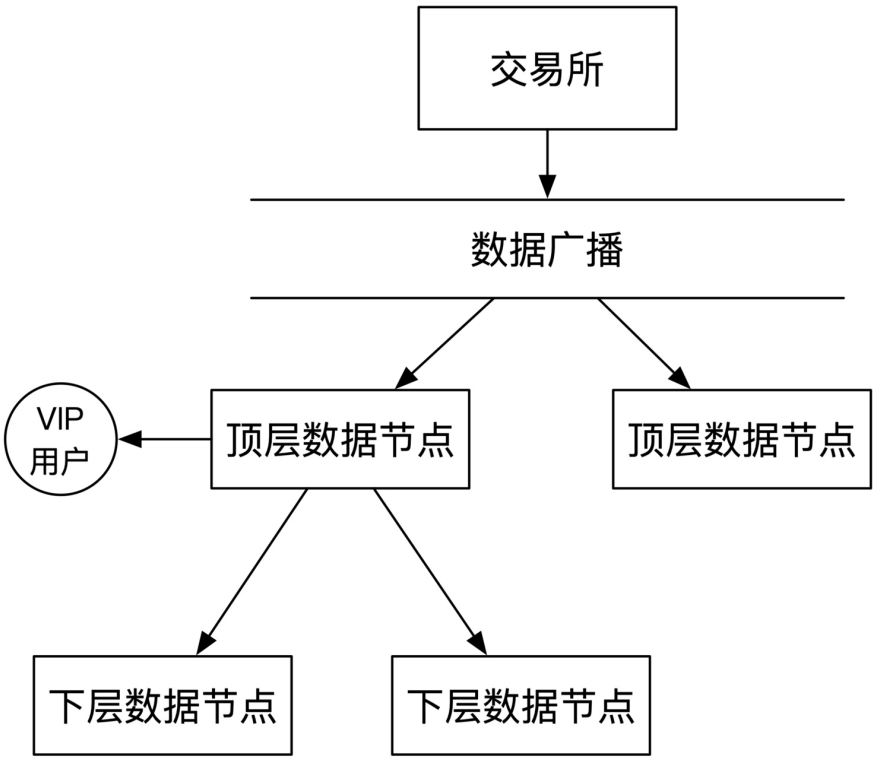
这里需要注意的是，低延时架构优化的是系统延时，而不是系统的吞吐量。互联网常见的**消息系统普遍采用的是消费者定时拉取数据的模式。**这样做的优点是能支持大量的数据消费者，但也会带来问题，就是两次拉取之间有一定的时间间隔。比如 Apache Kafka 的默认客户端就是这种行为。

数据的实时推送会消耗很多推送端的硬件资源，但是由于交易所的 VIP 客户数目很少，实时推送对系统的影响可控，所以数据可以通过顶层数据节点直接推送给用户。

其实我们还可以从经济学的角度来考虑这个问题。VIP 席位费的总盈利是总 VIP 客户数乘以席位费，VIP 客户数减少之后，VIP 之间的价格竞争会更激烈，所以席位费会增加。因为总盈利是这两者的乘积，很有可能总盈利会因为席位费的增加而大幅增加。

所以从利润的角度来考虑，系统也不一定需要支持很多的 VIP 用户。这也说明了我们在选择系统架构的时候，一定要结合业务来一起考虑。当然了，为了让这一点能成立，VIP 席位需要具备一定的**价格弹性**（Price Elasticity），你有兴趣的话可以了解一下经济学的相关内容，比如诺贝尔经济学奖获得者 Paul Samuelson 写的《经济学》。

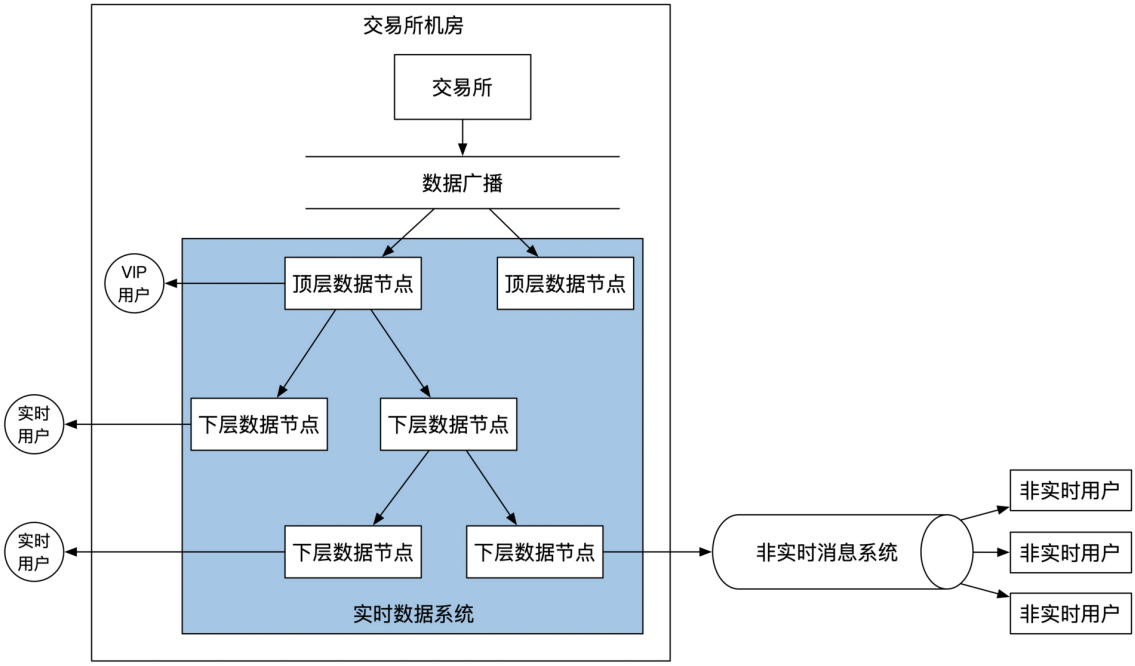
数据除了顶层的 VIP 用户之外，还有非 VIP 的付费实时数据用户。但是由于数据已经有了延时，这时候不一定需要用到数据广播，所以我们可以选择让顶层节点推送数据给下层节点。这时候的架构示意图如下：



以此类推，每一层都能往下继续分发数据。这时候数据分发的对象有交易所的非 VIP 付费客户，也有其他的数据节点。

当交易所解决完所有付费用户的实时数据推送问题之后，接下来要解决怎么把**实时数据变为非实时数据**，也就是怎么让非付费用户也能访问到数据。

其实这个过程很简单，只要将某一层的实时数据节点对接到非实时数据系统就可以了。这时候**数据由实时的推送方式变为非实时的拉取方式。**更新后的系统架构如下：



数据压缩

实时数据系统大部分的时间都花费在数据的解码和编码上。所以要想速度快，首先要数据量小，这也是金融系统架构和互联网架构很大的一个区别。

所以，你平时经常能见到的 Json 或者 XML 格式的数据一定不能用在实时数据系统里。因为这些传输格式是为人准备的，里面有多余的信息，所以你需要换成只有机器看得懂的二进制表达方式。

如果要求不高的话，一般来说 Google Prototol Buffer 协议就足够了。这个协议会按照你定义好的二进制表现形式来进行编码，能对数据进行很大幅度的压缩。

在要求更高的金融场景下，普遍会使用金融行业专用的 **FIX 通讯协议**。这个协议定义了通讯规则，同时也定义了数据传输方式。

市场数据有一个显著特点是，连续两个数据之间大部分内容都是一样的。比如说你去比较连续两个股票价格信息数据，就会发现它们很有可能只有价格这一个指标会有变化，其他的信息完全一样。

FIX 协议就利用了这个特性，在很多情况下**只需要传输数据变动的部分**，这样就能减少很多数据传输量。其实这个设计思想和视频压缩算法非常类似，视频压缩的时候是以关键帧为

基准，其他帧只存储相对于关键帧的变化。

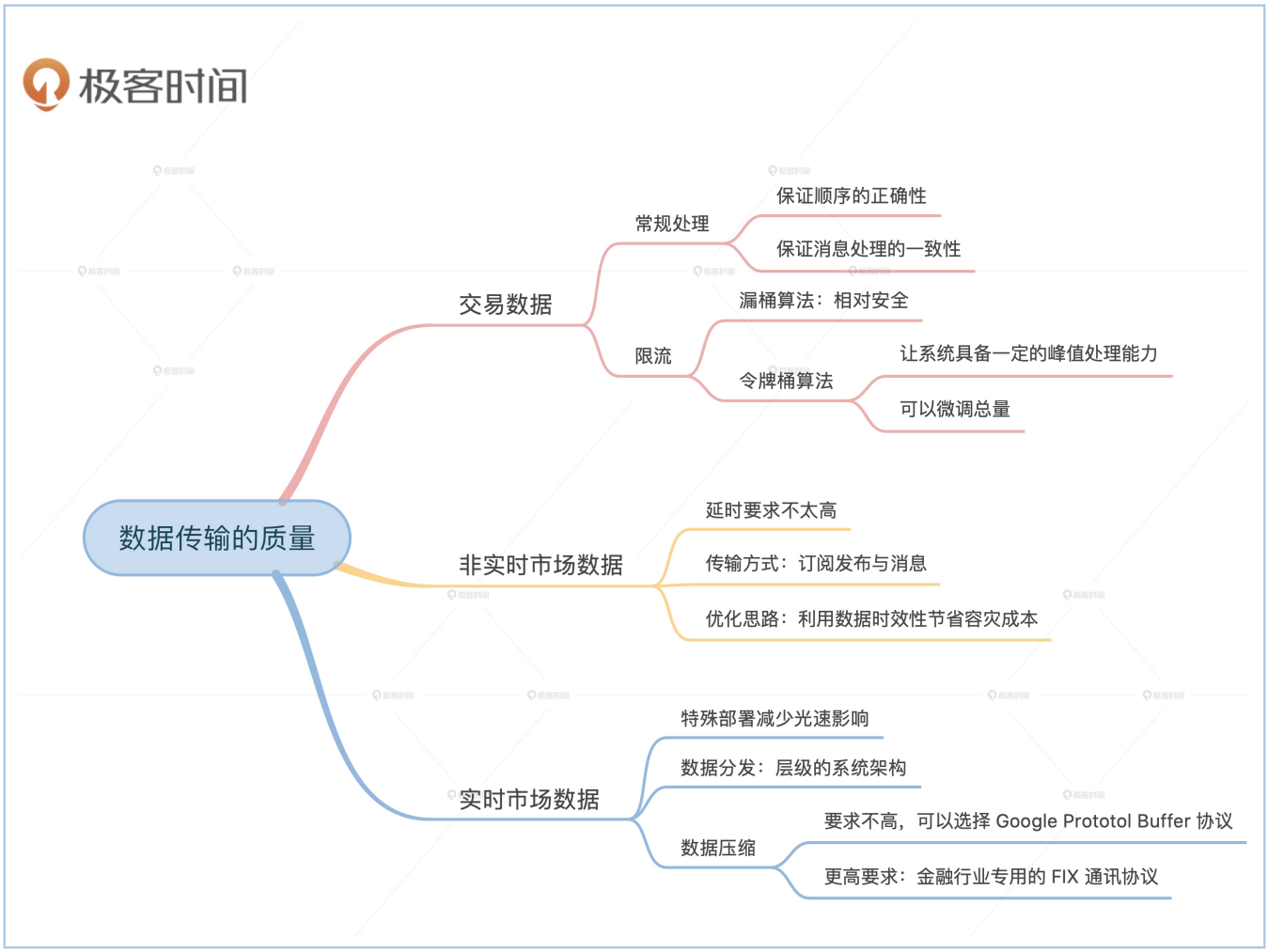
小结

这节课里我们学习了怎么做好金融数据的传输。

金融数据分为交易数据和市场数据两种。金融交易数据的处理和互联网处理方法非常类似，在处理的时候需要做好数据限流的架构选型。

市场数据的处理分为非实时和实时两种。非实时市场数据的处理也和互联网处理方法类似，在处理的时候，对订阅发布和消息这两种不同架构选择，我们要做好区分。因为市场数据具有实效性，我们可以容忍偶然的数据丢失，这也给了数据系统一个很大的优化空间。

实时市场数据的消费者分为不同的级别。实时数据系统的架构和用户一样，也是分为不同级别。数据会层层分发下去，不同层级有不同的延时情况和部署方案。实时系统的优化主要体现在数据压缩上，金融行业有自己特有的 FIX 二进制通讯协议。



思考题

实时数据系统的数据节点通常都是价格昂贵的机器。这些机器的处理速度极快。交易所机器运行太快了之后，会导致推送给实时用户的数据量过大，用户来不及处理。你这时候应该怎么处理这两者速度不一致导致的问题呢？

欢迎你在留言区和我交流互动。如果这节课对你有帮助的话，也欢迎你分享给朋友、同事，一起学习和讨论。

提建议

上一篇 08 | 计算结果的正确性：怎么保证计算结果是正确的？

下一篇 10 | 数据存储的合理性：金融业务可以不用关系型数据库吗？

精选留言 (2)

写留言

tt

2021-01-11

平衡速度不同的节点，就像CPU和主内存之间的差异，往往都是引入一个中间层比如一级缓存、二级缓存来解决，这些缓存对外是透明的。

这么来看，实时数据推送的层级结构与CPU缓存、内存的层级结构非常类似:

...

展开

4

webmin

2021-01-19

按时间窗合并明细数据，对于不同能力的用户给不同窗口大小的数据，比如：统计10，20，50毫秒窗口。