01 | 实时流计算的通用架构

为什么把本课时作为第一课时呢？因为通过本课时，你将构建起对流计算技术和系统的整体认识，这样既可以为后面的课时打下基础，又可以对设计和开发实时流计算应用有所启发。

任何一个系统的产生，都是为了解决一个具体的问题。实时流计算技术的诞生，就是为了更快更完整地获取数据，更快更充分地挖掘出数据价值。

我们不妨先来看看几个实时流计算技术的应用场景。根据这些场景，我们可以大体上知道它的通用架构。

实时流计算技术应用场景

图 1 是某打车软件公司交通热点路段分析及可视化系统的示意图。

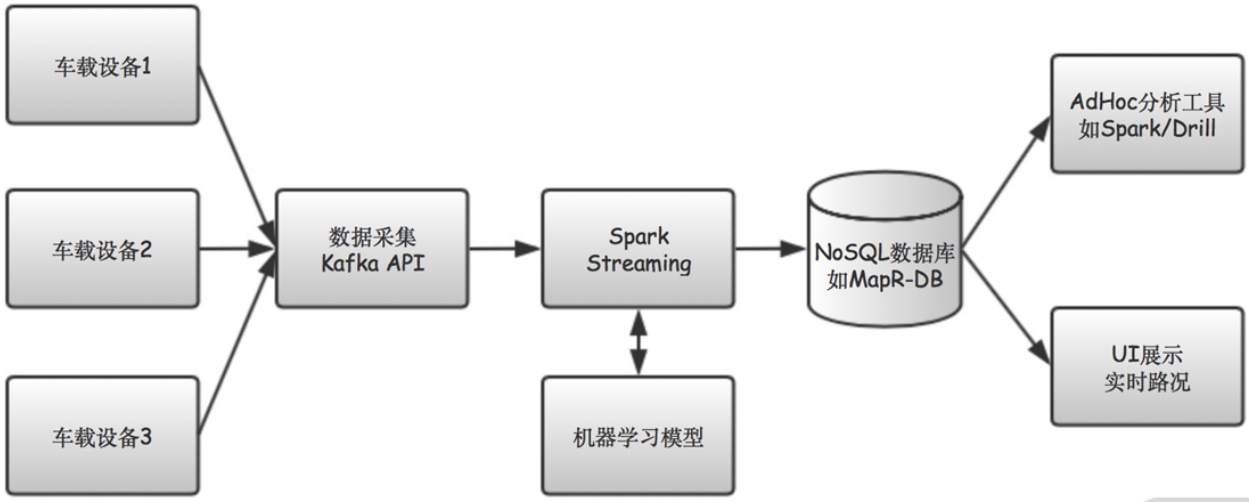


图 1 基于 Spark Streaming 的交通热点路段分析及可视化系统

在这个系统中，从车载设备上发出的数据，被一个基于 Kafka API 的数据采集模块接收，然后发送到 Spark Streaming 模块进行处理，并且还使用机器学习模型进行分析，然后分析的结果以 JSON 的形式存储到数据库中，并提供给可视化模块进行展示和分析。

我们再来看另一个金融风控的例子。图 2 是一个基于 Flink 的实时欺诈检测平台。

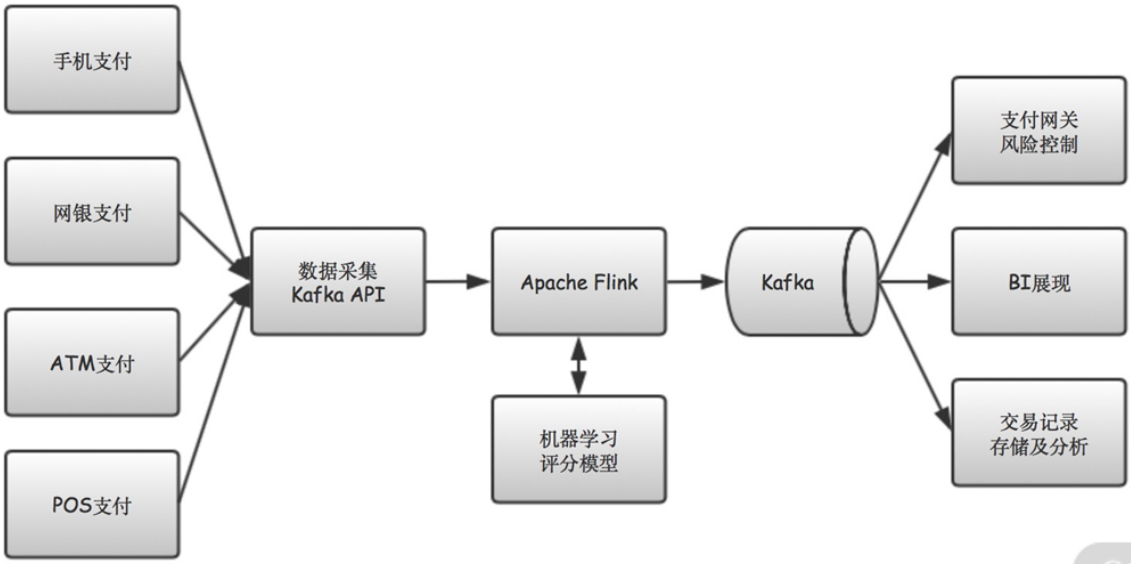


图 2 基于 Flink 的实时欺诈检测平台

在这个平台中，从手机等各种支付渠道产生的交易数据，被数据采集服务器收集起来，并发送到Kafka。然后 Flink 从 Kafka 中将交易数据取出来，采用基于机器学习的风控模型，进行风险分析和评估。然后分析的结果再次发送到 Kafka，后续支付网关就可以根据这些交易的欺诈风险等级，来允许或阻止交易进行。

实时流计算系统通用架构

比较上面两个场景的流计算系统组成，我们不难发现这些系统，都包含了五个部分：**数据采集、数据传输、数据处理、数据存储和数据展现**。

事实上，也正是这五个部分，构成了一般通用的实时流计算系统，它们之间的组成关系如下图 3 所示。

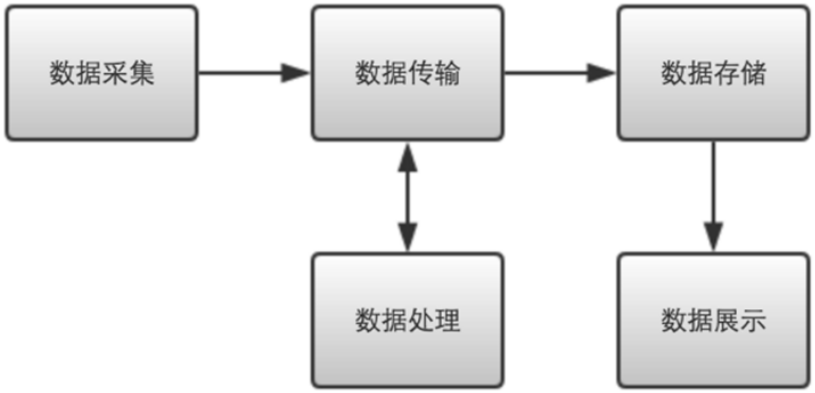


图 3 实时流计算系统的组成部分

在上图 3 中，数据采集模块用于接收来自各种数据源的数据，比如互联网上的各种移动设备、物联网上的各种传感器，内部网络中部署在各个服务模块上的日志代理等。数据采集模块收集到这些数据后，对数据进行一定整理，再将数据发送到数据传输模块。

数据传输模块通常是消息中间件，比如 Kafka，之后再由数据处理模块从数据传输模块中取出数据来进行处理。数据处理模块是流计算系统的核心，在这个模块中会实现流计算应用的各种业务功能。

之后，计算结果被重新发送到数据传输模块，并由数据存储模块取出后，保存到各种类型的数据库中。最后，数据展示模块会通过 API 或者 UI 的方式对结果进行展示。

下面我来逐一详细介绍下通用架构的五个部分。

数据采集

俗话说“巧妇难为无米之炊”，有数据了，我们才能进行流计算，所以我们先来看看应该怎样采集数据。

数据采集，就是从各种数据源收集数据的过程，比如浏览器、手机、工业传感器、日志代理等。怎样开发一个数据采集服务呢？最简单的方式，就是用 Spring Boot 开发一个 REST 服务，这样，我们就可以用 HTTP 请求的方式，从浏览器、手机等终端设备，将数据发送到数据采集服务器。

这么一看，数据采集服务器似乎很简单！其实不然，这中间还是有很多问题需要认真考虑。如果考虑不周的话，很可能你花冤枉钱买了许多服务器，但是系统的性能却依旧十分可怜。

为了避免在以后的开发中出现这种问题，这里我想跟你分享下我在日常开发 Web 服务时考虑的五个关键点。

- 第一点是**吞吐量**。我们一般用 TPS（Transactions Per Second），也就是每秒处理事务数，来描述系统的吞吐量。当吞吐量要求不高时，选择的余地往往更大些。你可以随意采用阻塞 IO ，或非阻塞 IO 的编程框架。但是当吞吐量要求很高时，通常就只能选择非阻塞 IO 的编程框架了。如果采用阻塞 IO 方式时，需要开启数千个线程，才能使吞吐量最大化，就可以考虑换成非阻塞 IO 的方案了。
- 第二点是**时延**。当吞吐量和时延同时有性能要求时，我一般是先保证能够满足时延要求，然后在此基础上，再尽可能提高吞吐量。如果一个服务实例的吞吐量，满足不了要求，就部署多个服务实例。对于互联网上的应用，如果吞吐量很大，为保证时延，还需要使用类似于 CDN 的方案。
- 第三点是**发送方式**。数据可以逐条发送，也可以批次发送。相比逐条发送而言，批次发送每次的网络 IO 耗时更多，为了提升接收服务器的吞吐能力，我一般也会采用 Netty 这样的非阻塞 IO 框架。
- 第四点是**连接方式**。使用长连接还是短连接，一般由具体的场景决定。当有大量连接需要维持时，就需要使用非阻塞 IO 服务器框架，比如 Netty。而当连接数量较少时，采用长连接和连接池的方案，一般也会非常显著提升请求处理的性能。
- 第五点是**连接数量**。如果数据源相对固定，比如微服务之间的调用，那我们可以采用长连接配合连接池的方案，这样一般会非常显著地提升请求处理的性能。但当数据源很多或经常变化时，应该将连接保持时间（Keep Alive Timeout）设置为一个合理的值。

总的来说，在大多数情况下，数据接收服务器选择诸如 Netty 的非阻塞 IO 方案，都会更加合适。

数据采集之后，我们一般还需要做些简单的处理，比如提取出感兴趣的字段，或者对字段进行调整等，然后再将调整好的字段，组成格式统一的数据，比如 JSON、AVRO、Protobuf 等。最后将整理好的数据，发往到数据传输系统。

数据传输

我们这里说的数据传输，是指流数据在各个模块间流转的过程。

流计算系统中，一般是采用消息中间件进行数据传输的，比如 Apache Kafka、RabbitMQ 等，在微服务系统中一般是采用 HTTP 或 RPC 的方式进行数据传输。这是流计算系统与微服务系统最明显的区别。

在选择消息中间件时，你需要重点考虑五个方面的问题：**吞吐量、时延、高可用、持久化和水平扩展**。为什么呢?

这是因为，**吞吐量和时延，通常是由产品和业务需求决定**的。比如，产品要求系统能够支持 10K 的 TPS，并且 99% 消息的时延不能超过 100ms，那我们部署的消息中间，吞吐量一定要显著超过 10K，时延要显著低于 100ms，因为还需要留出非常大的空间，来处理业务逻辑。

而高可用和持久化，则是保证我们系统，能够正确稳定运行的重要因素。

高可用是指消息中间件的一个或多个节点，在发生故障时，仍然能够持续提供正常服务。比如双 11 的零点，大家都在拼命剁手，此时如果因为一个节点磁盘写满，而导致整个系统不能下单，那真的就是瞬间错失一个亿的小目标了。

持久化则是指消息中间件里的消息，写入磁盘等存储介质后，重启时消息不会丢失。比如在消息中间件 Kafka 中，同一份数据在不同的物理节点上，保存多个副本，即使一个节点的数据，完全丢失，也能够通过其他节点上的数据副本，恢复出原来的数据。

水平扩展也是个非常重要的考量因素。当业务量逐渐增加时，原先的消息中间件处理能力逐渐跟不上，这时需要增加新的节点，以提升消息中间件的处理能力。比如 Kafka 可以通过增加 Kafka 节点和 topic 分区数的方式水平扩展处理能力。

总的来说，数据传输系统就像人体的血管，承载了实时流计算系统中数据的传输。一个高吞吐、低时延、支持高可用和持久化，且能水平扩展的数据传输系统，是构建优秀实时流计算应用的基础。目前，像 Kafka 和 Pulsar 都是不错的数据传输系统选择。

数据处理

接下来，我们来看下流计算系统的核心模块，即数据处理。为什么说数据处理，是流计算系统的核心呢？这是因为在数据处理模块，我们将实现各种业务功能，比如数据过滤、聚合计算、CEP、模型训练等。

我们构建实时流计算系统的目的，就是为了解决具体的业务问题。总的来说，这些业务问题可以分为以下四类。

- 第一类是**数据转化**。数据转化包括对流数据的抽取、清洗、转换和加载。比如使用 filter 函数过滤出符合条件的流数据，使用 map 函数给流数据增加新的字段。再比如更复杂的 Flink SQL CDC，也属于数据转化的内容。
- 第二类是在**流数据上，统计各种指标**，比如计数、求和、均值、标准差、极值、聚合、关联、直方图等。
- 第三类是**模式匹配**。模式匹配是指在流数据上，寻找预先设定的事件序列模式。比如我们常说的 CEP，也就是复杂事件处理，就属于模式匹配。
- 第四类是**模型学习和预测**。基于流的模型学习算法，可以实时动态地训练或更新模型参数，继而根据模型做出预测，能更加准确地描述数据背后当时正在发生的事情。

数据处理是流计算的核心，也是一个流计算应用开发人员最应该掌握的知识点。这部分的内容是非常丰富且有一定难度的，我将在本课程的模块三中，对数据处理问题进行详细讲解。

数据存储

使用实时流计算技术，一顿操作猛如虎，结果不记录，或者不输出结果的话，那就是算了个寂寞。所以数据处理过程中，必然会涉及，数据存储的问题。而数据存储，是一个非常麻烦的问题，特别是在实时流计算领域，这种大数据、低时延、高吞吐的场景，对我们的数据存储方案，挑战是非常大的。

不知道你是否考虑过这个问题，为什么软件行业，有那么多不同种类的数据库？MySQL、MongoDB、Redis、HBase、ElasticSearch、CockroachDB……随便想一下，就可以列举出数十种数据库。

这是因为每种数据库，其实都有其擅长的使用场景，没有一种数据库能够在所有场景下都能胜任，所以我在这里先抛砖引玉，针对实时流计算中几种最常见的场景，讲解下应该选择怎样的存储方案。

在实时风控场景下，我们经常需要计算诸如“过去一天同一设备上登录的不同用户数”这种类型的查询。在数据量较小时，使用传统关系型数据库和结构化查询语言是个不错的选择。

但当数据量变得很大后，这种基于关系型数据库的方案会变得越来越吃力，直到最后根本不可能在实时级别的时延内完成计算。这个时候，如果采用像 Redis 这样的 NoSQL 数据库并结合优化的算法设计，就能够做到实时查询，并获得更高的吞吐能力。所以相比传统 SQL 数据库，实时流计算中会更多地使用 NoSQL 数据库。

很多时候，我们需要将实时流计算的状态或者结果存储下来，以供其他服务根据一个或多个健，来查询一条特定的，实时计算记录。那这个时候，我们可以选择像 MongoDB 这样的 NoSQL 数据库。当然，这个时候如果在 MongoDB 之上，再配上一个 Redis 缓存也是极好的。

还有些时候，我们需要在 UI 上展现实时流计算的结果。不知道其他人是怎样想的，反正在我这个后端开发眼里，那些产品同学总喜欢在 UI 上设计一些“莫名其妙”的交互式查询，比如任意可选的查询条件和查询方式。那这个时候，我们选择的存储方案，就一定不能太“僵硬”，此时采用像 ElasticSearch 这样搜索引擎一类的存储方案，一定是个明智的选择。

总的来说，在相对复杂的业务场景下，实时流计算可能只是系统中的一个环节。我们需要针对不同的计算类型和查询目的，选择合适的存储方案。当一种数据库满足不了业务的需求时，我们还会将相同的数据，存入多种不同的存储。毕竟到目前为止，还没有一种能称之为“银弹”的数据库。

数据展现

最后就是数据展现模块了，数据展现是将数据呈现给最终用户的过程。

数据展现的形式，可以是 API，也可以是 UI。

- API 相对简单，比如用 Spring Boot 就很容易开发一个 REST API 服务。
- UI 目前越来越多的是采用 Web UI 的方式。

基于 Web 的 UI 有很多优点。一方面，其部署和访问都非常简单，只需要启动 Web 服务，然后在浏览器访问即可。另一方面，各种丰富的前端框架和数据可视化框架，为开发提供了更多的便利和选择，比如前端常用的框架，就有 React、Vue、Angular 等，然后

常用的数据可视化框架，则有 ECharts、D3.js 等。

由于数据展示，更加偏向于前端（包括 UI 设计、JS、CSS 和 HTML 等），这与实时流计算的主体，并无太强关联，所以我在本课程中，不会专门讨论数据展示的内容。

小结

今天，我依据几种不同场景的流计算系统，总结了一个通用的流计算系统架构，然后带你了解了这个架构中各个模块在整个系统中起到的作用。

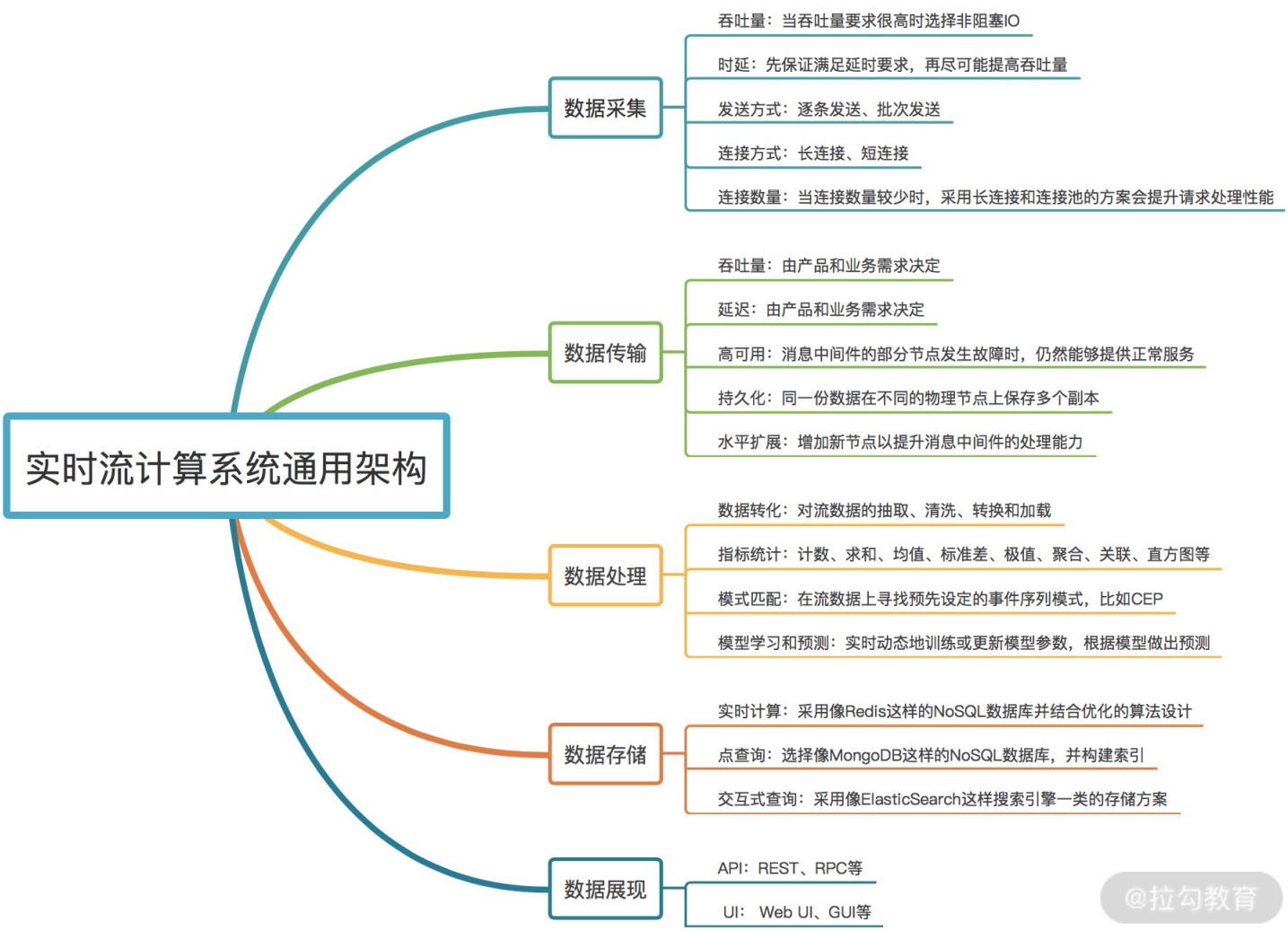
相信你在以后开发实时流计算应用时，十有八九会用到上面这种架构，并且一定会碰到我跟你讲的这些问题，尤其是数据采集、数据处理和数据存储三个模块。

- 数据采集模块的难点，一定会在高并发和高吞吐场景下暴露出来，这点需要你对 NIO 和异步编程有非常深刻的理解。
- 数据处理模块的难点，则主要表现在与业务的贴合。这要求你对流计算能够解决哪些问题有比较深刻的理解，并需要熟练掌握解决这些问题的算法。
- 数据存储模块的难点，则主要表现在能否根据具体的使用场景，选择最合适的存储方案。而实时流计算中，会涉及多种不同类型的数据存储问题。

不过不用担心，在接下来的课程中，我将会为你详细讲解 NIO 和异步编程的问题。至于数据处理和数据存储的内容，则会在本课程的模块三进行详细讨论。

那么你在工作中，有没有遇到比较难解决的实时计算或流计算问题呢？你可以先把这些问题放在留言区，我会时刻关注，并在后续文章为你着重强调哦！

> 本课时精华：



PB 级企业大数据项目实战 + 拉勾硬核内推，5 个月全面掌握大数据核心技能。点击链接，全面赋能！