阿里云
表格存储

数据模型

文档版本：20181008

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止：** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告：** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明：** 您也可以通过按**Ctrl** + **A**选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 **确定**。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

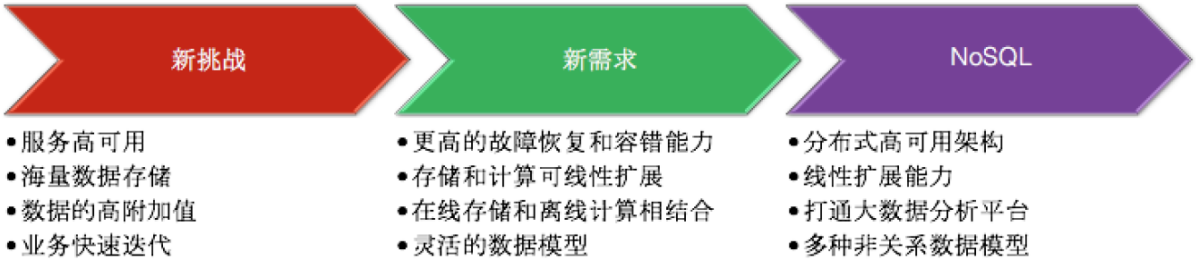
目录

法律声明.....I
通用约定.....I
1 前言.....1
2 Wide Column.....3
2.1 模型介绍.....3
2.2 基本概念.....4
2.2.1 数据版本和生命周期.....4
2.2.2 主键和属性.....5
2.2.3 读/写吞吐量.....7
2.3 数据访问API.....9
3 Timeline.....11

1 前言

Table Store 是阿里云自研的一款分布式 NoSQL 数据库，提到 NoSQL 数据库，现在对很多应用研发而言都已不再陌生。当前很多应用系统底层不再仅依赖于关系型数据库，而是会根据不同的业务场景来选择不同类型的数据库，例如缓存型 KeyValue 数据存储在 Redis、文档型数据存储在 MongoDB、图数据会存储在 Neo4J 等。

传统的关系型数据库很难承载如此海量的数据，需要一种具备高扩展能力的分布式数据库。但基于传统的关系数据模型，实现高可用和可扩展的分布式数据库非常困难。如果能打破关系模型，以一种更简单的数据模型对数据建模、弱化事务和约束、以高可用和可扩展、能更好地满足业务需求为设计理念，基于这样的理念推动了 NoSQL 的发展。



NoSQL 具有以下特征：

- 多数据模型

  为满足不同数据的需求，提供了多数据模型供您选择，例如 KeyValue、Document、Wide Column、Graph 以及 Time Series 等。NoSQL数据库打破了关系模型的约束，选择了多元的发展方向。多数据模型更贴近不同场景的实际需求。

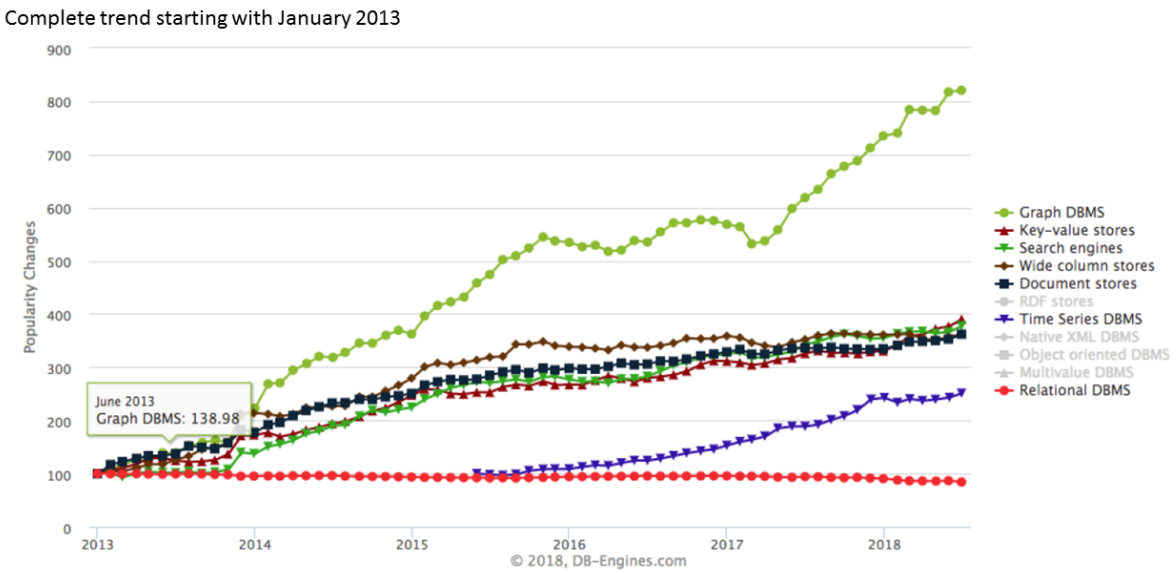
- 高并发、低延迟

  NoSQL 的设计目标是面向在线业务提供高并发、低延迟的访问。

- 高可扩展

  为应对爆发的数据量增长，可扩展是核心的设计目标之一。

DB-Engines 旨在收集和介绍数据库管理系统（DBMS）方面的信息。下面展示了 DB-Engines 收集到的各类 NoSQL 数据库从 2013 年到 2018 年的发展趋势。



从 DB-Engines 的发展趋势来看，各类 NoSQL 数据库在近几年都处于一个蓬勃发展的状态。阿里云 Table Store 作为一款分布式 NoSQL 数据库，在数据模型上选择了多模型的架构，同时支持 Wide Column 和 Timeline。

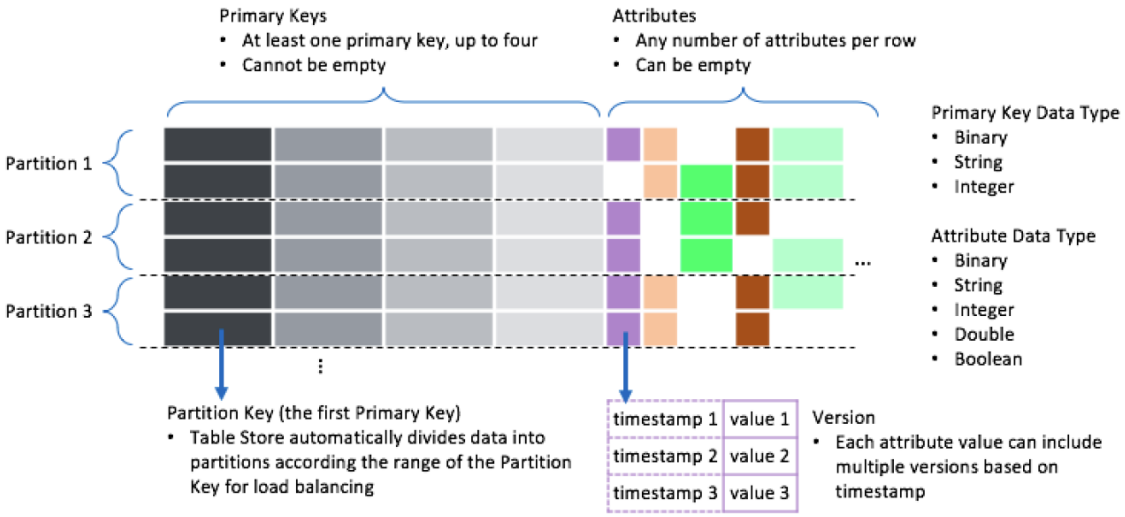
Wide Column 模型由 Bigtable 提出，后被其他同类型系统广泛应用的一种经典模型。目前绝大部分半结构化、结构化数据都存储在该模型系统中。除了 Wide Column 模型外，我们推出了另一种全新的数据模型 Timeline。Timeline 模型是一种用于消息数据的新一代模型，适用于 IM、Feeds 和物联网设备消息下推等消息系统中消息的存储和同步，目前已被广泛使用。接下来，我们详细了解下这两种模型。

2 Wide Column

2.1 模型介绍

Wide Column 模型对比关系模型区别如下：

- Wide Column 模型：三维结构（行、列和时间）、schema-free、宽行、多版本数据以及生命周期管理。
- 关系模型：二维（行、列）以及固定的Schema。



Wide Column 模型由以下几个部分组成：

- 主键（Primary Key）：每一行都会有主键，主键会由多列（1-4列）构成，主键的定义是固定Schema，主键主要用于唯一区分一行数据。
- 分区键（Partition Key）：主键的第一列称为分区键，分区键用于对表进行范围分区。每个分区会分布式的调度到不同的机器上进行服务。在同一个分区键内，我们提供跨行事务。更多信息，请参见主键和属性文档。
- 属性列（Attribute Column）：一行中除主键列外，其余都是属性列。属性列会对应多个值，不同值对应不同的版本，一行可存储不限个数个属性列。
- 版本（Version）：每一个值对应不同的版本，版本的值是一个时间戳，用于定义数据的生命周期。
- 数据类型（Data Type）：Table Store 支持多种数据类型，包含 String、Binary、Double、Integer 和 Boolean。

- 生命周期（Time To Live）：每个表可定义数据生命周期。例如生命周期配置为一个月，则该表数据中在一个月之前写入的数据就会被自动清理。数据的写入时间由版本来决定，版本一般由服务端在数据写入时根据服务器时间来定，也可由应用指定。更多详情，请参见数据版本和生命周期文档。
- 最大版本数（Max Versions）：每个表可定义每一列最多保存的版本数，用于控制一列的版本的个数。 当一个属性列的版本个数超过 Max Versions 时，最早的版本将被异步删除。

2.2 基本概念

2.2.1 数据版本和生命周期

最大版本数

最大版本数（Max Versions）是数据表的一个属性，表示该数据表中的属性列能够保留多少个版本的数据。当一个属性列的版本个数超过 Max Versions 时，最早的版本将被异步删除。

建表后，您可以通过 UpdateTable 接口动态更改数据表的 Max Versions。

说明：

- 超过 Max Versions 的数据版本为无效数据，即使数据还没有被真正删除，该数据对用户已经不可见，无法读出。
- 当调小 Max Versions 时，如果数据版本个数超过新设的 Max Versions，最早的版本会被系统异步删除。
- 当调大 Max Versions 时，如果以前版本个数超过旧的 Max Versions 但还没有被系统删除的，数据会被重新读出来。

有效版本偏差

有效版本偏差（Max Version Offset）是数据表的一个属性，单位为秒。

为了防止非期望的写入，服务端在处理写请求时会对属性列的版本号进行检查。当版本号小于当前写入时间减去 Max Version Offset，或者大于等于当前写入时间加上 Max Version Offset 的值时，该行数据写入失败。

属性列的有效版本范围为：[数据写入时间 - 有效版本偏差，数据写入时间 + 有效版本偏差)。数据写入时间为 1970-01-01 00:00:00 UTC 时间到当前写入时间的秒数。属性列版本号为毫秒，其除以 1000 换算成秒之后必须属于这个范围。

例如，当数据表的有效版本范围为 86400（一天），在 2016-07-21 00:00:00 UTC 时，只能写入版本号大于 1468944000000（换算成秒之后即 2016-07-20 00:00:00 UTC）并且小于 1469116800000（换算成秒之后即 2016-07-22 00:00:00 UTC）的数据。当某一行的某个属性列版本号为 1468943999000（换算成秒之后即 2016-07-19 23:59:59 UTC）时，该行数据写入失败。

- 建数据表时，用户若不设置有效版本偏差，将使用默认值 86400。
- 建表后，可以通过 UpdateTable 接口动态更改有效版本偏差。
- 有效版本偏差为非 0 值，可以大于 1970-01-01 00:00:00 UTC 时间到当前时间的秒数。

数据生命周期

数据生命周期（Time To Live，简称 TTL）是数据表的一个属性，即数据的存活时间，单位为秒。表格存储会在后台对超过存活时间的数据进行清理，以减少用户的数据存储空间，降低存储成本。

- TTL 由用户在建表时进行设置，如果希望数据永不过期，将其设置为 -1。
- 建表后，可以通过 UpdateTable 接口动态更改 TTL。
- TTL 的单位为秒，例如期望过期时间为 30 天，TTL 应设置为 2592000（即 30 * 24 * 3600）。

假设数据表的 TTL 设置为 86400（一天），在 2016-07-21 00:00:00 UTC 时，该数据表上所有版本号小于 1468944000000（除以 1000 换算成秒之后即 2016-07-20 00:00:00 UTC）的属性列都将过期，系统会自动清理这些过期的数据。

说明：

- 超过 TTL 的过期数据为无效数据，即使数据还没有被真正删除，该数据对用户已经不可见，无法读出。
- 当调小 TTL 时，可能会有数据因为 TTL 变小而过期，这部分数据会被系统异步删除。
- 当调大 TTL 时，如果有版本号在上个 TTL 之外的数据还没有被系统删除，数据会被重新读出。

2.2.2 主键和属性

主键

- 主键是表中每一行的唯一标识。主键由 1 到 4 个主键列组成。
- 创建表的时候，必须明确指定主键的组成、每一个主键列的名字和数据类型以及它们的顺序。
- 主键列的数据类型只能是 String、Integer 和 Binary。如果为 String 或者 Binary 类型，长度不超过 1 KB。

属性

属性存放行的数据。每一行包含的属性列个数没有限制。

版本号

在一个属性列上，当写入的版本数超过数据表的最大版本数时，较早版本的数据会被删除，只保留最新的 Max Versions 的版本数。

在写入数据时可以指定属性列的版本号，如果不指定版本号，服务端会将当前时间的毫秒单位时间戳（从 1970-01-01 00:00:00 UTC 计算起的毫秒数）作为属性列生成版本号。比如属性列版本号为 1468944000000（即 2016-07-20 00:00:00 UTC），当数据表的 TTL 设置为 86400（一天）时，该版本的数据将会在 2016-07-21 00:00:00 UTC 过期，随后会被后台系统自动删除。

读取一行数据时，可以指定每列最多读取多少版本或者读取的版本号范围。

说明：

- 版本号的单位为毫秒，在进行 TTL 比较和有效版本偏差计算时，需要除以 1000 换算成秒。
- 当数据的版本号（即时间戳）完全由服务端决定时，写入的数据在写入后经过 TTL 秒后会被系统清理。
- 为了防止无效的写入，写入过期数据将会直接失败。例如在 2016-07-21 00:00:00 向 TTL 为 86400 的数据表中写入版本号小于 1468944000000（即 2016-07-20 00:00:00 UTC）的数据将会直接失败。
- 为了防止错误的写入，写入的属性列的版本号换算成秒后，需要在 [数据写入时间-有效版本偏差，数据写入时间+有效版本偏差) 的范围内。

列名的命名规范

主键列和属性列遵循如下命名规范：

- 必须由英文字母、数字或下划线（_）组成
- 首字符必须为英文字母或下划线（_）
- 大小写敏感
- 长度在 1~255 个字符之间

列值类型

表格存储支持 5 种类型的列值：

数据类型	定义	是否可为主键	大小限制
String	UTF-8，可为空	是	为主键列时最大为 1 KB，为属性列时请参考限制说明。
Integer	64 bit，整型	是	8 Bytes
Double	64 bit，Double 类型	否	8 Bytes
Boolean	True/False，布尔类型	否	1 Byte
Binary	二进制数据，可为空	是	为主键列时最大为 1 KB，为属性列时请参考限制说明。

分区键

组成主键的第一个主键列又称为分区键。表格存储会根据表中每一行分区键的值所属的范围自动将这一行数据分配到对应的分区和机器上，以达到负载均衡的目的。

具有相同分区键值的行属于同一个数据分区，一个分区可能包含多个分区键值。分区键的值是最小的分区单位，相同的分区键值下的数据无法再做切分。为了防止分区过大无法切分，单个分区键值下所有行的大小总和建议不超过 10 GB。

表格存储服务会根据特定的规则对分区进行分裂和合并，以达到更好的负载均衡。这个过程是自动的，应用程序无需关心。

2.2.3 读/写吞吐量

读/写吞吐量的单位为读服务能力单元和写服务能力单元，简称 CU（Capacity Unit），是数据读写操作的最小计费单位。

- 1 单位读能力表示从数据表中读一条 4 KB 数据。
- 1 单位写能力表示向数据表写一条 4 KB 数据。
- 操作数据大小不足 4 KB 的部分向上取整，如写入 7.6 KB 数据消耗 2 单位写能力，读出 0.1 KB 数据消耗 1 单位读能力。

应用程序通过 API 进行表格存储读写操作时，会消耗对应的写服务能力单元和读服务能力单元

预留读/写吞吐量

预留读/写吞吐量是表的一个属性。应用程序在创建表的时候，可以为该表指定预留读/写吞吐量。预留读写吞吐量的配置不影响该数据表的访问性能和服务能力。

设置预留吞吐量之后，应用程序每秒不超过预留吞吐量的访问将会按照预留吞吐量的单价进行计费。

例如，假设应用程序从表中每秒读取 80 条记录，每条记录的大小均为 3 KB，在这种情况下，每秒将消耗80个读吞吐量。

将预留读吞吐量设置为80 ，一个小时的读费用为 80 * 预留读单价，能够处理的读操作为 80 * 3600 = 288000 次。

说明：

- 预留读/写吞吐量可以设置为 0。
- 当预留读/写吞吐量大于 0 时，表格存储根据该配置为表分配和预留相应的资源，从而获得更低的资源使用成本。
- 当预留读/写吞吐量大于 0 时，即使没有读写请求也会进行计费，所以表格存储限制用户能够自行设置的单表预留读写吞吐量最大为 5000 （读和写分别不超过 5000）。如果用户有单表预留读写吞吐量需要超出 5000 的需求，可以提交工单提高预留读写吞吐量。
- 不存在的表将被视作预留读和预留写吞吐量均为 0，访问不存在的表将根据操作类型消耗 1 个按量读 CU 或者 1 个按量写 CU。

应用程序可以通过 UpdateTable 操作动态修改表的预留读/写吞吐量配置。

按量读/写吞吐量

按量读/写吞吐量是数据表在每一秒钟实际消耗的读/写吞吐量中超出预留读/写吞吐量的部分，统计周期为 1 秒。

假如数据表设置的预留读吞吐量为 100，连续3秒的访问情况如下：

- T0：读操作实际消耗 120 读吞吐量，则这 1 秒内预留吞吐量为100，消耗的按量读吞吐量为 20。
- T1：读操作实际消耗 95 读吞吐量，则这 1 秒内预留吞吐量为100，消耗的按量读吞吐量为 0。
- T2：读操作实际消耗 110 读吞吐量，则这 1 秒内预留吞吐量为100，消耗的按量读吞吐量为 10。

T0 至 T2 时刻的消耗的读吞吐量为：100 预留读吞吐量以及 30 按量读吞吐量。

说明：

每个小时内，表格存储对预留吞吐量取平均值，对按量吞吐量取累加值来作为用户实际消耗的吞吐量。

由于按量读/写吞吐量的模式无法预估需要为数据表预留的计算资源，表格存储需要提供足够的服务能力以应对突发的访问高峰，所以按量吞吐量的单价高于预留吞吐量的单价。合理设置数据表的预留吞吐量能够有效地降低使用成本。

说明：

由于按量读/写吞吐量无法准确估计需要预留的资源，在某些极端访问情况下，若单个分片键每秒钟的访问需要消耗 10000 CU，表格存储可能会返回 OTSCapacityUnitExhausted 错误给应用程序。此时，应用程序需要使用退避重试等策略来减少访问该表的频率。

更多信息请参考*Table Store* 表和计费方式。

2.3 数据访问API

Wide Column 模型在数据操作层面，提供两类数据访问 API：Data API 和 Stream API。

Data API

关于 Data API 的详细文档，请参考表格存储的数据操作。Data API 是标准的数据 API，提供数据的在线读写，包括：

- PutRow：新插入一行，如果存在相同行，则覆盖。
- UpdateRow：更新一行，可增加、删除一行中的属性列，或者更新已经存在的属性列的值。如果该行 不存在，则新插入一行。
- DeleteRow：删除一行。
- BatchWriteRow：批量更新多张表的多行数据。
- GetRow：读取一行数据。
- GetRange：范围扫描数据，可正序扫描或者逆序扫描。
- BatchGetRow：批量查询多张表的多行数据。

Stream API

在关系模型数据库中没有对数据库内增量数据定义标准的 API，但是在传统关系数据库的很多应用场景 中，增量数据（Binlog）的用途不可忽视。Table Store将增量数据的能力挖掘出来后可以在技术架 构上实现：

- 异构数据源复制：MySQL 数据增量同步到 NoSQL，做冷数据存储。

- 对接流计算：可实时对MySQL内数据做统计，做一些大屏类应用。
- 对接搜索系统：可将搜索系统扩展为 MySQL 的二级索引，增强 MySQL 内数据的检索能力。

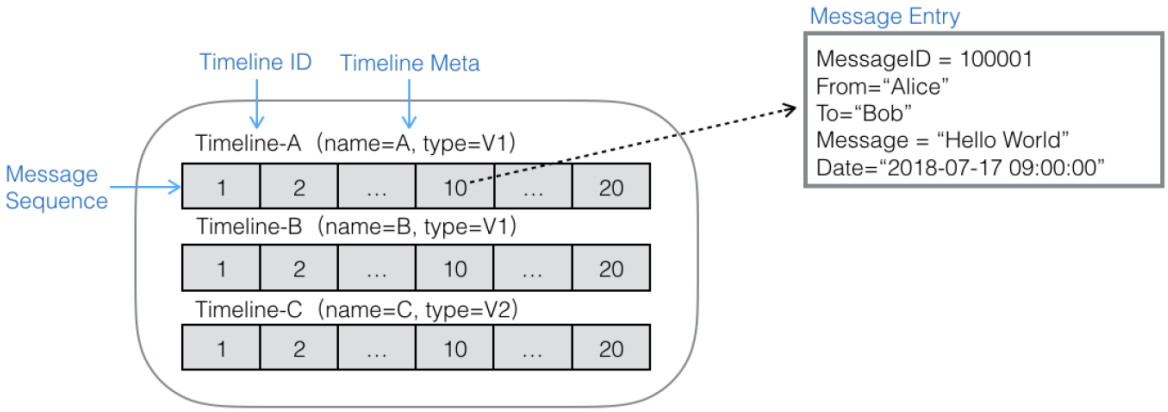
Table Store挖掘了增量数据存在的价值，并提供了标准的*Stream API*。Steam API提供的接口如下：

- ListStream：获取表的 Stream 信息，如 StreamID。
- DescribeStream：获取 Stream 的详细信息，如 Shard 列表、Shard 结构树等。
- GetShardIterator：获取 Shard 当前增量数据的 Iterator。
- GetStreamRecord：根据 ShardIterator 获取 Shard 内的增量数据。

Table Store Stream 的实现会比 MySQL Binlog 复杂得多。因为 Table Store 本身是一个分布式的架构， Stream 也是一个分布式的增量数据消费框架。Stream 的数据消费必须保序获取，Stream 的 Shard 与 Table Store 内部表的分区一一对应，且表的分区会存在分裂和合并。

3 Timeline

Timeline 模型是针对消息数据场景所设计的数据模型，它能满足消息数据场景对消息保序、海量消息存储、实时同步的特殊需求。



说明：

将一张大的数据表内的数据抽象为多个 Timeline，能够承载的 Timeline 个数无上限。

Timeline 的构成主要包括：

- Timeline ID：用于唯一标识 Timeline。
- Timeline Meta：Timeline 的元数据，元数据内可包含任意键值对属性。
- Message Sequence：消息队列，承载该 Timeline下的所有消息。

  说明：

  消息在队列里有序保存，并且根据写入顺序分配自增的 ID。一个消息队列可承载的消息个数无上限，在消息队列内部可根据消息 ID 随机定位某条消息，并提供正序或者反序扫描。
- Message Entry：消息体，包含消息的具体内容，可以包含任意键值对。