阿里云
表格存储

最佳实践

文档版本：20190815

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。
##	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 表操作篇..... 1
2 数据操作篇..... 7
3 亿量级订单管理方案.....8
3.1 行业背景.....8
3.2 方案选择.....9
3.3 准备工作.....11
3.4 搭建订单系统.....12
4 基于多元索引搭建亿量级店铺搜索系统..... 17
4.1 方案背景.....17
4.2 准备工作.....18
4.3 方案实现.....19
5 搭建海量智能元数据管理系统..... 22
5.1 方案背景.....22
5.2 准备工作.....25
5.3 方案实现.....28

1 表操作篇

本文将为您提供关于表设计的最佳实践。

设计良好的主键

表格存储会根据表的分区键将表的数据自动切分成多个分区，每个分区调度到一台服务节点上。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。为了防止某一个分区键值的数据成为访问热点达到单机服务能力上限，应用程序需要让数据的分布和访问量的分布尽可能均匀。

表格存储会对表中的行按主键进行排序，合理设计主键可以让数据在分区上的分布更加均匀，从而能够充分地利用表格存储水平扩展的特点。

选取分区键时，建议遵循以下几个原则：

- 单个分区键值中的数据不宜过大，建议不超过 10 GB。

  说明：
  10 GB 是限制为了避免访问热点，而不是数据存储的限制。
- 一张表内，不同分区键值中的数据在逻辑上是独立的。
- 访问压力不要集中在小范围连续的分区键值中。

使用示例

例如，有一张表，里面存储的是某大学内所有学生使用学生卡消费的记录。主键列有学生卡 ID（CardID），商家 ID（SellerID），消费终端 ID（DeviceID），订单号（OrderNumber）。同时我们有如下约定：

- 每一张学生卡对应一个 CardID，每一个商家对应一个 SellerID。
- 每一个消费终端对应 DeviceID，DeviceID 在全局是唯一的。
- 在每一个消费终端上产生的每一笔消费记录一个 OrderNumber。一个消费终端产生的 OrderNumber 是唯一的，但是在全局范围内 OrderNumber 不唯一。例如，不同的消费终端有可能产生两条完全不同的消费记录，但是它们的 OrderNumber 相同。
- 同一个消费终端产生的 OrderNumber 按时间排序，新的消费记录比老的消费记录拥有更大的 OrderNumber。
- 每笔消费记录均会被实时写入这张表中。

为高效利用表格存储，在设计表格存储的表的主键时，需考虑表的分区键：

分区方式	说明
使用 CardID 作为表的分区键	使用 CardID 作为表的分区键是一个比较好的选择。每天每张卡产生的消费记录数从总体上来讲是均匀的，每一个分区中的访问压力也应该是均匀的。以 CardID 作为表的分区键可以较好地利用预留读/写吞吐量资源。
使用 SellerID 作为表的分区键	使用 SellerID 作为表的分区键不是一个好的选择。因为学校内的商铺数量相对较少，同时一些商铺可能产生大量的消费记录成为热点，不利于访问压力的均匀分配。
使用 DeviceID 作为表的分区键	使用 DeviceID 作为表的分区键是一个比较好的选择。尽管每家商铺的消费记录数可能相差较大，但是每天每台消费终端上产生的消费记录数是可预期的。消费终端每天产生消费记录的条数取决于收银员操作的速度，这就决定了一台消费终端产生的消费记录数是受限的。因此，使用 DeviceID 作为表的分区键也可以保证访问压力的相对均匀。
使用 OrderNumber 作为表的分区键	使用 OrderNumber 作为表的分区键不是一个好的选择。因为 OrderNumber 是顺序增长的，因此在同一段时间内产生的消费订单的 OrderNumber 的值会集中在一个较小的范围内，这些消费订单记录会集中写入到个别的分区，以致预留读/写吞吐量没能得到高效利用。如果必须使用 OrderNumber 作为分区键，建议在 OrderNumber 上进行哈希散列，将哈希值作为 OrderNumber 的前缀，保证数据和访问压力的均匀。

总结

可以根据需求将 CardID 和 DeviceID 作为表的分区键，而不应该使用 SellerID 和 OrderNumber。之后再根据应用程序的实际需求来设计剩余的主键列。

通过拼接方式使用分区键

建议表格存储中表的同一分区键值下的数据量大小不超过 10 GB。如果您的表中单个分区键值的所有行的总数据量大小可能超过 10 GB，在设计表时可以将原来的多个主键列拼接成分区键。

使用示例

例如，上一小节中提到的学生卡消费记录表，假设主键为 [DeviceID, SellerID, CardID, OrderNumber]。DeviceID 是该表的分区键，单个 DeviceID 中所有行的数据量总大小可能超过 10 GB，可以将 DeviceID、SellerID 和 CardID 拼接作为表的第一个主键列（即分区键）。

原来的表如下所示：

DeviceID	SellerID	CardID	OrderNumber	attrs
16	'a100'	66661	200001	...
54	'a100'	6777	200003	...
54	'a1001'	6777	200004	...
167	'a101'	283408	200002	...

将 DeviceID、SellerID 和 CardID 拼接成分区键后的表如下所示：

CombineDeviceIDSellerIDCardID	OrderNumber	attrs
'16:a100:66661'	200001	...
'167:a101:283408'	200002	...
'54:a1001:6777'	200004	...
'54:a100:6777'	200003	...

在原来的表中，Device=54 的两行是属于同一个分区键值为 54 下的两条消费记录。在新的表中，这两条消费记录拥有不同的分区键值。通过拼接多列主键列形成分区键的表减少了单个分区键值下的总数据量大小。

选择将 DeviceID、SellerID 和 CardID 拼接成分区键，而不选择将 DeviceID 和 SellerID 进行拼接的原因是，前一节提到的消费记录表约定所有 DeviceID 相同的消费记录其 SellerID 也相同，因此仅仅拼接 DeviceID 和 SellerID 并不能解决单个分区键值的数据量过大的问题。

但是，拼接主键列形成表有一些小瑕疵。DeviceID 是一个 Integer 类型的主键列，在原来的表中，DeviceID=54 的消费记录在 DeviceID=167 的前面。将前三列主键列拼接成 String 类型的主键列后，DeviceID=54 的消费记录在 DeviceID=167 的后面。假如应用程序需要范围读取 DeviceID 在 [15, 100) 之间所有的消费记录，上面的表无法满足需求。

为了应对这种状况，可以在 DeviceID 高位补 0。补 0 的个数取决于 DeviceID 最大位数。假设 DeviceID 的取值范围是 [0, 999999]，可以将 DeviceID 高位补 0 至 6 位后再进行拼接，得到的表如下所示：

CombineDeviceIDSellerIDCardID	OrderNumber	attrs
'000016:a100:66661'	200001	...
'000054:a1001:6777'	200004	...
'000054:a100:6777'	200003	...
'000167:a101:283408'	200002	...

经过高位补 0 后的表依然有一些问题。在原来的表中，DeviceID=54 的两行，SellerID='a1001' 的行应该在 SellerID='a100' 的后面。产生这种现象的原因是，'000054:a1001' 的字典序小于 '000054:a100:'，但是 'a1001' 的字典序大于 'a100'，连接符 : 影响了字典序。在选取连接符时，应该选取比所有可用字符的 ASCII 码都小的字符作为连接符。在该表中，SellerID 的取值为数字、大小写英文字母。我们可以使用 , 作为连接符，因为 , 比所有 SellerID 可用字符的 ASCII 码都小。

使用 , 拼接后的表如下所示：

CombineDeviceiDSellerIDCardID	OrderNumber	attrs
'000016,a100,66661'	200001	...
'000054,a100,6777'	200003	...
'000054,a1001,6777'	200004	...
'000167,a101,283408'	200002	...

上面经过拼接形成的分区键的表的记录顺序就和原来的表保持一致了。

总结

当表中单个分区键值的所有行的数据量总大小可能超过 10 GB 时，可以将多个主键列拼接成分区键，以避免单分区键值的数据量大小限制。在拼接分区键时需要注意以下事项：

- 选取需要拼接的多个主键列，必须能有效地将原来表中相同的分区键值的记录，变成拥有不同分区键值的记录。
- 拼接 Integer 类型主键列时可以在高位补 0，保持记录的顺序一致。
- 选取连接符时需要考虑连接符对新的分区键的字典序的影响，选取比所有可用字符都小的连接符是一个比较安全的选择。

在分区键中加入哈希前缀

使用示例

在设计良好的主键一节中提到，尽量不要使用 OrderNumber 作为表的分区键。因为 OrderNumber 是顺序增长的，消费记录总是被写入最新的 OrderNumber 范围之内，旧的 OrderNumber 不再有写入压力，造成访问压力不均匀的现象，以致预留读/写吞吐量得不到高效利用。如果必须使用顺序增长的键值作为分区键，我们可以对分区键拼接哈希前缀，让相连的 OrderNumber 在表中随机分布，使访问压力分布均匀。

以 OrderNumber 为分区键的消费记录表如下所示：

OrderNumber	DeviceID	SellerID	CardID	attrs
200001	16	'a100'	66661	...
200002	167	'a101'	283408	...
200003	54	'a100'	6777	...
200004	54	'a1001'	6777	...
200005	66	'b304'	178994	...

对 OrderNumber 使用 md5 算法计算前缀（您也可以采取其他哈希散列算法），拼接成 HashOrderNumber。因为 md5 算法计算得到的哈希字符串可能过长，我们只需要取前几位就能达到让 OrderNumber 相连的记录在表中随机分布的目的。在这个例子中我们取前 4 位：

HashOrderNumber	DeviceID	SellerID	CardID	attrs
'2e38200004'	54	'a1001'	6777	...
'a5a9200003'	54	'a100'	6777	...
'c335200005'	66	'b304'	178994	...
'db6e200002	167	'a101'	283408	...
'ddba200001'	16	'a100'	66661	...

在后续访问消费记录时，使用相同的算法对 OrderNumber 计算哈希前缀，即可得到对应消费记录的 HashOrderNumber。在分区键中加入哈希前缀的弊端是，原来连续的记录会被打散，无法再使用 GetRange 操作读取一段范围内在逻辑上连续的记录。

并行写入数据

表格存储的表会被切分成多个分区，这些分区被分散在多个表格存储的服务器上。如果有一批数据要上传到表格存储中，同时这批数据是按主键排列顺序的，若按顺序写入数据，可能会导致写入压力集中在某个分区中，而其他的分区处于空闲状态，无法有效利用预留读/写吞吐量，影响数据导入速度。

可以采取以下任一措施来提升导入数据的速率：

- 将原始数据顺序打乱后再进行导入，以保证写入数据均匀地分配在各个分区上。
- 使用多个工作线程并行导入数据。把大的数据集合切分成很多个小集合，工作线程随机选取小集合进行数据导入。

区分冷数据和热数据

数据往往具有时效性。例如，在设计良好的主键一节中提到的存储消费记录表，近期产生的消费记录被访问的可能性较大，因为应用程序需要及时地对消费记录进行处理和统计，或者查询最近的消费记录。但是年代久远的消费记录被查询的可能性不大，这些数据渐渐成为冷数据，但仍然占用存储空间。

其次，表中存在大量冷数据会导致数据访问压力不均匀，从而导致表上配置的预留读/写吞吐量无法被充分利用。例如，已经毕业的学生的卡片，不会再产生消费记录。假如 CardID 是随着卡片申请时间递增的，以 CardID 作为分区键，会导致已经毕业的学生的 CardID 没有访问压力却被分配到预留读/写吞吐量，造成浪费。

为了解决这种问题，可以用不同的表来区分冷热数据，并设置不同的预留读/写吞吐量。例如，将消费记录按月份分表，每一个新的自然月就换一张新的表。当月的消费记录表需要不停写入新的消费记录，同时有查询操作。当月的消费记录表可以设置一个较大的预留读/写吞吐量配置来满足访问需求。前几个月的表由于不再写入新数据或者写入的新数据量较少，查询的请求较多，因此前几个月的消费记录表可以设置较小的预留写吞吐量，较大的预留读吞吐量。而历史超过一年的消费记录表，由于再被使用的可能性不大，可以设置较小的预留读/写吞吐量配置。已经超出维护年限的消费记录表可以将数据导出，存入 OSS（Object Storage Service）归档，或直接删除。

2 数据操作篇

本文将为您提供关于数据操作的最佳实践。

拆分属性列访问热度差异大的表

如果行的属性列较多，但是每次操作只访问一部分属性列，可以考虑将表拆分成多个表，将不同访问频率的属性列放到不同的表中。

例如，在商品管理系统中，每行存放商品数量、商品价格和商品简介。商品数量和商品价格均为占用空间较小的 Integer 类型，商品简介是 String 类型占用空间较大。大多数操作仅更新商品数量与商品价格，而不修改商品简介，所以商品简介的修改频率较低。这时候可以考虑将这个表拆分为两个，一个表存储商品数量和商品价格，另一个表存储商品简介。

压缩较大的属性列文本

如果属性列是较大的文本，应用程序可以考虑将属性列压缩之后再以 Binary 类型存储到表格存储中。这样做节省了空间、减少了访问的服务能力单元消耗，从而可以降低使用表格存储的成本。

将数据量超出限制的属性列存储到 OSS 中

表格存储限制单个属性列值不超过 2 MB。如果需要存储单个值超过 2 MB 的需求，如图片、音乐、文件等，可以使用 OSS（Object Storage Service）对其进行存储。OSS是阿里云提供的开放存储服务，用以应对海量数据的存储和访问。OSS 的存储单价比表格存储更低，更适合存储文件。

如果应用程序不方便使用 OSS，可以将超过 2 MB 的单个值拆分成多个行，存储在表格存储中。

错误重试加入时间间隔

表格存储可能会遇到软硬件问题，导致应用程序的部分请求失败，并返回可重试的错误。建议应用程序遇到此类错误时等待一段时间后再进行重试，每两次重试之间应该加大时间间隔或引入随机时间间隔，避免重试的请求堆积到一个时间点上引发雪崩效应。

3 亿量级订单管理方案

3.1 行业背景

订单系统存在于各行各业，如电商订单、银行流水、运营商话费账单等，是一个非常广泛、通用的系统。随着互联网的发展，以及各企业对数据的重视，需要存储和持久化的订单量越来越大。这些为订单系统管理来了新挑战。

需求场景

某电商平台A，需要存储和维护订单数据。同时，基于所有的订单数据，系统又需要向三类人群提供多元化查询服务：消费者、店家、平台。通过订单系统，消费者可以查询自己的历史订单，商家可以统计热销产品，平台也可以分析用户行为、平台交易规模等。主要查询方式涵盖订单的多维度检索，以及订单数据的分析、统计等，例如：

面向消费者：【A消费者】*【近1年】*【卖出电脑】订单查询。

面向售货员：【B售货员】*【近1个月】销售订单。

……

技术点

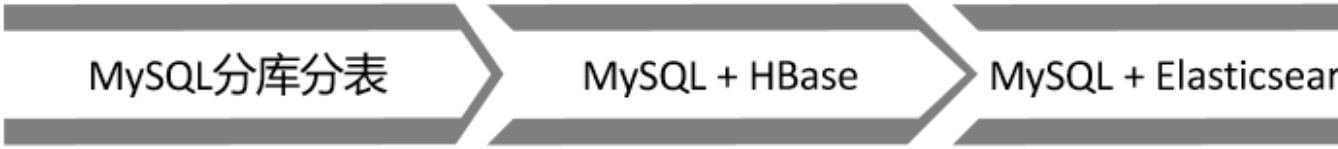
在订单场景中，需要考虑的技术点主要包含下几个方面：

- 查询能力：需要具备丰富的查询类型，如多维度、范围、模糊查询等，同时具备排序、统计等功能。
- 数据量：存储海量数据的同时，满足强一致、高可用、低成本等要求。
- 服务性能：应对高并发请求高并发的同时，保证低延迟。

其中，多维、实时查询功能是订单管理解决方案的核心功能。

3.2 方案选择

本章节主要为您介绍针对订单系统的一些传统解决方案，以及面对亿量级订单，表格存储（Table Store）提供的更全面的解决方案。



传统方案一：MySQL分库分表

MySQL自身拥有强大的数据查询、分析功能，基于MySQL创建订单系统，可以应对订单数据多维查询、统计场景。伴随着订单数据量的增加，采取分库、分表方案应对，通过这种伪分布式方案解决数据膨胀带来的问题。但数据一旦达到瓶颈，便出现明显的弊端：

- 数据纵向（数据规模）膨胀：采用分库分表方案，MySQL在部署时需要预估分库规模，数据量一旦达到上限后，重新部署并做数据全量迁移。
- 数据横向（字段维度）膨胀：schema需预定义，迭代新增新字段变更复杂。而维度到达一定量后影响数据库性能。

传统方案二： MySQL+HBase

引入双数据的方案应运而生，通过实时数据、历史数据分存的方案，可以一定程度解决数据量膨胀问题。该方案将数据归类成两部分存储：实时数据、历史数据。

- 实时订单数据（例如：近3个月的订单）：将实时订单存入MySQL数据库。实时订单的总量膨胀的速度得到了限制，同时保证了实时数据的多维查询、分析能力。
- 历史订单数据（例如：3个月以前的订单）：通过数据同步服务，将历史订单数据存入HBase，借助于HBase这一分布式NoSQL数据库，有效应对了订单数据膨胀困扰。也保证了历史订单数据的持久化。

但是，该方案牺牲了历史订单数据对用户、商家、平台的使用价值，假设了历史数据的需求频率极低。但是一旦有需求，便需要全表扫描，查询速度慢、I/O成本很高。而维护数据同步又带来了数据一致性、同步运维成本飙升等难题。

传统方案三：MySQL+Elasticsearch

此方案同样是将数据分两部分存储，可以一定程度解决订单索引维度增长问题。用户自己维护数据同步服务，保证两部分数据的一致性。

- 全量数据：将全量的订单数据存入MySQL数据库，订单ID之外的数据整体存为一个字段。该全量数据作为持久化存储，也用于非索引字段的反查。
- 查询数据：仅将需要检索的字段存入Elasticsearch（基于Lucene分布式索引数据库），借助于Elasticsearch的索引能力，提供可以应付维度膨胀的订单数据，然后必要时反查MySQL获取订单完整信息。

该方案应付了数据维度膨胀带来的困扰，但是随着订单量的不断膨胀，MySQL扩展性差的问题再次暴露出来。同时数据同步至Elasticsearch的方案，开发、运维成本很高，方案选择也存在弊端。

表格存储解决方案

由上可以看出，传统的方案并不能完全应对亿量级订单场景。使用Table Store研发的多元索引（SearchIndex）方案，则可以完美地解决亿量级订单系统问题。Table Store具有即开即用，按量收费等特点。多元索引随时创建，是海量电商订单元数据管理的优质方案。

Table Store作为阿里云提供的一款全托管、分布式NoSQL型数据存储服务，具有海量数据存储、热点数据自动分片、海量数据多维检索等功能，天然地解决了订单数据大爆炸这一挑战。同时，多元索引功能在保证用户数据高可用的基础上，提供了数据多维度搜索、统计等能力。针对多种场景创建多种索引，实现多种模式的检索。可以仅在需要的时候创建、开通索引。由Table Store来保证数据同步的一致性，这极大的降低了用户的方案设计、服务运维、代码开发等工作量。

下图为Table Store与传统订单方案使用的工具在各项性能上的对比：

能力分析	MySql	HBase	Elasticsearch	TableStore
存储方式	行存储	列存储	索引存储	列存储+索引存储
扩展性	单机、扩展性差	水平扩展	水平扩展	（自动）水平扩展
一致性	强一致性	强一致性、时序一致性		强一致性、时序一致性
检索	较弱的支持	不支持	支持	支持
数据量	~ 1T，~亿行	~10 PB，~万亿行	~1 PB，~千亿行	~10 PB，~万亿行

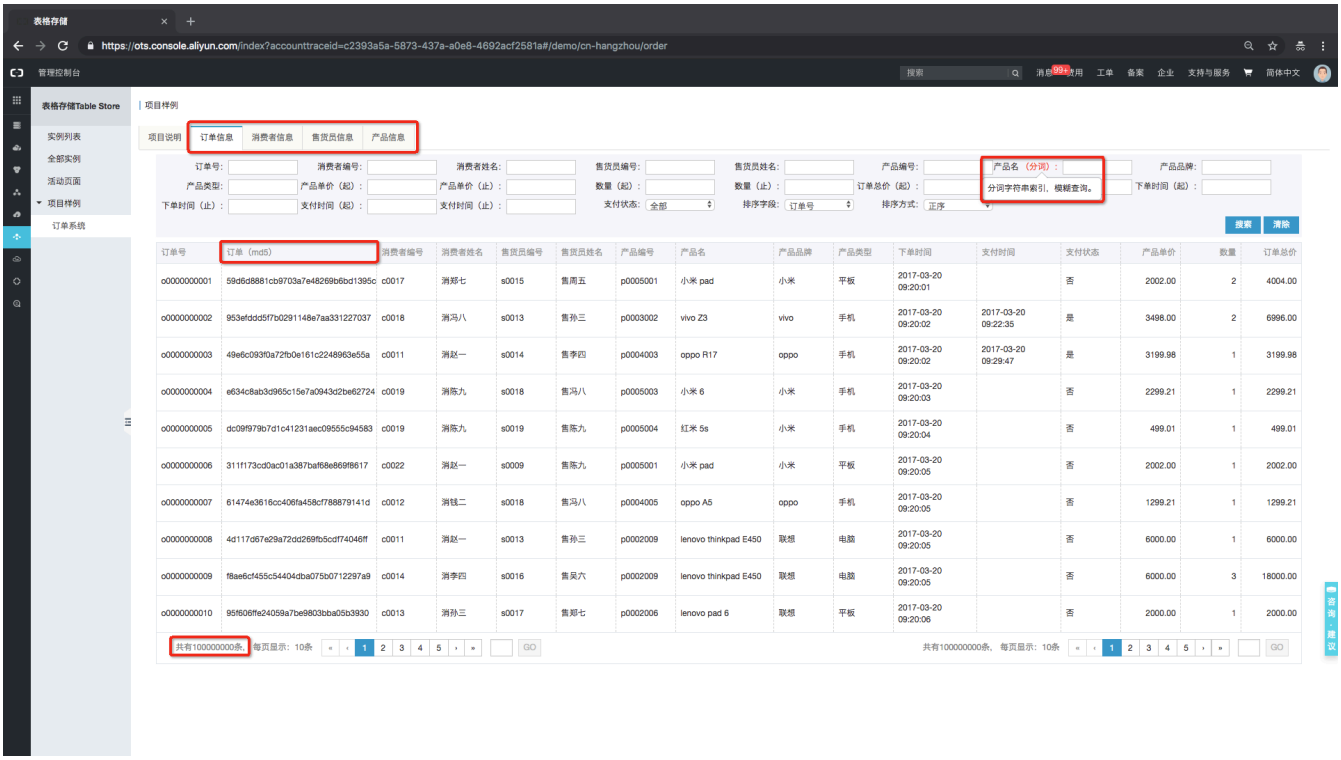
方案实现

参见准备工作以及搭建订单系统。

方案样例

基于表格存储搭建的订单系统样例内嵌在表格存储控制台中，用户可登录控制台体验系统。该样例提供了亿量级订单数据。官网控制台地址：项目样例。

> **说明:**
> 若为表格存储的新用户，需要点击开通服务后体验，开通免费，订单数据存储在公共实例中，体验不消耗用户存储、流量、读写吞吐。



3.3 准备工作

开始使用表格存储（Table Store）搭建订单系统前，您需要完成以下准备工作。

开通表格存储

具体操作步骤参见开通表格存储服务。

创建表格存储实例

实例（Instance）是您使用和管理表格存储服务的实体，每个实例相当于一个数据库。表格存储对应用程序的访问控制和资源计量都在实例级别完成。创建表格存储实例的具体操作步骤参见创建实例。

下载SDK

选择以下语言的SDK包进行下载安装：

- Java SDK
- Python SDK
- Go SDK
- Node.js SDK
- .NET SDK
- PHP SDK

设计数据表

订单系统不仅仅是订单一张数据表，它应包含：消费者表、售货员表、产品表、供货商表、交易订单表、支付订单表等。在本次搭建示例中，主要使用最基本的四张表（消费者表、售货员表、产品表、交易订单表），如下图：

表名：order_contract

列名	数据类型	索引类型	字段说明
_id(主键列)	String		MD5(oId)避免热点
oId(主键列)	String	KEYWORD	订单编号
pName	String	TEXT	产品名，TEXT类型索引可模糊查询，但
totalPrice	double	DOUBLE	订单总价
orderTime	long	LONG	下单时间（时间戳）
...	...	...	...

3.4 搭建订单系统

本章节主要为您介绍如何使用表格存储（Table Store）搭建订单系统。

前提条件

您已经完成了准备工作。

步骤一 创建数据表

创建四张表：订单表、消费者表、售货员表、产品表。您仅需将四张表创建在同一个实例，您可以通过以下两种方式创建数据表：

- 通过控制台创建、管理数据表，具体参见创建数据表。
- 通过SDK直接创建、管理数据表。

步骤二 创建数据表多元索引

Table Store自动做全量、增量的索引数据同步。您可以通过以下两种方式创建、管理多元索引：

- 通过控制台创建、管理多元索引，具体参见创建及使用多元索引。
- 通过SDK创建、管理多元索引。

步骤三 导入数据

插入部分测试数据。

说明：

控制台项目样例中插入了1亿条数据，您可以通过控制台插入少量测试数据。

订单号	订单（md5）（主键）	消费者编号	消费者姓名	售货员编号	售货员姓名	产品编号	产品名	产品品牌	产品类型	下单时间	支付时间	支付状态	产品单价	数量	总价钱
o00000000000000	c49f5fd5aba33159accae0d3ecd749a7	c0019	消陈九	s0020	售楚十	p0003004	vivox21	vivo	手机	2018-07-17 21:00:00		否	2498.999	2	4997.998

消费者编号（主键）	消费者姓名	消费者积分	注册时间
c0001	消赵一	818	2018-07-07 14:

售货员编号（主键）	售货员姓名	售货员积分	入职日期
s0001	售赵一	613	2018-07-07 14:

产品编号（主键）	产品名	产品品牌	产品类型	产品单价	新增时间
p0001001	iphone 6	苹果	手机	6969.00	2018-07-

步骤四 读取数据

数据读取分为两类：

- 主键读取

  基于原生表格存储的主键列获取：getRow、getRange、 batchGetRow等。主键读取用于索引（自动）反查，用户也可以提供主键（订单md5）的单条查询的页面，亿量级下查询速度极快。单主键查询方式不支持多维度检索。

- 索引读取

  基于新SearchIndex功能Query：search接口。用户可以自由设计索引字段的多维度条件组合查询。通过设置选择不同的查询参数，构建不同的查询条件、不同排序方式；目前支持：精确查询、范围查询、前缀查询、匹配查询、通配符查询、短语匹配查询、分词字符串查询，并通过布尔与、或组合。

  如【c0001号消费者，且消费在99.99以上的订单】组合方式如下：

```
List<Query> mustQueries = new ArrayList<Query>();

TermQuery termQuery = new TermQuery();
termQuery.setFieldName("cId");
termQuery.setTerm(ColumnValue.fromString("c0001"));
mustQueries.add(termQuery);

RangeQuery rangeQuery = new RangeQuery();
rangeQuery.setFieldName("totalPrice");
rangeQuery.setFrom(ColumnValue.fromDouble(99.99)) ;
mustQueries.add(rangeQuery);

BoolQuery boolQuery = new BoolQuery();
boolQuery.setMustQueries(mustQueries);
```

专家服务

表格存储 提供专业的免费的技术咨询服务，欢迎加入我们的钉钉讨论群。

表格存储公开交流群
1023人
扫一扫群二维码，立刻加入该群。

4 基于多元索引搭建亿量级店铺搜索系统

4.1 方案背景

本章节主要为您介绍基于多元索引搭建亿量级店铺搜索系统的需求场景及方案。

亿量级店铺搜索系统的核心点与瓶颈在于数据库是否有海量存储能力、低延迟读能力、高效查询能力以支持GEO+多维度数据检索。表格存储（Tablestore）作为阿里云提供的一款全托管、零运维的分布式NoSQL型数据存储服务，具有海量数据存储、热点数据自动分、海量数据多维检索等功能，可以有效的地解决了GEO数据量大膨胀这一挑战。

需求场景

某店铺搜索平台，提供了亿量级的店铺信息。用户通过平台提供的PC端、移动端网页，按照自己的需求维度组合，搜索用户心仪的店铺。平台需要在地图上展示店铺的具体位置、店铺详细信息、店铺主页的跳转。

维度一：【距离1JM内】【人均100以内】【评分最高】【奶茶店】

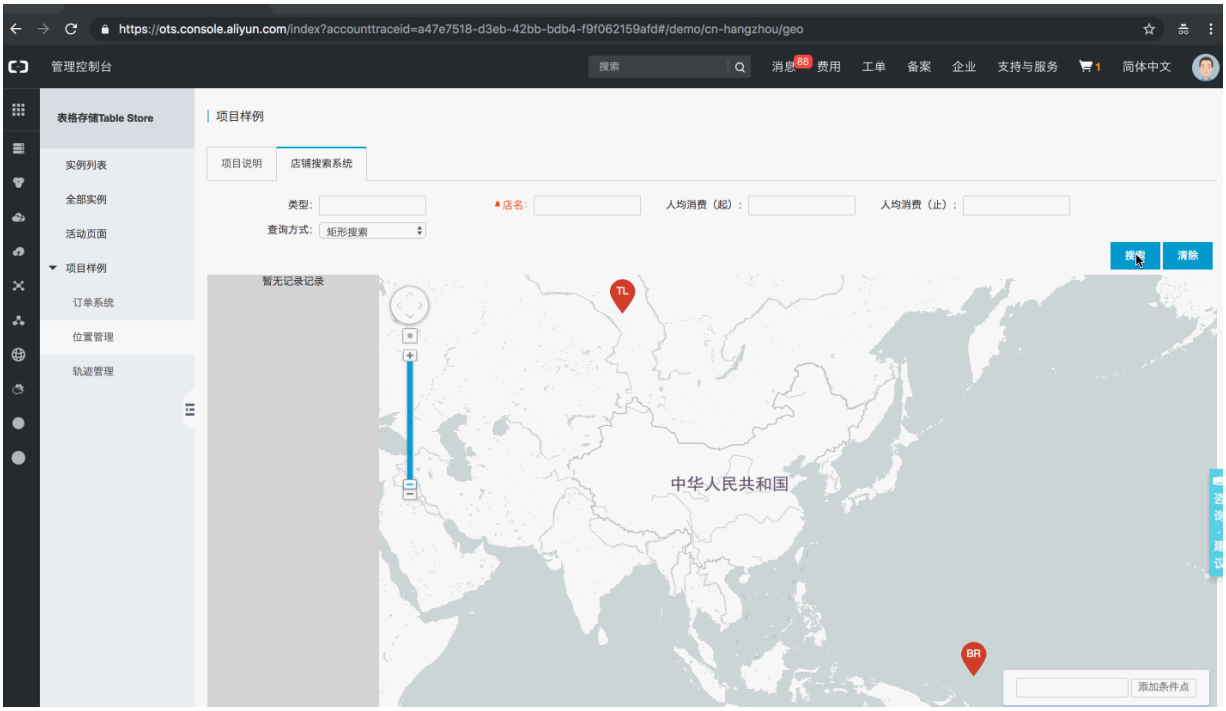
维度二：【杭州市内】【评分最高的】【沈家*】店铺

……

实现快速、多维GEO查询功能，是GEO管理解决方案的核心功能，样例如下：

说明:

注：该样例提供了【亿量级】店铺数据。官网控制台样例地址：项目样例



基于表格存储搭建的店铺搜索系统页面一览，样例内嵌在表格存储控制台中，用户可登录控制台体验系统（若为表格存储的新用户，需要点击开通服务后体验，开通免费，订单数据存储在公共实例中，体验不消耗用户存储、流量、CU）。

表格存储方案

使用表格存储研发的多元索引（SearchIndex）功能，可以轻松搭建一套亿量级店铺搜索系统。多元索引功能可以创建GEO索引、分词字符串索引等，为用户提供了GEO检索、多维组合检索等能力，用户可随时创建，存量、增量数据自动同步。

用户可以仅在需要的时候创建、开通索引。由Tablestore来保证数据同步的一致性，这极大的降低了用户的方案设计、服务运维、代码开发等工作量。

4.2 准备工作

开始使用表格存储（Tablestore）搭建亿量级店铺搜索系统前，您需要完成以下准备工作。

开通表格存储

具体操作步骤参见开通表格存储服务。

创建表格存储实例

实例（Instance）是您使用和管理表格存储服务的实体，每个实例相当于一个数据库。表格存储对应用程序的访问控制和资源计量都在实例级别完成。创建表格存储实例的具体操作步骤参见创建实例。

下载SDK

选择以下语言的SDK包进行下载安装：

- Java SDK
- Python SDK
- Go SDK
- Node.js SDK
- .NET SDK
- PHP SDK

设计数据表

店铺检索系统样例，仅简易使用一张店铺表，主要包含字段：店铺类型、店铺名称、店铺地理位置、店铺平均评分、人均消费消等。表设计如下：

表名：geo_positon

列名	数据类型	索引类型	字段说明
_id(主键列)	String		MD5(pId)避免热点
pId	Stirng		店铺编号
type	String	KEYWORD	类型
name	String	TEXT	店铺名，TEXT类型索引可模糊查询，但不能排序
pos	String	GEO_POINT	店铺位置："30.132,120.082"(纬度,精度)
point	double	DOUBLE	评分
...	...	...	...

4.3 方案实现

本章节主要为您介绍如何使用表格存储（Tablestore）搭建亿量级店铺搜索系统。

前提条件

您已经完成了准备工作。

步骤一 创建数据表

通过以下两种方式创建一张店铺信息表：

- 通过控制台创建、管理数据表，具体参见创建数据表。
- 通过SDK直接创建、管理数据表。

步骤二 创建数据表索引

Tablestore自动进行全量、增量的索引数据同步。您可以通过以下两种方式创建、管理多元索引：

- 通过控制台创建、管理多元索引，具体参见创建及使用多元索引。
- 通过SDK创建、管理多元索引。

步骤三 导入数据

控制台项目样例中插入了1亿条数据，您可以通过控制台插入少量测试数据。

店铺编号	店铺（md5）（主键）	类型	店铺名称	店铺位置	店铺评分	人均消费		
o00570 22192	0000000f470ef0f548b9 25ceffe1a7e3	杭帮菜	韩村杭帮菜	36.76613,11 1.41461	2.87	63.6 7		

步骤四 读取数据

数据读取分为两类：

- 主键读取

  基于原生表格存储的主键列获取：getRow, getRange, batchGetRow等。主键读取用于索引（自动）反查，用户也可以提供主键（订单md5）的单条查询的页面，亿量级下查询速度极快。单主键查询方式不支持多维度检索。

- 索引读取（店铺查询）

  基于新SearchIndex功能Query：search接口。用户可以自由设计索引字段的多维度条件组合查询。通过设置选择不同的查询参数，构建不同的查询条件、不同排序方式；目前支持：精确查询、范围查询、前缀查询、匹配查询、通配符查询、短语匹配查询、分词字符串查询，并通过布尔与、或组合。

  如【"36.76613,111.41461"周边1km米范围内的奶茶店】，查询条件如下：

```
List<Query> mustQueries = new ArrayList<Query>();

TermQuery termQuery = new TermQuery();
termQuery.setFieldName("type");
termQuery.setTerm(ColumnValue.fromString(奶茶));
mustQueries.add(termQuery);

GeoDistanceQuery geoDistanceQuery = new GeoDistanceQuery();
```

```
geoDistanceQuery.setFieldName("pos");
geoDistanceQuery.setCenterPoint("36.76613,111.41461");
geoDistanceQuery.setDistanceInMeter(1000);
mustQueries.add(geoDistanceQuery);

BoolQuery boolQuery = new BoolQuery();
boolQuery.setMustQueries(mustQueries);
```

专家服务

表格存储 提供专业的免费的技术咨询服务，欢迎加入我们的钉钉讨论群。

5 搭建海量智能元数据管理系统

5.1 方案背景

本章节主要为您介绍搭建海量智能元数据管理系统的场景、技术点以及搭建方案。

用户存储海量的文档、媒体文件等数据的同时，对文件元数据（Meta）的管理不可或缺。元数据拥有多维度的字段信息，基本信息包含文件大小、创建时间、用户等。随着人工智能的发展，通过AI技术提取文件核心要素也成为文件元数据的重要信息。以图片为例：用户通过智能媒体服务，获取分析图片核心标签并为标签打分，用户还可提取人脸识别相关信息，以及地理位置等信息，提取的信息也需要存储到文件元数据信息中。因而文件元数据的信息量不断增加，格式、类型也不断呈现多元化。

需求场景

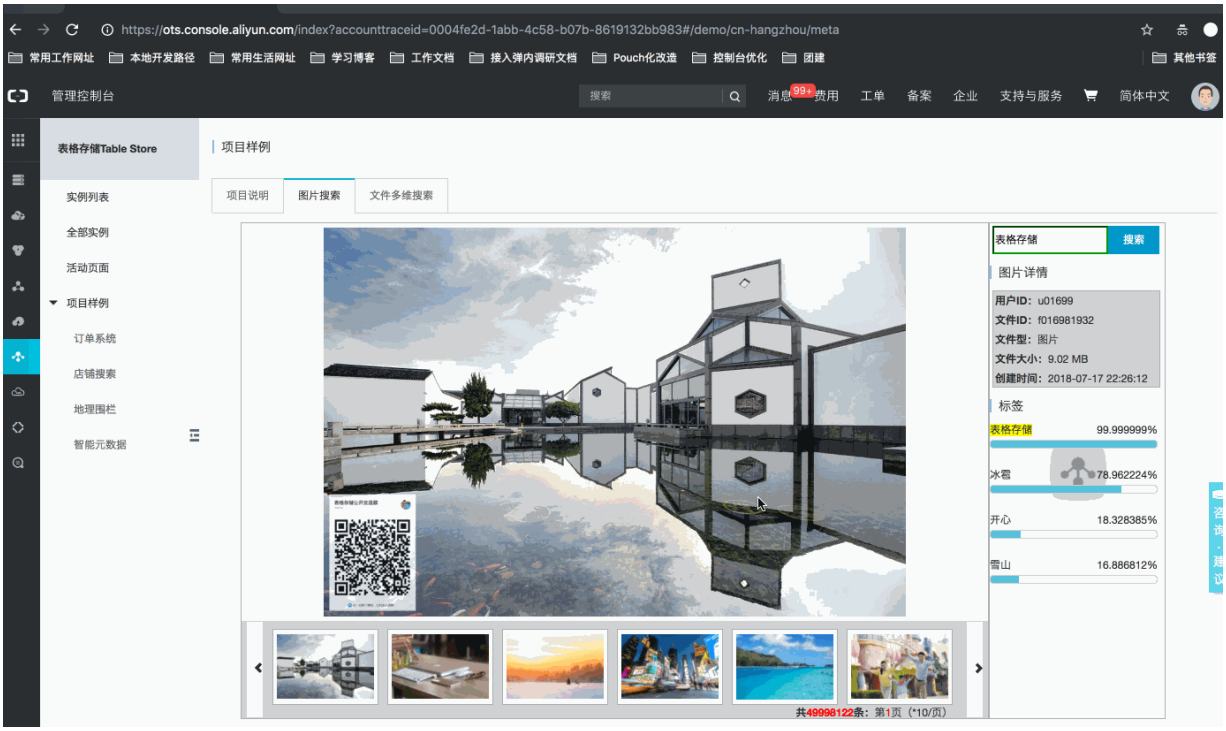
某智能媒体管理平台，为用户提供文件（图片、视频等）管理服务，用户通过自研（或售卖）的智能媒体分析工具，为目标文件进行分析。用分析后的信息丰富原有的元数据信息。因此，平台需要一套有效的元数据管理方案，为用户提供元数据信息的管理、分析、统计功能。例如：

用户A：【用户A的文件】*【近1年】*【标签含[开心]】*的所有图片，按标签分数排序

用户B：【用户B的文件】*【出现某某明星】*的所有视频，按明星相似度排序

……

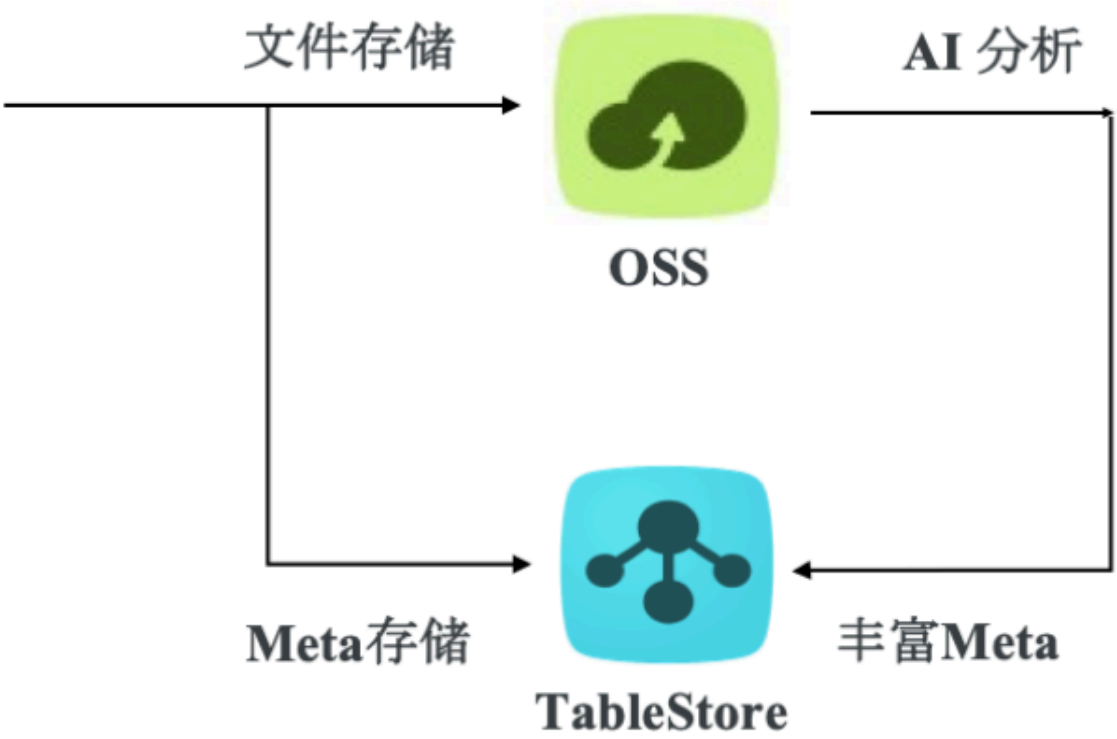
理系统样例，如下所示：__官网控制台地址：项目样例



技术点

对于智能元数据管理系统，通常需要考虑的技术点，包含以下方面：

- 查询能力：具备强大的查询能力，如多类型索引、多维度组合查询等，同时具备排序、统计等功能。
- 横向扩展（多字段）：元数据的字段类型丰富，字段变动、增删频繁，数据库尽量schema free来保证横向扩展能力。
- 纵向扩展（数据量）：海量文件就会对应海量元数据，面对数据膨胀，数据库要满足易扩展、低成本等基本要求。
- 服务性能：应对高并发请的同时，保证低延迟、强一致、高可用。



表格存储方案

使用表格存储（Tablestore）研发的多元索引（SearchIndex）方案，可以有效解决海量元数据的管理问题。Tablestore具有即开即用，按量收费等特点。

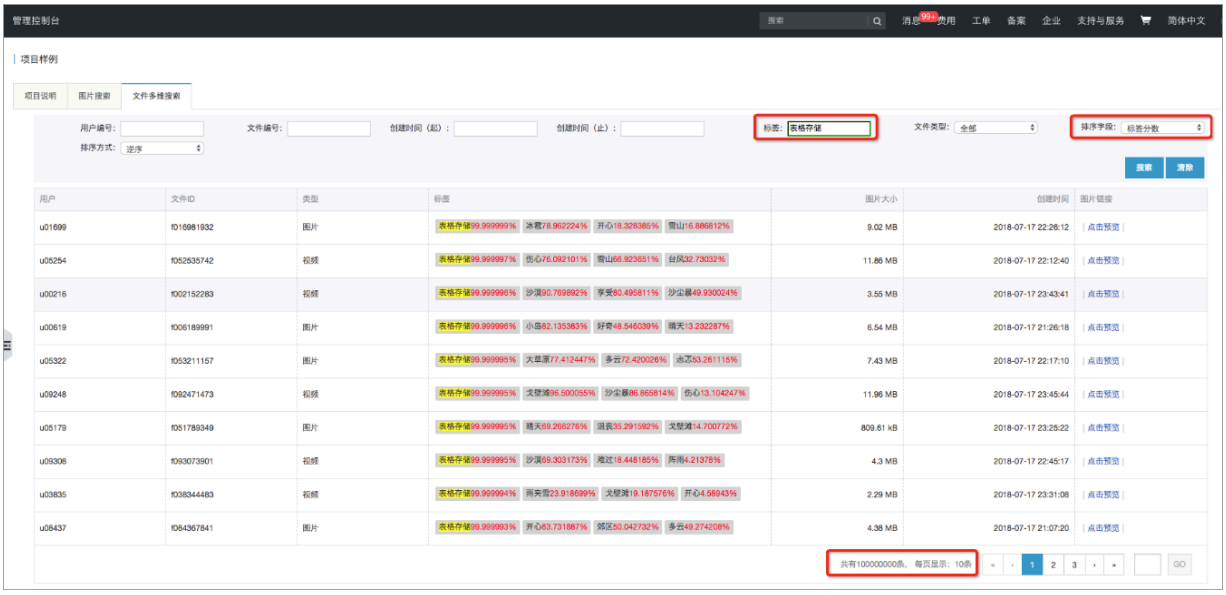
Tablestore作为阿里云提供的一款全托管、分布式NoSql型数据存储服务，具有【海量数据存储】、【热点数据自动分片】、【海量数据多维检索】等功能，天然地解决了数据大爆炸这一挑战；在应对数据横向、纵向扩展上，充分发乎其优势。多元索引随时创建，是Meta元数据管理的合适方案。同时，SearchIndex功能在保证用户数据高可用的基础上，提供了数据多维度搜索、统计等能力。针对多种场景创建多种索引，实现多种模式的检索。用户可以仅在需要的时候创建、开通索引。由Tablestore来保证数据同步的一致性，这极大的降低了用户的方案设计、服务运维、代码开发等工作量。

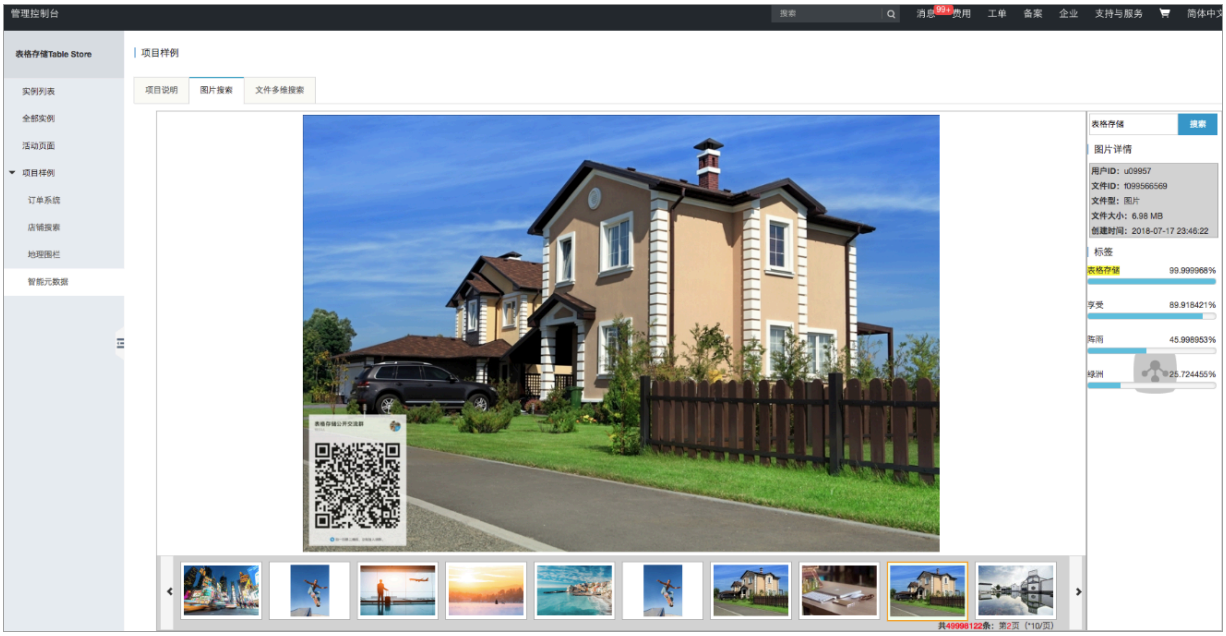
基于表格存储搭建的智能元数据管理系统页面一览

样例内嵌在表格存储控制台中，用户可登录控制台体验系统（若为表格存储的新用户，需要点击开通服务后体验，开通免费，Meta数据存储在公共实例中，体验不消耗用户存储、流量、Cu）。

说明:

该样例提供了【亿量级】文件元数据。官网控制台地址：项目样例





5.2 准备工作

本章节主要为您介绍使用表格存储（Tablestore）搭建海量智能元数据管理系统前的准备工作。

开通表格存储

具体操作步骤参见开通表格存储服务。

创建表格存储实例

实例（Instance）是您使用和管理表格存储服务的实体，每个实例相当于一个数据库。表格存储对应用程序的访问控制和资源计量都在实例级别完成。创建表格存储实例的具体操作步骤参见创建实例。

下载SDK

选择以下语言的SDK包进行下载安装：

- Java SDK
- Python SDK
- Go SDK
- Node.js SDK
- .NET SDK
- PHP SDK

设计数据表

表名：order_contract

列名	数据类型	索引类型	字段说明
_id(主键列)	String		MD5(fId)避免热点
fId	String	KEYWORD	文件编号
userId	String	KEYWORD	用户编号
tags	String	Nested: [{ tag: String, score: LONG }]	多标签使用嵌套索引（数组字符串） '[{"tag":"表格存储","score":97.317251}，{"score":50.770918,"tag":"沙漠"}]'
size	long	LONG	文件大小
createdAt	long	LONG	创建时间（时间戳）
url	String	KEYWORD	文件链接（存储于oss）
...	...	...	...

5.3 方案实现

本章节主要为您介绍如何使用表格存储（Tablestore）搭建海量智能元数据管理系统。

前提条件

您已经完成了准备工作。

步骤一 创建数据表

通过以下两种方式创建一张店铺信息表：

- 通过控制台创建、管理数据表，具体参见创建数据表。
- 通过SDK直接创建、管理数据表。

步骤二 创建数据表索引

Tablestore自动进行全量、增量的索引数据同步。您可以通过以下两种方式创建、管理多元索引：

- 通过控制台创建、管理多元索引，具体参见创建及使用多元索引。
- 通过SDK创建、管理多元索引。

步骤三 导入数据

控制台项目样例中插入了1亿条数据，您可以通过控制台插入少量测试数据。

文件编号	文件ID（md5主键）	用户编号	标签（数组字符串）	类型	链接	大小
f052535742	1bce....	u052554	[{"score":99.999999,"tag":"表格存储"},{"score":78.962224,"tag":"冰雹"},{"score":18.328385,"tag":"开心"},{"score":16.886812,"tag":"雪山"}]	image	https://prd-console-demo.oss-cn-hangzhou.aliyuncs.com/image/imm1.jpg	902206 6

读取数据

数据读取分为两类：

- 主键读取

  基于原生表格存储的主键列获取：getRow, getRange, batchGetRow等。主键读取用于索引（自动）反查，用户也可以提供主键（文件编号md5）的单条查询的页面，亿量级下查询速度保持在十毫秒量级。单主键查询方式不支持多维度检索。

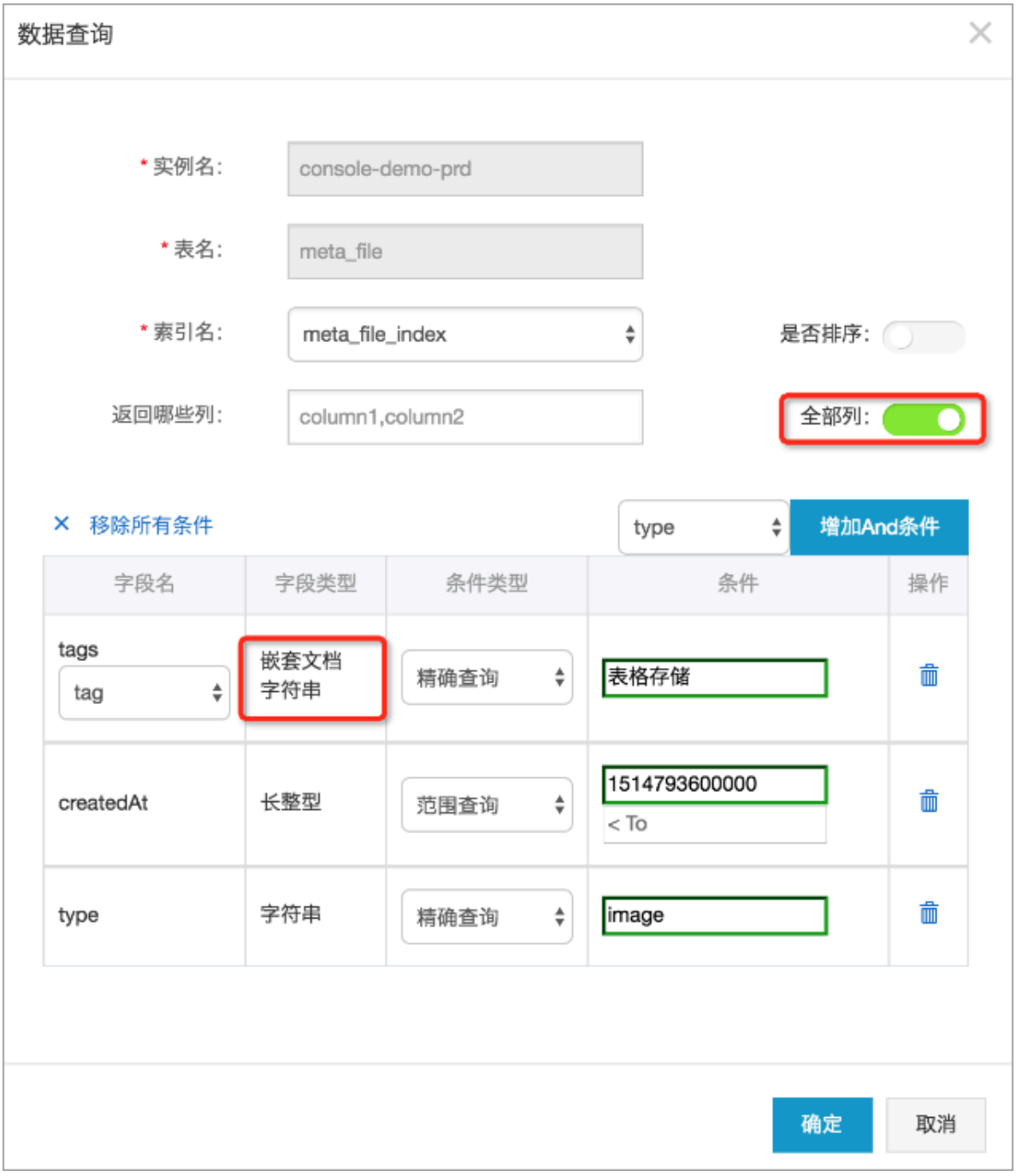
- 索引读取

  基于新SearchIndex功能Query：search接口。用户可以自由设计索引字段的多维度条件组合查询。通过设置选择不同的查询参数，构建不同的查询条件、不同排序方式；目前支持：精确查询、范围查询、前缀查询、匹配查询、通配符查询、短语匹配查询、分词字符串查询、嵌套查询、GEO查询，并通过布尔与、或组合。

  如【标签为：表格存储，创建时间[2018-01-01, 2018-12-01)】文件的信息：（SDK与控制查询）

```
List<Query> mustQueries = new ArrayList<Query>();

//嵌套字段Query
TermQuery termQuery = new TermQuery();
termQuery.setFieldName("tags.tag");
termQuery.setTerm(ColumnValue.fromString("表格存储"));

NestedQuery nestedQuery = new NestedQuery();
nestedQuery.setPath("tags");
nestedQuery.setScoreMode(ScoreMode.Avg);
nestedQuery.setQuery(termQuery);
mustQueries.add(nestedQuery);

//范围Query
RangeQuery rangeQuery = new RangeQuery();
rangeQuery.setFieldName("createdAt");
rangeQuery.setFrom(ColumnValue.fromLong(1514793600000, true);
rangeQuery.setTo(ColumnValue.fromLong(1543651200000, false);
mustQueries.add(rangeQuery);

//精确Query
TermQuery termQuery = new TermQuery();
termQuery.setFieldName("type");
termQuery.setTerm(ColumnValue.fromString("image"));
mustQueries.add(termQuery);

BoolQuery boolQuery = new BoolQuery();
```

```
boolQuery.setMustQueries(mustQueries);
```



专家服务

表格存储 提供专业的免费的技术咨询服务，欢迎加入我们的钉钉讨论群。

表格存储公开交流群
1023人
扫一扫群二维码，立刻加入该群。