大数据技术丛书

华章科技

Kyligence

Apache Kylin 权威指南

Apache Kylin核心团队◎著

Apache Kylin是首个中国人贡献的
Apache顶级开源项目

机械工业出版社
China Machine Press

大数据技术丛书

Apache Kylin权威指南

Apache Kylin核心团队 著

ISBN：978-7-111-55701-2

本书纸版由机械工业出版社于2017年出版，电子版由华章分社（北京华章图文信息有限公司，北京奥维博世图书发行有限公司）全球范围内制作与发行。

客服热线：+ 86-10-68995265

客服信箱：service@bbbvip.com

官方网址：www.hzmedia.com.cn

新浪微博 @华章数媒

微信公众号 华章电子书（微信号：hzebook）

目录

推荐序一

2016年早些时候，我曾经写过一篇有关联通Hadoop的文章，在其中的“展望篇”里谈到过OLAP on Hadoop的新技术Apache Kylin。今天《Apache Kylin权威指南》一书即将出版，我也有幸受本书作者之一韩卿（Luke）的邀请来写推荐序。

联通集团的BI是2010年建设的，由于全国有4亿用户的明细数据需要集中处理，再加上对移动互联网用户流量日志的采集，使得数据量急增。截至2013年已达PB级规模，并仍以指数级速度增长，传统数据仓库不堪重负，数据的存储和批量处理成了瓶颈。另一方面BI上提供的面向用户的数据查询和多维分析服务，使得后台生产的Cube越来越多，几年下来已有七八千个。用户需求对某一维度的改变往往会造成一个新Cube的产生，耗费资源不说，也为管理带来了极大的不便。2013年年底我们在传统数据仓库之外搭建了第一个Hadoop平台，节点数也从最初的几十个发展到了今天的3500个，大大提高了系统的存储及计算能力，为联通大数据对内对外的发展都起到了至关重要的作用。美中不足的是分布式存储和并行计算只解决了系统的性能问题，尽管我们也部署了像Hive、Impala这样的SQL on Hadoop技术，但在Hadoop体系上的多维联机分析（OLAP）却始终得不到满意的结果。Oracle+Hadoop的混搭架构还因为有对OLAP的需求而继续维持着，零散的Cube数还在继续增长，架构师们还在继续寻

找奇迹方案的出现。

Apache Kylin就是在这种大背景下出现在我们的视野中的。一个好的产品首先要有一个清晰的定位，要有一套能够明确解决行业痛点的方案。Kylin在这点上做得非常好，它把自己定义为Hadoop大数据平台上的一个开源OLAP引擎。三个关键词：Hadoop、开源、OLAP，使它的定位一目了然，不用过多地解释。同时，Kylin也是透明的，不像许多产品把自己使用的技术搞得很神秘，Kylin沿用了原来数据仓库技术中的Cube概念，把无限数据按有限的维度进行“预处理”，然后将结果（Cube）加载到HBase里，供用户查询使用，使得现有的分析师和业务人员能够快速理解和掌握。相比于IOE时代的BI，它非常巧妙地使用了Hadoop的分布式存储与并行计算能力，用横向可扩展的硬件资源来换取计算性能的极大提高。

为了能够将Kylin真正融入到联通的大数据架构中，我们正在紧锣密鼓地组织系统测试。比如对单用户级的数据查询、第三方可视化工具的集成、多维Cube建立的维度数极限等的测试。我们还计划用Kafka来导入数据，用Spark来加工Cube，用其他产品来代替HBase进而提高数据读取性能，用Kylin的路由选择来桥接新老Cube，等等。这时出版的《Apache Kylin权威指南》一书，对于我们来说无疑是雪中之炭，我们的许多疑惑都会在这本指南当中找到权威解答。

联通公司现在经历的这些过程很多企业都会遇到，“坑”我们愿意去

填，路希望大家来走。在向读者推荐《Apache Kylin权威指南》一书的同时，我们真诚期望Kylin（作为Apache开源社区第一个由中国人开发并主导的产品）能够成功，能够在不断的实践中提高自己，能够充分利用中国这个占世界数据量20%的大市场，把自己打造成大数据领域的一只独角兽。

范济安

国家千人计划专家

中国联通集团信息化部CTO

推荐序二

我是一个开源软件的爱好者，算是开源届的一名老兵。从1995年到美国留学起，就开始接触开源软件，当时的GNU、Linux、FreeBSD和Emacs等自由软件让刚出国门的我感到惊艳万分。从那时开始，我就再没有和自由软件、开源软件分开过：从读博士期间一直参与研发自由软件XSB、因个人爱好参与贡献GNU Emacs、在IBM工作期间基于一系列开源软件为团队开发DocBook文档写作工具链，到后来在LinkedIn工作期间研究作为5个核心成员开源的分布式实时搜索系统SenseiDB，再到近几年在小米大力推动开源战略，打造基于开源软件的小米云计算、大数据和机器学习技术及团队。20多年来，对开源软件的热爱，让我逐渐从一名早期的自由软件爱好者、信仰者、贡献者和管理者，变成了一名坚定的开源软件倡导者。在这期间，我见证了开源技术的萌芽、兴起和今天的繁荣，也经历了国内外不同文化下的开源发展历程。

作为一名参与开源软件较早的中国人，我也深深地感受到了最初西方世界对中国人使用开源技术、参与开源软件开发的质疑和冷落。因为互联网和自由软件进入我国较晚，也因为中国人在英语上的不足和东西方文化的差异，还因为早期国内的一些开源爱好者对开源软件的理解不足，使得在开源方面较为领先的西方开源人士对国人在开源上的使用和贡献存在极大偏见。中国开源力量融入国际开源社区的过程是缓慢和艰

苦的，幸运的是，近四五年来，随着GitHub的兴起和多个开源社区的迅猛发展，中国每年产生的计算机人才也多了起来，中国越来越多的互联网公司开始正确地拥抱开源，中国工程师在国际开源社区的贡献和影响力也越来越大（比如，作为一个很年轻的创业公司，小米就在不到一年半的时间里推出了3个HBase committer），这确实不是一件容易的事。但是，今天不管是在云计算、大数据，还是容器等诸多开源技术领域，真正由中国人自己主导、从零开始、自主研发、最后贡献到国际开源社区并成为顶级开源项目的，应该就只有Apache Kylin一个。Apache Kylin是2013年由eBay在上海的一个中国工程师团队发起的、基于Hadoop大数据平台的开源OLAP引擎，它利用空间换时间的方法，把很多分钟级别乃至小时级别的大数据查询速度一下子提升到了亚秒级别，极大地提高了数据分析的效率，填补了业界在这方面的空白。

我非常高兴能够看到一个来自国内的团队开源一个项目，并在短短不到一年的时间里顺利使其毕业，也使其成为Apache软件基金会的顶级项目，取得了可以和Hadoop、Spark等重大开源软件相提并论的成就。一支来自国内的工程师团队能够快速融入国际开源社区，被全球最大的开源软件基金会接纳并成功占领一席之地，这是一件非常不容易的事情，足以让国人欣慰和骄傲。这一切都和Apache Kylin项目背后的负责人韩卿（Luke）密不可分。我是在QCon北京2014全球软件开发大会上认识韩卿的，并由此第一次知道了Kylin这个项目，和韩卿开始交谈不久，我就觉得他是当时国内为数不多的、真正懂得开源软件打法的一个人。那次的交

谈非常愉快，从此我也开始关注这个项目并极度看好它。

开源项目，并不是将代码公开就完事了，团队需要做更多艰苦的工作来不断推广技术、经营社区和营销品牌，使得项目能够被广泛接纳和使用。韩卿及Kylin团队在这方面做得非常出色，在各种国内外的技术大会上、很多开源社区里都可以看到他们忙碌的身影。在短短的两年时间里，我就看到Kylin项目从Apache孵化器项目毕业成为顶级项目，也看到这个团队离开eBay并创立了Kyligence这家创业公司。今天，很多成功的重大开源项目背后都有一两个伟大的创业公司：Hadoop背后是Cloudera和Hortonworks、Spark后面是Databricks，等等。我也看好Apache Kylin后面的Kyligence！

小米不仅仅是一家手机公司，更是一个大数据公司，公司内部的很多产品和业务都深度依赖大数据分析，我们所面对的数据量、挑战和困难都是空前的。Apache Kylin独特的数据查询性能优势在小米中有很多应用场景，我希望将来我们能够更多地用到Apache Kylin技术，也希望和Kyligence能有深度的技术合作。

今年，深度学习和大数据引发了人工智能的热潮，人工智能的热潮反过来也会推动大数据领域相关技术的发展和演进，大数据领域必将诞生更多的新技术和新产品。相信在不久的未来，会有更多的、类似于Apache Kylin的、由中国人主导的项目从实际需求中产生、开源并被贡献到国际开源社区，向世界输出我们的技术实力。在将本书推荐给读者的

同时，我也希望更多的读者、团队和公司能一起参与、贡献和拥抱开源，努力提高我国技术人员在国际开源社区的影响力。Apache Kylin项目相关的经验也非常值得其他技术人员学习和借鉴！

崔宝秋

小米首席架构师

小米云平台负责人

推荐序三

在大数据处理技术领域，用户最普遍的诉求就是希望以很简易的方式从大数据平台上快速获取查询结果，同时也希望传统的商务智能工具能够直接和大数据平台连接起来，以便使用这些工具做数据分析。目前已经出现了很多优秀的SQL on Hadoop引擎，包括Hive、Impala及SparkSQL等，这些技术的出现和应用极大地降低了用户使用Hadoop平台的难度。为了进一步满足“在高并发、大数据量的情况下，使用标准SQL查询聚合结果集能够达到毫秒级”这一应用场景，Apache Kylin应运而生，在eBay孵化并最终贡献给开源社区。Apache Kylin是一种分布式分析引擎，提供Hadoop之上的标准SQL查询接口及多维分析（OLAP）功能。

Apache Kylin通过空间换时间的方式，实现在亚秒级别延迟的情况下，对Hadoop上的大规模数据集进行交互式查询；Kylin通过预计算，把计算结果集保存在HBase中，原有的基于行的关系模型被转换成基于键值对的列式存储；通过维度组合作为HBase的Rowkey，在查询访问时不再需要昂贵的表扫描，这为高速高并发分析带来了可能；Kylin提供了标准SQL查询接口，支持大多数的SQL函数，同时也支持ODBC/JDBC的方式和主流的BI产品无缝集成。

同时，Apache Kylin是目前国内少有的几个通过了Cloudera公司产品工程认证的大数据分析和查询引擎。Cloudera公司相信，作为唯一一个来

自中国的Apache顶级开源项目，Apache Kylin不仅仅代表了中国对国际开源社区的参与，同时也将为我国及全球企业用户探索大数据的价值的进程做出卓越的贡献。

在过去的一年中，我们有机会与Kyligence公司合作，共同为国内的企业客户提供基于Cloudera Hadoop平台上的大数据应用。本书的出版为开发人员和数据分析人员利用这一技术提供了极大的便利。更重要的是，这本书不仅能够指导开发人员安装和使用Apache Kylin，而且还深入探讨了Apache Kylin的核心技术架构，并且通过丰富的案例展示了如何通过优化来提升大数据的应用性能。本书的作者之一韩卿先生是Apache Kylin的主要创建者和项目委员会主席（PMC chair），对于Kylin的技术架构、应用及未来发展都有深刻的理解。我相信本书对于Kylin使用者和开发者来说，是及时的且不可或缺的。

凌琦

Cloudera全球副总裁兼大中华区总经理

推荐序四

大数据在近几年已经成为一个火爆的名词，而企业针对数据的分析也从未停止过。从早些年传统企业的数据仓库、BI，到近些年互联网公司的广告推荐、产品分析，再到现在基于IoT硬件的线下用户行为画像，无论是互联网企业还是传统企业，一直都在尝试通过数据帮助企业或企业的用户提升工作效率和体验。从过去的决策支持，到现在普及的精准推荐，乃至未来的基于实时分析的AI交互，大数据及相关技术将一直是这些业务发展的基石，因而在最近的10年，大数据技术有了日新月异的发展。

从海量数据的批量计算到实时分析，从精准推荐到OLAP查询，业界涌现了大量优秀的开源项目。Apache Kylin就是其中一颗由国人研发的璀璨的明星，是国内第一个Apache顶级开源项目（与Kafka、Spark齐名），它解决了海量数据下OLAP查询的关键技术。大数据本身并不能产生价值，针对数据的分析和运用才可以产生价值，而OLAP是企业对数据做深度分析必用的组件。在过去，它能帮助企业从不同维度汇总、下钻看到企业不同部门、地区的差异及发展趋势；现在，它能帮助企业针对不同用户画像的人群做相关行为分析、排行，也可以针对不同的点击事件深入分析不同渠道的转化率、客单价。OLAP技术曾经在百亿数据集、PB级别规模的时候，遇到了很大的瓶颈，无论是并行计算还是近似计算，都对I/O、CPU

和查询时长带来了挑战。Kylin运用它独有的技术，在数据存储不产生指数级增长的情况下，采用预计算技术以空间换回了时间，在百亿甚至万亿级别数据集上实现了毫秒级的查询响应速度。同时也利用了模糊计算等技术在允许一定误差的情况下，对10亿级别用户、几千种用户行为标签的数据实现了用户行为的即时查询，帮助企业极大地降低了大数据OLAP实施的门槛，也降低了大数据平台实施的TCO，是企业建设大数据平台的优质OLAP引擎。本书可以帮助企业的技术管理者、开发者详细了解Kylin并将应用部署到自己的企业当中，规避其中的实施风险、提高部署与处理效率。

数据是一种新的能源，它与石油、电力不同，产生于企业和用户的行为，能通过不断地深入使用和反复分析利用来帮助企业增收、节支、提效、避险，其中各个环节都要有适用的工具，Apache Kylin就是其中之一。大数据从过去的批量数据处理发展到现在的实时数据分析，我非常高兴地看到Kylin也支持了相关特性，让数据不止是用于实时计算，还可以帮助管理者看到实时的联机分析处理结果。当然，数据的实时OLAP只是实时分析中的一种，要结合数据实时采集、数据实时计算、数据流挖掘、实时场景引擎等技术，才可以让企业从T+1的分析发展到实时数据分析，进而实现实时决策与反馈，最终实现企业自身的智能分析与交互。数据的实时分析是每个企业实现AI的必经之路，而数据实时分析的应用又离不开Kylin这样的OLAP引擎。

最后，很荣幸可以为本书写推荐序，本书作者之一韩卿（Luke）也是我多年的好友，从他在eBay之时我们就有很多交流，我也有幸看着Apache Kylin项目逐步成为国际著名的开源项目。大数据的发展不是一个项目或一个企业就可以独立推动的，也希望更多的人才和企业加入大数据分析的行业中来，使得我国能够涌现出更多像Apache Kylin一样的优秀项目，让数据成为每一个企业的再生能源！

郭炜

易观CTO

前言

“麒麟出没，必有祥瑞。”

——中国古谚语

“于我而言，与Apache Kylin团队一起合作使Kylin通过孵化成为顶级项目是非常激动人心的，诚然，Kylin在技术方面非常振奋人心，但同样令人兴奋的是Kylin代表了亚洲国家，特别是中国，在开源社区中越来越高的参与度。”

——Ted Dunning Apache孵化项目副总裁，MapR首席应用架构师

今天，随着移动互联网、物联网、AI等技术的快速兴起，数据成为了所有这些技术背后最重要，也是最有价值的“资产”。如何从数据中获得有价值的信息？这个问题驱动了相关技术的发展，从最初的基于文件的检索、分析程序，到数据仓库理念的诞生，再到基于数据库的商业智能分析。而现在，这一问题已经变成了如何从海量的超大规模数据中快速获取有价值的信息，新的时代、新的挑战、新的技术必然应运而生。

在数据分析领域，大部分的技术都诞生在国外，特别是美国，从最初的数据库，到以Hadoop为首的大数据技术，再到今天各种DL（Deep Learning）、AI，等等。但我国拥有着世界上独一无二的“大”数据，最多的

人口、最多的移动设备、最活跃的应用市场、最复杂的网络环境等，应对这些挑战，我们需要有自己的核心技术，特别是在基础领域的突破和研发方面。今天，以Apache Kylin为首的各种来自中国的先进技术不断涌现，甚至在很多方面都大大超越了国外的其他技术，这一点也彰显了中国的技术实力。

自Hadoop选取大象伊始，上百个项目，以动物居之者为多，而其中唯有Apache Kylin（麒麟）来自中国，在众多项目中分外突出。在全球最大的开源基金会——Apache软件基金会（Apache Software Foundation，ASF）的160多个顶级项目中，Apache Kylin是唯一一个来自中国的顶级开源项目，与Apache Hadoop、Apache Spark、Apache Kafka、Apache Tomcat、Apache Struts、Apache Maven等顶级项目一起以The Apache Way构建了开源大数据领域的国际社区，并拓展了生态系统。

大数据与传统技术最大的区别就在于数据的体量对查询带来的巨大挑战。从最早使用大数据技术来做批量处理，到现在越来越多地需要大数据平台也能够如传统数据仓库技术一样支持交互式分析。随着数据量的不断膨胀，数据平民化的不断推进，低延迟、高并发地在Hadoop之上提供标准SQL查询的能力成为必须要攻破的技术难题。而Apache Kylin的诞生正是基于这个背景，并成功地完成了很多人认为不可能实现的突破。Apache Kylin最初诞生于eBay中国研发中心（坐落于上海浦东新区的德国中心），在2013年9月底，eBay中国研发中心的技术人员开始对此进行POC

并组建团队，经过一年的艰苦开发和测试，于2014年9月30日使其正式上线，并在第二天（2014年10月1日）正式开源。

在这个过程中，使用何种技术，如何进行架构，如何突破那些看似无法完成的挑战，整个开发团队和用户一起经历了一段艰难的历程。今天呈现出的Apache Kylin已经经历了上千亿乃至上万亿规模数据量的分析请求，以及上百家公司的实际生产环境的检验，成为各个公司大数据分析平台不可替代的重要部分。本书将从Apache Kylin的架构和设计、各个模块的使用、与第三方的整合、二次开发及开源实践等方面进行讲解，为各位读者呈现最核心的设计理念和哲学、算法和技术等。

Apache Kylin社区的发展不易，自2014年10月开源到今天已有两年，从最初的几个人发展到今天的几十个贡献者，国内外上百家公司在正式使用，连续两年获得InfoWorld Bossie Awards最佳开源大数据工具奖。来自核心团队、贡献者、用户、导师、基金会等的帮助和无私的奉献铸就了这个活跃的社区，也使得Apache Kylin得以在越来越多的场景下发挥作用。现在，由Apache Kylin核心团队撰写了本书，相信能更好地将相关的理论、设计、技术、架构等展现给各位朋友，希望能够让更多的朋友更加充分地理解Kylin的优点和使用的场景，更多地挖掘出Kylin的潜力。同时也希望本书能够鼓励并吸引更多的人参与Kylin项目和开源项目，影响更多人贡献更多的项目和技术到开源世界来。

韩卿

Apache Kylin联合创建者及项目委员会主席

2016年10月

第1章　Apache Kylin概述

Apache Kylin是Hadoop大数据平台上的一个开源OLAP引擎。它采用多维立方体预计算技术，可以将大数据的SQL查询速度提升到亚秒级别。相对于之前的分钟乃至小时级别的查询速度，亚秒级别速度是百倍到千倍的提升，该引擎为超大规模数据集上的交互式大数据分析打开了大门。

Apache Kylin也是中国人主导的、唯一的Apache顶级开源项目，在开源社区有世界级的影响力。

本章将对Apache Kylin的历史和背景做一个完整的介绍，并从技术的角度对Kylin做一个概览性的介绍。

本书内容以Apache Kylin v1.5为基础。

1.1 背景和历史

今天，大数据领域的发展如火如荼，各种新技术层出不穷，整个生态欣欣向荣。作为大数据领域最重要的技术——Apache Hadoop，从诞生至今已有10周年。它最初只是致力于简单的分布式存储，然后在其之上实现大规模并行计算，到如今它已在实时分析、多维分析、交互式分析、机器学习甚至人工智能等方面都有着长足的发展。

2013年年初，eBay内部使用的传统数据仓库及商业智能平台应用碰到了瓶颈，即传统的架构只支持垂直扩展，通过在一台机器上增加CPU和内存等资源来提升数据处理能力，相对于数据指数级的增长，单机扩展很快就达到了极限。另一方面，Hadoop大数据平台虽然能存储和批量处理大规模数据，但与BI平台的连接技术依然不成熟，无法提供高效的交互式查询。于是寻找更好的方案便成为了当务之急。正好在2013年年中的时候eBay公司启动了一个大数据项目，其中的一块内容就是BI on Hadoop的预研。当时eBay中国卓越中心组建了一支很小的团队，他们在分析和测试了多种开源和商业解决方案之后，发现没有一种方案能够完全满足当时的需求，即在超大规模数据集上提供秒级的查询性能，并能基于Hadoop与BI平台无缝整合等。在研究了多种可能性之后，最终eBay的Apache Kylin核心团队决定自己实现一套OLAP on Hadoop的解决方案，以弥补业界的这个空白。与此同时，eBay公司也非常鼓励开源各个项目，

回馈社区，eBay的Apache Kylin核心团队在向负责整个技术平台的高级副总裁做汇报的时候，得到的一个反馈就是“要从第一天就做好开源的准备”。

图1-1 Hortonworks CTO在Twitter上对Apache Kylin的评论

经过一年多的研发，在2014年的9月底，Kylin平台在eBay内部正式上线。它一上线便吸引了多个种子客户。Kylin在Hadoop上提供了标准的、友好的SQL接口，外加查询速度非常迅速，原本要用几分钟的查询现在几秒钟就能返回结果，BI分析的工作效率得到了几百倍的提升，因此Kylin获得了公司内部客户、合作伙伴及管理层的高度评价。2014年10月1日，项目负责人韩卿将Kylin的源代码提交到github.com并正式开源，当天就获得了业界专家的关注和认可，如图1-1所示的是Hortonworks的CTO在

Twitter上对此给出的评价。

很快，Hadoop社区的许多朋友都鼓励eBay的Apack Kylin核心团队将该项目贡献到Apache软件基金会（ASF），让它能够与其他大数据项目一起获得更好的发展，在经过一个月的紧张筹备和撰写了无数个版本的项目建议书之后，Kylin项目于2014年11月正式加入Apache孵化器项目，并有多位资深的社区活跃成员作我们的导师。

在项目组再次付出无数努力之后，2015年的11月，Apache软件基金会宣布Apache Kylin正式成为顶级项目。这是第一个也是唯一一个完全由我国团队贡献到全球最大的开源软件基金会的顶级项目。项目负责人韩卿成为Apache Kylin的项目管理委员会（PMC）主席，也是Apache软件基金会160多个顶级项目中的唯一一个中国人，Apache Kyln创造了历史。正如Kylin的导师——Apache孵化器副总裁Ted Dunning在ASF官方新闻稿中的评价："…Apache Kylin代表了亚洲国家，特别是中国，在开源社区中越来越高的参与度…"。

2016年3月，由Apache Kylin核心开发者组建的创业公司Kyligence正式成立。就如每一个成功的开源项目背后都有一家创业公司一样（Hadoop领域有Cloudera、Hortoworks等；Spark领域有Databricks；Kafka领域有Conf？luent），Kylin也可以通过Kyligence的进一步投入保持高速研发，并且Kylin的社区和生态圈也会得到不断的发展和壮大，可以预见这个开源项目将会越来越好。

在业界最负盛名的技术类独立评选中，InfoWorld的Bossie Award每年都会独立挑选和评论相关的技术、应用和产品等。2015年9月，Apache Kylin获得了2015年度的“最佳开源大数据工具奖”，2016年9月，Apache Kylin再次蝉联此国际大奖，与Google TensorFlow齐名。这是业界对Apache Kylin的充分认可和褒奖。

1.2 Apache Kylin的使命

Kylin的使命是超高速的大数据OLAP（Online Analytical Processing），也就是要让大数据分析像使用数据库一样简单迅速，用户的查询请求可以在秒内返回，交互式数据分析将以前所未有的速度释放大数据里潜藏的知识和信息，让我们在面对未来的挑战时占得先机。

1.2.1 为什么要使用Apache Kylin

自从10年前Hadoop诞生以来，大数据的存储和批处理问题均得到了妥善解决，而如何高速地分析数据也就成为了下一个挑战。于是各式各样的“SQL on Hadoop”技术应运而生，其中以Hive为代表，Impala、Presto、Phoenix、Drill、SparkSQL等紧随其后。它们的主要技术是“大规模并行处理”（Massive Parallel Processing，MPP）和“列式存储”（Columnar Storage）。大规模并行处理可以调动多台机器一起进行并行计算，用线性增加的资源来换取计算时间的线性下降。列式存储则将记录按列存放，这样做不仅可以在访问时只读取需要的列，还可以利用存储设备擅长连续读取的特点，大大提高读取的速率。这两项关键技术使得Hadoop上的SQL查询速度从小时提高到了分钟。

然而分钟级别的查询响应仍然离交互式分析的现实需求还很远。分析师敲入查询指令，按下回车，还需要去倒杯咖啡，静静地等待查询结果。得到结果之后才能根据情况调整查询，再做下一轮分析。如此反复，一个具体的场景分析常常需要几小时甚至几天才能完成，效率低下。

这是因为大规模并行处理和列式存储虽然提高了计算和存储的速度，但并没有改变查询问题本身的时间复杂度，也没有改变查询时间与数据量成线性增长的关系这一事实。假设查询1亿条记录耗时1分钟，那

么查询10亿条记录就需10分钟，100亿条记录就至少需要1小时40分钟。当然，可以用很多的优化技术缩短查询的时间，比如更快的存储、更高效的压缩算法，等等，但总体来说，查询性能与数据量呈线性相关这一点是无法改变的。虽然大规模并行处理允许十倍或百倍地扩张计算集群，以期望保持分钟级别的查询速度，但购买和部署十倍或百倍的计算集群又怎能轻易做到，更何况还有高昂的硬件运维成本。

另外，对于分析师来说，完备的、经过验证的数据模型比分析性能更加重要，直接访问纷繁复杂的原始数据并进行相关分析其实并不是很友好的体验，特别是在超大规模的数据集上，分析师将更多的精力花在了等待查询结果上，而不是在更加重要的建立领域模型上。

1.2.2 Apache Kylin怎样解决关键问题

Apache Kylin的初衷就是要解决千亿条、万亿条记录的秒级查询问题，其中的关键就是要打破查询时间随着数据量成线性增长的这个规律。仔细思考大数据OLAP，可以注意到两个事实。

·大数据查询要的一般是统计结果，是多条记录经过聚合函数计算后的统计值。原始的记录则不是必需的，或者访问频率和概率都极低。

·聚合是按维度进行的，由于业务范围和分析需求是有限的，有意义的维度聚合组合也是相对有限的，一般不会随着数据的膨胀而增长。

基于以上两点，我们可以得到一个新的思路——“预计算”。应尽量多地预先计算聚合结果，在查询时刻应尽量使用预算的结果得出查询结果，从而避免直接扫描可能无限增长的原始记录。

举例来说，使用如下的SQL来查询10月1日那天销量最高的商品：

```
select item, sum(sell_amount)
from sell_details
where sell_date='2016-10-01'
group by item
order by sum(sell_amount) desc
```

用传统的方法时需要扫描所有的记录，再找到10月1日的销售记录，

然后按商品聚合销售额，最后排序返回。假如10月1日有1亿条交易，那么查询必须读取并累计至少1亿条记录，且这个查询速度会随将来销量的增加而逐步下降。如果日交易量提高一倍到2亿，那么查询执行的时间可能也会增加一倍。

而使用预计算的方法则会事先按维度[sell_date, item]计算sum(sell_amount)并存储下来，在查询时找到10月1日的销售商品就可以直接排序返回了。读取的记录数最大不会超过维度[sell_date, item]的组合数。显然这个数字将远远小于实际的销售记录，比如10月1日的1亿条交易包含了100万条商品，那么预计算后就只有100万条记录了，是原来的百分之一。并且这些记录已经是按商品聚合的结果，因此又省去了运行时的聚合运算。从未来的发展来看，查询速度只会随日期和商品数目的增长而变化，与销售记录的总数不再有直接联系。假如日交易量提高一倍到2亿，但只要商品的总数不变，那么预计算的结果记录总数就不会变，查询的速度也不会变。

“预计算”就是Kylin在“大规模并行处理”和“列式存储”之外，提供给大数据分析的第三个关键技术。

1.3 Apache Kylin的工作原理

Apache Kylin的工作原理本质上是MOLAP（Multidimensional Online Analytical Processing）Cube，也就是多维立方体分析。这是数据分析中相当经典的理论，在关系数据库年代就已经有了广泛的应用，下面将对其做简要介绍。

1.3.1 维度和度量简介

在说明MOLAP Cube之前需要先介绍一下维度(Dimension)和度量(Measure)这两个概念。

简单来讲，维度就是观察数据的角度。比如电商的销售数据，可以从时间的维度来观察(如图1-2的左侧所示)，也可以进一步细化，从时间和地区的维度来观察(如图1-2的右侧所示)。维度一般是一组离散的值，比如时间维度上的每一个独立的日期，或者商品维度上的每一件独立的商品。因此统计时可以把维度值相同的记录聚合在一起，然后应用聚合函数做累加、平均、去重复计数等聚合计算。

时间（维度）	销售额（度量）
2016 1Q	1.7 M
2016 2Q	2.1 M
2016 3Q	1.6 M
2016 4Q	1.8 M

时间（维度）	地区（维度）	销售额（度量）
2016 1Q	中国	1.0 M
2016 1Q	北美	0.7 M
2016 2Q	中国	1.5 M
2016 2Q	北美	0.6 M
2016 3Q	中国	0.9 M
2016 3Q	北美	0.7 M
2016 4Q	中国	0.9 M
2016 4Q	北美	0.9 M

图1-2 维度和度量的例子

度量就是被聚合的统计值，也是聚合运算的结果，它一般是连续的值，如图1-2中的销售额，抑或是销售商品的总件数。通过比较和测算度量，分析师可以对数据进行评估，比如今年的销售额相比去年有多大的增长，增长的速度是否达到预期，不同商品类别的增长比例是否合理等。

1.3.2 Cube和Cuboid

有了维度和度量，一个数据表或数据模型上的所有字段就可以分类了，它们要么是维度，要么是度量（可以被聚合）。于是就有了根据维度和度量做预计算的Cube理论。

给定一个数据模型，我们可以对其上的所有维度进行组合。对于N个维度来说，组合的所有可能性共有2^N种。对于每一种维度的组合，将度量做聚合运算，然后将运算的结果保存为一个物化视图，称为Cuboid。所有维度组合的Cuboid作为一个整体，被称为Cube。所以简单来说，一个Cube就是许多按维度聚合的物化视图的集合。

下面来列举一个具体的例子。假定有一个电商的销售数据集，其中维度包括时间（Time）、商品（Item）、地点（Location）和供应商（Supplier），度量为销售额（GMV）。那么所有维度的组合就有2^4=16种（如图1-3所示），比如一维度（1D）的组合有[Time]、[Item]、[Location]、[Supplier]4种；二维度（2D）的组合有[Time，Item]、[Time，Location]、[Time、Supplier]、[Item，Location]、[Item，Supplier]、[Location，Supplier]6种；三维度（3D）的组合也有4种；最后零维度（0D）和四维度（4D）的组合各有1种，总共就有16种组合。

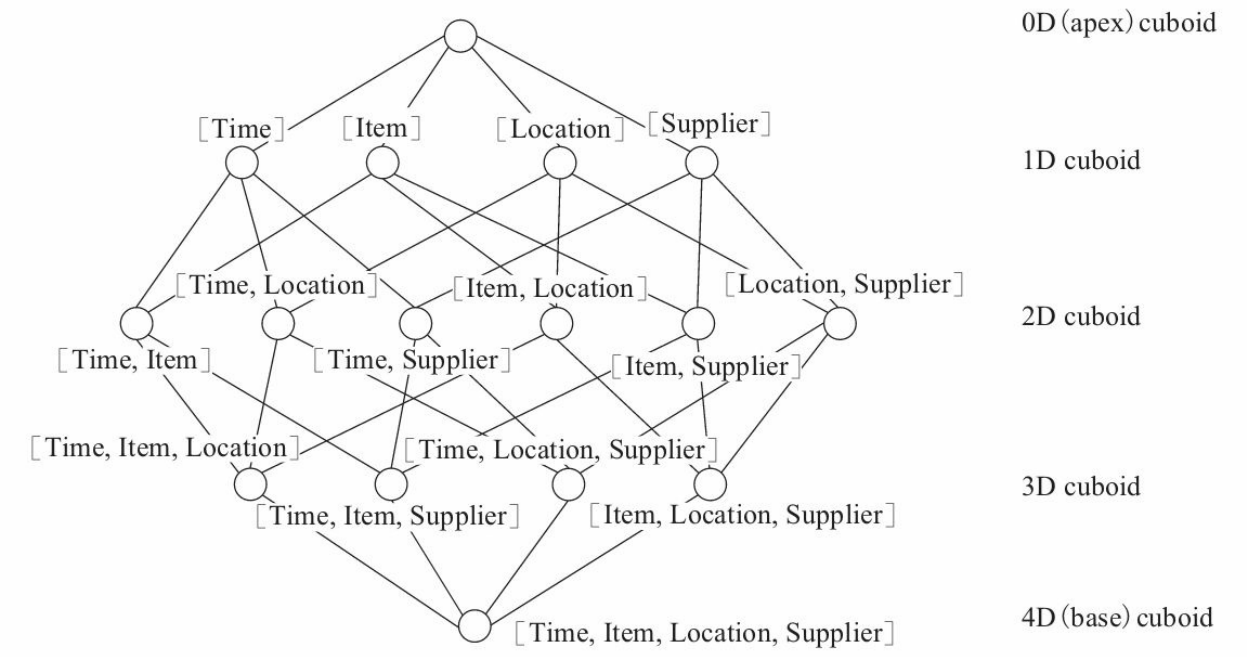


图1-3 一个四维Cube的例子

计算Cuboid，即按维度来聚合销售额。如果用SQL语句来表达计算Cuboid[Time，Loca-tion]，那么SQL语句如下：

```
select Time, Location, Sum(GMV) as GMV from Sales group by Time, Location
```

将计算的结果保存为物化视图，所有Cuboid物化视图的总称就是Cube。

1.3.3 工作原理

Apache Kylin的工作原理就是对数据模型做Cube预计算，并利用计算的结果加速查询，具体工作过程如下。

1）指定数据模型，定义维度和度量。

2）预计算Cube，计算所有Cuboid并保存为物化视图。

3）执行查询时，读取Cuboid，运算，产生查询结果。

由于Kylin的查询过程不会扫描原始记录，而是通过预计算预先完成表的关联、聚合等复杂运算，并利用预计算的结果来执行查询，因此相比非预计算的查询技术，其速度一般要快一到两个数量级，并且这点在超大的数据集上优势更明显。当数据集达到千亿乃至万亿级别时，Kylin的速度甚至可以超越其他非预计算技术1000倍以上。

1.4 Apache Kylin的技术架构

Apache Kylin系统可以分为在线查询和离线构建两部分，技术架构如图1-4所示，在线查询的模块主要处于上半区，而离线构建则处于下半区。

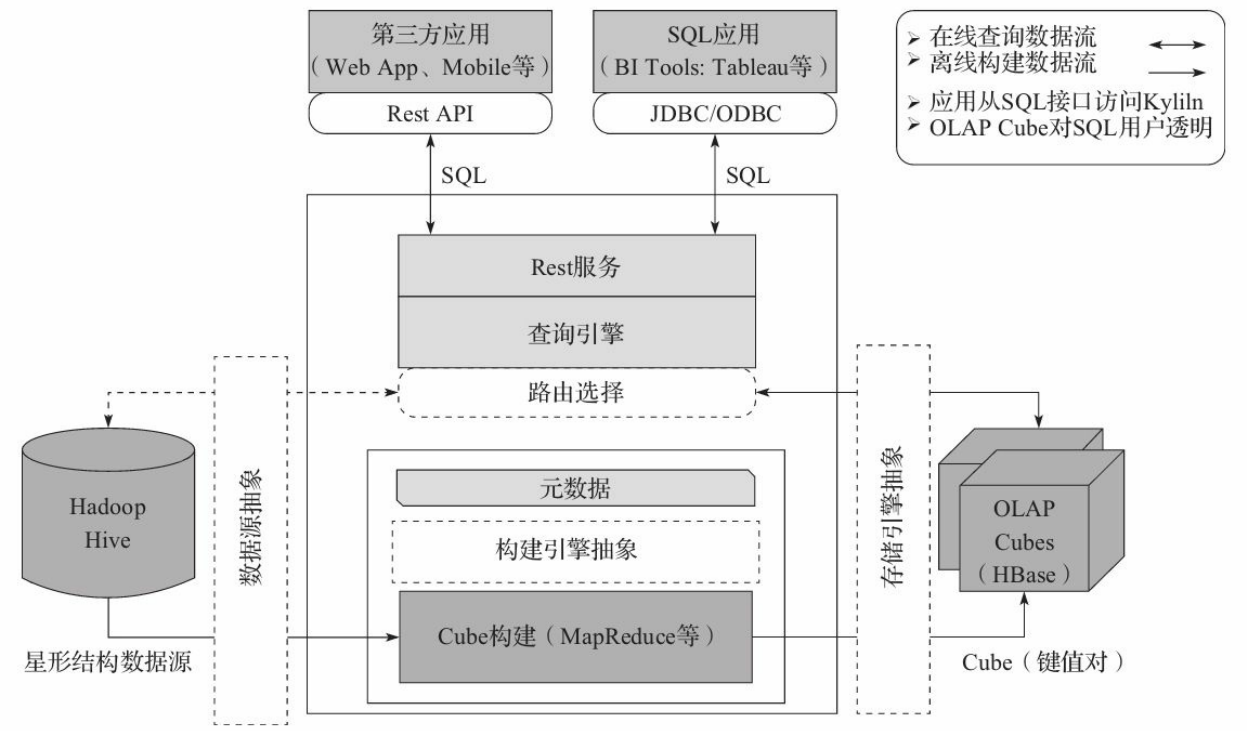


图1-4 Kylin的技术架构

我们首先来看看离线构建的部分。从图1-4可以看出，数据源在左侧，目前主要是Hadoop Hive，保存着待分析的用户数据。根据元数据的定义，下方构建引擎从数据源抽取数据，并构建Cube。数据以关系表的形式输入，且必须符合星形模型（Star Schema）（更复杂的雪花模型在成文时

还不被支持，可以用视图将雪花模型转化为星形模型，再使用Kylin）。MapReduce是当前主要的构建技术。构建后的Cube保存在右侧的存储引擎中，一般选用HBase作为存储。

完成了离线构建之后，用户可以从上方查询系统发送SQL进行查询分析。Kylin提供了各种Rest API、JDBC/ODBC接口。无论从哪个接口进入，SQL最终都会来到Rest服务层，再转交给查询引擎进行处理。这里需要注意的是，SQL语句是基于数据源的关系模型书写的，而不是Cube。Kylin在设计时刻意对查询用户屏蔽了Cube的概念，分析师只需要理解简单的关系模型就可以使用Kylin，没有额外的学习门槛，传统的SQL应用也很容易迁移。查询引擎解析SQL，生成基于关系表的逻辑执行计划，然后将其转译为基于Cube的物理执行计划，最后查询预计算生成的Cube并产生结果。整个过程不会访问原始数据源。

·注意　对于查询引擎下方的路由选择，在最初设计时曾考虑过将Kylin不能执行的查询引导去Hive中继续执行，但在实践后发现Hive与Kylin的速度差异过大，导致用户无法对查询的速度有一致的期望，很可能大多数查询几秒内就返回结果了，而有些查询则要等几分钟到几十分钟，因此体验非常糟糕。最后这个路由功能在发行版中默认关闭，因此在图1-4中是用虚线表示的。

Apache Kylin1.5版本引入了“可扩展架构”的概念。在图1-4中显示为三个粗虚线框表示的抽象层。可扩展指Kylin可以对其主要依赖的三个模

块做任意的扩展和替换。Kylin的三大依赖模块分别是数据源、构建引擎和存储引擎。在设计之初，作为Hadoop家族的一员，这三者分别是Hive、MapReduce和HBase。但随着推广和使用的深入，渐渐有用户发现它们均存在不足之处。比如，实时分析可能会希望从Kafka导入数据而不是从Hive；而Spark的迅速崛起，又使我们不得不考虑将MapReduce替换为Spark，以期大幅提高Cube的构建速度；至于HBase，它的读性能可能还不如Cassandra或Kudu等。可见，是否可以将一种技术替换为另一种技术已成为一个常见的问题。于是我们对Kylin1.5版本的系统架构进行了重构，将数据源、构建引擎、存储引擎三大依赖抽象为接口，而Hive、MapReduce、HBase只是默认实现。深度用户可以根据自己的需要做二次开发，将其中的一个或多个替换为更适合的技术。

这也为Kylin技术的与时俱进埋下了伏笔。如果有一天更先进的分布式计算技术取代了MapReduce，或者更高效的存储系统全面超越了HBase，Kylin可以用较小的代价将一个子系统替换掉，从而保证Kylin能够紧跟技术发展的最新潮流，从而保持最高的技术水平。

可扩展架构也带来了额外的灵活性，比如，它可以允许多个引擎同时并存。例如Kylin可以同时对接Hive、Kafka和其他第三方数据源；抑或用户可以为不同的Cube指定不同的构建引擎或存储引擎，以期达到最极致的性能和功能定制。

1.5 Apache Kylin的主要特点

Apache Kylin的主要特点包括支持SQL接口、支持超大数据集、秒级响应、可伸缩性、高吞吐率、BI工具集成等。

1.5.1 标准SQL接口

Apache Kylin以标准SQL作为对外服务的主要接口。因为SQL是绝大多数分析人员最熟悉的工具，同时也是大多数应用程序使用的编程接口。尽管Kylin内部以Cube技术为核心，对外却没有选用MDX（MultiDimensional eXpressions）作为接口。虽然MDX作为OLAP查询语言，从学术上来说，它是更加适合Kylin的选择，然而实践表明，SQL简单易用，代表了绝大多数用户的第一需求，这也是Kylin能够快速推广的一个关键。

SQL需要以关系模型作为支撑。Kylin使用的查询模型是数据源中的关系模型表，一般而言，也就是指Hive表。终端用户只需要像原来查询Hive表一样编写SQL，就可以无缝地切换到Kylin，几乎不需要额外的学习，甚至原本的Hive查询也因为与SQL同源，大多都无须修改就能直接在Kylin上运行。

Apache Kylin在将来也可能会推出MDX接口。事实上已经有方法可以通过MDX转SQL的工具，让Kylin也能支持MDX。

1.5.2 支持超大数据集

Apache Kylin对大数据的支撑能力可能是目前所有技术中最为领先的。早在2015年eBay的生产环境中Kylin就能支持百亿记录的秒级查询，之后在移动的应用场景下又有了千亿记录秒级查询的案例。这些都是实际场景的应用，而非实验室中的理论数据。

因为使用了Cube预计算技术，在理论上，Kylin可以支撑的数据集大小没有上限，仅受限于存储系统和分布式计算系统的承载能力，并且查询速度不会随数据集的增大而减慢。Kylin在数据集规模上的局限性主要在于维度的个数和基数。它们一般由数据模型来决定，不会随着数据规模的增长而线性增长，这也意味着Kylin对未来数据的增长有着更强的适应能力。

如今（截至2016年5月），对于Apache Kylin，除了eBay将其作为孵化公司有广泛应用之外，国内外一线的互联网公司对此几乎都有大规模的使用，包括百度、网易、京东、美团、唯品会、Expedia等。此外，其在传统行业中也有非常多的实际应用，包括中国移动、银联、国美等。据不完全统计，真实上线的Apache Kylin用户已经超过了一百多家，在开源后一年多一点的时间内能有如此大的全球用户基础，足见Kylin在处理超大规模数据集上的能力和优势。

1.5.3 亚秒级响应

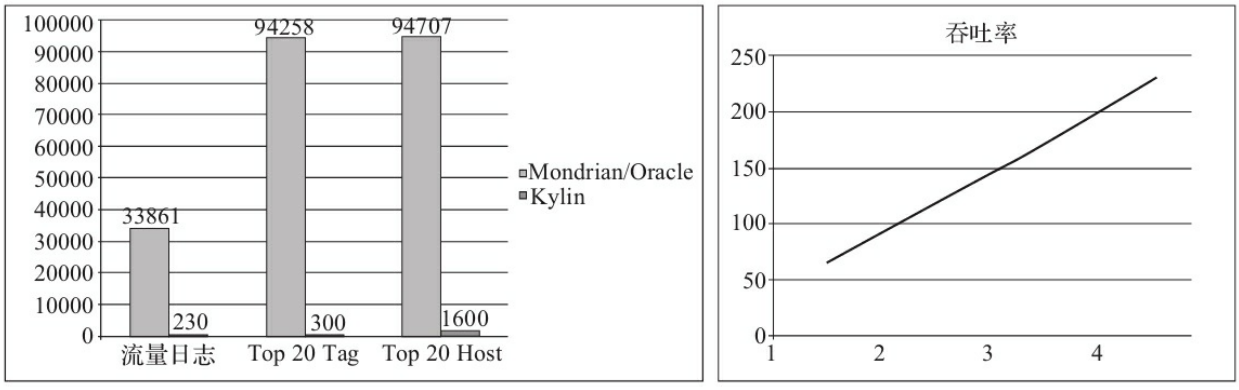
Apache Kylin拥有优异的查询响应速度，这点得益于预计算，很多复杂的计算，比如连接、聚合，在离线的预计算过程中就已经完成，这大大降低了查询时刻所需要的计算量，提高了响应速度。

根据可查询到的公开资料可以得知，Apache Kylin在某生产环境中90%的查询可以在3s内返回结果。这并不是说一小部分SQL相当快，而是在数万种不同SQL的真实生产系统中，绝大部分的查询都非常迅速；在另外一个真实的案例中，对1000多亿条数据构建了立方体，90%的查询性能都在1.18s以内，可见Kylin在超大规模数据集上表现优异。这与一些只在实验室中，只在特定查询情况下采集的性能数据不可同日而语。当然并不是使用Kylin就一定能获得最好的性能。针对特定的数据及查询模式，往往需要做进一步的性能调优、配置优化等，性能调优对于充分利用好Apache Kylin至关重要。

1.5.4 可伸缩性和高吞吐率

在保持高速响应的同时，Kylin有着良好的可伸缩性和很高的吞吐率。图1-5是来自网易的性能分享。图1-5中左侧是Kylin查询速度与Mondrian/Oracle的对比，可以看到在3个测试查询中，Kylin分别比Mondrian/Oracle快147倍、314倍和59倍。

同时，图1-5中右侧展现了Kylin的吞吐率及其可伸缩性。在只有1个Kylin实例的情况下，Kylin每秒可以处理近70个查询，已经远远高于每秒20个查询的一般水平。更为理想的是，随着服务器的增加，吞吐率也呈线性增加，存在4个实例时可达到每秒230个查询左右，而这4个实例仅部署在一台机器上，理论上添加更多的应用服务器后可以支持更大的并发率。



By NetEase:
http://www.bitstech.net/2016/01/04/kylin-olap/

图1-5 Kylin的可伸缩性和吞吐率

这主要还是归功于预计算降低了查询时所需的计算总量，令Kylin可以在相同的硬件配置下承载更多的并发查询。

1.5.5 BI及可视化工具集成

Apache Kylin提供了丰富的API，以与现有的BI工具集成，具体包括如下内容。

·ODBC接口，与Tableau、Excel、Power BI等工具集成。

·JDBC接口，与Saiku、BIRT等Java工具集成。

·Rest API，与JavaScript、Web网页集成。

分析师可以沿用他们最熟悉的BI工具与Kylin一同工作，或者在开放的API上做二次开发和深度定制。

另外，Kylin核心开发团队也贡献了Apache Zeppelin的插件，现在已经可以用Zeppelin来访问Kylin服务。

1.6 与其他开源产品比较

与Apache Kylin一样致力于解决大数据查询问题的其他开源产品也有不少，比如Apache Drill、Apache Impala、Druid、Hive、Presto（Facebook）、SparkSQL等。本节试图将Kylin与它们做一个简单的比较。

从底层技术的角度来看，这些开源产品有很大的共性，一些底层技术几乎被所有的产品一致采用，Kylin也不例外。

·大规模并行处理：可以通过增加机器的方式来扩容处理速度，在相同的时间里处理更多的数据。

·列式存储：通过按列存储提高单位时间里数据的I/O吞吐率，还能跳过不需要访问的列。

·索引：利用索引配合查询条件，可以迅速跳过不符合条件的数据块，仅扫描需要扫描的数据内容。

·压缩：压缩数据然后存储，使得存储的密度更高，在有限的I/O速率下，在单位时间里读取更多的记录。

综上所述，我们可以注意到，所有这些方法都只是提高了单位时间内处理数据的能力，当大家都一致采用这些技术时，它们之间的区别将

只停留在实现层面的代码细节上。最重要的是，这些技术都不会改变一个事实，那就是处理时间与数据量之间的正比例关系。当数据量翻倍时，MPP（在不扩容的前提下）需要翻倍的时间来完成计算；列式存储需要翻倍的存储空间；索引下符合条件的记录数也会翻倍；压缩后的数据大小也还是之前的两倍。因此查询速度也会随之变成之前的两倍。当数据量成十倍百倍地增长时，这些技术的查询速度就会成十倍百倍地下降，最终变得不能接受。

Apache Kylin的特色在于，在上述的底层技术之外，另辟蹊径地使用了独特的Cube预计算技术。预计算事先将数据按维度组合进行了聚合，将结果保存为物化视图。经过聚合，物化视图的规模就只由维度的基数来决定，而不再随着数据量的增长呈线性增长。以电商为例，如果业务扩张，交易量增长了10倍，只要交易数据的维度不变（供应商/商品数量不变），聚合后的物化视图将依旧是原先的大小，查询的速度也将保持不变。

与那些类似产品相比，这一底层技术的区别使得Kylin从外在功能上呈现出了不同的特性，具体如下。

·SQL接口：除了Druid以外，所有的产品都支持SQL或类SQL接口。巧合的是，Druid也是除了Kylin以外，查询性能相对更好的一个。这点除了Druid有自己的存储引擎之外，可能还得益于其较为受限的查询能力。

·大数据支持：大多数产品的能力在亿级到十亿级数据量之间，再大的数据量将显著降低查询的性能。而Kylin因为采用预计算技术，因此查询速度不受数据量限制。有实际案例证明数据量在千亿级别时，Kylin系统仍然能够保有秒级别的查询性能。

·查询速度：如前文所述，一般产品的查询速度都会不可避免地随着数据量的增长而下降，而Kylin则能够在数据量成倍增长的同时，查询速度保持不变，而且这个差距也将随着数据量的成倍增长而变得愈加明显。

·吞吐率：根据之前的实验数据，Kylin的单例吞吐量一般在每秒70个查询左右，并且可以线性扩展，而普通的产品因为所有计算都在查询时完成，所以需要调动集群的更多资源才能完成查询，通常极限在每秒20个查询左右，而且扩容成本较高，需要扩展整个集群。相对的，Kylin系统因为瓶颈不在整个集群，而在于Kylin服务器，因此只需要增加Kylin服务器就能成倍地提高吞吐率，扩容成本低廉。

1.7 小结

本章介绍了Apache Kylin的历史背景和技术特点。尤其是它基于预计算的大数据查询原理，理论上可以在任意大的数据规模上达到O(1)常数级别的查询速度，这一点也是Apache Kylin与传统查询技术的关键区别，如图1-6所示。传统技术，如大规模并行计算和列式存储的查询速度都在O(N)级别，与数据规模增线性关系。如果数据规模增长10倍，那么O(N)的查询速度就会下降到十分之一，无法满足日益增长的数据需求。依靠Apache Kylin，我们不用再担心查询速度会随着数据量的增长而减慢，面对未来的数据挑战时也能更有信心。

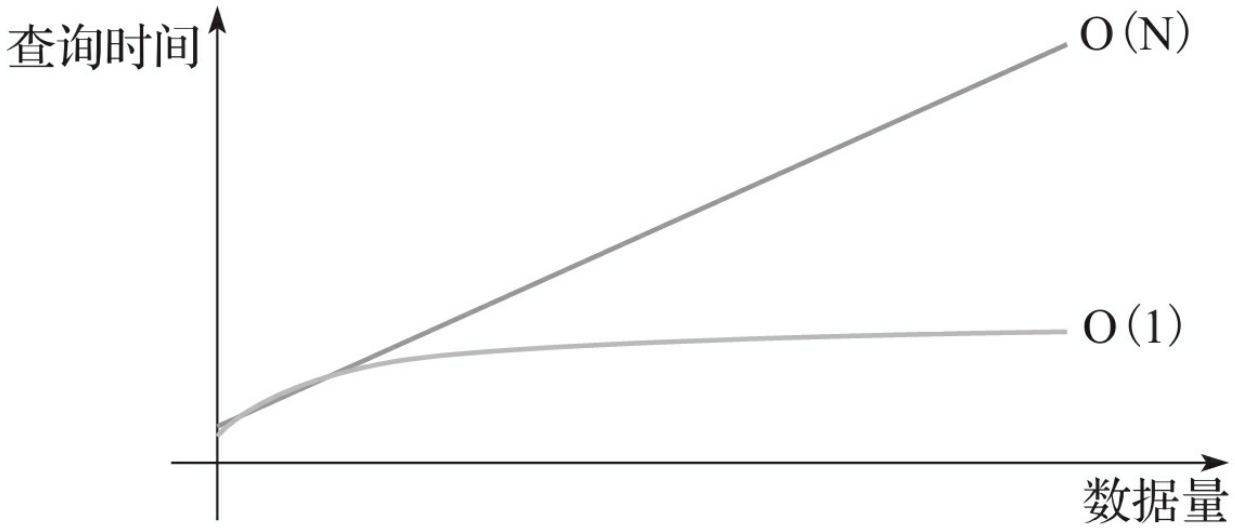


图1-6 查询时间复杂度O(1)比O(N)

第2章　快速入门

第1章介绍了Kylin的概况，以及与其他SQL on Hadoop技术的比较，相信读者对Kylin已经有了一个整体的认识。本章将详细介绍Kylin的一些核心概念，然后带领读者逐步定义数据模型，创建Cube，并通过SQL来查询Cube，以帮助读者对Kylin有更为直观的了解。

2.1 核心概念

在开始使用Kylin之前，我们有必要先了解一下Kylin里的各种概念和术语，为后续章节的学习奠定基础。

2.1.1 数据仓库、OLAP与BI

数据仓库(Data Warehouse)是一种信息系统的资料储存理论，此理论强调的是利用某些特殊的资料储存方式，让所包含的资料特别有利于分析和处理，从而产生有价值的资讯，并可依此做出决策。

利用数据仓库的方式存放的资料，具有一旦存入，便不会随时间发生变动的特性，此外，存入的资料必定包含时间属性，通常一个数据仓库中会含有大量的历史性资料，并且它可利用特定的分析方式，从其中发掘出特定的资讯。

OLAP(Online Analytical Process)，联机分析处理，以多维度的方式分析数据，而且能够弹性地提供上卷(Roll-up)、下钻(Drill-down)和透视分析(Pivot)等操作，它是呈现集成性决策信息的方法，多用于决策支持系统、商务智能或数据仓库。其主要的功能在于方便大规模数据分析及统计计算，可对决策提供参考和支持。与之相区别的是联机交易处理(OLTP)，联机交易处理，更侧重于基本的、日常的事务处理，包括数据的增删改查。

·OLAP需要以大量历史数据为基础，再配合上时间点的差异，对多维度及汇整型的信息进行复杂的分析。

·OLAP需要用户有主观的信息需求定义，因此系统效率较佳。

OLAP的概念，在实际应用中存在广义和狭义两种不同的理解方式。广义上的理解与字面上的意思相同，泛指一切不会对数据进行更新的分析处理。但更多的情况下OLAP被理解为其狭义上的含义，即与多维分析相关，基于立方体(Cube)计算而进行的分析。

BI(Business Intelligence)，即商务智能，指用现代数据仓库技术、在线分析技术、数据挖掘和数据展现技术进行数据分析以实现商业价值。

今天，许多企业已经建立了自己的数据仓库，用于存放和管理不断增长的数据；这些数据中蕴含着丰富的价值，但只有通过使用一系列的分析工具，进行大量的筛选、计算和展示，数据中蕴含的规律和潜在的信息才能被人们所发现；分析人员可结合这些信息展开商业决策和市场活动，从而为用户提供更好的服务，或为企业产生更大的价值。

2.1.2 维度和度量

维度和度量是数据分析中的两个基本概念。

维度是指审视数据的角度，它通常是数据记录的一个属性，例如时间、地点等。度量是基于数据所计算出来的考量值；它通常是一个数值，如总销售额、不同的用户数等。分析人员往往要结合若干个维度来审查度量值，以便在其中找到变化规律。在一个SQL查询中，Group By的属性通常就是维度，而所计算的值则是度量。如下面的示例：

```
    select part_dt, lstg_site_id, sum(price) as total_selled, count(distinct
seller_id) as sellers from kylin_sales group by part_dt, lstg_site_id
```

在上面的这个查询中，part_dt和lstg_site_id是维度，sum(price)和count(distinct seller_id)是度量。

2.1.3 事实表和维度表

事实表(Fact Table)是指存储有事实记录的表，如系统日志、销售记录等；事实表的记录在不断地动态增长，所以它的体积通常远大于其他表。

维度表(Dimension Table)或维表，有时也称查找表(Lookup Table)，是与事实表相对应的一种表；它保存了维度的属性值，可以跟事实表做关联；相当于将事实表上经常重复出现的属性抽取、规范出来用一张表进行管理。常见的维度表有：日期表(存储与日期对应的周、月、季度等的属性)、地点表(包含国家、省／州、城市等属性)等。使用维度表有诸多好处，具体如下。

·缩小了事实表的大小。

·便于维度的管理和维护，增加、删除和修改维度的属性，不必对事实表的大量记录进行改动。

·维度表可以为多个事实表重用，以减少重复工作。

2.1.4 Cube、Cuboid和Cube Segment

Cube（或Data Cube），即数据立方体，是一种常用于数据分析与索引的技术；它可以对原始数据建立多维度索引。通过Cube对数据进行分析，可以大大加快数据的查询效率。

Cuboid在Kylin中特指在某一种维度组合下所计算的数据。

Cube Segment是指针对源数据中的某一个片段，计算出来的Cube数据。通常数据仓库中的数据数量会随着时间的增长而增长，而Cube Segment也是按时间顺序来构建的。

2.2 在Hive中准备数据

2.1节介绍了Kylin中的常见概念。本节将介绍准备Hive数据的一些注意事项。需要被分析的数据必须先保存为Hive表的形式，然后Kylin才能从Hive中导入数据，创建Cube。

Apache Hive是一个基于Hadoop的数据仓库工具，最初由Facebook开发并贡献到Apache软件基金会。Hive可以将结构化的数据文件映射为数据库表，并可以将SQL语句转换为MapReduce或Tez任务进行运行，从而让用户以类SQL（HiveQL，也称HQL）的方式管理和查询Hadoop上的海量数据。

此外，Hive还提供了多种方式（如命令行、API和Web服务等）可供第三方方便地获取和使用元数据并进行查询。今天，Hive已经成为Hadoop数据仓库的首选，是Hadoop上不可或缺的一个重要组件，很多项目都已兼容或集成了Hive。基于此情况，Kylin选择Hive作为原始数据的主要来源。

在Hive中准备待分析的数据是使用Kylin的前提；将数据导入到Hive表中的方法有很多，用户管理数据的技术和工具也各式各样，因此具体步骤不在本书的讨论范围之内。如有需要可以参考Hive的使用文档。这里将着重阐述需要注意的几个事项。

2.2.1 星形模型

数据挖掘有几种常见的多维数据模型，如星形模型(Star Schema)、雪花模型(Snowf? lake Schema)、事实星座模型(Fact Constellation)等。

星形模型中有一张事实表，以及零个或多个维度表；事实表与维度表通过主键外键相关联，维度表之间没有关联，就像很多星星围绕在一个恒星周围，故取名为星形模型。

如果将星形模型中某些维度的表再做规范，抽取成更细的维度表，然后让维度表之间也进行关联，那么这种模型称为雪花模型。

星座模型是更复杂的模型，其中包含了多个事实表，而维度表是公用的，可以共享。

不过，Kylin只支持星形模型的数据集，这是基于以下考虑。

·星形模型是最简单，也是最常用的模型。

·由于星形模型只有一张大表，因此它相比于其他模型更适合于大数据处理。

·其他模型可以通过一定的转换，变为星形模型。

2.2.2 维度表的设计

除了数据模型以外，Kylin还对维度表有一定的要求，具体要求如下。

1）要具有数据一致性，主键值必须是唯一的；Kylin会进行检查，如果有两行的主键值相同则会报错。

2）维度表越小越好，因为Kylin会将维度表加载到内存中供查询；过大的表不适合作为维度表，默认的阈值是300MB。

3）改变频率低，Kylin会在每次构建中试图重用维度表的快照，如果维度表经常改变的话，重用就会失效，这就会导致要经常对维度表创建快照。

4）维度表最好不要是Hive视图（View），虽然在Kylin1.5.3中加入了对维度表是视图这种情况的支持，但每次都需要将视图进行物化，从而导致额外的时间开销。

2.2.3 Hive表分区

Hive表支持多分区(Partition)。简单地说，一个分区就是一个文件目录，存储了特定的数据文件。当有新的数据生成的时候，可以将数据加载到指定的分区，读取数据的时候也可以指定分区。对于SQL查询，如果查询中指定了分区列的属性条件，则Hive会智能地选择特定分区(也就是目录)，从而避免全量数据的扫描，减少读写操作对集群的压力。

下面列举的一组SQL演示了如何使用分区：

```
    Hive> create table invites (id int, name string) partitioned by (ds string) row
format delimited fields terminated by 't' stored as textfile;

    Hive> load data local inpath '/user/hadoop/data.txt' overwrite into table
invites partition (ds='2016-08-16');

    Hive> select * from invites where ds ='2016-08-16';
```

Kylin支持增量的Cube构建，通常是按时间属性来增量地从Hive表中抽取数据。如果Hive表正好是按此时间属性做分区的话，那么就可以利用到Hive分区的好处，每次在Hive构建的时候都可以直接跳过不相干日期的数据，节省Cube构建的时间。这样的列在Kylin里也称为分割时间列(Partition Time Column)，通常它应该也是Hive表的分区列。

2.2.4 了解维度的基数

维度的基数(Cardinality)指的是该维度在数据集中出现的不同值的个数;例如“国家”是一个维度，如果有200个不同的值，那么此维度的基数就是200。通常一个维度的基数会从几十到几万个不等，个别维度如“用户ID”的基数会超过百万甚至千万。基数超过一百万的维度通常被称为超高基数维度(Ultra High Cardinality，UHC)，需要引起设计者的注意。

Cube中所有维度的基数都可以体现出Cube的复杂度，如果一个Cube中有好几个超高基数维度，那么这个Cube膨胀的概率就会很高。在创建Cube前需要对所有维度的基数做一个了解，这样就可以帮助设计合理的Cube。计算基数有多种途径，最简单的方法就是让Hive执行一个count distinct的SQL查询;Kylin也提供了计算基数的方法，在2.3.1节中会进行介绍。

2.2.5 Sample Data

如果需要一些简单数据来快速体验Apache Kylin，也可以使用Apache Kylin自带的Sample Data。运行“${KYLIN_HOME}/bin/sample.sh”来导入Sample Data，然后就能按照下面的流程继续创建模型和Cube。

具体请执行下面命令，将Sample Data导入Hive数据库。

```
cd ${KYLIN_HOME}
bin/sample.sh
```

Sample Data测试的样例数据集总共仅1MB左右，共计3张表，其中事实表有10000条数据。因为数据规模较小，有利于在虚拟机中进行快速实践和操作。数据集是一个规范的星形模型结构，它总共包含了3个数据表：

·KYLIN_SALES是事实表，保存了销售订单的明细信息。各列分别保存着卖家、商品分类、订单金额、商品数量等信息，每一行对应着一笔交易订单。

·KYLIN_CATEGORY_GROUPINGS是维表，保存了商品分类的详细介绍，例如商品分类名称等。

·KYLIN_CAL_DT也是维表，保存了时间的扩展信息。如单个日期所在的年始、月始、周始、年份、月份等。

这3张表一起构成了整个星形模型。

2.3 设计Cube

如果数据已经在Hive中准备好了，并且已经满足了2.2节中介绍的条件，那么就可以开始设计和创建Cube了。本节将按通常的步骤介绍Cube是如何进行创建的。

2.3.1 导入Hive表定义

登录Kylin的Web界面，创建新的或选择一个已有的项目之后，需要做的就是将Hive表的定义导入到Kylin中。

单击Web界面的Model→Data source下的“Load Hive Table”图标，然后输入表的名称（可以一次导入多张表，以逗号分隔表名，如图2-1所示），单击按钮“Sync”，Kylin就会使用Hive的API从Hive中获取表的属性信息。

导入成功后，表的结构信息会以树状的形式显示在页面的左侧，可以单击展开或收缩，如图2-2所示。

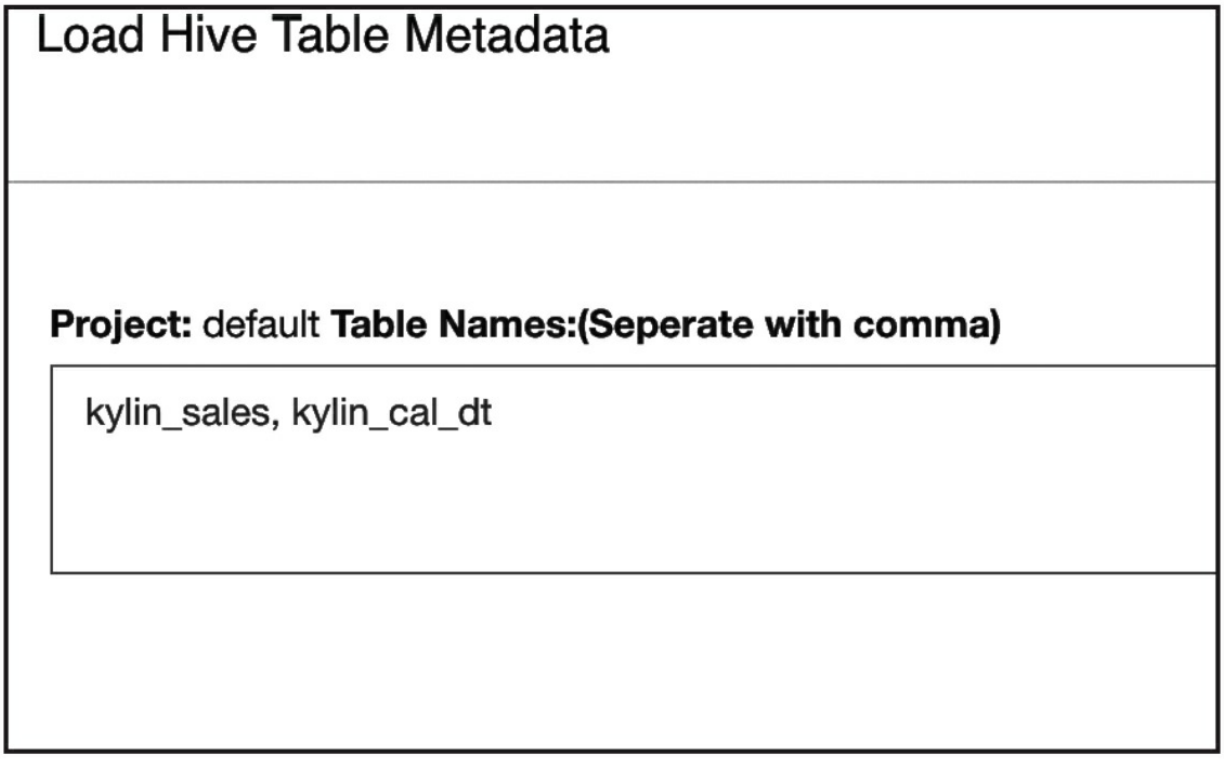


图2-1　输入Hive表名

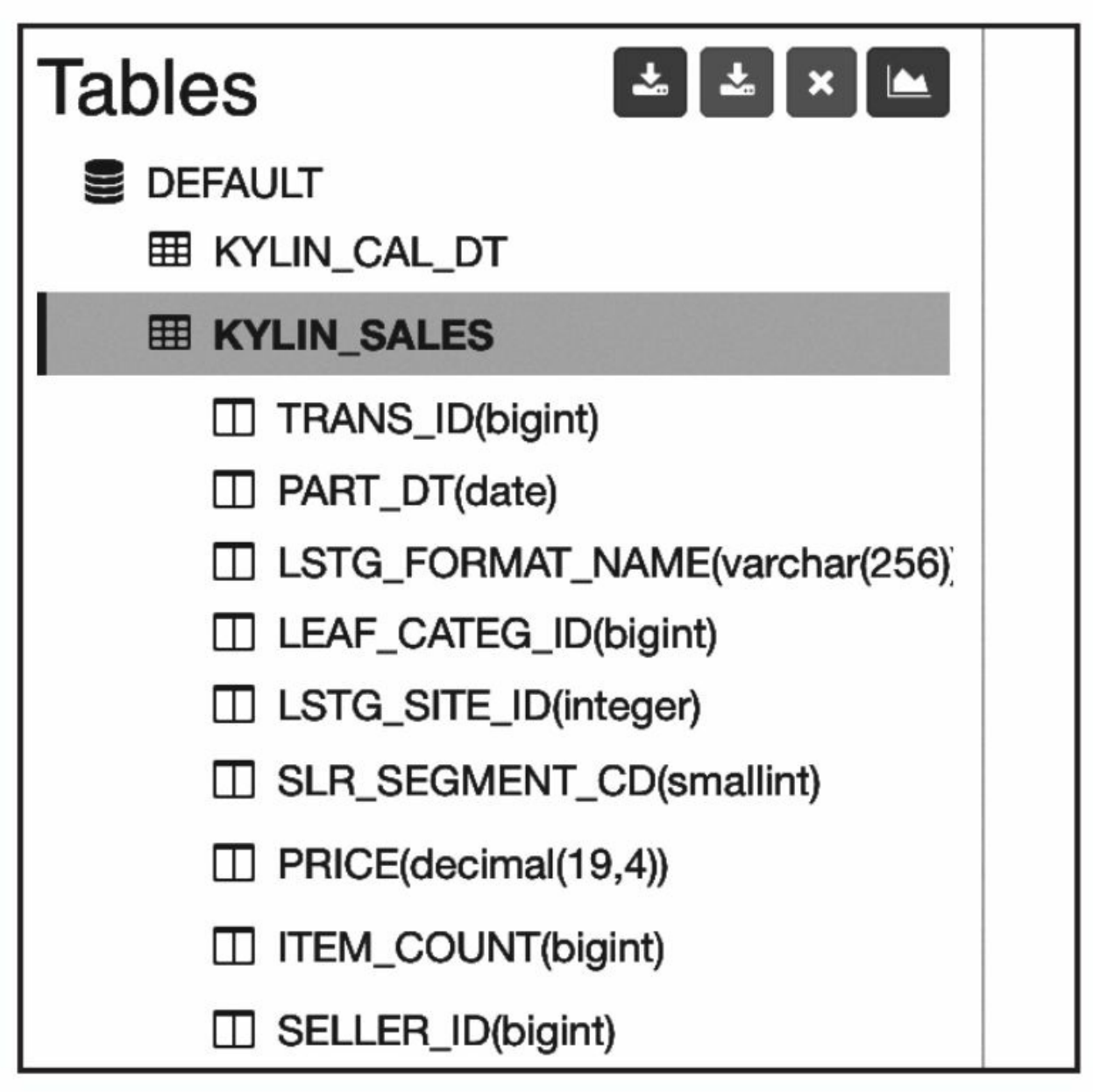


图2-2 完成导入的Hive表

同时，Kylin会在后台触发一个MapReduce任务，计算此表每个列的基数。通常稍过几分钟之后再刷新页面，就会看到显示出来的基数信息，如图2-3所示。

ID ▲	Name ⬍	Data Type ⬍	Cardinality ⬍
1	TRANS_ID	bigint	1
2	PART_DT	date	736
3	LSTG_FORMAT_NAME	varchar(256)	5
4	LEAF_CATEG_ID	bigint	136
5	LSTG_SITE_ID	integer	7
6	SLR_SEGMENT_CD	smallint	8
7	PRICE	decimal(19,4)	10000
8	ITEM_COUNT	bigint	1
9	SELLER_ID	bigint	955

图2-3 计算后各列的基数

需要注意的是，这里Kylin对基数的计算方法采用的是HyperLogLog的近似算法，与精确值略有误差，但作为参考值已经足够了。

2.3.2 创建数据模型

有了表信息之后，就可以开始创建数据模型（Data Model）了。数据模型是Cube的基础，它主要用于描述一个星形模型。有了数据模型以后，定义Cube的时候就可以直接从此模型定义的表和列中进行选择了，省去重复指定连接（join）条件的步骤。基于一个数据模型还可以创建多个Cube，以方便减少用户的重复性工作。

在Kylin界面的“Models”页面中，单击“New”→“New Model”，开始创建数据模型。给模型输入名称之后，选择一个事实表（必需的），然后添加维度表（可选），如图2-4所示。

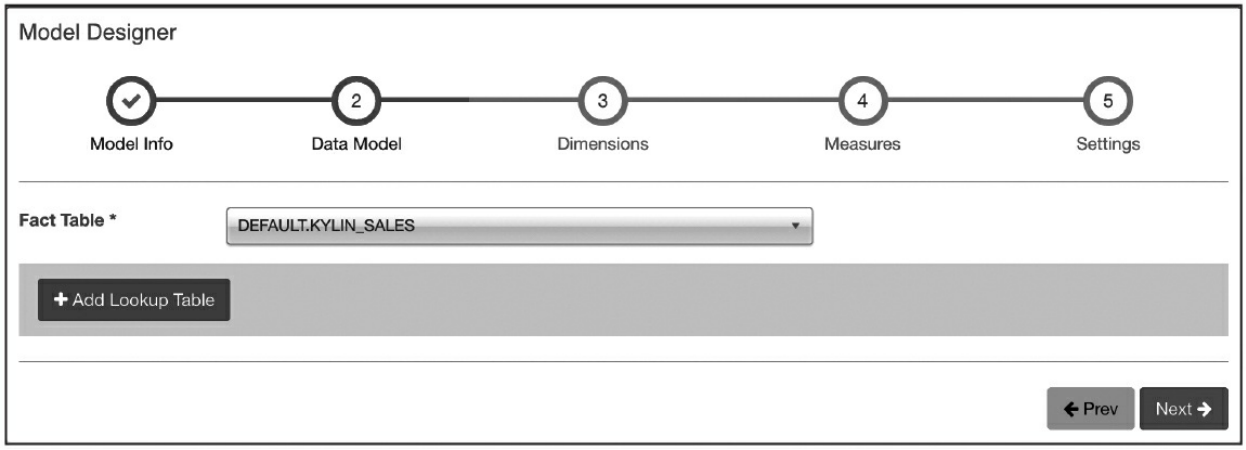


图2-4 选择事实表

添加维度表的时候，需要选择连接的类型：是Inner还是Left，然后选择连接的主键和外键，这里也支持多主键，如图2-5所示。

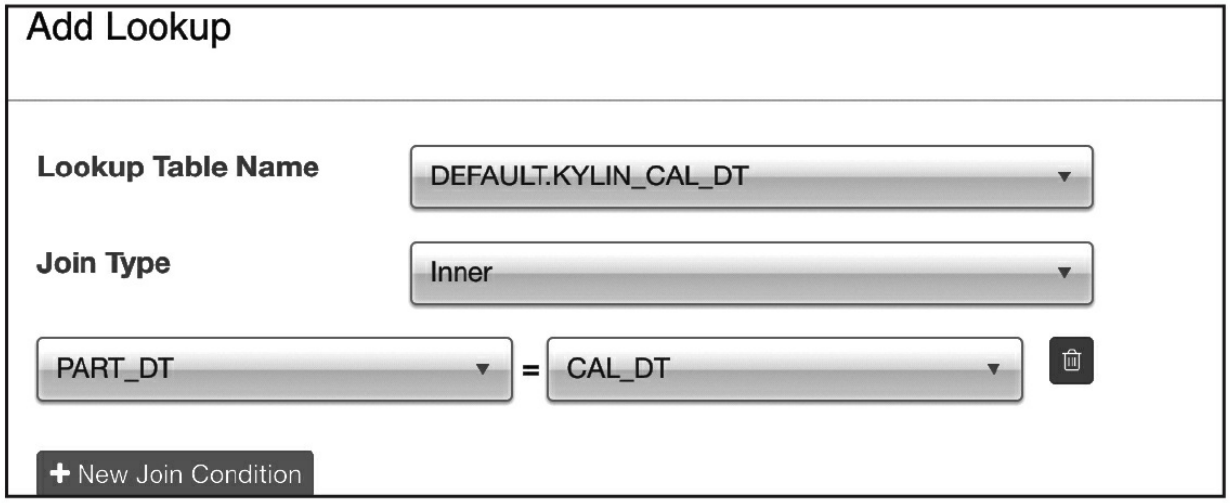


图2-5 选择维度表

接下来选择会用作维度和度量的列。这里只是选择一个范围，不代表这些列将来一定要用作Cube的维度或度量，你可以把所有可能会用到的列都选进来，后续创建Cube的时候，将只能从这些列中进行选择。

选择维度列时，维度可以来自事实表或维度表，如图2-6所示。

选择度量列时，度量只能来自事实表，如图2-7所示。

最后一步，是为模型补充分割时间列信息和过滤条件。如果此模型中的事实表记录是按时间增长的，那么可以指定一个日期／时间列作为模型的分割时间列，从而可以让Cube按此列做增量构建，关于增量构建的具体内容请参见第3章。

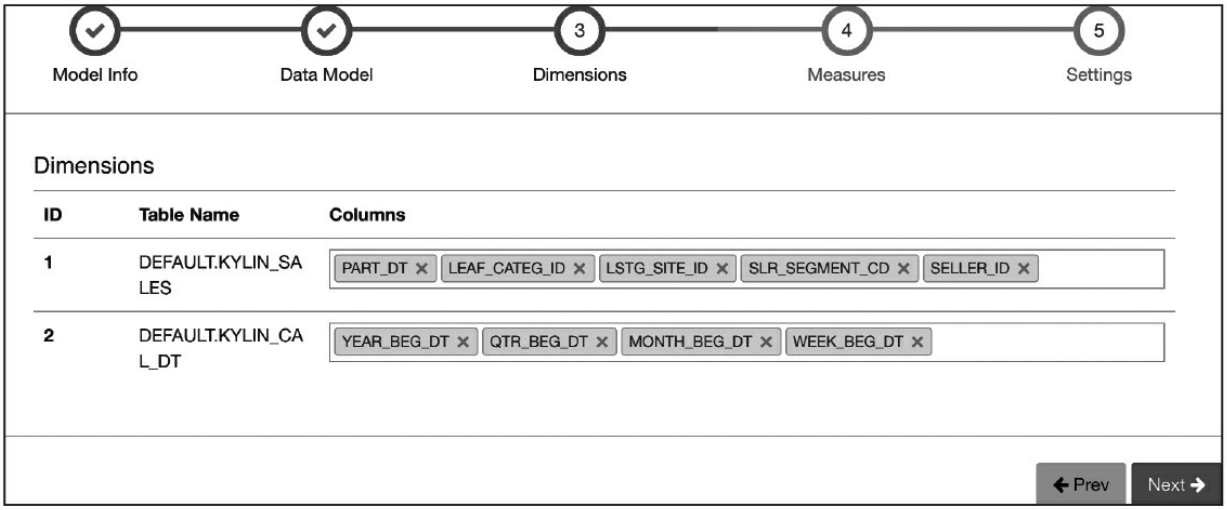


图2-6 选择维度列

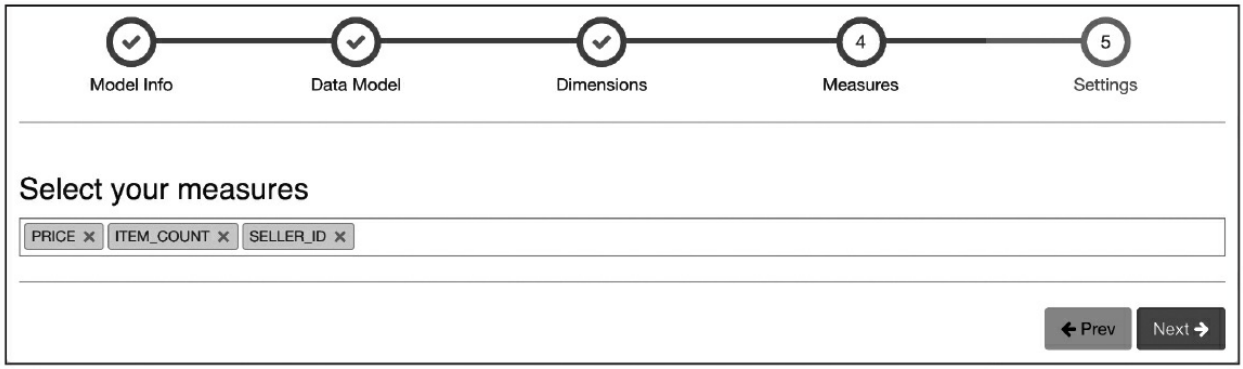


图2-7 选择度量列

过滤(Filter)条件是指，如果想把一些记录忽略掉，那么这里可以设置一个过滤条件。Kylin在向Hive请求源数据的时候，会带上此过滤条件。在图2-8所示的示例中，会直接排除掉金额小于等于0的记录。

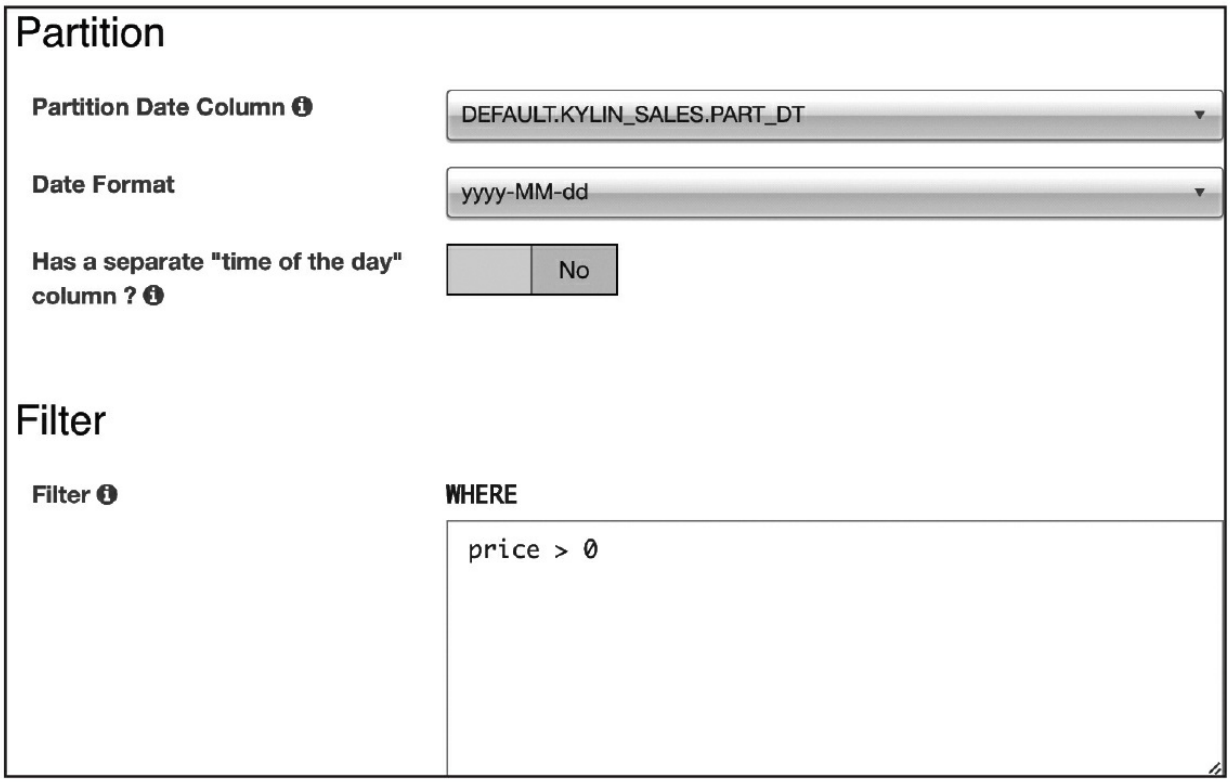


图2-8 选择分区列和设定过滤器

最后，单击“Save”保存此数据模型，随后它将出现在“Models”的列表中。

2.3.3 创建Cube

本节将快速介绍创建Cube时的各种配置选项，但是由于篇幅的限制，这里将不会对Cube的配置和Cube的优化进行深入的展开介绍。读者可以在后续的章节（如第6章“Cube优化”）中找到关于Cube的更详细的介绍。接下来开始Cube的创建；单击“New”，选择“New Cube”，会开启一个包含若干步骤的向导。

第一页，选择要使用的数据模型，并为此Cube输入一个唯一的名称（必需的）和描述（可选的）（如图2-9所示）；这里还可以输入一个邮件通知列表，用于在构建完成或出错时收到通知。如果不想接收处于某些状态的通知，那么可以从“Notif？ication Events”中将其去掉。

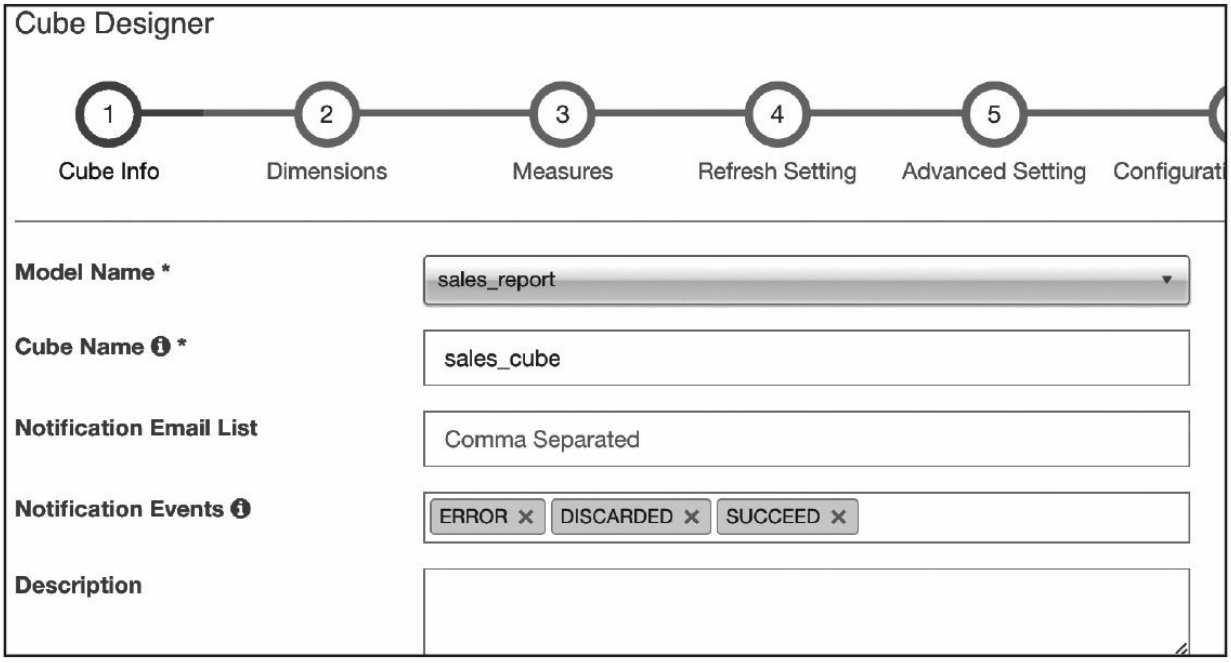


图2-9 Cube基本信息

第二页，选择Cube的维度。可以通过以下两个按钮来添加维度。

·“Add Dimension”：逐个添加维度，可以是普通维度也可以是衍生（Derived）维度。

·“Auto Generator”：批量选择并添加，让Kylin自动完成其他信息。

使用第一种方法的时候，需要为每个维度起个名字，然后选择表和列（如图2-10所示）。

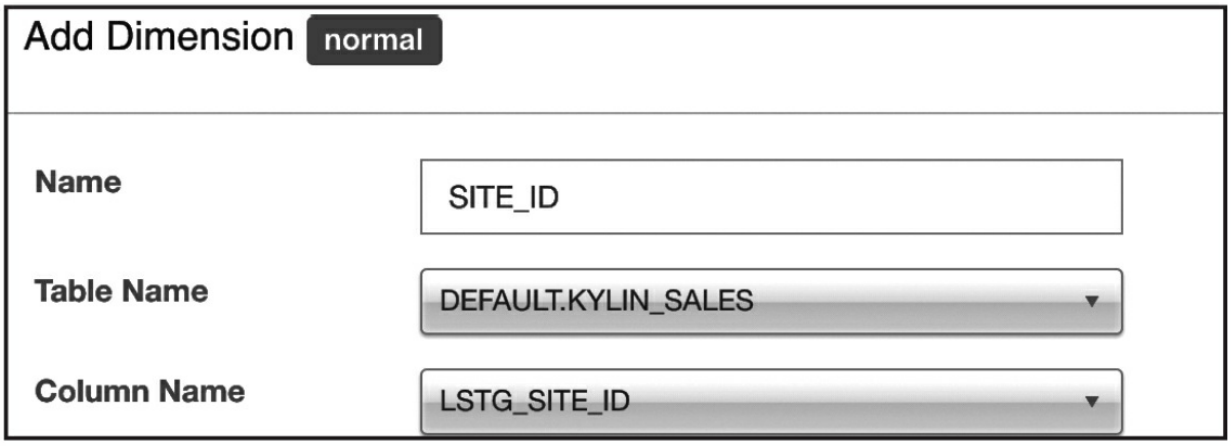


图2-10 添加普通维度

如果是衍生维度的话，则必须是来自于某个维度表，一次可以选择多个列（如图2-11所示）；由于这些列值都可以从该维度表的主键值中衍生出来，所以实际上只有主键列会被Cube加入计算。而在Kylin的具体实现中，往往采用事实表上的外键替代主键进行计算和存储。但是在逻辑

上可以认为衍生列来自于维度表的主键。

使用第二种方法的时候，Kylin会用一个树状结构呈现出所有的列，用户只需要勾选所需要的列即可，Kylin会自动补齐其他信息，从而方便用户的操作（如图2-12所示）。请注意，在这里Kylin会把维度表上的列都创建成衍生维度，这也许不是最合适的，在这种情况下，请使用第一种方法。

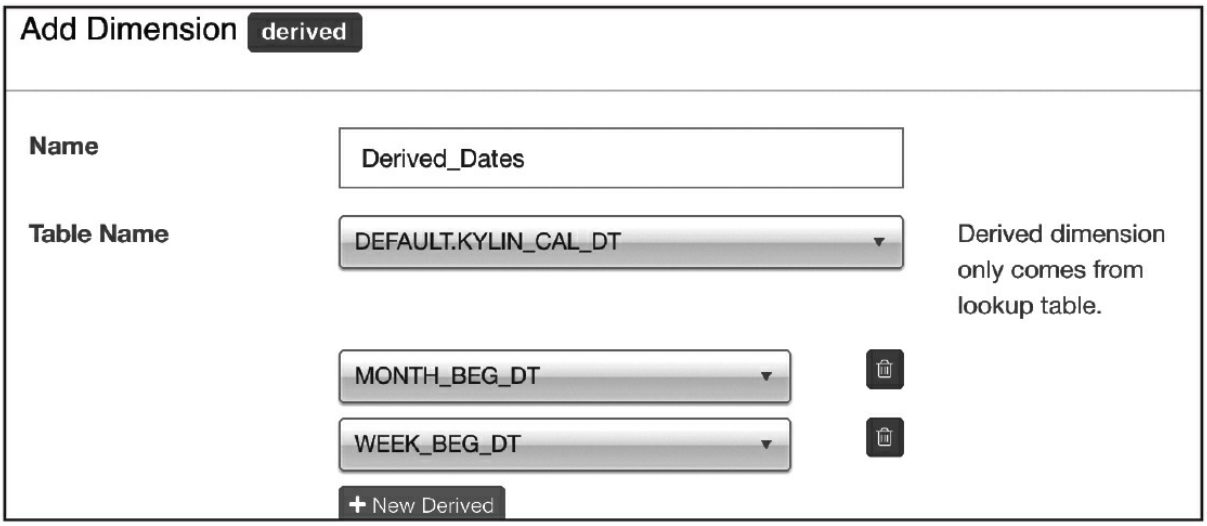


图2-11 添加衍生维度

第三页，创建度量。Kylin默认会创建一个Count（1）的度量。可以单击“+Measure”按钮来添加新的度量。Kylin支持的度量有：SUM、MIN、MAX、COUNT、COUNT DISTINCT、TOP_N、RAW等。请选择需要的度量类型，然后再选择适当的参数（通常为列名）。图2-13是一个SUM（price）的示例。

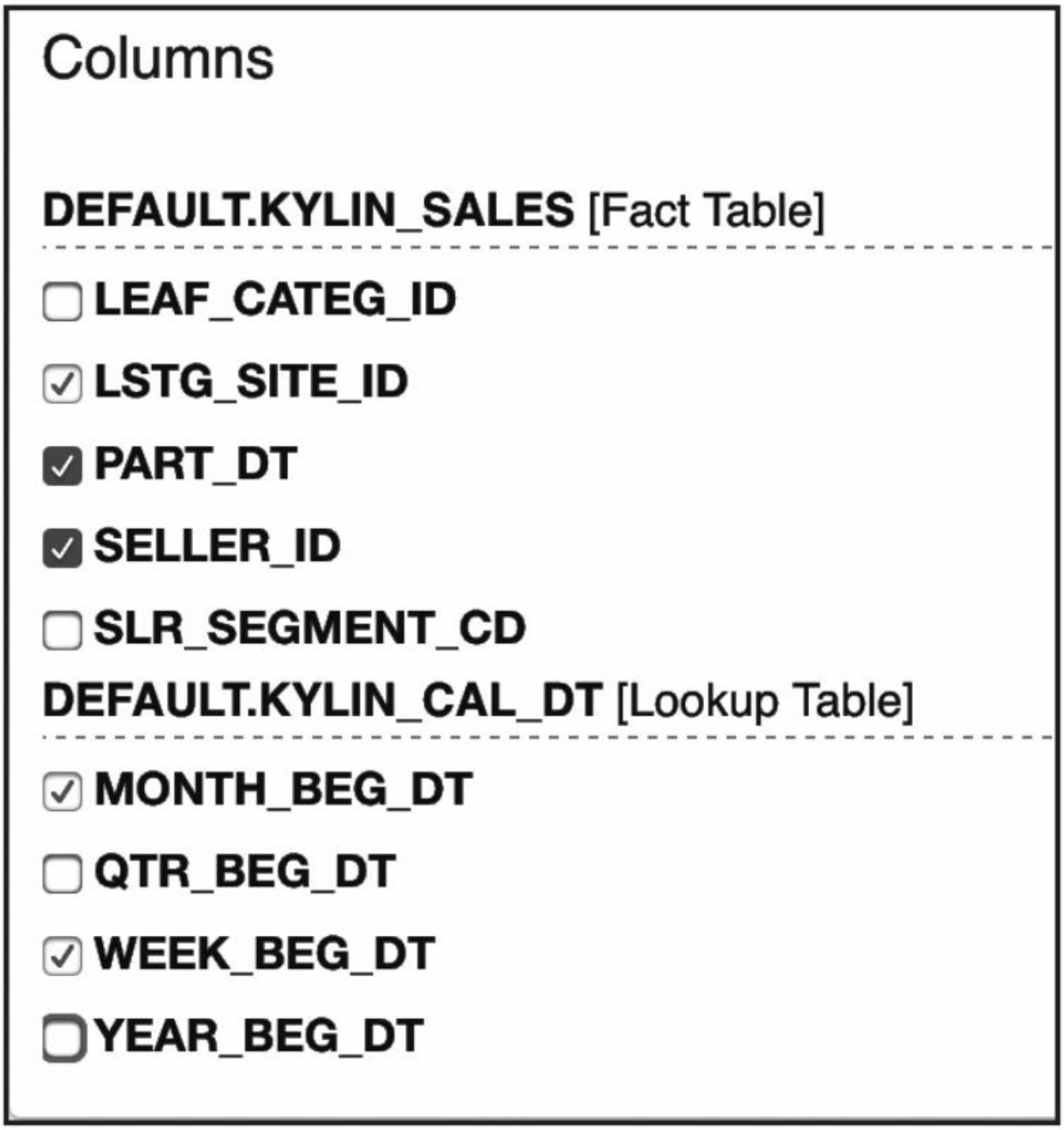

Columns
DEFAULT.KYLIN_SALES [Fact Table]
LEAF_CATEG_ID
LSTG_SITE_ID
PART_DT
SELLER_ID
SLR_SEGMENT_CD
DEFAULT.KYLIN_CAL_DT [Lookup Table]
MONTH_BEG_DT
QTR_BEG_DT
WEEK_BEG_DT
YEAR_BEG_DT

图2-12 批量添加维度

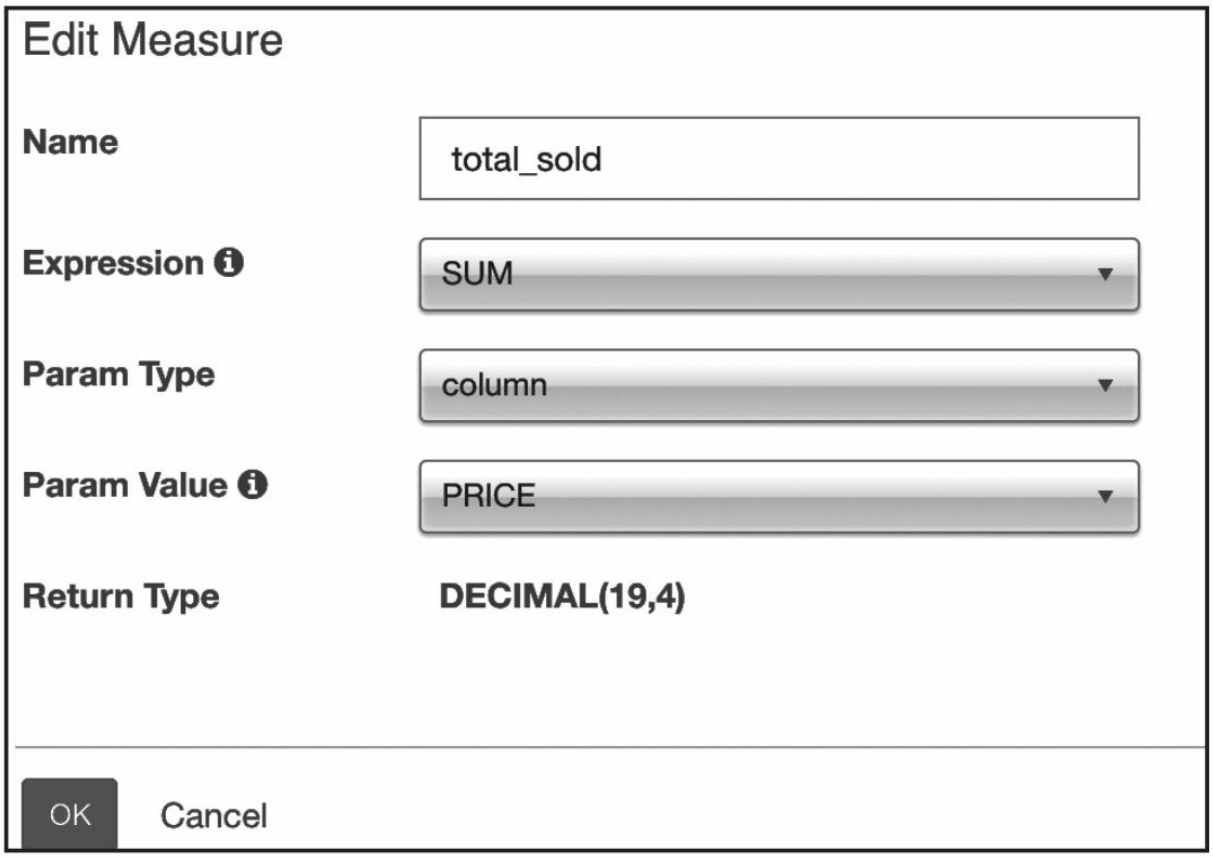


图2-13 添加度量

重复上面的步骤，创建所需要的度量。Kylin可以支持在一个Cube中添加多达上百个的度量；添加完所有度量之后，单击“Next”，如图2-14所示。

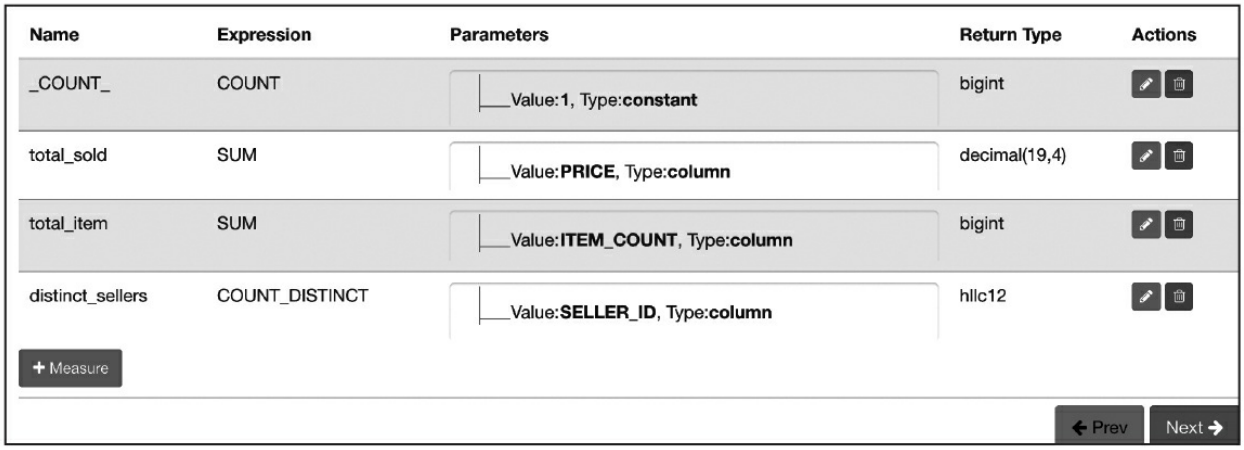


图2-14 度量列表

第四页，是关于Cube数据刷新的设置。在这里可以设置自动合并的阈值、数据保留的最短时间，以及第一个Segment的起点时间（如果Cube有分割时间列的话），详细内容请参考第3章。

第五页，高级设置。在此页面上可以设置聚合组和Rowkey（如图2-16所示）。

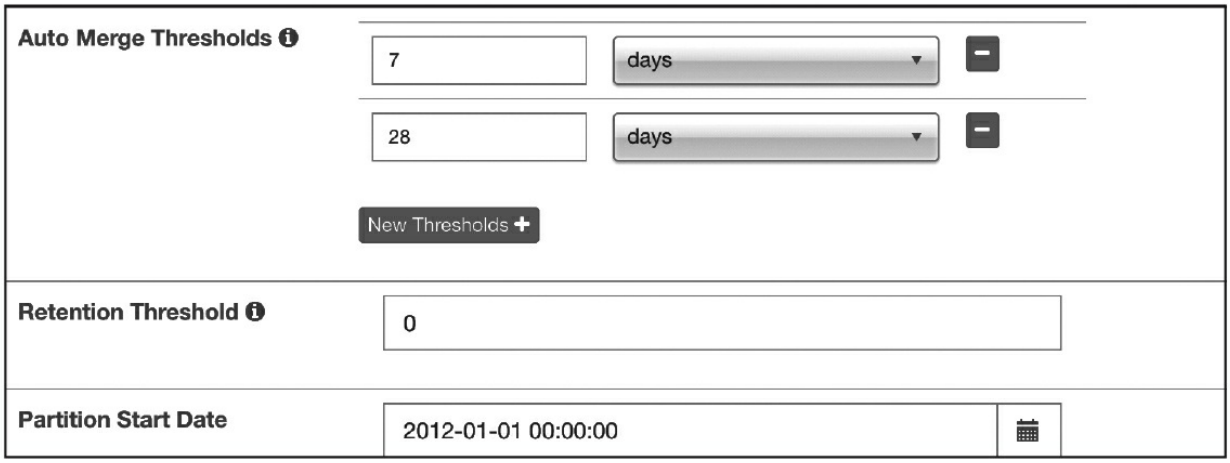


图2-15 刷新设置

Kylin默认会把所有维度都放在同一个聚合组中；如果维度数较多（例如>10），那么建议用户根据查询的习惯和模式，单击“New Aggregation Group+”，将维度分为多个聚合组。通过使用多个聚合组，可以大大降低Cube中的Cuboid数量。下面来举例说明，如果一个Cube有（M+N）个维度，那么默认它会有2^{m+n}个Cuboid；如果把这些维度分为两个不相交的聚合组，那么Cuboid的数量将被减少为2^m+2^n。

在单个聚合组中，可以对维度设置高级属性，如Mandatory、Hierarchy、Joint等。这几种属性都是为优化Cube的计算而设计的，了解这些属性的含义对日后更好地使用Cube至关重要。

Mandatory维度指的是那些总是会出现在Where条件或Group By语句里的维度；通过将某个维度指定为Mandatory，Kylin就可以不用预计算那些不包含此维度的Cuboid，从而减少计算量。

Hierarchy是一组有层级关系的维度，例如“国家”“省”“市”，这里的“国家”是高级别的维度，“省”“市”依次是低级别的维度。用户会按高级别维度进行查询，也会按低级别维度进行查询，但在查询低级别维度时，往往都会带上高级别维度的条件，而不会孤立地审视低级别维度的数据。例如，用户会单击“国家”作为维度来查询汇总数据，也可能单击“国家”＋“省”，或者“国家”＋“省”＋“市”来查询，但是不会跨越国家直接Group By“省”或“市”。通过指定Hierarchy，Kylin可以省略不满足此模式的Cuboid。

Joint是将多个维度组合成一个维度，其通常适用于如下两种情形。

·总是会在一起查询的维度。

·基数很低的维度。

Kylin以Key-Value的方式将Cube存储到HBase中。HBase的key，也就是Rowkey，是由各维度的值拼接而成的；为了更高效地存储这些值，Kylin会对它们进行编码和压缩；每个维度均可以选择合适的编码（Encoding）方式，默认采用的是字典（Dictionary）编码技术；除了字典以外，还有整数（Int）和固定长度（Fixed Length）的编码。

字典编码是将此维度下的所有值构建成一个从string到int的映射表；Kylin会将字典序列化保存，在Cube中存储int值，从而大大减小存储的大小。另外，字典是保持顺序的，即如果字符串A比字符串B大的话，那么A编码后的int值也会比B编码后的值大；这样可以使得在HBase中进行比较查询的时候，依然使用编码后的值，而无需解码。

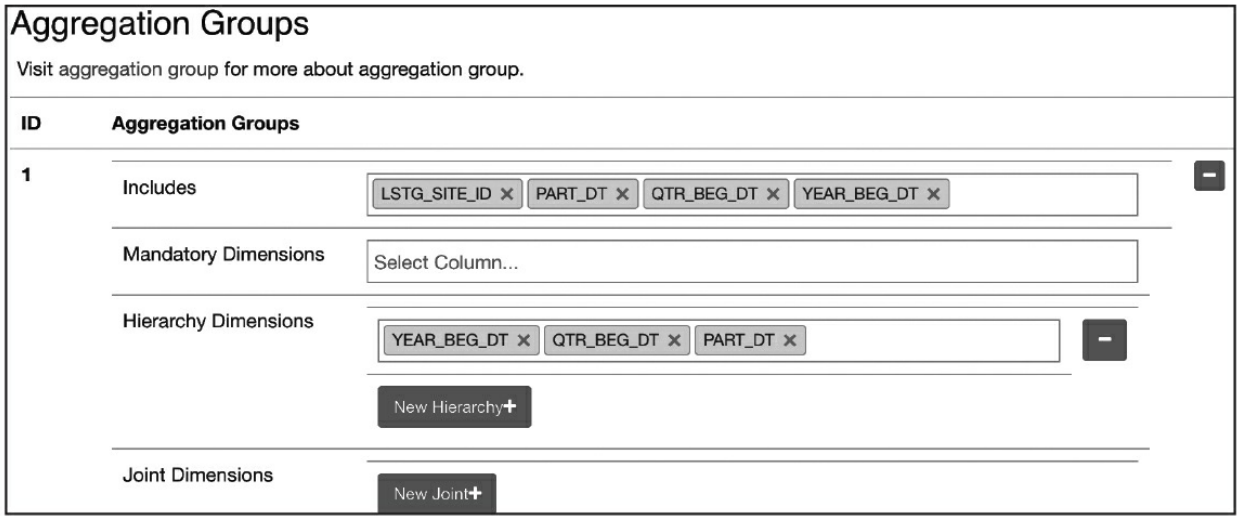


图2-16 高级设置

字典非常适合于非固定长度的string类型值的维度，而且用户无需指定编码后的长度；但是由于使用字典需要维护一张映射表，因此如果此维度的基数很高，那么字典的大小就非常可观，从而不适合于加载到内存中，在这种情况下就要选择其他的编码方式了。Kylin中字典编码允许的基数上限默认是500万（由参数"kylin.dictionary.max.cardinality"配置）。

整数（int）编码适合于对int或bigint类型的值进行编码，它无需额外存储，同时还可以支持很大的基数。用户需要根据值域选择编码的长度。例如有一个"手机号码"的维度，它是一个11位的数字，如13800138000，我们知道它大于2^{31}，但是小于2^{39} -1，那么使用int（5）即可满足要求，每个值占用5字节，比按字符存储（11字节）要少占用一半以上的空间。

当上面几种编码方式都不适合的时候，就需要使用固定长度的编码了；此编码方式其实只是将原始值截断或补齐成相同长度的一组字节，没

有额外的转换，所以空间效率较差，通常只是作为一种权宜手段。

各维度在Rowkeys中的顺序，对于查询的性能会产生较明显的影响。在这里用户可以根据查询的模式和习惯，通过拖曳的方式调整各个维度在Rowkeys上的顺序（如图2-17所示）。通常的原则是，将过滤频率较高的列放置在过滤频率较低的列之前，将基数高的列放置在基数低的列之前。这样做的好处是，充分利用过滤条件来缩小在HBase中扫描的范围，从而提高查询的效率。

第五页，为Cube配置参数。和其他Hadoop工具一样，Kylin使用了很多配置参数以提高灵活性，用户可以根据具体的环境、场景等配置不同的参数进行调优。Kylin全局的参数值可在conf/kylin.properties文件中进行配置；如果Cube需要覆盖全局设置的话，则需要在此页面中指定。单击“+Property”按钮，然后输入参数名和参数值，如图2-18所示，指定“kylin.hbase.region.cut=1”，这样此Cube在存储的时候，Kylin将会为每个HTable Region分配1GB来创建一个HTable Region。

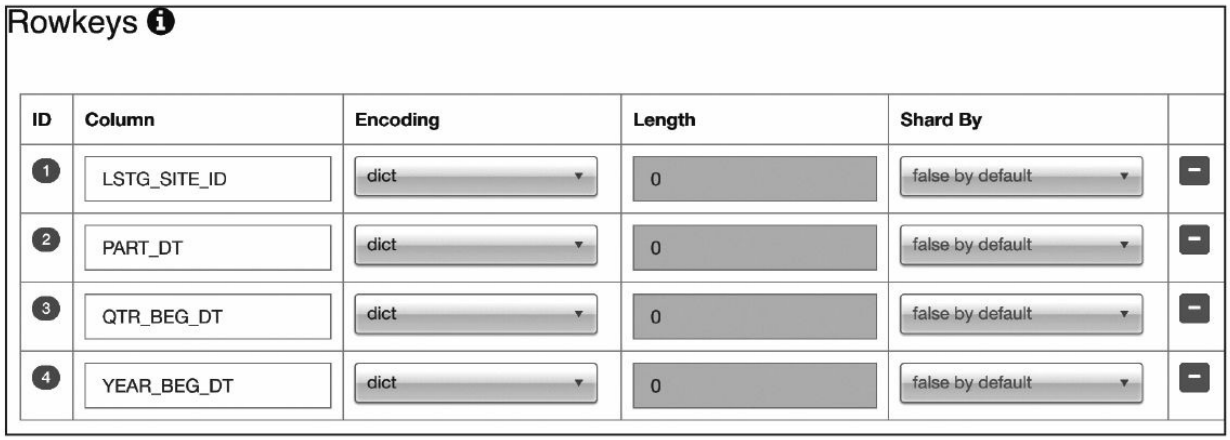


图2-17 Rowkey设置

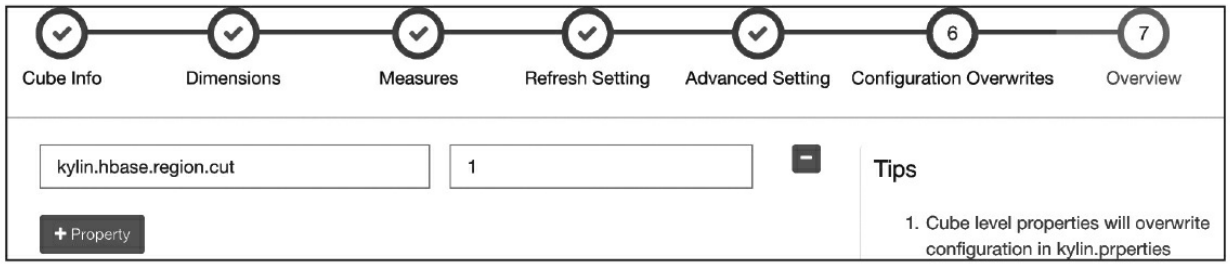


图2-18 覆盖默认参数

然后单击Next跳转到最后一个确认页面，如有修改，则单击“Prev”按钮返回以修改，最后再单击“Save”按钮进行保存，一个Cube就创建完成了。创建好的Cube会显示在“Cubes”列表中，如要对Cube的定义进行修改，只需单击“Edit”按钮就可以进行修改。也可以展开此Cube行以查看更多的信息，如JSON格式的元数据、访问权限、通知列表等。

2.4 构建Cube

·注意 本节将快速介绍构建Cube相关的操作说明和设置，因受到篇幅的限制，许多具体内容无法深入展开，读者可以从后续的第3章和第4章中获得更详细的介绍。

新创建的Cube只有定义，而没有计算的数据，它的状态是“DISABLED”，是不会被查询引擎挑中的。要想让Cube有数据，还需要对它进行构建。Cube的构建方式通常有两种：全量构建和增量构建；两者的构建步骤是完全一样的，区别只在于构建时读取的数据源是全集还是子集。

Cube的构建包含如下步骤，由任务引擎来调度执行。

1）创建临时的Hive平表（从Hive读取数据）。

2）计算各维度的不同值，并收集各Cuboid的统计数据。

3）创建并保存字典。

4）保存Cuboid统计信息。

5）创建HTable。

6）计算Cube（一轮或若干轮MapReduce）。

7）将Cube的计算结果转成HFile。

8）加载HFile到HBase。

9）更新Cube元数据。

10）垃圾回收。

以上步骤中，前5步是为计算Cube而做的准备工作，例如遍历维度值来创建字典，对数据做统计和估算以创建HTable等；第6）步是真正的Cube计算，取决于所使用的Cube算法，它可能是一轮MapReduce任务，也可能是N（在没有优化的情况下，N可以被视作是维度数）轮迭代的MapReduce。由于Cube运算的中间结果是以SequenceFile的格式存储在HDFS上的，所以为了导入到HBase中，还需要第7）步将这些结果转换成HFile（HBase文件存储格式）。第8）步通过使用HBase BulkLoad工具，将HFile导入进HBase集群，这一步完成之后，HTable就可以查询到数据了。第9）步更新Cube的数据，将此次构建的Segment的状态从“NEW”更新为“READY”，表示已经可供查询了。最后一步，清理构建过程中生成的临时文件等垃圾，释放集群资源。

Monitor页面会显示当前项目下近期的构建任务。图2-19显示了一个正在运行的Cube构建的任务，当前进度为46%多。

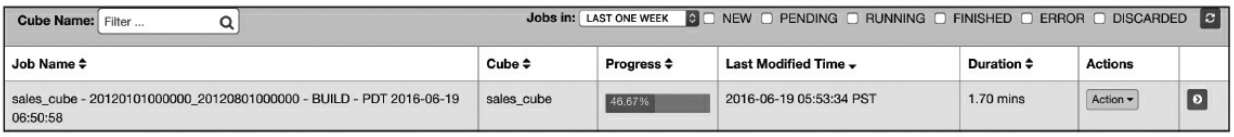


图2-19 任务列表

单击任务右边的“⬍”按钮，展开可以得到任务每一步的详细信息，如图2-20所示。

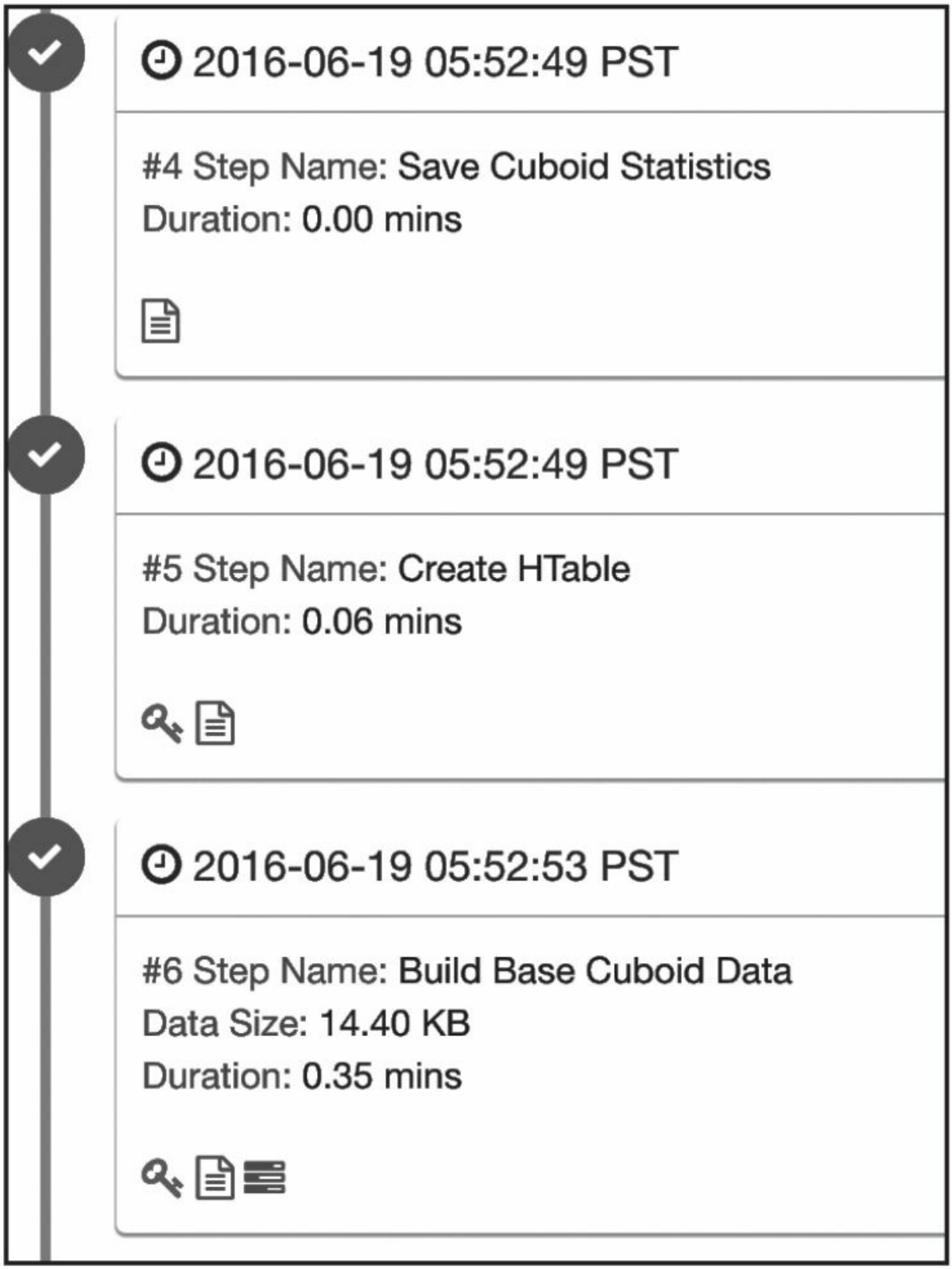

2016-06-19 05:52:49 PST
#4 Step Name: Save Cuboid Statistics
Duration: 0.00 mins
2016-06-19 05:52:49 PST
#5 Step Name: Create HTable
Duration: 0.06 mins
2016-06-19 05:52:53 PST
#6 Step Name: Build Base Cuboid Data
Data Size: 14.40 KB
Duration: 0.35 mins

图2-20 任务步骤列表

如果任务中的某一步是执行Hadoop任务的话，那么会显示Hadoop任务的链接，单击即可跳转到对应的Hadoop任务监测页面，如图2-21所示。

如果任务执行中的某一步出现报错，那么任务引擎会将任务状态置为“ERROR”并停止后续的执行，等待用户排错。在错误排除之后，用户可以单击“Resume”从上次失败的地方恢复执行。或者如果需要修改Cube或重新开始构建，那么用户需要单击“Discard”来丢弃此次构建。

接下来将介绍几种不同的构建方式。

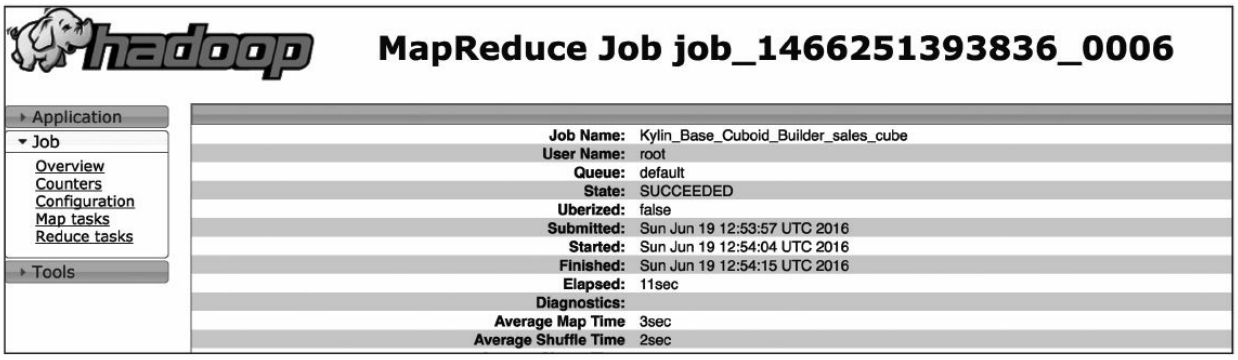


图2-21 MapReduce任务监测页面

2.4.1 全量构建和增量构建

1.全量构建

对数据模型中没有指定分割时间列信息的Cube，Kylin会采用全量构建，即每次从Hive中读取全部的数据来开始构建。通常它适用于以下两种情形。

·事实表的数据不是按时间增长的。

·事实表的数据比较小或更新频率很低，全量构建不会造成太大的开销。

2.增量构建

增量构建的时候，Kylin每次都会从Hive中读取一个时间范围内的数据，然后进行计算，并以一个Segment的形式进行保存。下次再构建的时候，会自动以上次结束的时间为起点时间，再选择新的终止时间进行构建。经过多次构建，Cube中将会有多个Segment依次按时间顺序进行排列，如Seg-1，Seg-2，…，Seg-N。查询的时候，Kylin会查询一个或多个Segment然后再做聚合计算，以便返回正确的结果给请求者。

使用增量构建的好处是，每次只需要对新增数据进行计算，从而避

免了对历史数据进行重复计算。对于数据量很大的Cube，使用增量构建是非常有必要的。

图2-22是构建一个Segment的Cube时的输入框，需要用户选择时间范围。

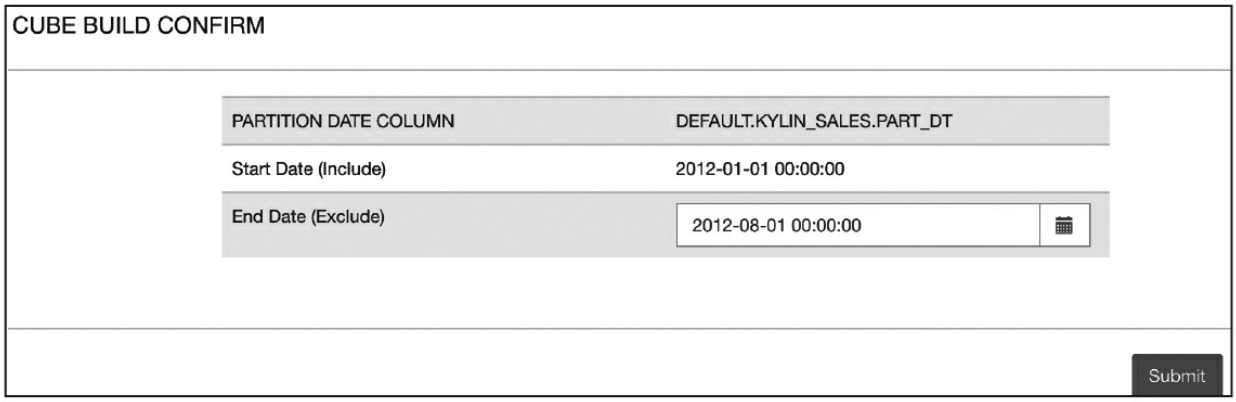


图2-22 提交增量构建

在从Hive读取源数据的时候，Kylin会带上此时间条件，如图2-23所示。

```
INSERT OVERWRITE TABLE kylin_intermediate_sales_cube_20120101000000_20120801000000 SELECT
KYLIN_SALES.LSTG_SITE_ID
,KYLIN_SALES.PART_DT
,KYLIN_CAL_DT.QTR_BEG_DT
,KYLIN_CAL_DT.YEAR_BEG_DT
,KYLIN_SALES.PRICE
,KYLIN_SALES.ITEM_COUNT
,KYLIN_SALES.SELLER_ID
FROM DEFAULT.KYLIN_SALES as KYLIN_SALES
INNER JOIN DEFAULT.KYLIN_CAL_DT as KYLIN_CAL_DT
ON KYLIN_SALES.PART_DT = KYLIN_CAL_DT.CAL_DT
WHERE (price > 0)  AND (KYLIN_SALES.PART_DT >= '2012-01-01' AND KYLIN_SALES.PART_DT < '2012-08-01')
;
```

图2-23 增量构建的SQL

·注意　增量构建抽取数据的范围，采用了前包后闭的原则，即包含了开始时间，但不包含结束时间，从而保证上一个Segment的结束时间与下一个Segment的起始时间相同，但数据不会重复。

下一次构建的时候，起始时间必须是上一次的结束时间。如果使用Kylin的Web GUI触发，那么起始时间会被自动填写，用户只需要选择结束时间。如果使用Rest API触发，用户则需要确保时间范围不会与已有的Segment有重合。

2.4.2 历史数据刷新

Cube构建完成以后，如果某些历史数据发生了改动，那么需要针对相应的Segment进行重新计算，这种构建称为刷新。刷新通常只针对增量构建的Cube而言，因为全量构建的Cube只要重新全部构建就可以得到更新；而增量更新的Cube因为有多个Segment，因此需要先选择要刷新的Segment，然后再进行刷新。

图2-24是提交刷新的请求页面，用户需要在下拉列表中选择一个时间区间。

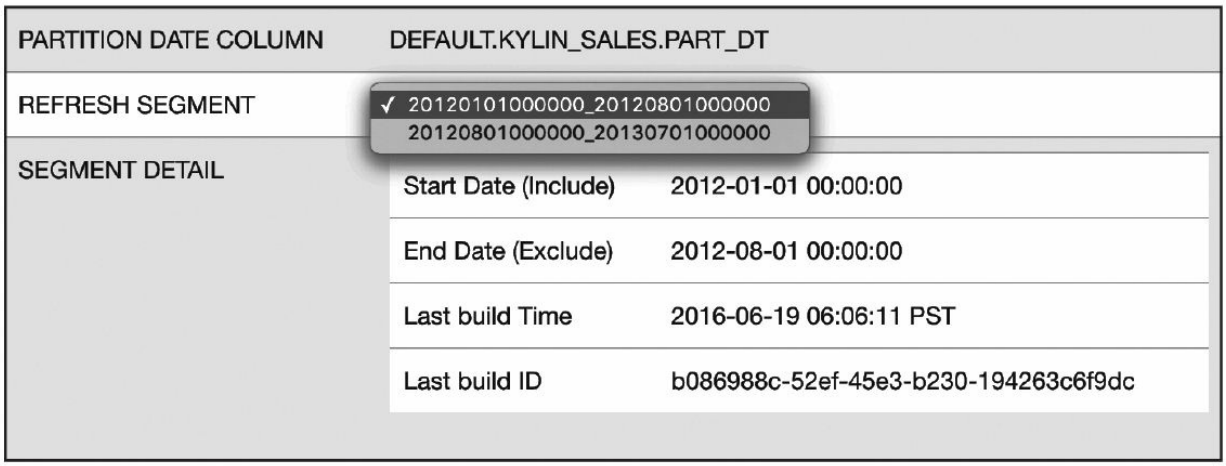


图2-24 刷新已有的Segment

提交以后，生成的构建任务与最初的构建任务完全一样。

在刷新的同时，Cube仍然可以被查询，只不过返回的是陈旧数据。当

Segment刷新完毕时，新的Segment会立即生效，查询开始返回最新的数据。老Segment则成为垃圾，等待回收。

2.4.3 合并

随着时间的迁移，Cube中可能会存在较多数量的Segment，使得查询性能下降，并且会给HBase集群管理带来压力。对此，需要适时地将一些Segment进行合并，将若干个小Segment合并成较大的Segment。

合并的好处具体如下。

·合并相同的Key，从而减少Cube的存储空间。

·由于Segment减少了，因此可以减少查询时的二次聚合，提高了查询性能。

·HTable的数量得以减少，更便于集群的管理。

下面来看看合并的操作步骤，图2-25中的Cube有两个Segment。

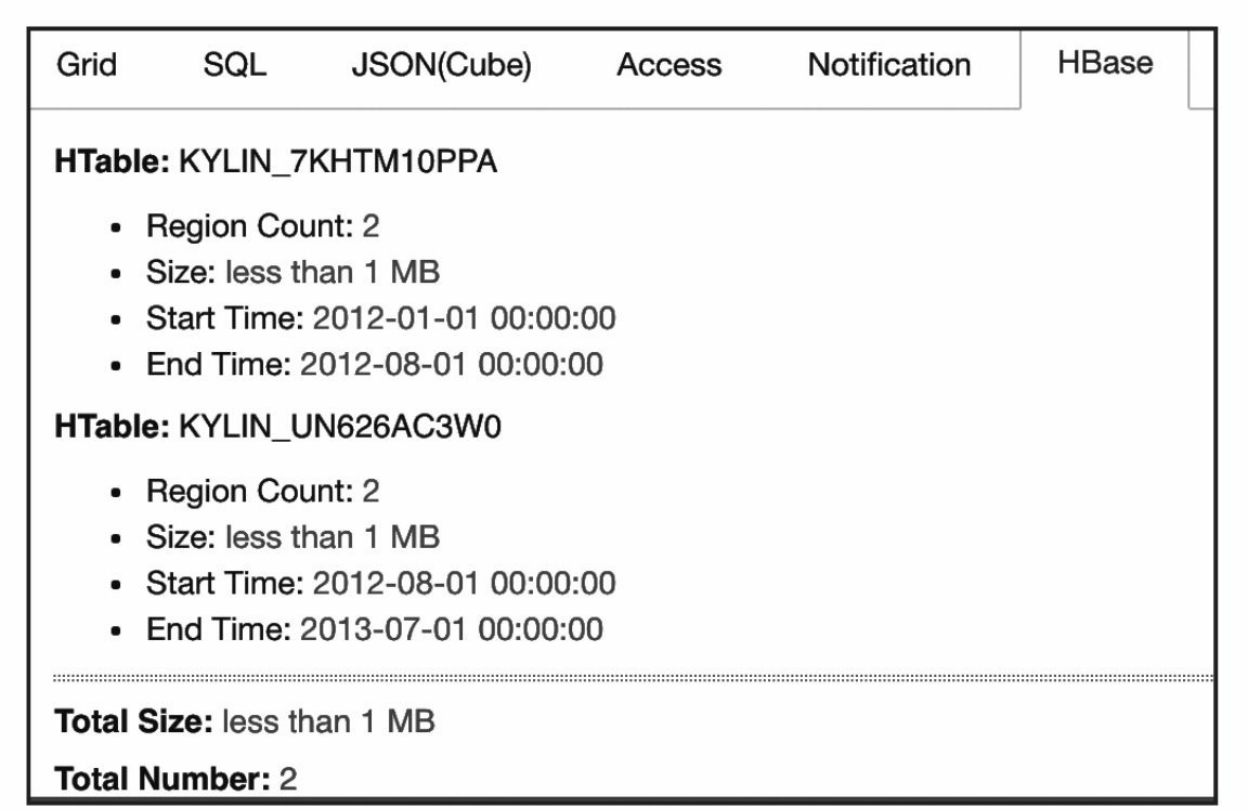


图2-25 Cube Segment列表

现在触发一个合并，单击Actions→Merge；选择要合并的起始Segment和结束Segment，生成一个合并的任务，如图2-26所示。

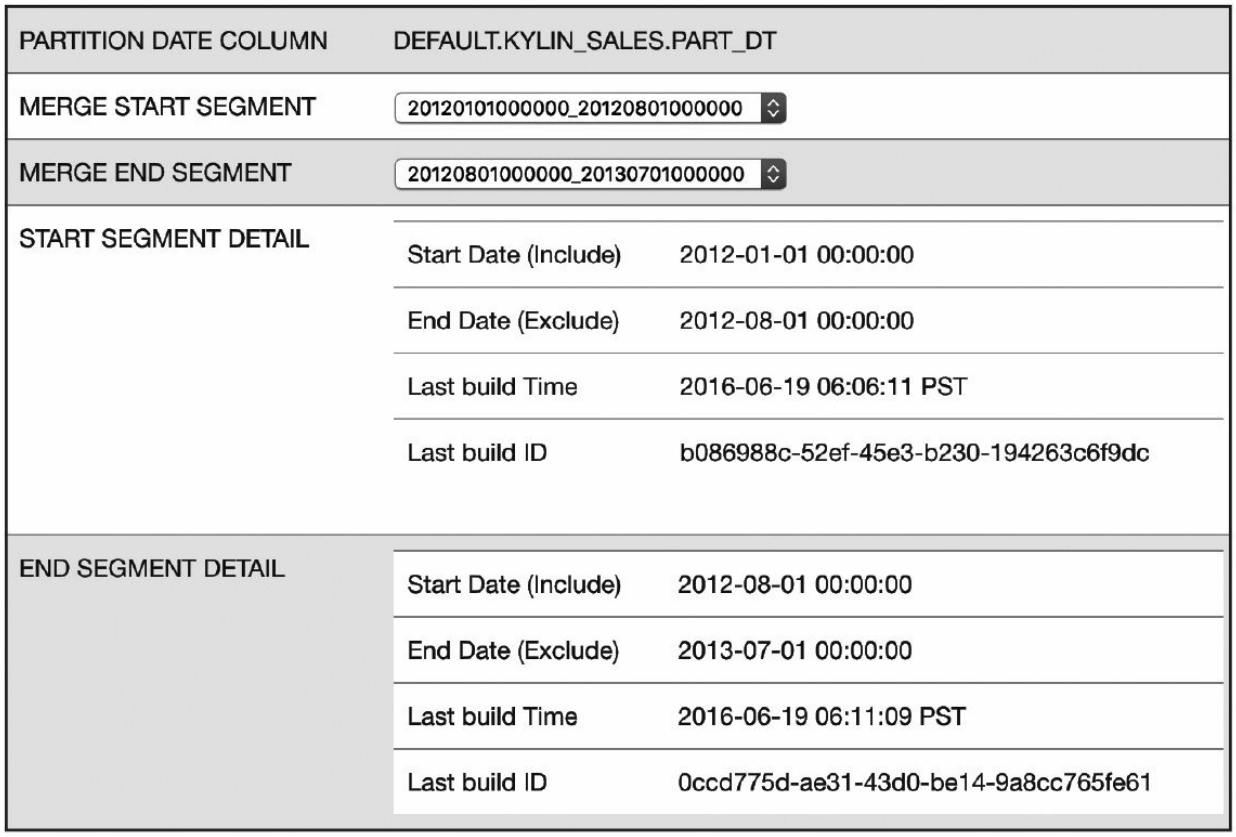


图2-26 提交合并任务

合并的时候，Kylin将直接以当初各个Segment构建时生成的Cuboid文件作为输入内容，而不需要从Hive加载原始数据。后续的步骤跟构建时基本一致。直到新的HTable加载完成后，Kylin才会卸载旧的HTable，从而确保在整个合并过程中，Cube都是可以查询的。

合并完成之后，此Cube的Segment减少为1个，如图2-27所示。

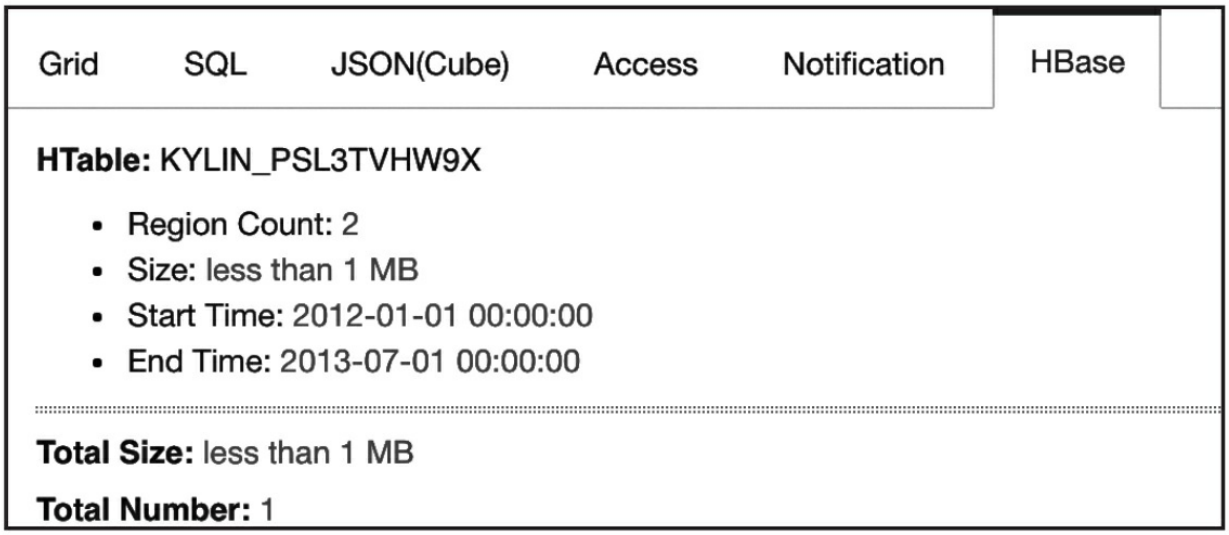


图2-27 合并后的Segment

2.5 查询Cube

·**注意** 本节将简要介绍如何查询Cube。更多内容请参考后续的章节（如第5章）。

Cube构建好以后，状态变为“READY”，就可以进行查询了。Kylin的查询语言是标准SQL的SELECT语句，这是为了获得与大多数BI系统和工具无缝集成的可能性。通常的一个查询语句类似于如下的SQL:

```
SELECT DIM1, DIM2, …, MEASURE1, MEASURE2… FROM FACT_TABLE
    INNER JOIN LOOKUP_1 ON FACT_TABLE.FK1 = LOOKUP_1.PK
    INNER JOIN LOOKUP_2 ON FACT_TABLE.FK2 = LOOKUP_2.PK
WHERE FACT_TABLE.DIMN = '' AND …
    GROUP BY DIM1, DIM2…
```

需要了解的是，只有当查询的模式跟Cube定义相匹配的时候，Kylin才能够使用Cube的数据来完成查询。Group By的列和Where条件里的列，必须是在Dimension中定义的列，而SQL中的度量，应该跟Cube中定义的度量相一致。

在一个项目下，如果有多个基于同一模型的Cube，而且它们都满足查询对表、维度和度量的要求；那么，Kylin会挑选一个“最优的”Cube来进行查询；这是一种基于成本（cost）的选择，Cube的成本计算中包括多方面的因素，例如Cube的维度数、度量、数据模型的复杂度等。查询引擎将为每个Cube为完成此SQL估算一个成本值，然后选择成本最小的Cube来完

成此查询。

如果查询是在Kylin的Web GUI上进行的，那么查询结果会以表的形式展现出来，如图2-28所示。所执行的Cube名称也会一同显示。用户可以单击“Visualization”按钮生成简单的可视化图形，或单击“Export”按钮将结果集下载到本地。

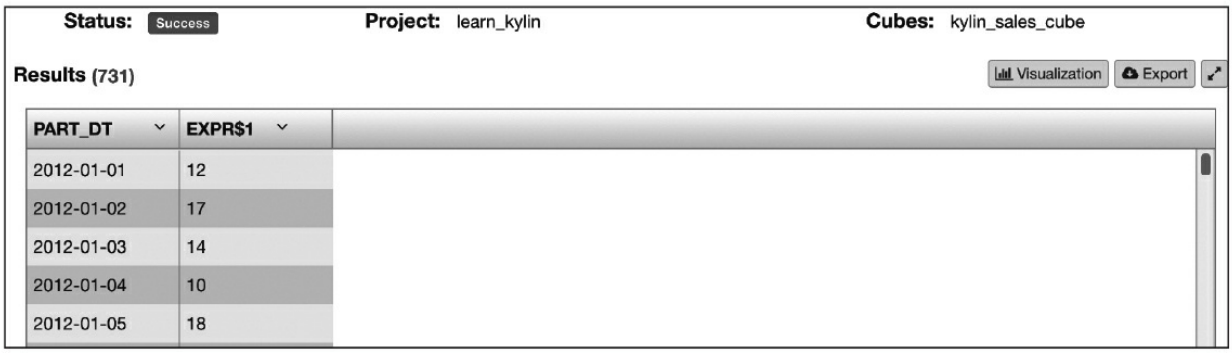


图2-28 查询结果展示

2.6 SQL参考

Apache Kylin支持标准SQL作为查询语言，但是SQL有很多变体，Kylin支持的只是SQL所有变体中的一个子集，并不是支持所有现存的SQL语句和语法。用户在使用Kylin之前，需要对Kylin所支持的SQL有一个了解，以避免走弯路。

首先，Kylin作为OLAP引擎，只支持查询，而不支持其他操作，如插入、更新等，即所有的SQL都必须是SELECT语句，否则Kylin会报错。

第二，查询Kylin中SQL语句的表名、列名、度量、连接关系时，需要至少跟一个Cube的模型相匹配；在设计Cube的时候，需要充分考虑查询的需求，避免遗漏表、列等信息。

第三，Kylin使用Apache Calcite做SQL语法分析。Apache Calcite是一个开源的SQL引擎，它提供了标准SQL解析、多种查询优化和连接各种数据源的能力；Calcite项目在Hadoop中越来越引人注意，并且已被众多项目集成为SQL解析器。

一条SQL语句首先需要被Calcite解析，然后才可以被Kylin执行。下面是Calcite中的SELECT语句的语法（引自https://calcite.apache.org/docs/reference.html）：

```
SELECT [ STREAM ] [ ALL | DISTINCT ]
            { * | projectItem [, projectItem ]* }
        FROM tableExpression
        [ WHERE booleanExpression ]
        [ GROUP BY { groupItem [, groupItem ]* } ]
        [ HAVING booleanExpression ]
        [ WINDOW windowName AS windowSpec [, windowName AS windowSpec ]* ]

projectItem:
        expression [ [ AS ] columnAlias ]
    |   tableAlias . *

tableExpression:
        tableReference [, tableReference ]*
    |   tableExpression [ NATURAL ] [ LEFT | RIGHT | FULL ] JOIN tableExpression
[ joinCondition ]

joinCondition:
        ON booleanExpression
    |   USING '(' column [, column ]* ')'
```

第四，不是所有的Calcite能够解析的SELECT语句都可以被Kylin执行；还有一些SQL功能，现阶段Kylin（截止v1.5.3）还不支持，未来会考虑加以实现，目前已知的有如下三项SQL功能。

·Window函数：https://issues.apache.org/jira/browse/KYLIN-1732

·Union：https://issues.apache.org/jira/browse/KYLIN-1206

·Between AND：https://issues.apache.org/jira/browse/KYLIN-1770

上述三个功能已经在Apache Kylin主分支上得以实现，但目前（2016年8月）还未包含在最新的发行版中。如无意外，应该会在下一个发行版中发布。

2.7 小结

本章介绍了使用Apache Kylin必备的基本概念，如星形数据模型、事实表、维表、维度、度量等，并在这些基础上快速创建了基于Sample Data的模型，构建Cube，最后执行SQL查询。带领读者体验了Apache Kylin的主要使用过程。后续的章节将继续展开和探讨这个过程中的一些关键技术，比如增量构建、可视化和Cube优化等。

第3章 增量构建

第2章介绍了如何构建Cube并利用其完成在线多维分析的查询。每次Cube的构建都会从Hive中批量读取数据，而对于大多数业务场景来说，Hive中的数据处于不断增长的状态。为了支持Cube中的数据能够不断地得到更新，且无需重复地为已经处理过的历史数据构建Cube，因此对于Cube引入了增量构建的功能。

我们将Cube划分为多个Segment，每个Segment用起始时间和结束时间来标志。Segment代表一段时间内源数据的预计算结果。在大部分情况下（例外情况见第4章“流式构建”），一个Segment的起始时间等于它之前那个Segment的结束时间，同理，它的结束时间等于它后面那个Segment的起始时间。同一个Cube下不同的Segment除了背后的源数据不同之外，其他如结构定义、构建过程、优化方法、存储方式等都完全相同。

本章将首先介绍如何设计并创建能够增量构建的Cube，然后介绍实际测试或生产环境中触发增量构建的方法，最后将会介绍如何处理由于增量构建而导致的Segment碎片，以保持Kylin的查询性能。

3.1 为什么要增量构建

全量构建可以看作增量构建的一种特例：在全量构建中，Cube中只存在唯一的一个Segment，该Segment没有分割时间的概念，因此也就没有起始时间和结束时间。全量构建和增量构建各有其适用的场景，用户可以根据自己的业务场景灵活地进行切换。全量构建和增量构建的详细对比如表3-1所示。

表3-1 全量构建和增量构建的对比

全量构建	增量构建
每次更新时都需要更新整个数据集	每次只对需要更新的时间范围进行更新，因此离线计算量相对较小
查询时不需要合并不同 Segment 的结果	查询时需要合并不同 Segment 的结果，因此查询性能会受到影响
不需要后续的 Segment 合并	累计一定量的 Segment 之后，需要进行合并
适合小数据量或全表更新的 Cube	适合大数据量的 Cube

对于全量构建来说，每当需要更新Cube数据的时候，它不会区分历史数据和新加入的数据，也就是说，在构建的时候会导入并处理所有的原始数据。而增量构建只会导入新Segment指定的时间区间内的原始数据，并只对这部分原始数据进行预计算。为了验证这个区别，可以到Kylin的Monitor页面观察构建的第二步——创建Hive中间表（Create Intermediate Flat Hive Table），单击纸张形的LOG按钮即可观察该步骤的参数：

```
    INSERT OVERWRITE TABLE
    kylin_intermediate_test_kylin_cube_without_slr_left_join_desc_20120601000000_2013
0101000000 SELECT
    TEST_KYLIN_FACT.CAL_DT
    ,TEST_KYLIN_FACT.LEAF_CATEG_ID
    ,TEST_KYLIN_FACT.LSTG_SITE_ID
    ,TEST_CATEGORY_GROUPINGS.META_CATEG_NAME
    ,TEST_CATEGORY_GROUPINGS.CATEG_LVL2_NAME
    ,TEST_CATEGORY_GROUPINGS.CATEG_LVL3_NAME
    ,TEST_KYLIN_FACT.LSTG_FORMAT_NAME
    ,TEST_KYLIN_FACT.SLR_SEGMENT_CD
    ,TEST_KYLIN_FACT.PRICE
    ,TEST_KYLIN_FACT.ITEM_COUNT
    ,TEST_KYLIN_FACT.SELLER_ID
    ,TEST_SITES.SITE_NAME
    FROM DEFAULT.TEST_KYLIN_FACT as TEST_KYLIN_FACT
    LEFT JOIN EDW.TEST_CAL_DT as TEST_CAL_DT
    ON TEST_KYLIN_FACT.CAL_DT = TEST_CAL_DT.CAL_DT
    LEFT JOIN DEFAULT.TEST_CATEGORY_GROUPINGS as TEST_CATEGORY_GROUPINGS
    ON TEST_KYLIN_FACT.LEAF_CATEG_ID = TEST_CATEGORY_GROUPINGS.LEAF_CATEG_ID AND
TEST_KYLIN_FACT.LSTG_SITE_ID = TEST_CATEGORY_GROUPINGS.SITE_ID
    LEFT JOIN EDW.TEST_SITES as TEST_SITES
    ON TEST_KYLIN_FACT.LSTG_SITE_ID = TEST_SITES.SITE_ID
    LEFT JOIN EDW.TEST_SELLER_TYPE_DIM as TEST_SELLER_TYPE_DIM
    ON TEST_KYLIN_FACT.SLR_SEGMENT_CD = TEST_SELLER_TYPE_DIM.SELLER_TYPE_CD
    WHERE (TEST_KYLIN_FACT.CAL_DT >= '2012-06-01' AND TEST_KYLIN_FACT.CAL_DT <
'2013-01-01')
     distribute by rand();
```

该构建任务对应于名为test_kylin_cube_without_slr_left_join_empty的Cube构建，其Seg-ment所包含的时间段为从2012-06-01（包含）到2013-01-01（不包含），可以看到在导入数据的Hive命令中带入了包含这两个日期的过滤条件，以此保证后续构建的输入仅包含2012-06-01到2013-01-01这段时间内的数据。这样的过滤能够减少增量构建在后续的预计算中所需要处理的数据规模，有利于减少集群的计算量，加速Segment构建的时间。

其次，增量构建的Cube和全量构建的Cube在查询时也有不同。对于

增量构建的Cube，由于不同时间的数据分布在不同的Segment之中，因此为了获得完整的数据，查询引擎需要向存储引擎请求读取各个Segment的数据。当然，查询引擎会根据查询中的条件自动跳过不感兴趣的Segment。对于全量构建的Cube，查询引擎只需要向存储引擎访问单个Segment所对应的数据，从存储层返回的数据无需进行Segment之间的聚合，但是这也并非意味着查询全量构建的Cube不需要查询引擎做任何额外的聚合，为了加强性能，单个Segment的数据也有可能被分片存储到引擎的多个分区上（参考第6章），从而导致查询引擎可能仍然需要对单个Segment不同分区的数据做进一步的聚合。当然，整体来说，增量构建的Cube上的查询会比全量构建的做更多的运行时聚合，而这些运行时聚合都发生在单点的查询引擎之上，因此通常来说增量构建的Cube上的查询会比全量构建的Cube上的查询要慢一些。

可以看到，日积月累，增量构建的Cube中的Segment越来越多，根据上一段的分析可以猜测到该Cube的查询性能也会越来越慢，因为需要在单点的查询引擎中完成越来越多的运行时聚合。为了保持查询性能，Cube的管理员需要定期地将某些Segment合并在一起，或者让Cube根据Segment保留策略自动地淘汰那些不会再被查询到的陈旧Segment。关于这部分的详细内容会在3.4.1节中展开详细讨论。

最后，我们可以得到这样的结论：对于小数据量的Cube，或者经常需要全表更新的Cube，使用全量构建需要更少的运维精力，以少量的重复

计算降低生产环境中的维护复杂度。而对于大数据量的Cube，例如，对于一个包含两年历史数据的Cube，如果需要每天更新，那么每天为了新数据而去重复计算过去两年的数据就会变得非常浪费，在这种情况下需要考虑使用增量构建。

3.2 设计增量Cube

3.2.1 设计增量Cube的前提

并非所有的Cube都适用于增量构建，Cube的定义必须包含一个时间维度，用来分割不同的Segment，我们将这样的维度称为分割时间列（Partition Date Column）。尽管由于历史原因该命名中存在“date”的字样，但是分割时间列既可以是Hive中的Date类型、也可以是Timestamp类型或String类型。无论是哪种类型，Kylin都要求用户显式地指定分割时间列的数据格式，例如精确到年月日的Date类型（或者String类型）的数据格式可能是yyyyMMdd或yyyy-MM-dd，如果是精确到时分秒的Timestamp类型（或者String类型），那么数据格式可能是YYYY-MM-DD HH:MM:SS。

在一些场景中，时间由长整数Unix Time来表示，由于对该类型的支持存在争议（详情可参见https://issues.apache.org/jira/browse/KYLIN-1698），因此在目前的版本中并不支持使用长整数类型作为分割时间列。作为一种变通的方法，可以在ETL过程中克服这个问题。具体来说，就是在Hive中为包含长整数时间列的表创建一个视图，将长整数时间列转化为符合Kylin规范的任意类型，在后续的Cube设计中，应使用该视图而不是原始的表。

满足了设计增量Cube的前提之后，在进行增量构建时，将增量部分的起始时间和结束时间作为增量构建请求的一部分提交给Kylin的任务引擎，任务引擎会根据起始时间和结束时间从Hive中抽取相应时间的数据，并对这部分数据做预计算处理，然后将预计算的结果封装成为一个新的Segment，并将相应的信息保存到元数据和存储引擎中。一般来说，增量部分的起始时间等于Cube中最后一个Segment的结束时间。

3.2.2 增量Cube的创建

创建增量Cube的过程和创建普通Cube的过程基本类似，只是增量Cube会有一些额外的配置要求。

1.Model层面的设置

每个Cube背后都关联着一个Model，Cube之于Model就好像Java中的Object之于Class。如同3.2.1节中所描述的，增量构建的Cube需要指定分割时间列。同一个Model下不同分割时间列的定义应该是相同的，因此我们将分割时间列的定义放到了Model之中。Model的创建和修改在第2章中已经介绍过，这里将跳过重复的部分，直接进入Model Designer的最后一步Settings来添加分割时间列，如图3-1所示。

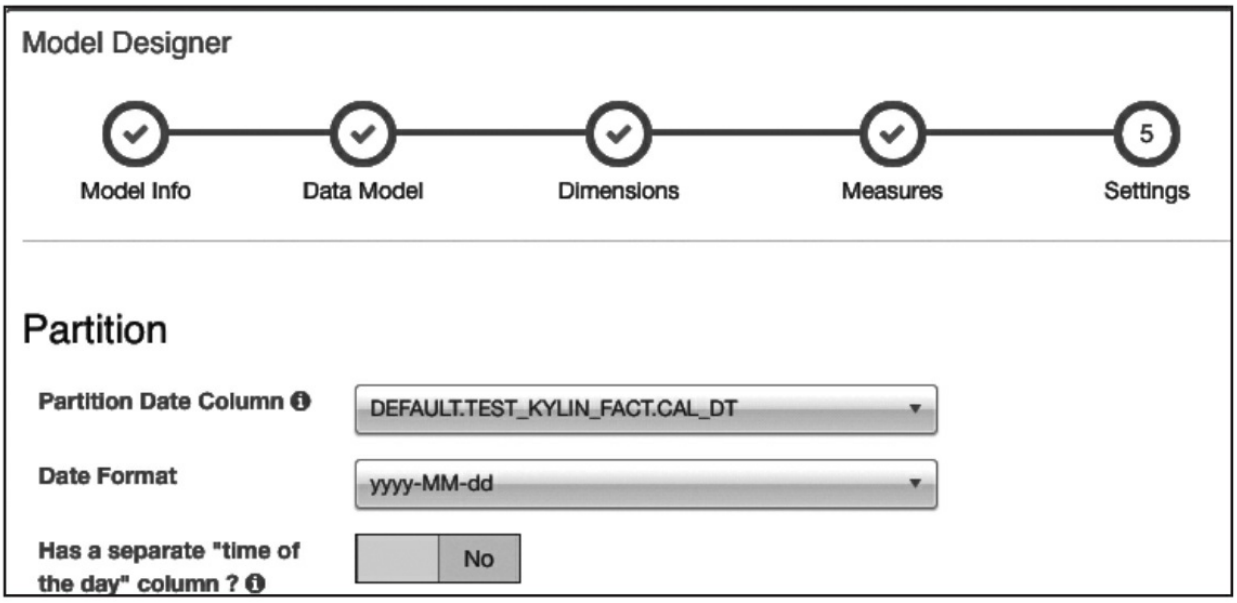


图3-1 定义分割时间列

目前分割时间列必须是事实表上的列，且它的格式必须满足3.2.1节中所描述的要求。一般来说如果年月日已经足够帮助分割不同的Segment，那么在大部分情况下日期列是分割时间列的首选。当用户需要更细的分割粒度时，例如用户需要每6小时增量构建一个新的Segment，那么对于这种情况，则需要挑选包含年月日时分秒的列作为分割时间列。

在一些用户场景中，年月日和时分秒并不体现在同一个列上，例如在用户的事实表上有两个列，分别是“日期”和“时间”，分别保存记录发生的日期（年月日）和时间（时分秒），对于这样的场景，允许用户指定一个额外的分割时间列来指定除了年月日之外的时分秒信息。为了区分，我们将之前的分割时间列称为常规分割时间列，将这个额外的列称为补充分割时间列。在勾选了“Has aseparate”time of the day“column？”选项之后（如图3-2所示），用户可以选择一个符合时分秒时间格式的列作为补充的分割时间列。由于日期的信息已经体现在了常规的分割时间列之上，因此补充的分割时间列中不应该再具有日期的信息。反过来说，如果这个列中既包含年月日信息，又包含时分秒信息，那么用户应该将它指定为格式是YYYY-MM-DD HH:MM:SS的常规分割时间列，而不需要勾选“Has aseparate”time of the day“column？”。在大部分场景下用户可以跳过补充分割时间列。

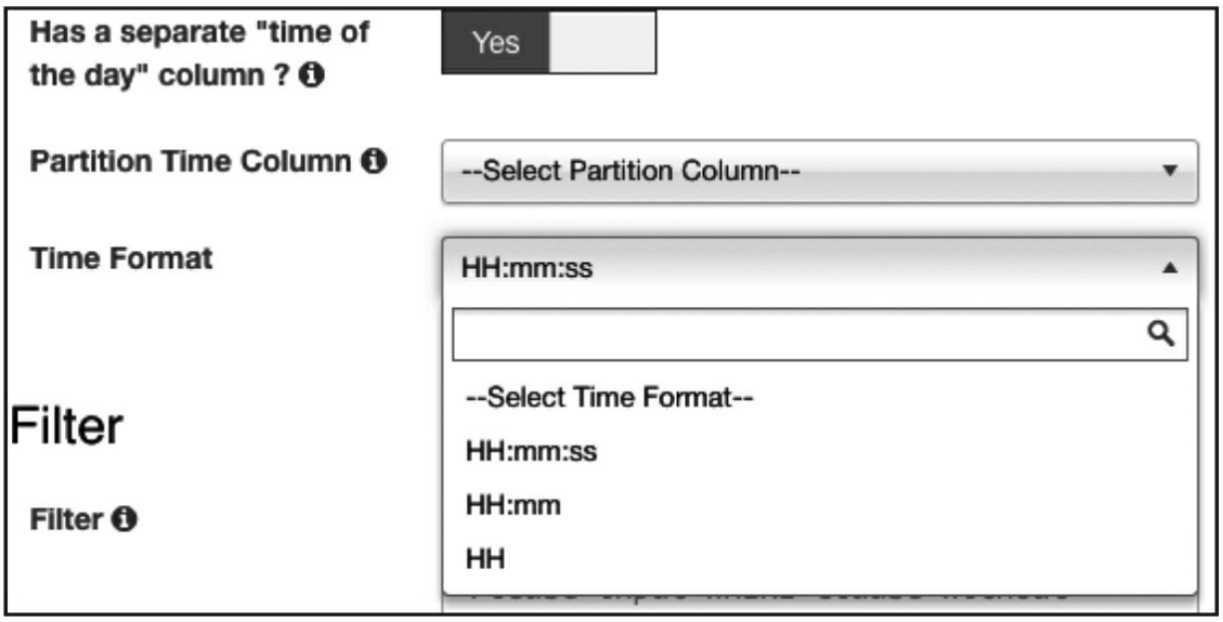


图3-2 补充时间分割列

2.Cube层面的设置

Cube的创建和修改在第2章中已经做过介绍，这里将跳过重复的部分，直接进入Cube Designer的“Refresh Settings”。这里的设置目前包含“Auto Merge Thresholds”、“Retention Threshold”和“Partition Start Date”。“Partition Start Date”是指Cube默认的第一个Segment的起始时间。同一个Model下不同的Cube可以指定不同的起始时间，因此该设置项出现在Cube Designer之中。“Auto Merge Thresholds”用于指定Segment自动合并的阈值，而“Retention Threshold”则用于指定将过期的Segment自动抛弃。3.4节将详细介绍这两个功能。

3.3 触发增量构建

3.3.1 Web GUI触发

在Web GUI上触发Cube的增量构建与触发全量构建的方式基本相同。在Web GUI的Model页面中，选中想要增量构建的Cube，单击Action→Build，如图3-3所示。

不同于全量构建，增量构建的Cube会在此时弹出对话框让用户选择“End Date”（如图3-4所示），目前Kylin要求增量Segment的起始时间等于Cube中最后一个Segment的结束时间，因此当我们为一个已经有Segment的Cube触发增量构建的时候，“Start Date”的值已经被确定，且不能修改。如果在触发增量构建的时候Cube中不存在任何的Segment，那么“Start Date”的值会被系统设置为“Partition Start Date”的值（参见3.2.2节）。

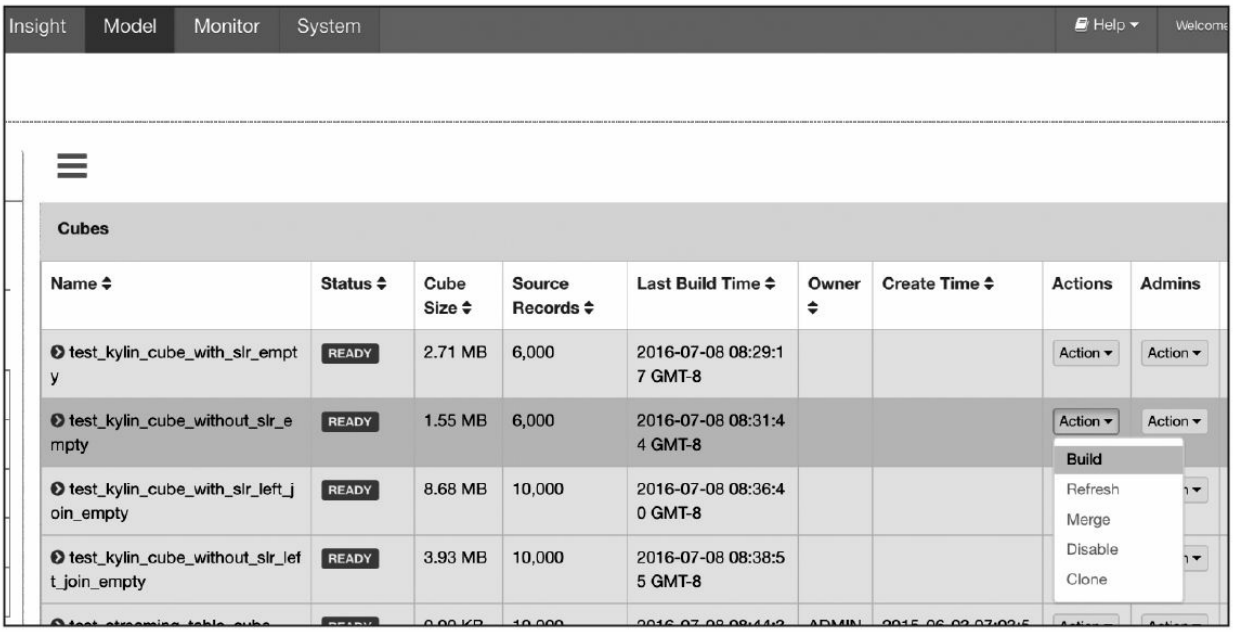


图3-3 触发增量构建

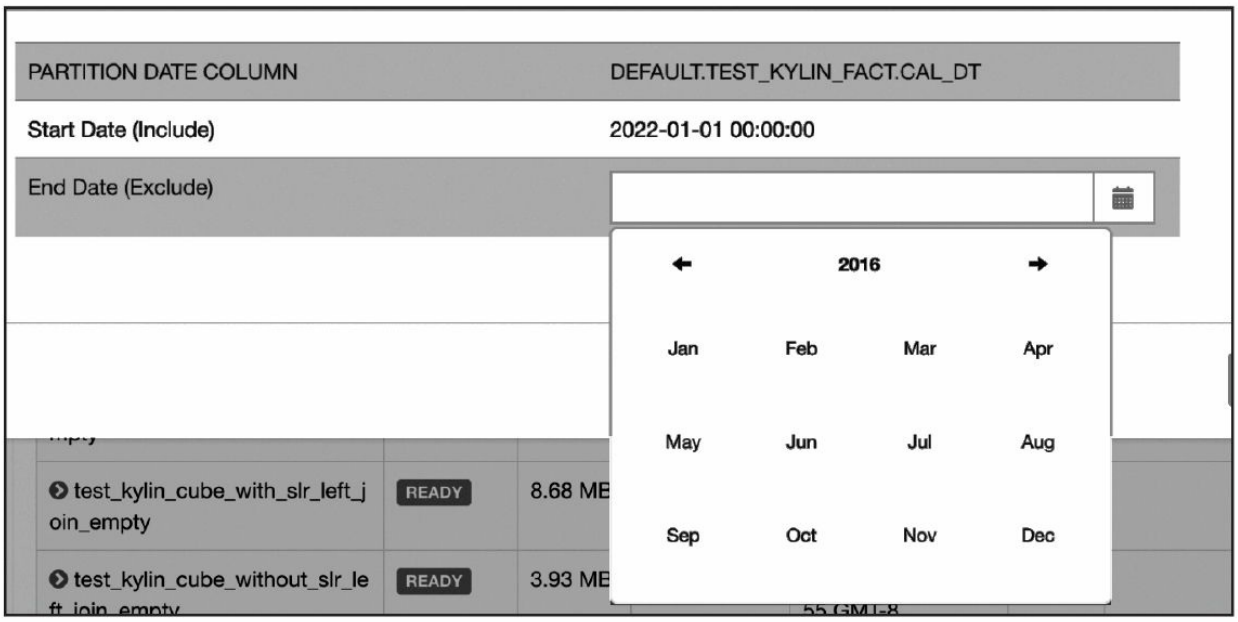


图3-4 选择增量构建End Date

仅当Cube中不存在任何Segment，或者不存在任何未完成的构建任务时，Kylin才接受该Cube上新的构建任务。未完成的构建任务不仅包含正

在运行中的构建任务，还包括已经出错并处于ERROR状态的构任务。如果存在一个ERROR状态的构建任务，那么用户需要先处理好该构建任务，然后才能成功地向Kylin提交新的构建任务。处理ERROR状态的构建任务的方式有两种：比较正常的做法是首先在Web GUI或后台的日志中查找构建失败的原因，解决问题后回到Monitor页面，选中失败的构建任务，单击Action→Resume，恢复该构建任务的执行。我们知道构建任务分为多个子步骤，Resume操作会跳过之前所有已经成功了的子步骤，直接从第一个失败的子步骤重新开始执行。举例来说，如果某次构建任务失败，我们在后台Hadoop的日志中发现失败的原因是由于Mapper和Reducer分配的内存过小导致了内存溢出，那么我们可以在更新了Hadoop相关的配置之后再恢复失败的构建任务。

3.3.2 构建相关的Rest API

Kylin提供了Rest API以帮助自动化地触发增量构建。该API同样也适用于非增量构建的Cube。关于Kylin API的更详细的介绍可以参见Kylin官网：

http://kylin.apache.org/docs15/howto/howto_build_cube_with_restapi.html和

http://kylin.apache.org？/docs15/howto/howto_use_restapi.html。

本节将着重介绍增量构建相关的API。事实上我们在Web GUI上进行的所有操作，其背后调用的都是同一套Rest API，所以在使用Rest API触发构建的时候，应当谨记之前进行Web GUI构建时所遇到的限制和经验。

1.获取Segment列表

首先可以通过以下的Rest API来获取某个Cube所包含的所有的Segment列表信息。返回的列表信息可以帮助客户端分析Cube的状态，并且决定下一步增量构建的参数：

```
GET http://hostname:port/kylin/api/Cubes?CubeName={CubeName}

Path Variable
CubeName - 必须的，Cube 名字
```

举例而言，假设在本地的7070端口启动了Kylin Server，那么可以通

过如下的Rest请求获取名为test_kylin_cube_without_slr_empty的Cube的Segment列表：

```
curl -X GET -H "Authorization: Basic QURNSU46S1lMSU4=" -H "Content-Type: application/json" http://localhost:7070/kylin/api/Cubes?CubeName=test_kylin_cube_without_slr_empty
```

格式化之后，该请求的返回结果如下所示：


```
[
    {
        "uuid": "daa53e80-41be-49a5-90ca-9fb7294db186",
        "version": "1.5.3",
        "name": "test_kylin_cube_without_slr_empty",
        "owner": null,
        "cost": 50,
        "status": "READY",
        "segments": [
            {
                "uuid": "f492158b-0910-4ced-bc51-26e78b9b8b81",
                "name": "19700101000000_20220101000000",
                "status": "READY",
                "dictionaries": {
                    "DEFAULT.TEST_KYLIN_FACT/LSTG_SITE_ID": "/dict/EDW.TEST_SITES/SITE_ID/a9f93c23-9eca-4e2e-a814-17b81344a816.dict",
                    "DEFAULT.TEST_CATEGORY_GROUPINGS/CATEG_LVL2_NAME": "/dict/DEFAULT.TEST_CATEGORY_GROUPINGS/CATEG_LVL2_NAME/58372aa5-6d42-4045-a32f-e6ae41c219a8.dict",
```

```
                        "DEFAULT.TEST_KYLIN_FACT/LSTG_FORMAT_NAME": "/dict/DEFAULT.
TEST_KYLIN_FACT/LSTG_FORMAT_NAME/2e2e8137-5600-4c63-ba3f-3f382f452227.dict",
                        "DEFAULT.TEST_KYLIN_FACT/LEAF_CATEG_ID": "/dict/DEFAULT.TEST_
CATEGORY_GROUPINGS/LEAF_CATEG_ID/ee675200-8c5c-4112-99fa-763bb0aa689a.dict",
                        "DEFAULT.TEST_CATEGORY_GROUPINGS/META_CATEG_NAME": "/dict/DEFAULT.
TEST_CATEGORY_GROUPINGS/META_CATEG_NAME/8bc37a42-5577-4c18-b6a5-bd1c7eb55c73.dict",
                        "DEFAULT.TEST_KYLIN_FACT/SLR_SEGMENT_CD": "/dict/EDW.TEST_SELLER_
TYPE_DIM/SELLER_TYPE_CD/0c356b8c-74fa-4e58-b8b8-bbbd5095a6be.dict",
                        "DEFAULT.TEST_KYLIN_FACT/CAL_DT": "/dict/EDW.TEST_CAL_DT/CAL_
DT/5e4b4f35-0fc8-4940-b123-b18c9f77da19.dict",
                        "DEFAULT.TEST_KYLIN_FACT/PRICE": "/dict/DEFAULT.TEST_KYLIN_FACT/
PRICE/94d429fc-60ef-4635-af1e-2b47679bb494.dict",
                        "DEFAULT.TEST_CATEGORY_GROUPINGS/CATEG_LVL3_NAME": "/dict/DEFAULT.
TEST_CATEGORY_GROUPINGS/CATEG_LVL3_NAME/759e5fd6-9c7e-47ed-9293-e7c8695b6bb4.dict"
                    },
                    "snapshots": {
                        "EDW.TEST_SITES": "/table_snapshot/test_sites/1c3d3b91-8afa-
4d12-8743-5376133185eb.snapshot",
                        "EDW.TEST_CAL_DT": "/table_snapshot/test_cal_dt/96a2ad25-4279-
4c7f-9c0a-7e1f0132ae77.snapshot",
                        "DEFAULT.TEST_CATEGORY_GROUPINGS": "/table_snapshot/test_category
_groupings/3ed9f146-2a8b-4bdb-8899-45f2d765c25a.snapshot",
                        "EDW.TEST_SELLER_TYPE_DIM": "/table_snapshot/test_seller_type_
dim/f7f7b3c8-cfe8-49ea-8230-8c296d0e03ef.snapshot"
                    },
                    "storage_location_identifier": "KYLIN_KZO9NPAWGC",
                    "date_range_start": 0,
                    "date_range_end": 1640995200000,
                    "source_offset_start": 0,
                    "source_offset_end": 0,
                    "size_kb": 1589,
                    "input_records": 6000,
                    "input_records_size": 154637,
                    "last_build_time": 1467995504950,
                    "last_build_job_id": "f3f49487-e5bc-4fd0-a571-58c13c9311e9",
                    "create_time_utc": 1467995076271,
                    "cuboid_shard_nums": {},
                    "total_shards": 1,
                    "blackout_cuboids": [],
                    "binary_signature": null,
                    "index_path": "/kylin/kylin_metadata/kylin-f3f49487-e5bc-4fd0-a571-
58c13c9311e9/test_kylin_cube_without_slr_empty/secondary_index/",
                    "rowkey_stats": [
                        [
                            "LEAF_CATEG_ID",
                            134,
                            1
```

```
 ],
 [
     "META_CATEG_NAME",
     44,
     1
],
[
    "CATEG_LVL2_NAME",
    94,
    1
],
[
    "CATEG_LVL3_NAME",
    127,
    1
],
[
    "LSTG_SITE_ID",
    262,
    2
],
[
    "SLR_SEGMENT_CD",
    8,
    1
],
[
    "CAL_DT",
```

```
                    3652427,
                    3
                ],
                [
                    "LSTG_FORMAT_NAME",
                    5,
                    1
                ],
                [
                    "PRICE",
                    5999,
                    2
                ]
            ]
        }
    ],
    "last_modified": 1467995504950,
    "descriptor": "test_kylin_cube_without_slr_desc",
    "create_time_utc": 0,
    "size_kb": 1589,
    "input_records_count": 6000,
    "input_records_size": 154637
  }
]
```


尽管输出比较复杂，但是我们仍然能够迅速地观察到当前的test_kylin_cube_without_slr_empty包含一个Segment，该Segment的分割时间为1970-01-01到2022-01-01。我们还能看到该Segment的状态（“status”）均为READY，表示这个Segment背后的构建任务均已正常完成，并且这个Segment已经可以正常使用。

2.获取构建任务详情

如果Segment的状态显示为“NEW”，则说明该Segment背后的构建任务尚未完成，需要提取该构建任务的标识符（job id），即Segment中的

last_build_job_id字段的值，然后以此为参数向Kylin提交如下的Rest请求以获取该构建任务的详情：

```
GET http://hostname:port/kylin/api/jobs/{job_uuid}
```

Path Variable

Job_uuid - 必需的，构建任务标识符

该请求的返回会带上相应的任务步骤清单，步骤中可能包含MapReduce作业或其他作业。每一个步骤都有相应的状态信息“step_status”。

·PENDING：表示该步骤处于等待被执行的状态。

·RUNNING：表示该步骤处于执行状态。

·ERROR：表示该步骤的执行已经结束，并且该步骤执行失败。

·DISCARDED：表示该步骤由于这个构建任务被取消而处于取消状态。

·FINISHED：表示该步骤的执行已经结束，并且该步骤执行成功。

test_kylin_cube_without_slr_empty的第一个Segment的last_build_job_id为f3f49487-e5bc-4fd0-a571-58c13c9311e9，通过以上的Rest接口可以得到如下的结果：

```
    {
        "uuid": "f3f49487-e5bc-4fd0-a571-58c13c9311e9",
        "version": "1.5.3",
        "name": "test_kylin_cube_without_slr_empty - 19700101000000_20220101000000 - BUILD -
GMT-08:00 2016-07-08 08:24:36",
        "type": "BUILD",
        "duration": 393,
        "steps": [
            {
                "id": "f3f49487-e5bc-4fd0-a571-58c13c9311e9-00",
                "name": "Count Source Table",
                "info": {
                    "endTime": "1467995164094",
                    "source_records_size": "571743",
                    "mr_job_id": "job_1466095360365_0611",
                    "hdfs_bytes_written": "6",
                     "yarn_application_tracking_url": "http://sandbox.hortonworks.com:
8088/proxy/application_1466095360365_0611/",
                    "startTime": "1467995122900"
                },
                "interruptCmd": null,
                "sequence_id": 0,
                "exec_cmd": "hive -e \"SET dfs.replication=2;\nSET hive.exec.compress.
output=true;\nSET hive.auto.convert.join.noconditionaltask=true;\nSET hive.
auto.convert.join.noconditionaltask.size=300000000;\nSET hive.merge.size.per.
task=32000000;\n\nset hive.exec.compress.output=false;\n\ndfs -mkdir -p /kylin/
kylin_metadata/kylin-f3f49487-e5bc-4fd0-a571-58c13c9311e9/row_count;INSERT OVERWRITE
DIRECTORY '/kylin/kylin_metadata/kylin-f3f49487-e5bc-4fd0-a571-58c13c9311e9/row_count'
SELECT count(*) from DEFAULT.TEST_KYLIN_FACT TEST_KYLIN_FACT\nWHERE (TEST_KYLIN_FACT.
CAL_DT < '2022-01-01')\n\n\"",
                "interrupt_cmd": null,
                "exec_start_time": 1467995122900,
                "exec_end_time": 1467995164094,
                "exec_wait_time": 0,
                "step_status": "FINISHED",
                "cmd_type": "SHELL_CMD_HADOOP",
                "run_async": false
            },
            {
                "id": "f3f49487-e5bc-4fd0-a571-58c13c9311e9-01",
                "name": "Create Intermediate Flat Hive Table",
                "info": {
                    "endTime": "1467995223284",
```

```
            "startTime": "1467995164168"
        },
        "interruptCmd": null,
        "sequence_id": 1,
        "exec_cmd": null,
        "interrupt_cmd": null,
        "exec_start_time": 1467995164168,
        "exec_end_time": 1467995223284,
        "exec_wait_time": 0,
        "step_status": "FINISHED",
        "cmd_type": "SHELL_CMD_HADOOP",
        "run_async": false
    },

...

    {
        "id": "f3f49487-e5bc-4fd0-a571-58c13c9311e9-16",
        "name": "Garbage Collection",
        "info": {
            "endTime": "1467995516662",
            "startTime": "1467995505050"
        },
      "interruptCmd": null,
      "sequence_id": 16,
      "exec_cmd": null,
```

```
            "interrupt_cmd": null,
            "exec_start_time": 1467995505050,
            "exec_end_time": 1467995516662,
            "exec_wait_time": 0,
            "step_status": "FINISHED",
            "cmd_type": "SHELL_CMD_HADOOP",
            "run_async": false
        }
    ],
    "submitter": "TEST",
    "progress": 100,
    "last_modified": 1467995516721,
    "related_cube": "test_kylin_cube_without_slr_empty",
    "related_segment": "f492158b-0910-4ced-bc51-26e78b9b8b81",
    "exec_start_time": 0,
    "exec_end_time": 0,
    "mr_waiting": 94,
    "job_status": "FINISHED"
}
```


由于篇幅的限制，此处省略了中间14个子步骤的信息，但是仍然可以观察到每个子步骤的信息都描述了步骤的参数等元信息，另外每个子步骤还有一个唯一的字符串标识符“id”。这些信息可以帮助快速定位问题的所在。

3.获取构建步骤的输出

一般情况下，构建触发的客户端会首先获取Cube的Segment列表，如果所有Segment的状态都是READY，那么客户端就可以开始构建新的Segment。反之，如果存在状态不是READY的Segment，那么客户端需要获取构建任务详情来观察各个子步骤的状态：如果某个子步骤的状态为

ERROR，或者长时间PENDING，或者运行了非常长的时间，那么客户端有必要检查一下该步骤中究竟正在发生什么。Kylin提供了另外一个Rest接口允许用户获取构建任务中某个特定子步骤的输出，接口的请求如下：

```
GET http://hostname:port/kylin/api/jobs/{job_uuid}/steps/{step_id}/output
```

Path Variable
Job_uuid - 必需的，构建任务标识符
Step_id - 必需的，构建任务子步骤标识符

该接口的输出为该步骤的日志，根据输出的结果，用户可以在触发构建的客户端中找到问题并修复问题，并且可调用以下的RESUME Rest接口重新执行该次构建任务。RESUME接口会跳过之前所有已经成功了的子步骤，直接从第一个失败的子步骤开始重新执行：

```
PUT http://hostname:port/kylin/api/jobs/{job_uuid}/resume
```

Path Variable
Job_uuid - 必需的，构建任务标识符

由于自动修复的复杂性，触发构建的客户端也可以选择只向管理员发送邮件通知该次失败。Kylin服务器中自带的Web GUI客户端中暂时没有自动修复的逻辑，在遇到构建失败的情况时，Web GUI会根据Cube层面的配置向不同的人员发送构建失败的消息，并且将整个构建任务置于ERROR状态，并等待管理人员重新登录Web GUI查看详情。关于出错时通知方式的配置可以参考第10章。

4.触发构建

首先介绍一下具体的API规范，代码如下：

```
PUT http://hostname:port/kylin/api/Cubes/{CubeName}/rebuild

Path Variable
CubeName - 必需的，Cube 名字
Request Body
startTime - 必需的，长整数类型的起始时间，例如使用 1388563200000 代表起始时间为 2014-01-01
endTime - 必需的，长整数类型的结束时间
buildType - 必需的，构建类型，可能的值为‘BUILD’‘MERGE’和‘REFRESH’，分别对应于新建 Segment、合并多个 Segment，以及刷新某个 Segment
```

举例而言，假设在本地的7070端口启动了Kylin Server，那么可以通过如下的Rest请求申请名为test_kylin_cube_without_slr_empty的Cube，用于增量地构建[2022-01-01，2023-01-01）这个时间段的新Segment：

```
curl -X PUT -H "Authorization: Basic QURNSU46S1lMSU4=" -H "Content-Type: application/json" -d '{"startTime": 1640995200000, "endTime": 1672560000000, "buildType": "BUILD"}' http://localhost:7070/kylin/api/Cubes/test_kylin_cube_without_slr_empty/rebuild
```

如果当前Cube不存在任何Segment，那么可以将startTime设置为0，这样kylin就会自动选择Cube的Partition Start Date（见3.2.2节）作为startTime。如果当前Cube不为空，那么对于BUILD类型的构建任务，请求中的startTime必须等于最后一个Segment的endTime，否则请求会返回500错误。

3.4 管理Cube碎片

增量构建的Cube每天都可能会有新的增量。日积月累，这样的Cube中最终可能包含上百个Segment，这将会导致查询性能受到严重的影响，因为运行时的查询引擎需要聚合多个Segment的结果才能返回正确的查询结果。从存储引擎的角度来说，大量的Segment会带来大量的文件，这些文件会充斥所提供的命名空间，给存储空间的多个模块带来巨大的压力，例如Zookeeper、HDFS Namenode等。因此，有必要采取措施控制Cube中Segment的数量。

另外，有时候用户场景并不能完美地符合增量构建的要求，由于ETL过程存在延迟，数据可能一直在持续地更新，有时候用户不得不在增量更新已经完成后又回过头来刷新过去已经构建好了的增量Segment，对于这些问题，需要在设计Cube的时候提前进行考虑。

3.4.1 合并Segment

Kylin提供了一种简单的机制用于控制Cube中Segment的数量：合并Segments。在Web GUI中选中需要进行Segments合并的Cube，单击Action→Merge，然后在对话框中选中需要合并的Segment，可以同时合并多个Segment，但是这些Segment必须是连续的。单击提交后系统会提交一个类型为“MERGE”的构建任务，它以选中的Segment中的数据作为输入，将这些Segment的数据合并封装成为一个新的Segment（如图3-5所示）。这个新的Segment的起始时间为选中的最早的Segment的起始时间，它的结束时间为选中的最晚的Segment的结束时间。

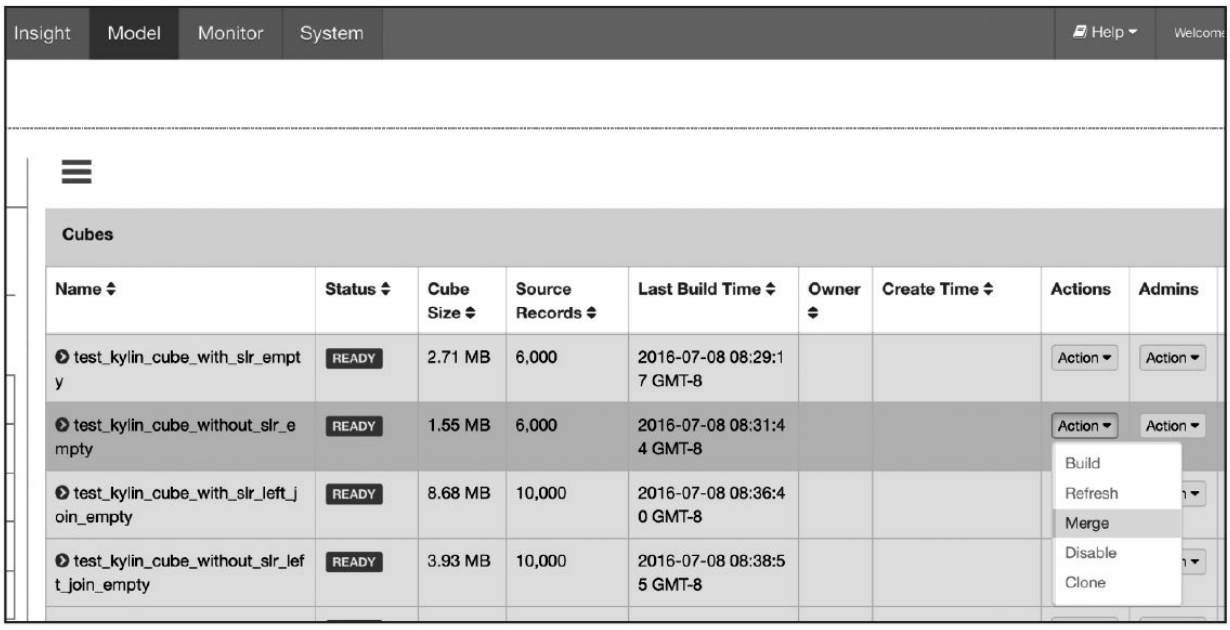


图3-5 合并Segment

在MERGE类型的构建完成之前，系统将不允许提交这个Cube上任何类型的其他构建任务。但是在MERGE构建结束之前，所有选中用来合并的Segment仍然处于可用的状态。当MERGE构建结束的时候，系统将选中合并的Segment替换为新的Segment，而被替换下的Segment等待将被垃圾回收和清理，以节省系统资源。

用户也可以使用Rest接口触发合并Segments，该API在之前的触发增量构建中也已经提到过：

```
PUT http://hostname:port/kylin/api/Cubes/{CubeName}/rebuild

Path Variable
CubeName - 必须的，Cube 名字
Request Body
startTime - 必须的，长整数类型的起始时间，例如使用 1388563200000 代表起始时间为 2014-01-01
endTime - 必须的，长整数类型的结束时间
buildType - 必须的，构建类型，可能的值为 'BUILD' 'MERGE' 和 'REFRESH'，分别对应于新建
Segment、合并多个 Segment，以及刷新某个 Segment
```

我们需要将buildType设置为MERGE，并且将startTime设置为选中的需要合并的最早的Segment的起始时间，将endTime设置为选中的需要合并的最晚的Segment的结束时间。

合并Segment非常简单，但是需要Cube管理员不定期地手动触发合并，尤其是当生产环境中存在大量的Cube时，对每一个Cube单独触发合并操作会变得非常繁琐，因此，Kylin也提供了其他的方式来管理Segment碎片。

3.4.2 自动合并

在3.2.2节中曾提到过，在Cube Designer的“Refresh Settings”的页面中有“Auto Merge Thresholds”和“Retention Threshold”两个设置项可以用来帮助管理Segment碎片。虽然这两项设置还不能完美地解决所有业务场景的需求，但是灵活地搭配使用这两项设置可以大大减少对Segment进行管理的麻烦。

“Auto Merge Thresholds”允许用户设置几个层级的时间阈值，层级越靠后，时间阈值就越大。举例来说，用户可以为一个Cube指定（7天、28天）这样的层级。每当Cube中有新的Segment状态变为READY的时候，就会触发一次系统试图自动合并的尝试。系统首先会尝试最大一级的时间阈值，结合上面的（7天、28天）层级的例子，首先查看是否能将连续的若干个Segment合并成为一个超过28天的大Segment，在挑选连续Segment的过程中，如果遇到已经有个别Segment的时间长度本身已经超过了28天，那么系统会跳过该Segment，从它之后的所有Segment中挑选连续的累积超过28天的Segment。如果满足条件的连续Segment还不能够累积超过28天，那么系统会使用下一个层级的时间阈值重复寻找的过程。每当找到了能够满足条件的连续Segment，系统就会触发一次自动合并Segment的构建任务，在构建任务完成之后，新的Segment被设置为READY状态，自动合并的整套尝试又需要重新再来一遍。

举例来说，如果现在有A~H8个连续的Segment，它们的时间长度分别为28天（A）、7天（B）、1天（C）、1天（D）、1天（E）、1天（F）、1天（G）、1天（H）。此时第9个Segment I加入，它的时间长度为1天，那么现在Cube中总共存在9个Segment。系统首先尝试能否将连续的Segment合并到28天这个阈值上，由于Segment A已经超过28天，它会被排除。接下来的B到H加起来也不足28天，因此第一级的时间阈值无法满足，退一步系统尝试第二级的时间阈值，也就是7天。系统重新扫描所有的Segment，发现A和B已经超过7天，因此跳过它们，接下来发现将Segment C到I合并起来可以达到7天的阈值，因此系统会提交一个合并Segment的构建请求，将Segment C到I合并为一个新的Segment X。X的构建完成之后，Cube中只剩下三个Segment，分别是原来的A（28天），B（7天）和新的X（7天）。由于X的加入，触发了系统重新开始整个合并尝试，但是发现已经没有满足自动合并的条件，既没有连续的、满足条件的、累积超过28天的Segment，也没有连续的、满足条件的、累积超过7天的Segment，尝试终止。

再举一个例子，如果现在有A~J10个连续的Segment，它们的时间长度分别为28天（A）、7天（B）、7天（C）、7天（D）、1天（E）、1天（F）、1天（G）、1天（H）、1天（I）、1天（J）。此时第11个Segment K加入，它的时间长度为1天，那么现在Cube中总共存在11个Segment。系统首先尝试能否将连续的Segment合并到28天这个阈值上，由于Segment A已经超过28天，它会被排除。系统接着从Segment B开始观察，发现若把Segment B至K这10个连续的Segment合并在一起正好可以达到第一级的阈值28天，因此系统提交一

个合并构建任务把B至K合并为一个新的Segment X，最终Cube中存在两个长度均为28天的Segment，依次对应原来的A和新的X。由于X的加入，触发了系统重新开始整个合并尝试，但是发现已经没有满足自动合并的条件，尝试终止。

“Auto Merge Thresholds”的设置非常简单，在Cube Designer的“Refresh Setting”中，单击“Auto Merge Thresholds”右侧的“New Thresholds”按钮，即可在层级的时间阈值中添加一个新的层级，层级一般按照升序进行排列（如图3-6所示）。从前面的介绍中不难得出结论，除非人为地增量构建一个非常大的Segment，自动合并的Cube中，最大的Segment的时间长度等于层级时间阈值中最大的层级。也就是说，如果层级被设置为（7天、28天），那么Cube中最长的Segment也不过是28天，不会出现横跨半年甚至一年的大Segment。

在一些场景中，用户可能更希望系统能以自然日的星期、月、年作为单位进行自动合并，这样在只需要查询个别月份的数据时，就能够只访问该月的Segment，而非两个毗邻的28天长度的Segment。对此，https://issues.apache.org/jira/browse/KYLIN-1865记录了这个问题。

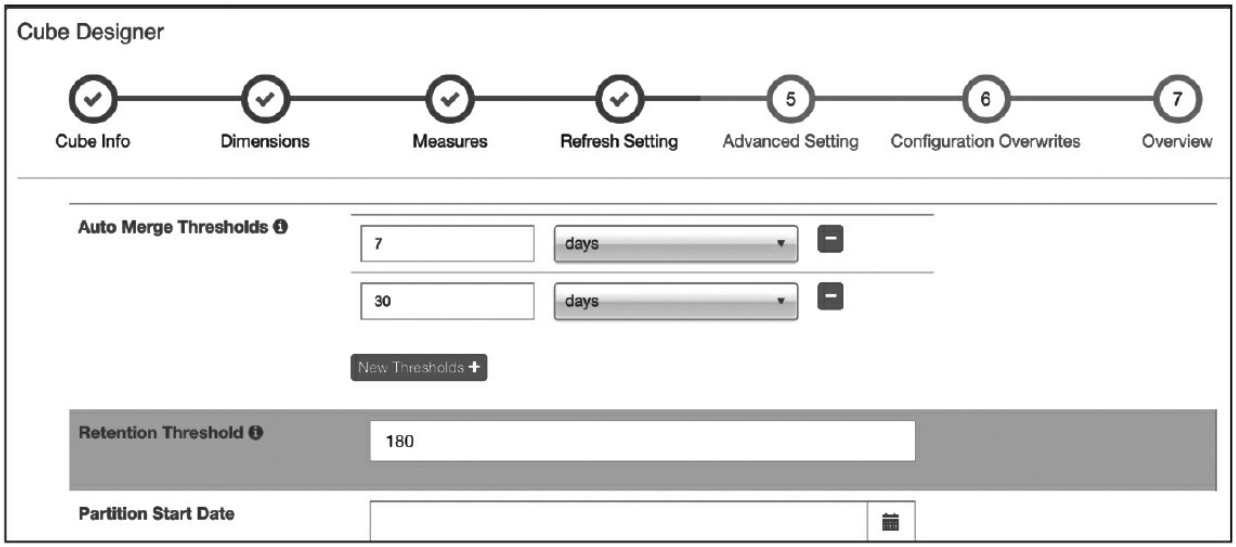


图3-6 设置自动合并阈值

3.4.3 保留Segment

从碎片管理的角度来说，自动合并是将多个Segment合并为一个Segment，以达到清理碎片的目的。保留Segment则是从另外一个角度帮助实现碎片管理，那就是及时清理不再使用的Segment。在很多业务场景中，只会对过去一段时间内的数据进行查询，例如对于某个只显示过去1年数据的报表，支撑它的Cube事实上只需要保留过去一年内的Segment即可。由于数据在Hive中往往已经存在备份，因此无需再在Kylin中备份超过一年的历史数据。

在这种情况下，我们可以将“Retention Threshold”设置为365。每当有新的Segment状态变为READY的时候，系统会检查每一个Segment：如果它的结束时间距离最晚的一个Segment的结束时间已经大于“Retention Threshold”，那么这个Segment将被视为无需保留。系统会自动地从Cube中删除这个Segment。

如果启用了“Auto Merge Thresholds”，那么在使用“Retention Threshold”的时候需要注意，不能将“Auto Merge Thresholds”的最大层级设置得太高。假设我们将“Auto Merge Thresholds”的最大一级设置为1000天，而将“Retention Threshold”设置为365天，那么受到自动合并的影响，新加入的Segment会不断地被自动合并到一个越来越大的Segment之中，

糟糕的是，这会不断地更新这个大Segment的结束时间，从而导致这个大Segment永远不会得到释放。因此，推荐自动合并的最大一级的时间不要超过1年。

3.4.4 数据持续更新

在实际应用场景中，我们常常会遇到这样的问题：由于ETL过程的延迟，业务每天都需要刷新过去N天的Cube数据。举例来说，客户有一个报表每天都需要更新，但是每天的源数据更新不仅包含了当天的新数据，还包括了过去7天内数据的补充。一种比较简单的方法是，每天在Cube中增量构建一个长度为一天的Segment，这样过去7天的数据就会以7个Segment的形式存在于Cube之中。Cube的管理员除了每天要创建一个新的Segment代表当天的新数据（BUILD操作）以外，还需要对代表过去7天的7个Segment进行刷新（REFRESH操作，Web GUI上的操作及Rest API参数与BUILD类似，这里不再详细展开）。这样的方法固然可以奏效，但是每天为每个Cube触发的构建数量太多，容易造成Kylin的任务队列堆积大量未能完成的任务。

上述简单方案的另外一个弊端是，每天一个Segment也会让Cube中迅速地累积大量的Segment，需要Cube管理员手动地对历史已经超过7天的Segment进行合并，期间还必须小心翼翼地，不能错将7天内的Segment一起合并了。举例来说，假设现在有100个Segment，每个Segment代表过去的一天的数据，Segment按照起始时间排序。在合并时，我们只能挑选前面93个Segment进行合并，如果不小心把第94个Segment也一起合并了，那么当我们试图刷新过去7天（94~100）的Segment的时候，会发现为了刷新第

94天的数据，不得不将1~93的数据一并重新计算，因为此时第94天的数据已经和1~93这93天的数据糅合在一个Segment之中了。这对于刷新来说是一种极大的浪费。糟糕的是，即使使用之前所介绍的自动合并的功能，类似的问题也仍然存在，目前为止，还没有一种机制能够有效阻止自动合并试图合并近期N天的Segment，因此使用自动合并仍然有可能将最近N天内的某些Segment与更早的其他Segment合并成一个大的Segment，这个问题将在https://issues.apache.org/jira/browse/KYLIN-1864中获得解决。

目前来说，比较折中的一种方案是不以日为单位创建新的Segment，而是以N天为单位创建新的Segment。举例来说，假设用户每天更新Cube的时候，前面7天的数据都需要更新一下，也就是说，如果今天是01-08，那么用户不仅要添加01-08的新数据，还要同时更新01-01到01-07的数据。在这种情况下，可设置N=7作为最小Segment的长度。在第一天01-01，创建一个新的Segment A，它的时间是从01-01到01-08，我们知道Segment是起始时间闭，结束时间开，因此Segment A的真实长度为7天，也就是01-01到01-07。即使在01-01当天，还没有后面几天的数据，Segment A也能正常地构建，只不过构建出来的Segment其实只有01-01一天的数据而已。从01-02到01-07的每一天，我们都要刷新Segment A，以保证1日到7日的数据保持更新。由于01-01已经是最早的日期，所以不需要对更早的数据进行更新。

到01-08的时候，创建一个新的Segment B，它的时间是从01-08到01-

15。此时我们不仅需要构建Segment B，还需要去刷新Segment A。因为01-01到01-07中的数据在01-08当天仍然可能处于更新状态。在接下来的01-09到01-14，每天刷新A、B两个Segment。等到了01-15这天的时候，首先创建一个新的Segment C，它的时间是从01-15到01-22。在01-15当天，Segment A的数据应当已经被视作最终状态，因为Segment A中的最后一天（01-07）已经不再过去N天的范围之内了。因此此时接下来只需要照顾Segment B和Segment C即可。

由此可以看到，在任意一天内，我们只需要同时照顾两个Segment，第一个Segment主要以刷新近期数据为主，第二个Segment则兼顾了加入新数据与刷新近期数据。这个过程中可能存在少量的多余计算，但是每天多余计算的数据量不会超过N天的数据量。这对于Kylin整体的计算量来说是可以接受的。根据业务场景的不同，N可能是7天，也有可能是30天，我们可以适度地把最小的Segment设置成比N稍微大一点的数字，例如N为7的时候，我们可以设置为10天，这样即使ETL有时候没有能够遵守N=7的约定，也仍然能够刷新足够的数据。值得一提的是，在https://issues.apache.org/jira/browse/KYLIN-1864得到解决之前，我们不要重叠使用自动合并和本节中所描述的处理数据陆续更新的策略。

3.5 小结

增量构建是使用Apache Kylin的关键步骤。因为对于大多数使用场景，数据都是日积月累逐渐增长的。如何合理地安排增量构建，保证用户在Cube中可以及时查询到最新的数据，是Apache Kylin运行维护的日常。第4章将延续本章的内容，继续探讨流式构建，将Apache Kylin的数据延迟缩短到分钟级别。

第4章　流式构建

第3章介绍的增量构建，可用来满足业务的数据更新需求。增量构建和全量构建一样，都需要从Hive中抽取数据，在经过若干轮的Map Reduce作业之后，才能对源数据进行预计算，最后将预计算的结果适配成存储引擎所需要的格式，并导入到存储引擎中。一般来说，增量构建的数据量明显小于全量构建，因此增量构建的时间少于全量构建。但是增量构建仍然无法满足分钟级的实时数据更新需求，其中很大一部分原因是实时数据落地到Hive，再由Kylin触发构建任务，并从Hive中拉取数据这个过程就需要花费大量的时间。另外，尽管它的数据量少于全量构建，但是增量构建的子步骤和全量构建的子步骤相同，调度的成本也不可忽略。因此，为了满足分钟级别的实时数据更新需求，我们不得不寻求其他的数据源和构建引擎，甚至还考虑过单独的存储引擎。流式构建则是应对实时数据更新需求的解决方案。

本章的屏幕截图和具体命令只适用于Apache Kylin v1.5。从v1.6版本开始，流式构建有了较大的变化，但基本设计仍然相通，不过具体的操作和命令已有所不同，请参考Apache Kylin官网的最新文档。

4.1 为什么要流式构建

实时数据更新是一种普遍的需求，快速更新的数据分析能够帮助分析师快速地判断业务的变化趋势，从而能够在风险仍然可控的阶段做出决策。在监控领域，通常需要非常实时的数据更新来抓捕异常的数据特征，这样一来，对数据的延迟需求可能必须是秒级别甚至毫秒级别。Kylin擅长的在线多维分析领域不同于监控领域，虽然普遍存在准实时的更新需求，但是分钟级别的更新与秒级别的更新在业务决策、趋势判断等方面的功能已经十分接近。市场上也已经存在其他的秒级别的在线多维分析产品可供选择，例如Druid（http://druid.io/），但是为了支持秒级的更新需求，该类产品不得不在内存中维护复杂的数据结构以接受实时数据更新（例如Druid中的IncrementalIndex）。这种结构需要系统全局地处理该结构的高可用性和持久化问题等，这会造成系统易用性的下降。更大的问题是在内存中维护所有的数据会对可处理的数据量造成明显的限制（具体详情请参见https://www.quora.com/What-are-the-differences-between-Druid-and-AWS-Redshift）。为了明确自身的定位，保持系统整体的简单易用性，我们认为Kylin瞄准分钟级的更新需求就已经能够满足大部分的实时在线多维分析需求。

由于Kylin不打算自己在内存中维护数据结构以保障实时数据更新，因此Kylin本身无需像Druid一样处理复杂的集群资源调度、容灾容错、数

据扩容等问题。无论是全量构建、增量构建还是流式处理，都有计算引擎的可扩展问题，由于自身并不维护数据，因此Kylin的核心部分可以只关注预计算的优化、查询的优化等核心问题，将计算的扩展性委托给其他的计算框架如Map Reduce、Spark等，将存储的高可用性问题，扩容问题交给其他的存储框架如HBase等。换而言之，Kylin可以更好地复用用户生产环境中已经广泛部署的其他组件，更好地融入Hadoop、Spark这样的生态圈之中。这样不仅节省了用户在基础设施上的投入，也节约了运维的管理成本，最主要的是使得Kylin产品本身非常灵活，而且容易部署。

4.2 准备流式数据

4.2.1 数据格式

由于实时数据更新频繁，因此对于流式构建，不再要求数据必须提前落地到Hive之中，因为那样做开销过大。Kylin假设在流式构建中，数据是以消息流的形式传递给流式构建引擎的。消息流中的每条消息需要包含如下信息。

·所有的维度信息。

·所有的度量信息。

·业务时间戳。

在消息流中，每条消息中的数据结构应该相同，并且可以用同一个分析器实例将每条消息中的维度、度量及时间戳信息提取出来。目前默认的分析器为org.apache.kylin.source.kafka.TimedJsonStreamParser，该分析器假设每条消息为一个单独的JSON，所有的信息都以键值对的形式保存在该JSON之中。它还假设键值对中存在一个特殊的键值代表消息的业务时间，该键值称为业务时间戳。该键值的键名是可配的，其值为长整数的Unix Time时间类型。

业务时间戳的粒度对于在线多维分析而言可能过高，无法在时间维度上完成深度的聚合。因此，Kylin允许用户挑选一些从业务时间戳上衍生出来的时间维度（Derived Time Dimension），具体来说有如下几种。

·minute_start：业务时间戳所在的分钟起始时间，类型为Timestamp（yyyy-MM-dd HH：mm：ss）。

·hour_start：业务时间戳所在的小时起始时间，类型为Timestamp（yyyy-MM-dd HH：mm：ss）。

·day_start：业务时间戳所在的天起始时间，类型为Date（yyyy-MM-dd）。

·week_start：业务时间戳所在的周起始时间，类型为Date（yyyy-MM-dd）。

·month_start：业务时间戳所在的月起始时间，类型为Date（yyyy-MM-dd）。

·quarter_start：业务时间戳所在的季度起始时间，类型为Date（yyyy-MM-dd）。

·year_start：业务时间戳所在的年起始时间，类型为Date（yyyy-MM-dd）。

这些衍生时间维度都是可选的，如果用户选择了这些衍生维度，那么在对应的时间粒度上进行聚合时就能够获得更好的查询性能，一般来说不推荐把原始的业务时间戳选择成一个单独的维度，因为该列的基数一般都是很大的。

4.2.2 消息队列

由于Kafka（http://kafka.apache.org/）的性能表现出色，且具有高可用性和可扩展性，因此被广泛地选择为实时消息队列。尽管Kafka之于Kylin是一个类似于Hive的可扩展组件，理论上也存在一些其他消息队列可作为流式构建的数据源，但是因为Kafka的流行程度，它已经成为Kylin事实上的流式构建消息队列标准。Kafka提供了两套读取访问接口：高层读取接口（High Level Consumer API）和底层读取接口（Simple Consumer API），由于需要直接控制读取队列的偏移量（Offset），因此这里选择了底层的读取接口。

流式构建的用户需要使用Kafka的Producer将数据源源不断地加入某个Topic中，并且将Kafka的一些基本信息（例如Broker节点信息和Topic名称）告知流式构建任务。流式构建任务在启动的时候会启动Kafka客户端，然后根据配置向Kafka集群读取相应的Topic中的消息，并进行预处理计算。

4.2.3 创建Schema

由于Kylin对查询客户端暴露的是ANSI SQL接口，因此用户最终将以SQL接口查询流式构建的数据。对于全量构建和增量构建，它们的源数据都是Hive中的某些表，因此Kylin可以从Hive中导入这些表的Schema，用户在进行查询的时候也可以像直接查询Hive表一样使用相应的SQL语法。但是流式构建的问题是，数据是以键值对的形式传入消息队列的，并不像Hive一样存在可以用来导入的表定义。但是由于消息队列中的键值对是基本固定的，甚至包括衍生时间维度，一经选择也变成是固定的，因此可以创造一个虚拟的表，用表中的各个列来对应消息队列中的维度、度量及衍生时间维度。在查询时，用户可以直接对这张虚拟的表发起各种SQL查询，就好像这张表真实存在一样。

在Kylin的Web GUI上，选择Model页面，单击“Data Source”，可以找到“Add Streaming Table”的按钮，这就是用来为流式构建创造虚拟表的入口，如图4-1所示。

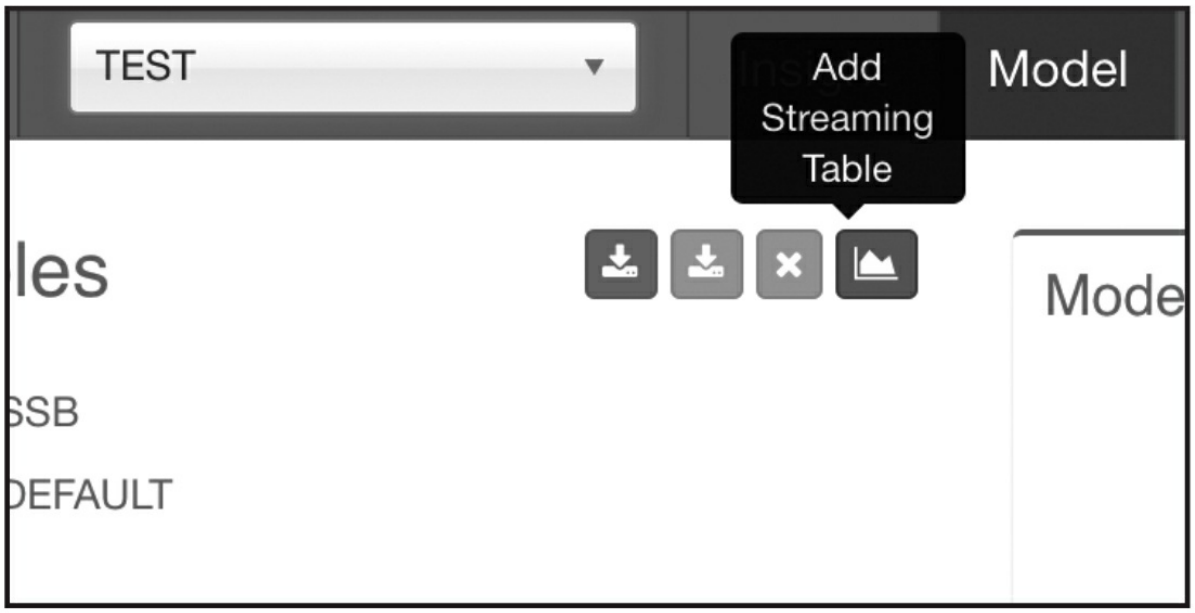


图4-1 添加Streaming Table

此时Web GUI会弹出向导对话框，以帮助用户完成虚拟表的创建，如图4-2所示。

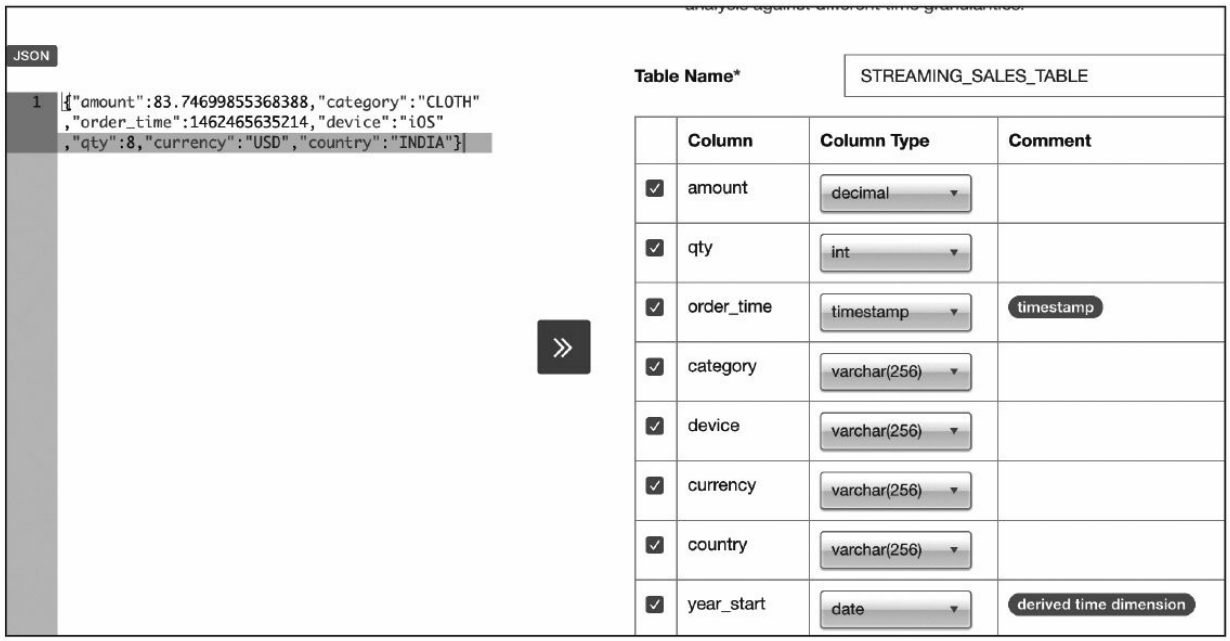


图4-2 创建Streaming Table

在图4-2左侧的数据框中，我们需要输入消息队列中的一段数据样本，数据样本可以是用户消息队列中的任意一条，但是需要保证想要被收录进虚拟表的键值对都应该出现在该数据样本之中。例如输入一段随意的样本：

```
{"amount":83.74699855368388,"category":"CLOTH","order_time":1462465635214,"device":"iOS","qty":8,"currency":"USD","country":"INDIA"}
```

单击中间的“>>”按钮，系统会自动地分析JSON中的键值对，将它们转化成虚拟表中的列。在右侧的虚拟表区域中，用户首先需要给虚拟表起一个名字，这个名字将在后续的SQL查询中被用到。表名的下方是表中各个可能的列名称及类型。虚拟表必须有至少一个Timestamp类型的列，而且该列必须是一个长整数类型。如果虚拟表中有多个Timestamp类型的列，那也不会有问题，后续的向导会要求从中选择一个作为真正的业务时间戳。JSON中其他的键值对会被视为维度和度量自动加入到虚拟表中，用户可以勾选掉虚拟表中不需要的列，也可以调整它们的数据类型。最后，在虚拟表的最下方有各种可供选择的衍生时间维度，用户可以根据业务的需求选择有用的时间维度并将其放入虚拟表中。在虚拟表层面，衍生时间维度和其他维度是同等的。

单击向导对话框右下方的Next，接下来的对话框会引导用户输入与这个虚拟表相关联的Kafka消息队列。有了相应的关联，后续再有Cube使

用这张虚拟表的时候，我们就能够知道数据需要从哪里获取了，它会从Kafka消息队列而非默认地从Hive中获取。在对话框中输入Kafka的Topic名称，接着在Cluster选项卡中添加Kafka Broker的主机名和端口（如图4-3所示）。

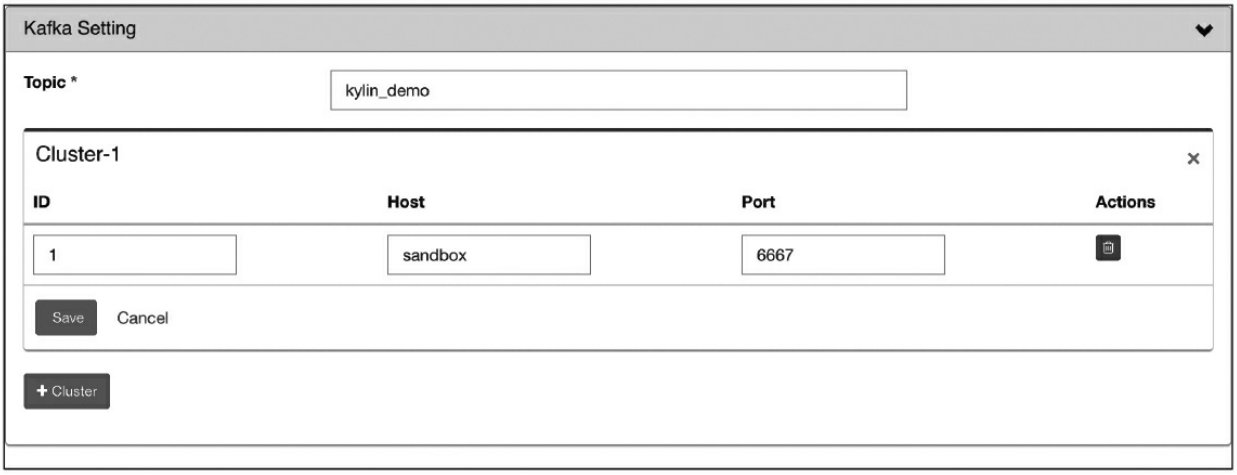


图4-3 关联Kafka Topic

同一对话框下方的高级设置中（如图4-4所示），有如下三个参数可供配置，选择默认情况。

·Timeout：可配置的，Kafka客户端读取超时时间。

·Buffer Size：可配置的，Kafka客户端读取缓冲区大小。

·Margin：可配置的，代表消息可能延迟的程度，具体说明参见后文。

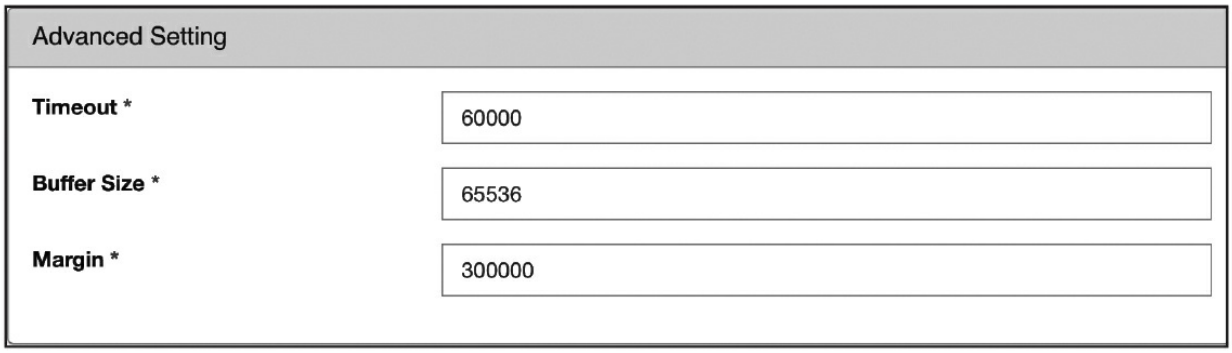


图4-4 Kafka高级设置

最后的分析器设置中允许用户对消息分析器做一定的配置，用户甚至还可以定制自己的分析器。一般情况下，默认的TimedJsonStreamParser分析器已经足够使用。如果虚拟表中存在多个Timestamp类型的列，那么用户需要告诉分析器使用哪个Timestamp列作为业务时间戳。打开第二项的下拉菜单即可完成选择（如图4-5所示）。如果虚拟表中只存在一个Timestamp类型的列，则无须选择。默认情况下用户不用修改Parser Name和Parser Properties。

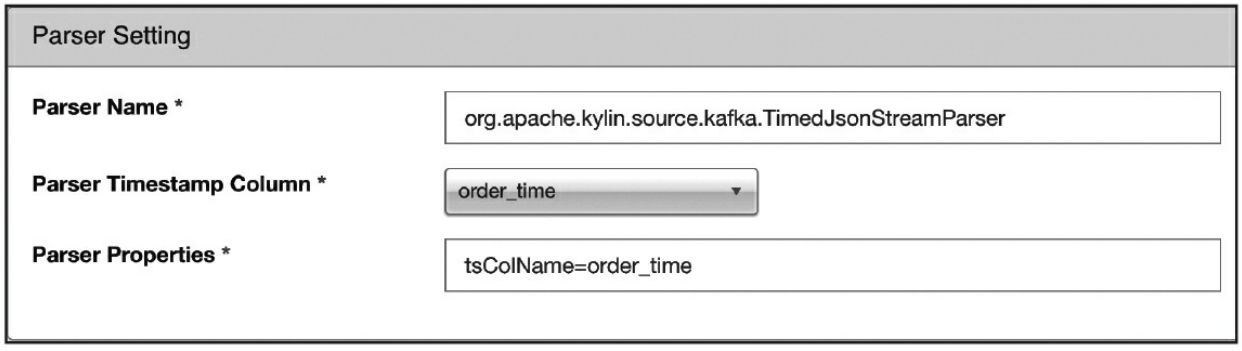


图4-5 配置消息分析器

最后单击Submit按钮，一个与Kafka消息队列数据源关联成功的流式

构建虚拟表就创建成功了。我们可以像查看从Hive中导入的表一样，到Model→Data Source中查看这个虚拟表，如图4-6所示。

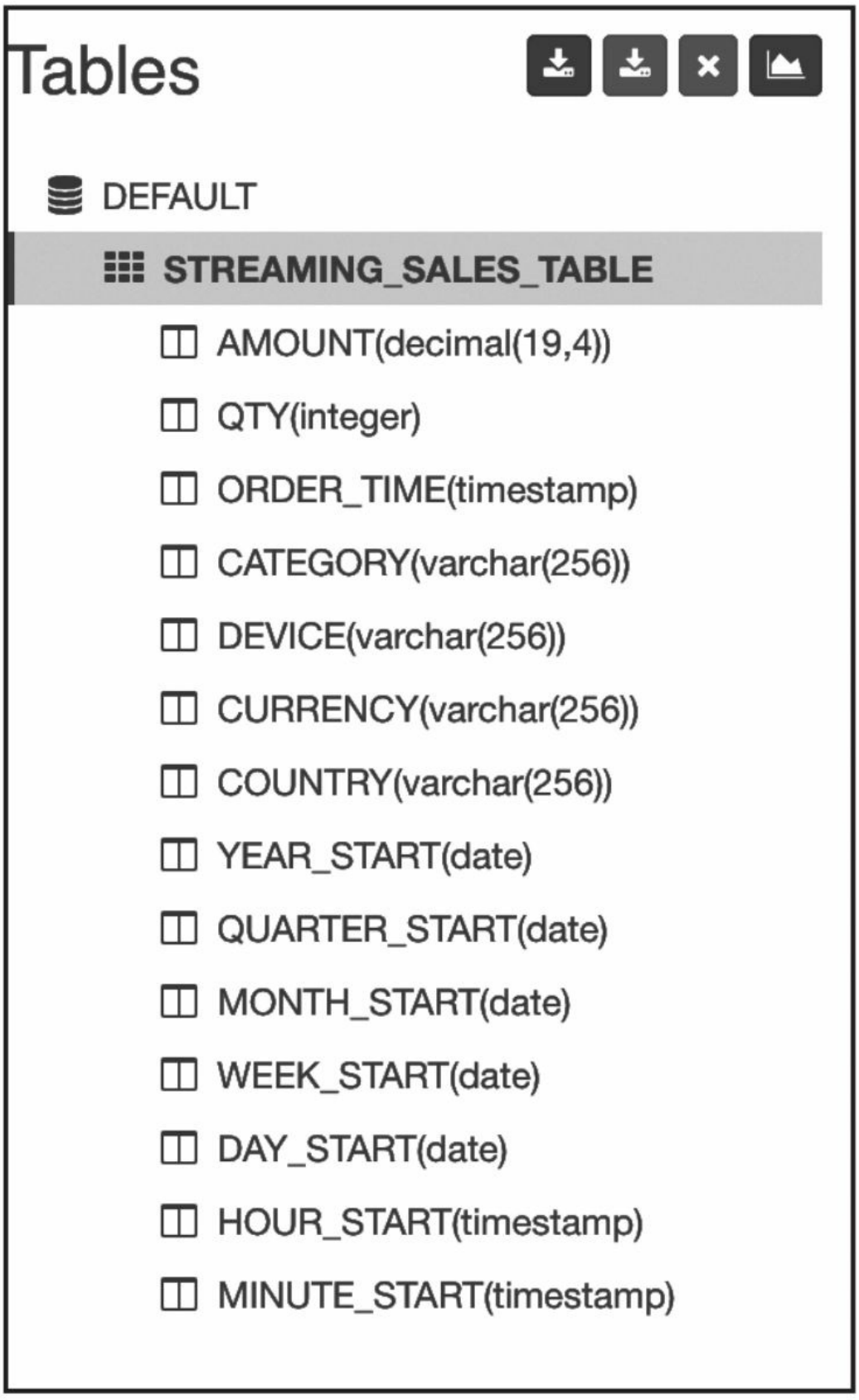

Tables
DEFAULT
STREAMING_SALES_TABLE
AMOUNT(decimal(19,4))
QTY(integer)
ORDER_TIME(timestamp)
CATEGORY(varchar(256))
DEVICE(varchar(256))
CURRENCY(varchar(256))
COUNTRY(varchar(256))
YEAR_START(date)
QUARTER_START(date)
MONTH_START(date)
WEEK_START(date)
DAY_START(date)
HOUR_START(timestamp)
MINUTE_START(timestamp)

图4-6 查看Kafka虚拟表

4.3 设计流式Cube

4.3.1 创建Model

和增量构建的流程一样，我们也要为流式构建的Cube创建数据模型（Model）。关于创建数据模型的一些细节内容已经在第2章中进行过介绍，在此不再赘述，这里只介绍与流式构建相关的配置项。

在创建Model对话框的第三步（如图4-7所示，创建Model的具体步骤详见2.3节）维度选择时，我们既可以选择普通的维度，又可以选择衍生的时间维度。注意，一般不推荐直接选择业务时间戳作为维度，因为业务时间戳的精度往往是精确到秒级甚至是毫秒级的，使用它作为一个维度失去了聚合的意义，也会让整个Cube的体积变得非常庞大。

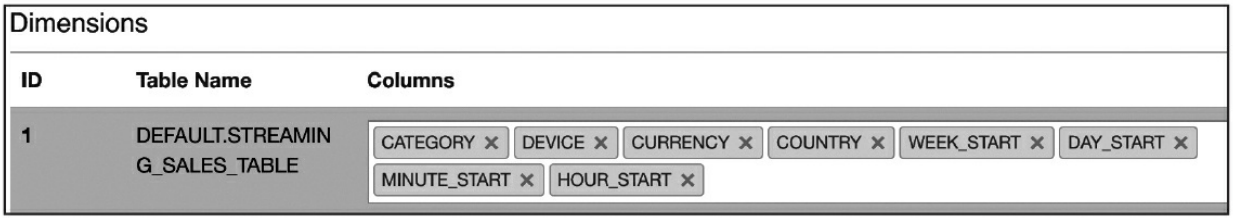


图4-7 创建Model对话框的第三步，选择维度

在创建Model对话框的第五步设置中，一般选择最小粒度的衍生时间维度作为分割时间列，在这里我们选择MINUTE_START，它的数据格

式前文也已经介绍过，即yyyy-MM-dd HH:mm:ss。有了分割时间列，就可以对Cube进行分钟级的流式构建了，其余设置均保持默认状态（如图4-8所示）。

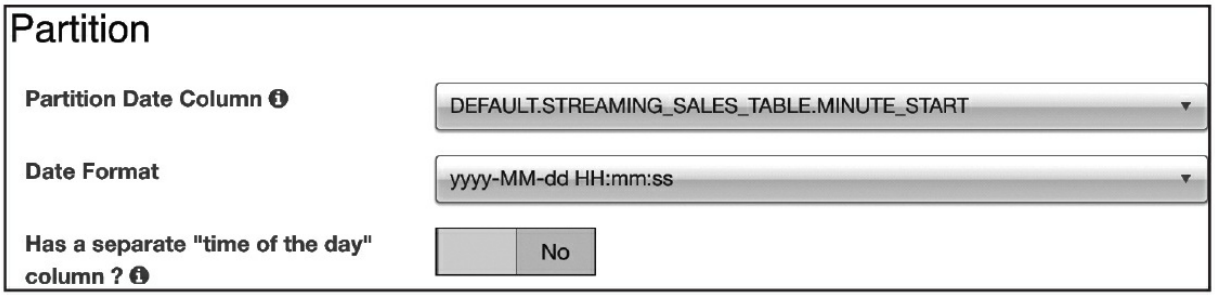


图4-8 创建Model的第五步，配置时间分割列

最终，单击“Save”按钮，保存所创建的数据模型。当看到成功提示时，数据模型就创建成功了。

4.3.2 创建Cube

接下来，基于创建好的数据模型开始在Kylin中创建流式构建的Cube。创建Cube的说明在第2章中也有详细描述，此处不再赘述，这里只介绍与流式构建相关的部分。

创建Cube对话框的第二步是为Cube添加维度，但因为只有一个事实表，所以所有的维度都是普通类型（Normal）（如图4-9所示）。

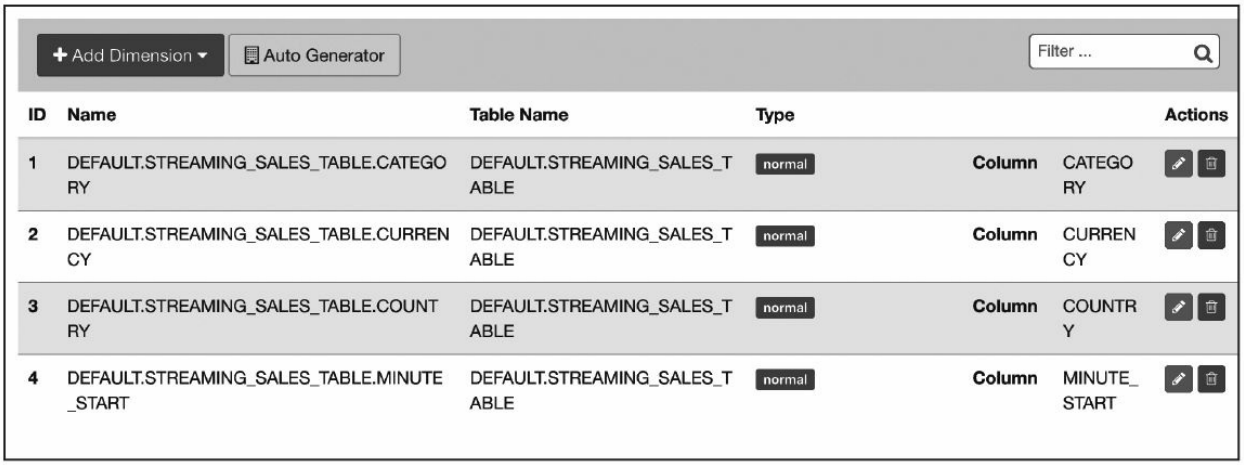


图4-9 选择Cube的维度

创建Cube的第四步是设置Cube的自动合并时间。因为流式构建需要频繁地构建较小的Segment，为了不对存储器造成过大的压力，同时也为了获取较好的查询性能，因此需要通过自动合并将已有的多个小Segment合并成一个较大的Segment。所以，这里将设置一个层级的自动合并时间：0.5小时、4小时、1天、7天、28天。此外，设置保留时间为30天（如图4-10所

示)。

在第五步的Aggregation Groups设置中，可以把衍生时间维度设置为Hierarchy关系，设置的方法和普通维度一样。在RowKeys部分，也可以像调整普通维度的顺序一样合理地调整衍生时间维度(如图4-11所示)。

图4-10 设置Cube的自动合并

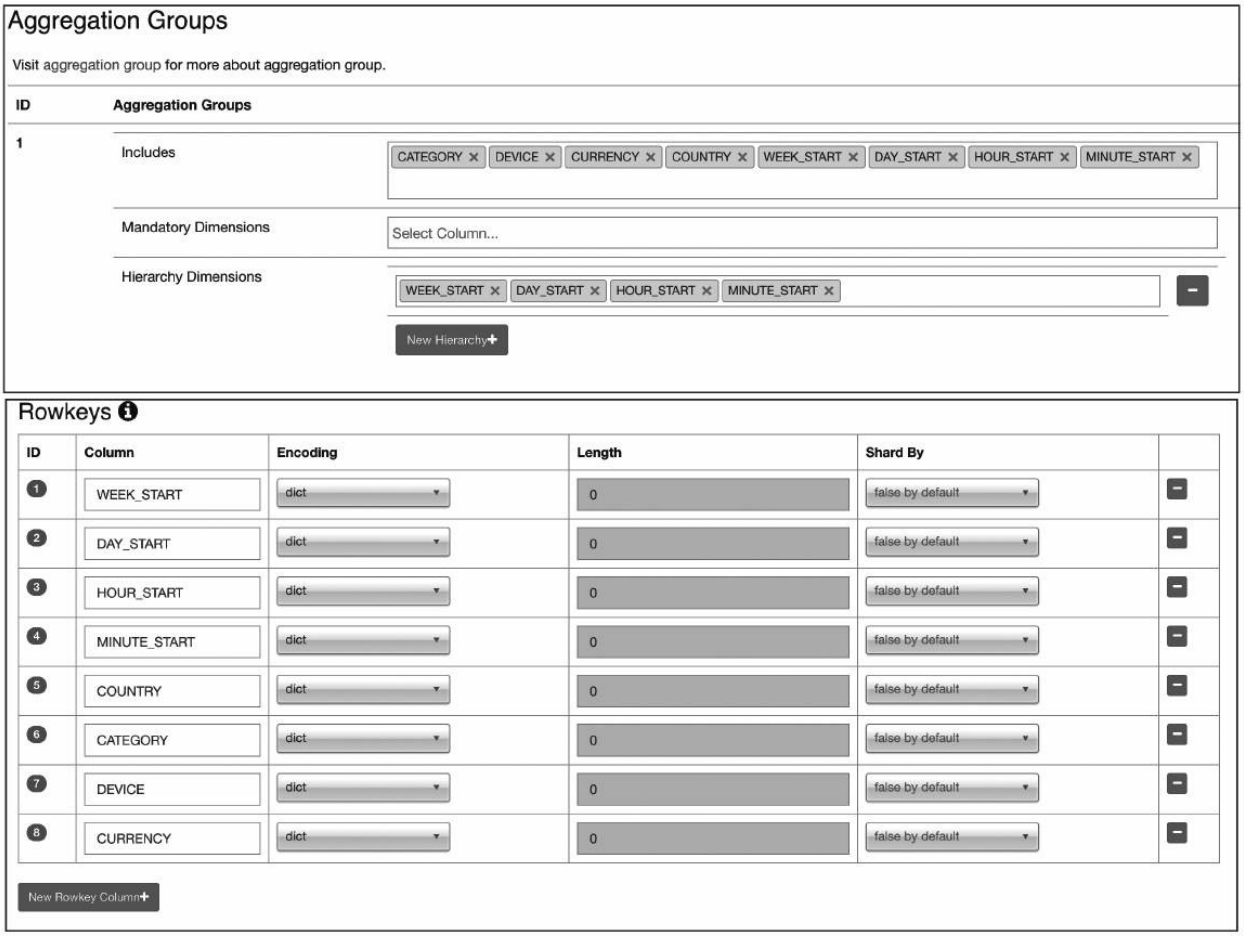


图4-11 设置Aggregation Group

最终，单击“Save”保存Cube，当看到成功提示时，一个流式构建的Cube就创建完成了。

4.4 流式构建原理

由于分布式网络存在延迟等因素，因此从消息队列中取出的消息并不一定必须要按照业务时间升序排列。事实上，Kylin假设消息队列中的所有消息均按照业务时间呈基本递增的趋势，图4-12描绘了基本的情况，其中x轴代表消息的序号，y轴代表消息中的业务时间。

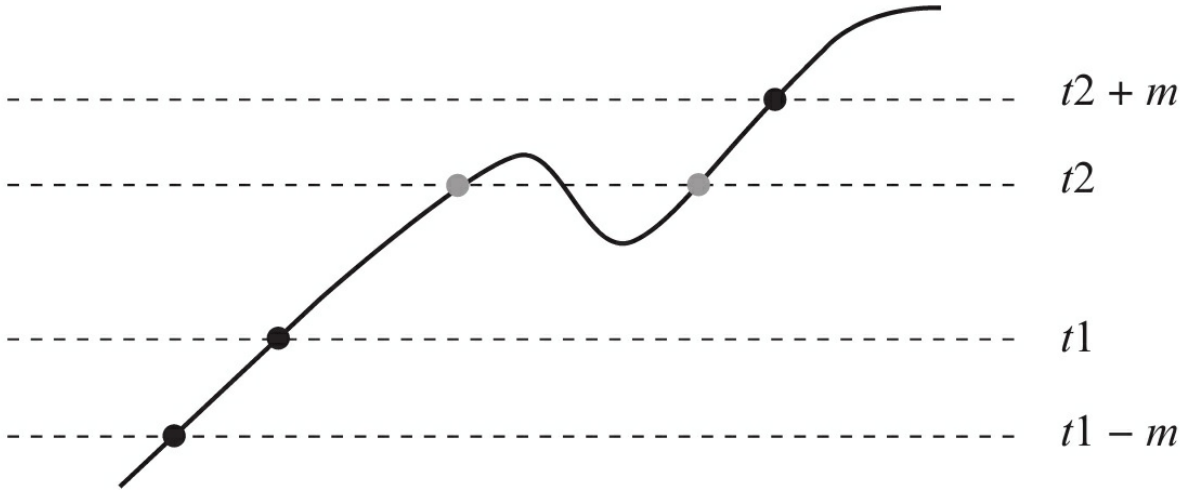


图4-12 消息的时序情况

流式构建需要达到分钟级的数据更新频率，Kylin的做法是每隔数分钟就启动一次微构建，用于处理最新的一批数据。这种做法的理念有一些类似于Spark Streaming，它们也是将流数据视作一种特殊的微批次来处理的。由于消息可能存在延迟，因此不能为某一时刻刚刚过去的那几分钟立刻构建微批次。举例来说，如果在每个微构建中要处理5分钟的增量数据，假设消息队列中的消息最多可能有10分钟的延迟（对应于4.2.3节中

的“Margin”)，那么就不能在1:00的时候立刻尝试去构建0:55到1:00这5分钟的数据，因为这部分数据的消息最迟可能在1:10分才会到齐，否则构建出来的Segment就存在很大的遗漏数据的风险。此时，需要像增量构建中提到的“数据持续更新”的情形一样，对过往的Segment进行刷新操作。但是目前流式构建并不支持Segment刷新操作，所以，最早只能在1:10开始构建0:55到1:00这部分的数据。也就是说即使构建在瞬间完成，最早也得在1:10才能在查询端看到0:55的数据。这中间的延迟我们称之为DELAY，它等于每个微构建批次的时间(INTERVAL)加上消息最长可能延迟的时间(MARGIN)，在上面的示例中，DELAY为10分钟+5分钟=15分钟。

弄清楚了延迟的概念，接下来我们假设已经到了1:10分，现在想要开始构建0:55到1:00这段时间的Segment。我们需要的输入是所有业务时间戳处于这个时间范围内的消息，但是由于消息队列并不是按照时间顺序来排序的，因此无法精确地知道应该获取消息队列中哪部分的消息。消息队列可能存储着过去相当长一段时间内的数据，如果每次流式构建都去扫描整个消息队列中的所有消息显然不符合实际。

为了尽量不丢失数据，这里采用了一种变种的二分查找法来定位所需要读取的消息队列的起始序号和结束序号。假设现在需要获取处于t1和t2之间的所有消息，如果使用简单的二分查找法，那么定位t2的时候可能就会找到图4-12中左侧的红点，这样一来，两个红点之间处于t1和t2之

间的那部分数据就会被遗失。在变种二分查找法中，会寻找业务时间戳等于t1-margin和t2+margin这两个时间点的消息的序列号，然后读取这两个消息序列号之间的所有消息。通常，只要数据能够保证在Margin时间内到达，流式构建就不会丢失数据。但是，通常情况下数据产生方无法保证100%的消息都会在Margin时间内到达，因此，理论上目前的流式构建仍然存在丢失数据的风险。因此，合理地设置Margin显得非常重要，如果Margin设置得过小，那么数据丢失的可能性就会大大增加。但是如果Margin过大，又会导致DELAY增加，那样在客户端就会明显感觉到数据的Time to Market延迟增加。

流式构建有数据量小、速度要求高的特点，前文也提到了流式构建区别于全量构建和增量构建，不从Hive中获取数据，因此，不能再沿用全量构建和增量构建中的构建引擎。作为流式构建的一个初级版本，目前提供了一个独立进程的内存流式构建引擎。它的工作方式如图4-13所示。调度器每隔一段时间(INTERVAL)就触发流式构建引擎开始工作，目的是构建一个新的Segment。在启动后，流式构建引擎会按照本节中所描述的方法，从消息队列中提取出一部分的消息。这些消息中有一些并不属于当前工作的Segment，因此需要一个过滤器将不需要的消息进行过滤。随后进入深层次的预计算中，流式构建引擎根据配置好的消息分析器，从每个消息中提取出有用的维度和度量，并结合配置从业务时间戳上衍生计算出所有需要的衍生时间维度。这些数据将按照Cube的设计被预计算，结果会被封装成一个新的Segment存到存储引擎中。

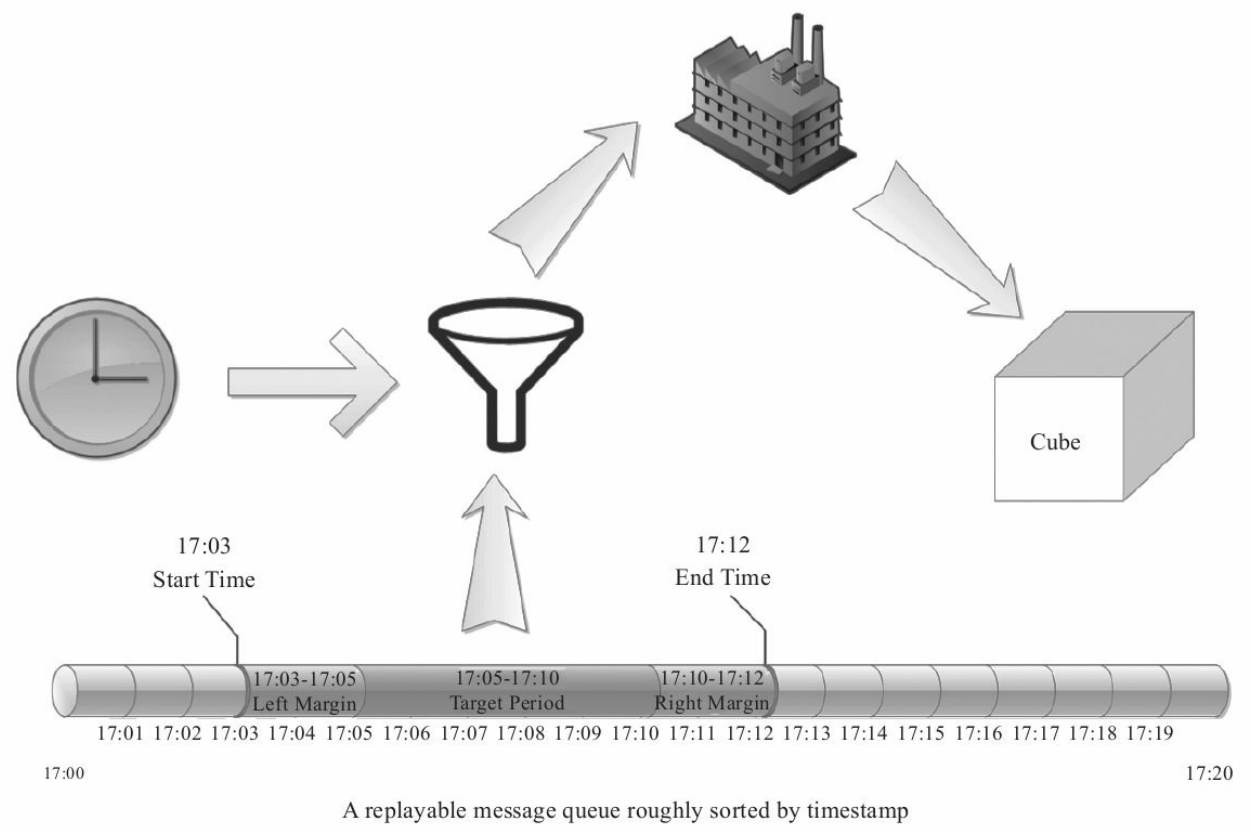


图4-13 流式构建引擎工作过程

4.5 触发流式构建

4.5.1 单次触发

Kylin目前暂时不支持从Web GUI上触发流式构建。为了触发一次流式构建，用户需要在一台能够访问Kafka集群和存储引擎的机器上执行以下的命令：

```
$KYLIN_HOME/bin/streaming_build.sh CUBE_NAME INTERVAL DELAY
```

其中的参数及说明分别如下。

·CUBE_NAME：代表所需流式构建的Cube。

·INTERVAL：代表此次流式构建的时间长度。

·DEALY：代表构建多久之前的数据，一般来说至少要设置为INTERVAL+MARGIN。

举例来说，假设STREAMING_CUBE是一个流式构建的Cube，每隔5分钟构建一次，而且它的Kafka数据源的Margin是10分钟，那么我们每隔5分钟需要调用如下命令：

```
$KYLIN_HOME/bin/streaming_build.sh STREAMING_CUBE 300000 900000
streaming started name: STREAMING_CUBE id: 1462471500000_1462471800000
```

构建任务的日志保存在$KYLIN_HOME/logs目录下，以JOB ID命名，如streaming_STREAMING_CUBE_1462471500000_1462471800000.log。待任务完成之后，可以在Monitor页面查看执行结果。

任务执行成功之后，需要手动启用该Cube。即在Cube列表中找到该Cube，单击右侧Actions按钮，并选择Enable。该操作只需要执行一次，在Enable之后，用户就可以像查询普通的Cube一样，在Web GUI上或使用查询Rest API进行SQL查询了。查询的过程和普通的增量构建是一样的，用户只需要根据左侧列出的表和列的信息并结合Cube上维度和度量的定义编写SQL语句即可。

这里给出一个SQL语句的例子，用户可以自行在SQL输入框中进行执行和测试：

```
    select minute_start, count(*), sum(amount), sum(qty) from streaming_sales_table
group by minute_start
```

以下是执行的结果，如图4-14所示。

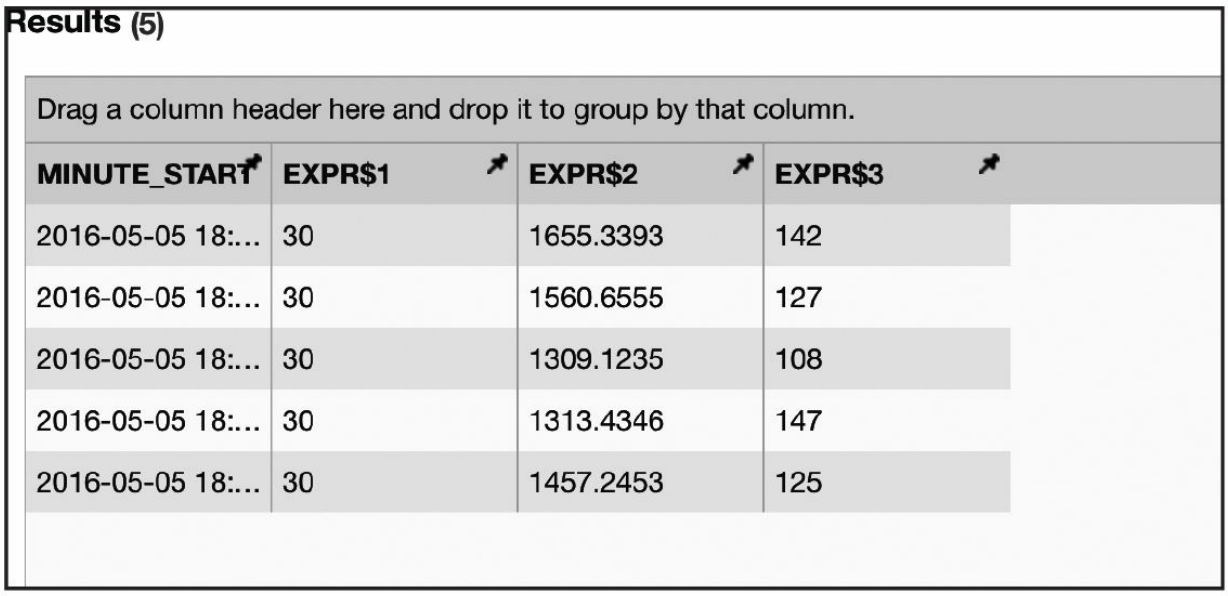


图4-14 流式构建Cube的查询结果

4.5.2 自动化多次触发

如果流式构建特别频繁，比如每5分钟构建一次，那么一天就要构建数百次，这样一来，每次都进行手动的触发显然就不切实际了。由于目前单次触发可以通过命令行的方式来执行，因此用户可以结合自己喜欢的调度工具来完成自动化的触发。在这里我们将介绍基于Cron Job的一种方案。

首先，由于用来构建的命令行脚本需要环境变量KYLIN_HOME，因此我们首先在bash_prof？ile中Export该变量：

```
    vi ~/.bash_profile
## add the KYLIN_HOME here
export KYLIN_HOME="/root/apache-Kylin-1.5.3-bin"
```

随后，添加一个Cron Job：

```
crontab -e
*/5 * * * * sh $KYLIN_HOME/bin/streaming_build.sh STREAMING_CUBE 300000 900000
```

于是每隔5分钟，CRON就会触发一个流式构建引擎实例来构建一段新的Segment。注意，如果命令的INTERVAL不是5分钟，那么Cron Job的定义也需要做相应的调整。

在之前的4.3.2节中就已经引导用户在创建流式构建Cube的时候就设

置好自动合并了。随着5分钟的Segment不断地堆积，自动合并也会被触发，Kylin会使用增量构建中的合并构建把小Segment陆续合并成大Segment，以保证查询性能。对于合并操作，Kylin使用的是与增量构建中相同的方式进行合并，而不再依赖于特殊的流式构建引擎。

4.5.3 出错处理

1.流式构建出错

如果出现Kafka读取出错的情况，那么可能会导致某个单次的微构建失败。这个失败的构建会导致Cube中缺少相应的Segment，在连续的Segment中形成一个“Gap”。Gap会阻碍自动合并的进行，而且会导致用户查询的时候得到的数据不完整。因此遇到微构建失败时，需要去查看相应的日志排除问题，并尽快将Gap补回来。Kylin提供了另外的命令行工具帮助弥补丢失的Segment，其语法如下：

```
$KYLIN_HOME/bin/streaming_fillgap.sh CUBE_NAME

CUBE_NAME 代表所需流式构建的 Cube
```

2.合并出错

如果出现自动合并的构建出错的情况，那么它会阻碍所有其他的在这个Cube上的合并操作，因为一个Cube同时只允许有一个未完成的构建操作。在流式构建中，如果自动合并停止，那么新产生的Segment会迅速地堆积，Cube的查询性能就会迅速下降。因此，每当合并构建出错时，管理员需要立刻到Web GUI查看合并失败的原因，排除故障并在Web GUI中恢复该合并构建。合并构建的失败往往与流式构建本身没有直接的关

系，因为合并不是流式构建引擎的专有功能。出错的原因往往和增量构建一样，出在Hadoop集群本身的问题上。

4.6 小结

总的来说，目前的流式构建基于增量构建的整体框架，使用了一个特殊的流式构建引擎，可从消息队列中迅速地获取数据，并把源数据预计算成为Segment。流式构建和增量构建大体相同，主要区别之处在于数据源不同，前者的数据源是Kafka这样的消息队列，而后者的数据源是Hive这样的数据仓库。另外一个不同之处在于，增量构建是由MapReduce作业来产生Cube的HDFS数据文件的，它会使用MapReduce将HDFS数据文件转化为符合存储引擎（HBase）的数据格式（HFile），然而在流式构建中，HDFS数据文件并不是由MapReduce产生的，而是由一个单进程的流式构建引擎独立完成的。因此当数据量变大的时候，整个系统的出错频率可能会增加。

目前，流式构建引擎的设计还比较初级。当单个微构建的数据量过大时，现有的单进程流式构建引擎可能会因为内存溢出而崩溃。同时，出错恢复的过程也会相对繁琐，可能需要在运维上花费更多的精力。另外目前版本的流式构建也不能提供对消息处理Exact-Once的语义，如果数据的延迟超过了预计的Margin，那么可能会存在丢失数据的风险。这些问题将在今后的版本中尽量得到逐个解决，敬请期待。

第5章　查询和可视化

基于之前的讲解，相信读者已经创建了自己的模型、立方体，并且顺利对其进行构建。在构建完成之后，我们再回到最初的目标——查询数据，这里首先会讲解Kylin自己的查询页面，再介绍如何通过Rest API、JDBC、ODBC或其他工具来访问Kylin。希望本章的内容能够帮助读者了解Kylin的查询和可视化页面，更全面地认识如何通过编程接口开发基于Kylin的可视化界面。

5.1 Web GUI

Apache Kylin的Insight页面即为查询页面（如图5-1所示），单击该页面，左边侧栏会将所有可以查询的表列出来，当然，这些表需要在Cube构建好以后才会显示出来。

图5-1 选择Insight页面

5.1.1 查询

提供一个输入框输入SQL，单击提交即可查询结果。在输入框的右下角有一个Limit字段，用来保护Kylin不会返回超大结果集，拖垮浏览器（或其他客户端）。如果SQL中没有Limit子句，那么这里默认会拼接上limit50000；如果SQL中有Limit子句，那么这里将以SQL中的为准。假如用户想去掉Limit限制，可以在SQL中不加Limit的同时将右下角的LIMIT输入框中的值也改为0（如图5-2所示）。

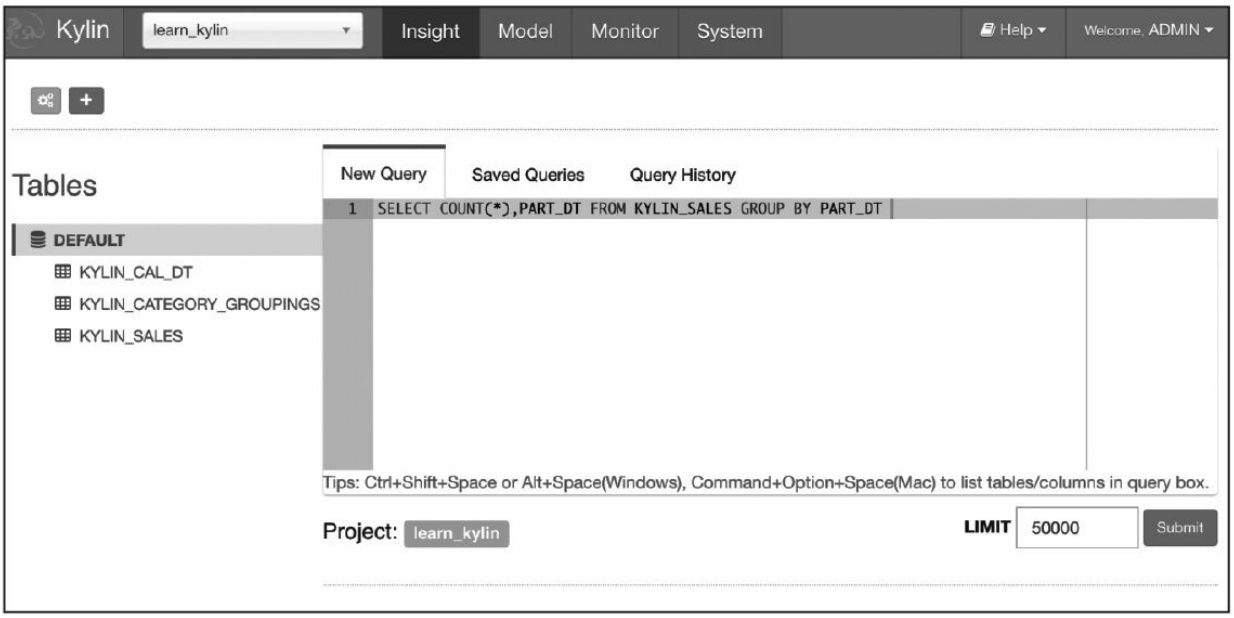


图5-2 查询页面

如果SQL因为Limit限制没有返回所有结果，那么前台会显示一个字样的按钮，单击后即可再次查询，并获取所有结果集（如图5-3所示）。

图5-3 “Show All”返回所有结果集

5.1.2 显示结果

1.表格形式展现结果

对于上面的查询，默认会以表格的形式显示结果，如果需要以图标的形式展示数据，则可单击表格右上角的Visualization按钮（如图5-4所示）。

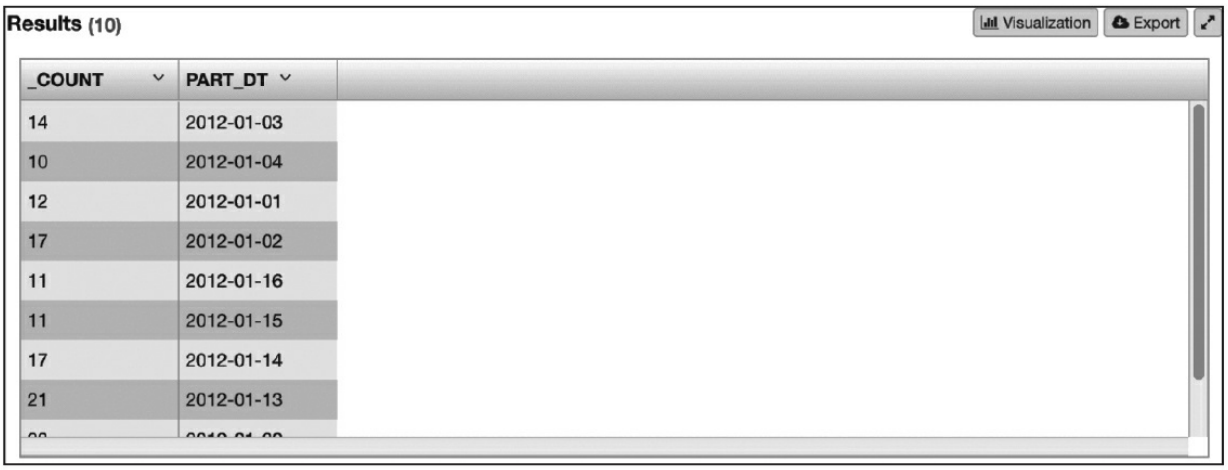


图5-4 表格展示结果集

2.图形化显示结果

若要以图形化来显示结果，那么前端图形化需要支持折线图（Line）、柱状图（Bar）、饼图（Pie）这三种类型（如图5-5、图5-6、图5-7所示）。这三种图形是比较常见的数据展示图，折线图可以展现数据在不同时间内的变化趋势，柱状图可以展示数据在不同条件下的对比情况，饼图可以较

好地展现数据在全局所占的比例大小。

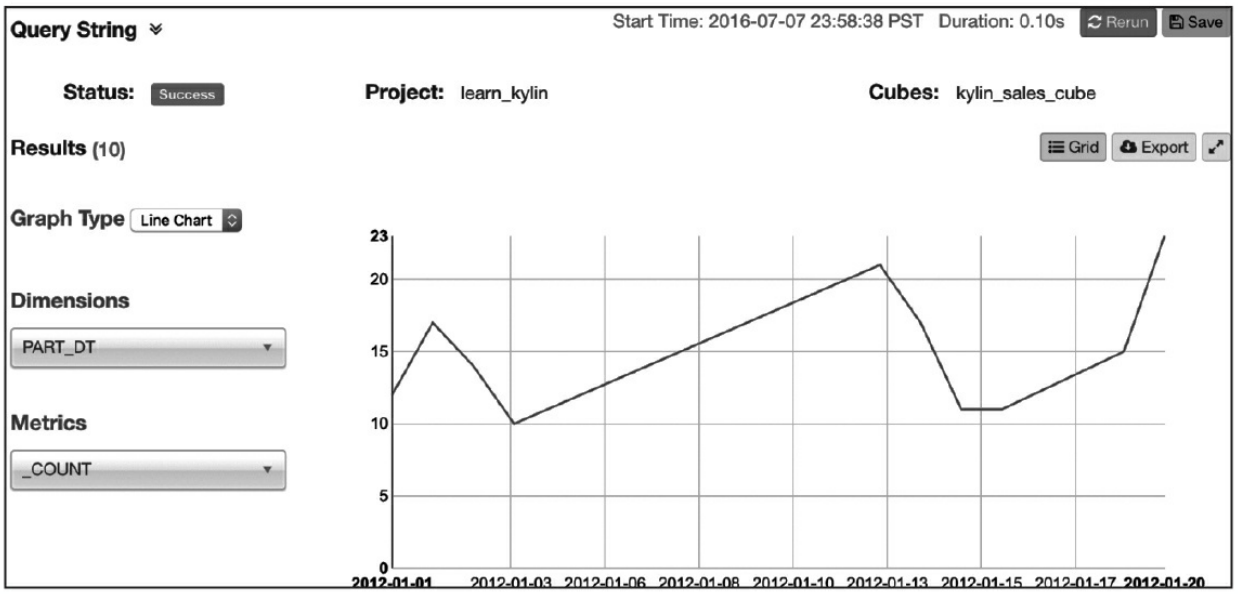


图5-5 折线图

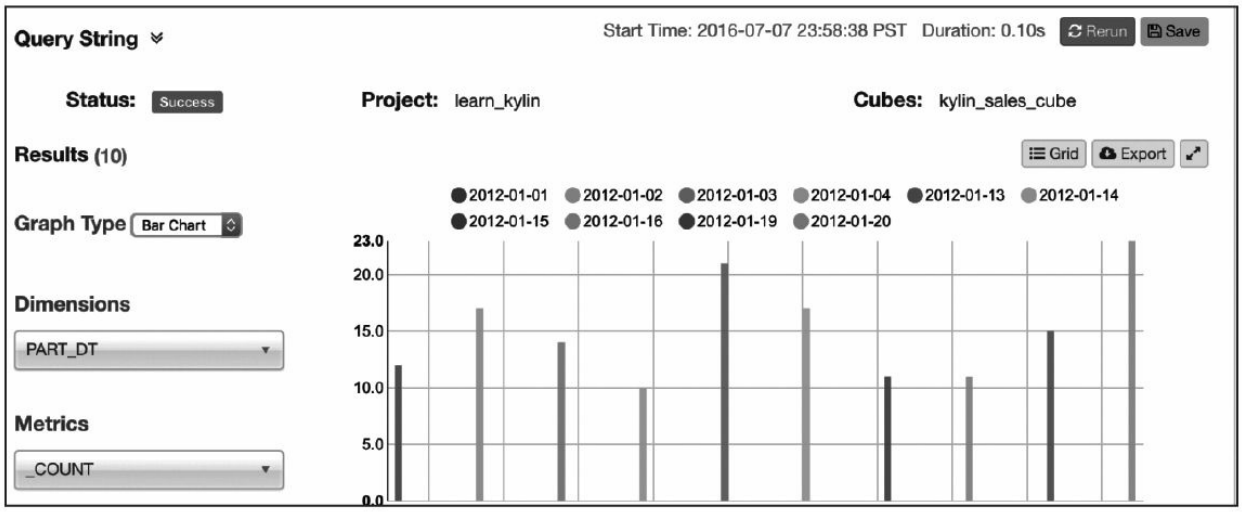


图5-6 柱状图

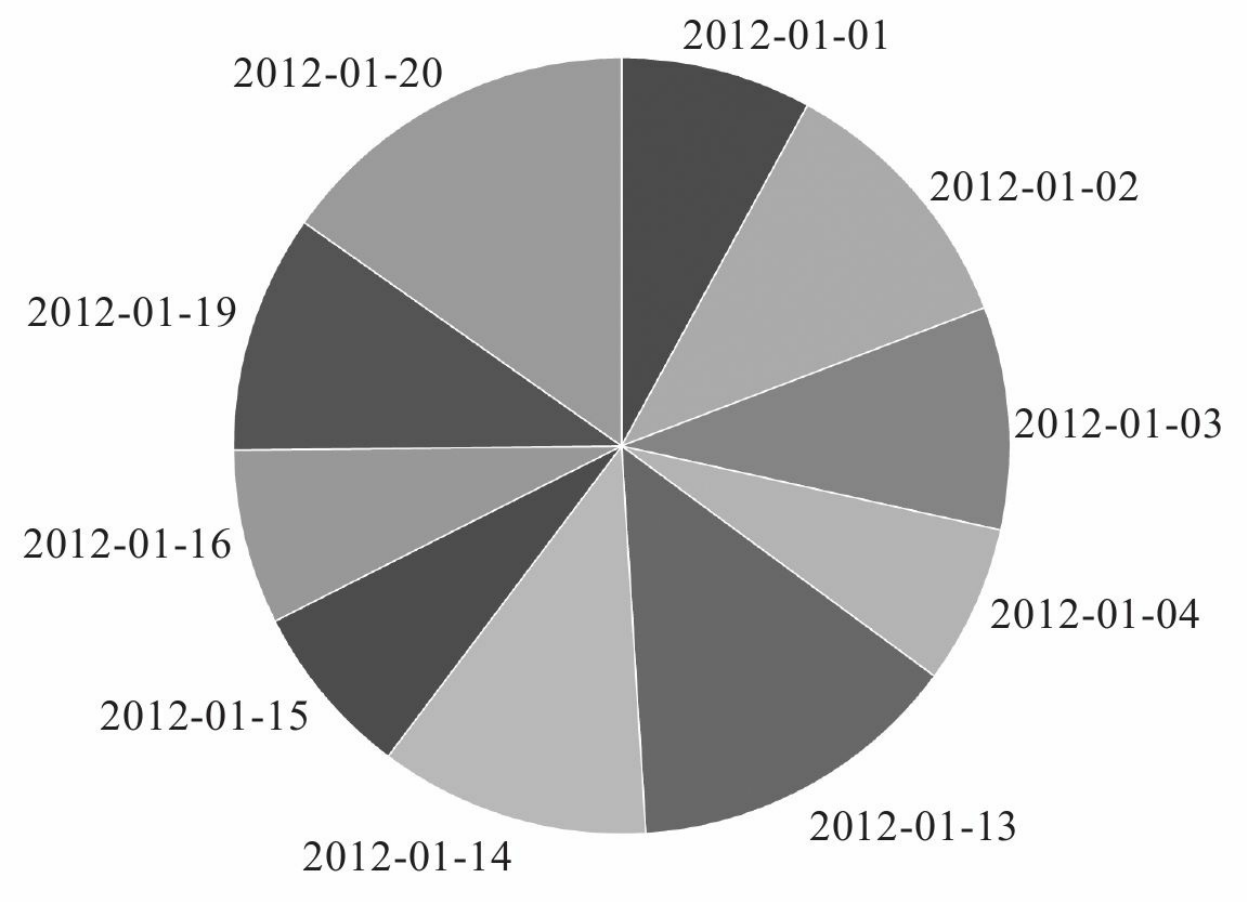

2012-01-01
2012-01-02
2012-01-03
2012-01-04
2012-01-13
2012-01-14
2012-01-15
2012-01-16
2012-01-19
2012-01-20

图5-7 饼图

5.2 Rest API

前面说过Kylin查询页面主要是基于一个查询Rest API，这里将详细介绍应该如何使用该API，读者了解后便可以基于该API在各种场景下灵活获取Apache Kylin的数据了。

5.2.1 查询认证

Kylin查询请求对应的URL为http://<hostname>:<port>/kylin/api/query，HTTP的请求方式为POST。Kylin所有的API都是基于Basic Authentication认证机制的，Basic Authentication是一种非常简单的访问控制机制，它先对账号密码基于Base64编码，然后将其作为请求头添加到HTTP请求头中，后端会读取请求头中的账号密码信息以进行认证。以Kylin默认的账号密码ADMIN/KYLIN为例，对相应的账号密码进行编码后，结果为“Basic QURNSU46S1lMSU4=”，那么HTTP对应的头信息则为“Authorization:Basic QURNSU46S1lMSU4=”。

·若要增强认证安全性，可以启用HTTPS协议，并将URL替换为https://。这样就能保证用户名和密码在传输过程中受到最好的安全保密。

5.2.2 查询请求参数

查询API的Body部分要求发送一个JSON对象，下面将对请求对象的各个属性逐一进行说明。注意，描述中的‘必填’是指该属性在查询时不能为空，必须加上；‘可选’表示查询时这个字段不是必须要填的，可根据实际需要决定是否加上该字段。

·sql：必填，字符串类型，请求的SQL。

·offset：可选，整型，查询默认从第一行返回结果，可以设置该参数以决定返回数据从哪一行开始往后返回。

·limit：可选，整型，加上limit参数后会从offset开始返回对应的行数，返回数据行数小于limit的将以实际行数为准。

·acceptPartial：可选，布尔类型，默认是“true”，如果为true，那么实际上最多会返回一百万行数据；如果要返回的结果集超过了一百万行，那么该参数需要设置为“false”。

·project：可选，字符串类型，默认为“DEFAULT”，在实际使用时，如果对应查询的项目不是“DEFAULT”，那就需要设置为自己的项目。

下面是一个HTTP请求内容的完整示例，读者通过这个示例可以明白

查询的请求体是一个什么样的结构:

```
{
    "sql":"select * from TEST_KYLIN_FACT",
    "offset":0,
    "limit":50000,
    "acceptPartial":false,
    "project":"DEFAULT"
}
```

5.2.3 查询返回结果

查询结果返回的也是一个JSON对象，下面给出的是返回对象中每一个属性的解释。

·columnMetas：每个列的元数据信息。

·results：返回的结果集。

·cube：这个查询对应使用的CUBE。

·affectedRowCount：这个查询关系到的总行数。

·isException：这个查询的返回是否异常。

·exceptionMessage：如果查询返回异常，则给出对应的内容。

·duration：查询消耗的时间，单位为毫秒。

·partial：这个查询结果是否仅为部分结果，这取决于请求参数中的？acceptPartial？为true还是false。

下面是一个查询返回格式示例：

```
{
    "columnMetas":[
        {
            "isNullable":1,
            "displaySize":0,
            "label":"CAL_DT",
            "name":"CAL_DT",
            "schemaName":null,
            "catelogName":null,
            "tableName":null,
            "precision":0,
            "scale":0,
            "columnType":91,
            "columnTypeName":"DATE",
            "readOnly":true,
            "writable":false,
            "caseSensitive":true,
            "searchable":false,
            "currency":false,
            "signed":true,
            "autoIncrement":false,
```

```
            "definitelyWritable":false
        },
        ….../ 此处省略
    ],
    "results":[
        [
            "2013-08-07",
            "32996",
            "15",
            "15",
            "Auction",
            "10000000",
            "49.048952730908745",
            "49.048952730908745",
            "49.048952730908745",
            "1"
        ],
        ….../ 此处省略
    ],
    "cube":"test_kylin_cube_with_slr_desc",
    "affectedRowCount":0,
    "isException":false,
    "exceptionMessage":null,
    "duration":3451,
    "partial":false
}
```

5.3 ODBC

Apache Kylin提供了32位和64位两种ODBC驱动，支持ODBC的应用可以基于该驱动访问Kylin。该驱动程序目前只提供Windows版本，在Tableau和Microsoft Excel上已经过充分的测试。

在安装KylinODBC之前，需要先安装Microsoft Visual C++2012Redistributable，其在Kylin的官网上可以下载。此外，因为ODBC需要从Rest API获取数据，所以在使用之前需要确保你有正在运行的Apache Kylin服务，有可以访问的Rest API接口。最后，如果以前安装过Apache Kylin ODBC驱动，那么需要先卸载老版本。

到Apache Kylin官网下载ODBC驱动，上面分别提供了KylinODBCDriver(x86).exe和KylinODBCDriver(x64).exe，供32位和64位的操作系统使用。

安装好驱动后，需要继续配置DSN，下面分步介绍如何配置DSN。

第一步，打开ODBC Data Source Administrator，然后安装驱动，如图5-8所示。这里又涉及如下两种情况：

安装32位驱动时，对应的打开位置为C:\Windows\SysWOW64\odbcad32.exe。

安装64位驱动时，依次打开Windows的控制面板→管理工具→数据源（ODBC）。

第二步，打开“System DSN”，单击“Add”，找到KylinODBCDriver这个选项，单击“Finish”继续下一步。如图5-9所示。

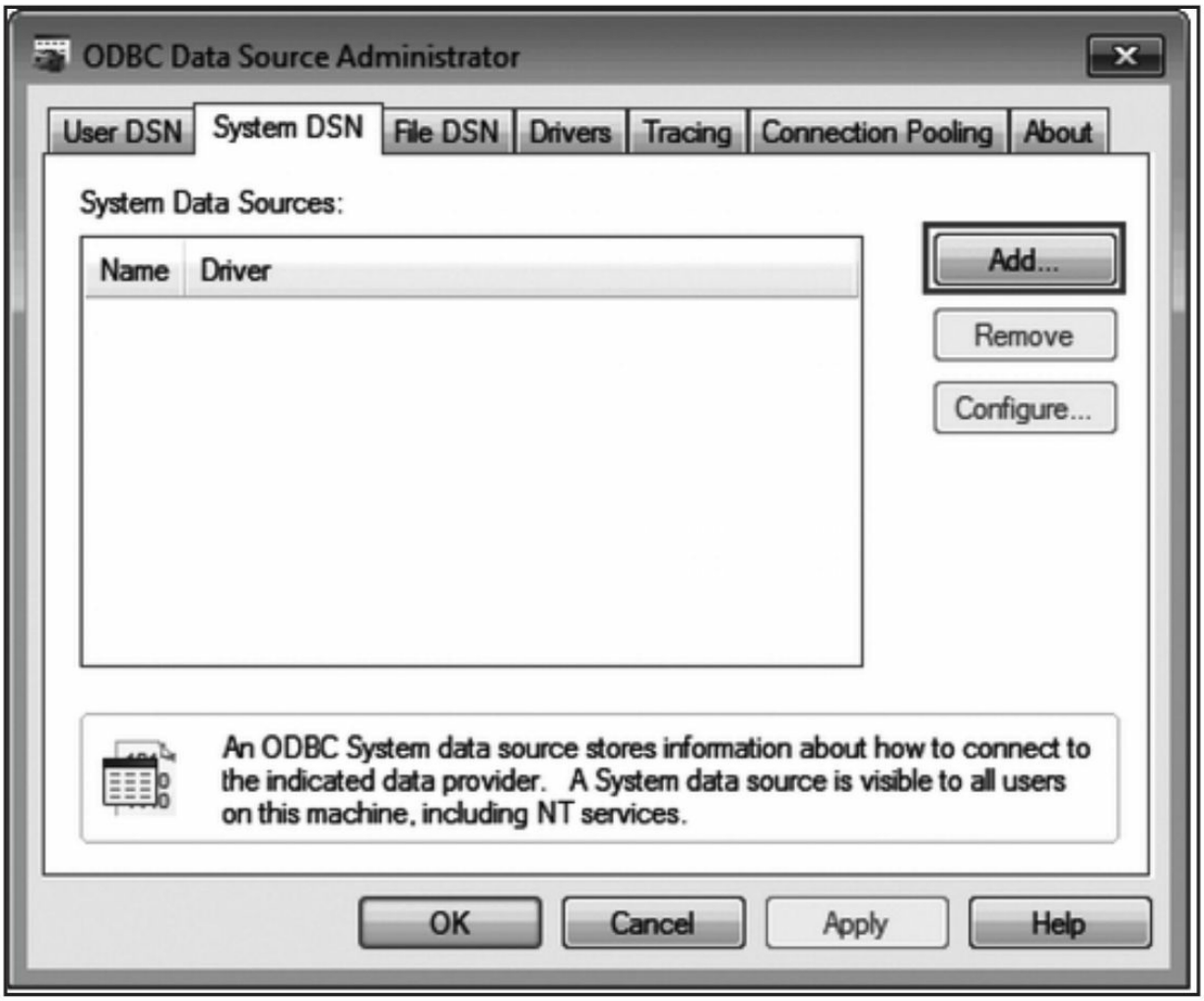


图5-8 打开ODBC Data Source Administrator

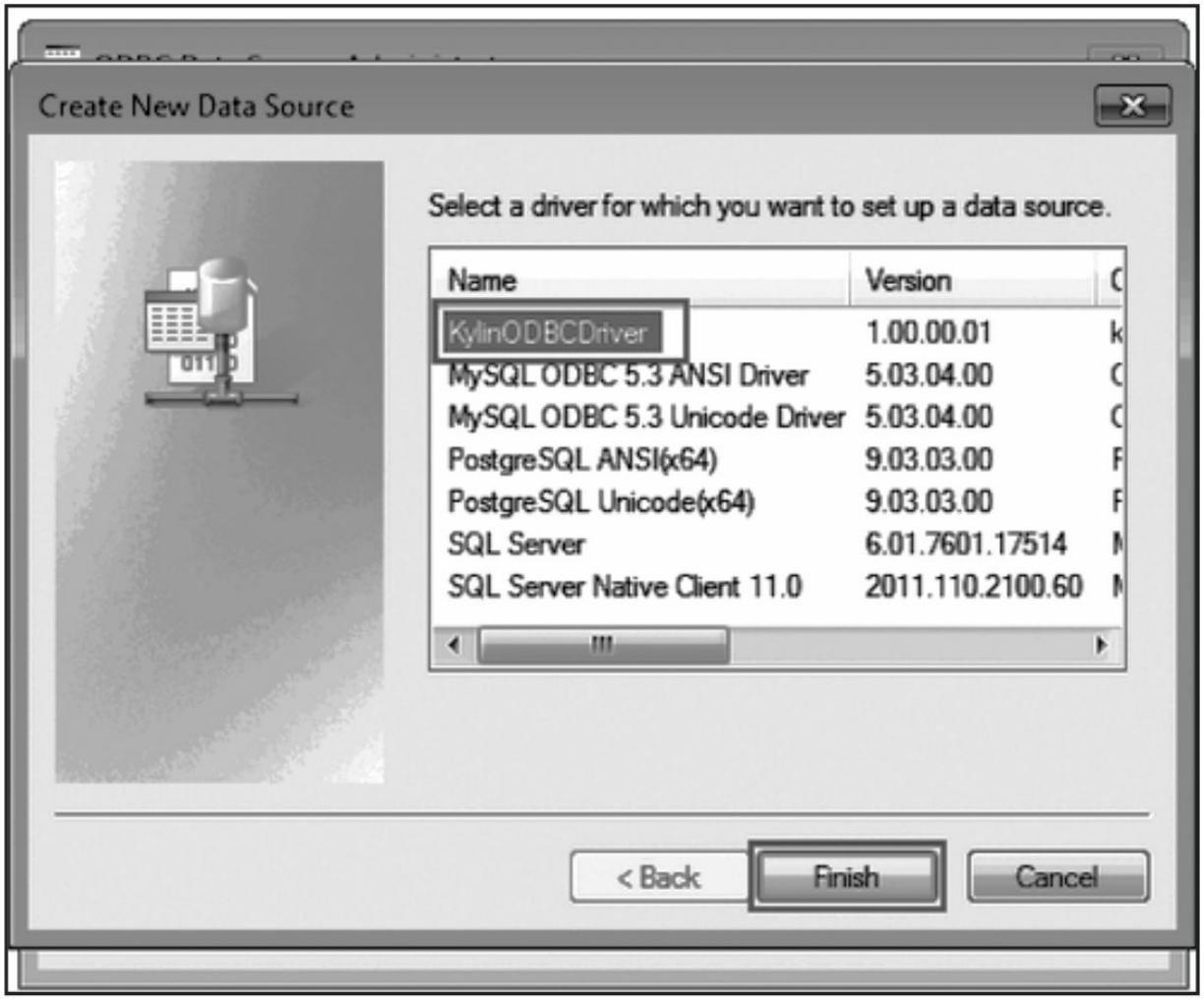


图5-9 利用KylinODBCDriver创建新的Data Source

第三步，在弹出的对话框中，填上对应的选项，服务器地址和端口分别为对应Rest API的IP和端口，如图5-10所示。注意图示中的端口号为443，是启用了HTTPS协议的一种情况。对于初次使用Apache Kylin的用户，默认的Rest API服务端口应该为7070。

第四步，单击“Done”按钮，在系统DSN中就可以看到新建的DSN了，如图5-11所示，然后就可以使用了。

图5-10 填写Rest API服务器和端口

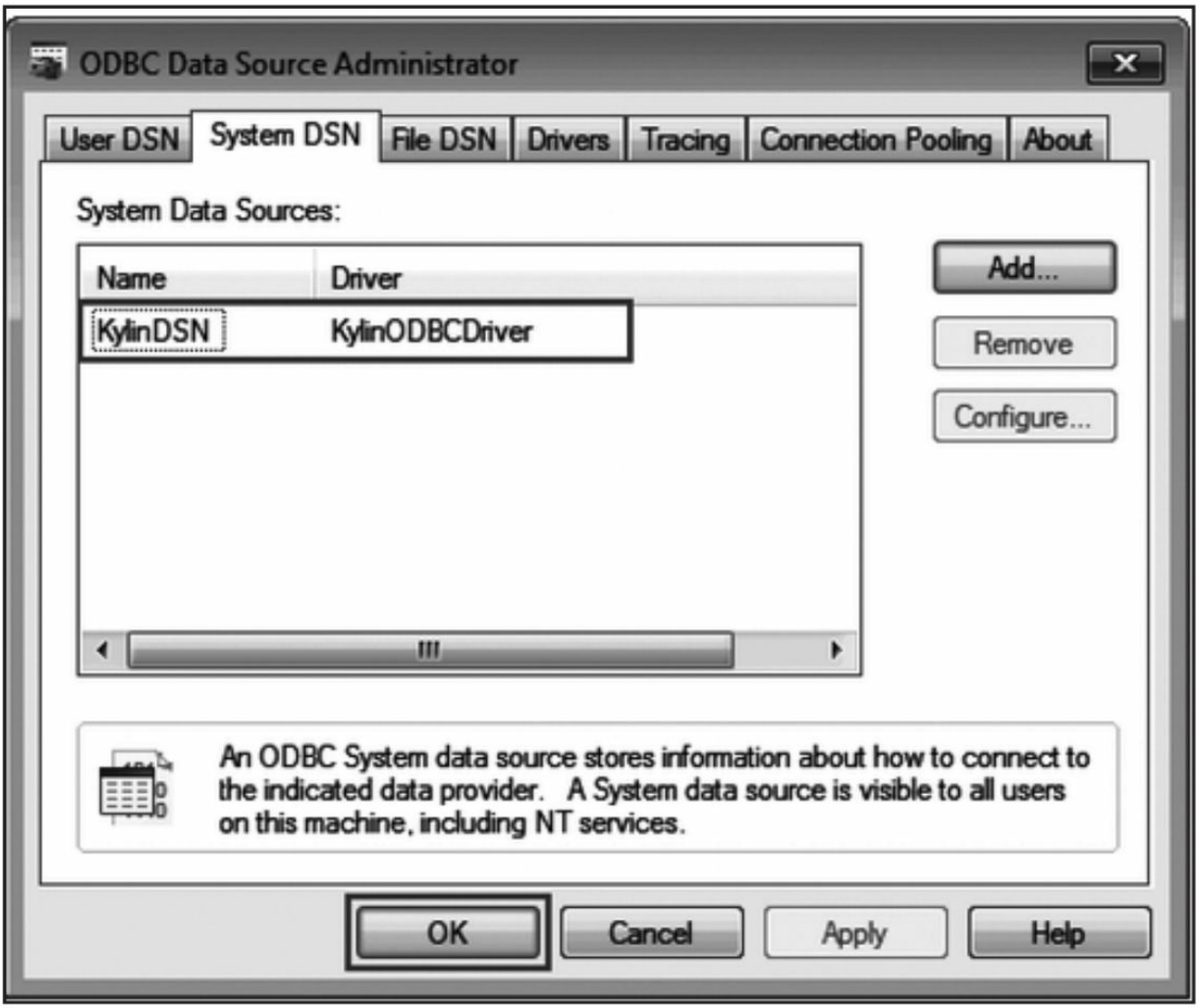


图5-11　添加DSN完成

5.4 JDBC

Kylin也为用户提供了JDBC驱动，用户通过本节可以了解如何正确使用Kylin提供的JDBC驱动包。本节会对JDBC的认证方式，以及URL的格式进行说明，同时通过示例代码直观地展示如何基于Statement和PreparedStatement查询Kylin的数据，读者在自己的环境中修改对应的URL及表名信息就可以直接运行示例代码了。

5.4.1 获得驱动包

在默认发布的二进制包中，对应lib目录下有名称为kylin-jdbc-{version}-SNAPSHOT.jar的jar包，这就是Apache Kylin的JDBC驱动包。

5.4.2 认证

创建JDBC连接时，有三个属性需要填写，下面是对各个属性的说明。

·user：Kylin用户的名称。

·password：Kylin用户的密码。

·ssl：默认值为false。如果为true，则所有的访问都将基于HTTPS。

5.4.3 URL格式

JDBC访问Kylin对应的URL格式为“jdbc:kylin://<hostname>:<port>/<kylin_project_name>”。URL中需要填写端口的信息，如果JDBC连接属性对应的“ssl”设置为true，那么端口将对应为Kylin服务器的HTTPS端口，一般为443；此外Apache Kylin默认的HTTP服务端口是7070；“kylin_project_name”是Apache Kylin服务端的项目名称，该项目必须存在。

下面是Kylin JDBC基于Statement的Query示例代码：

```
    Driver driver = (Driver) Class.forName("org.apache.kylin.jdbc.Driver").
newInstance();
    Properties info = new Properties();
    info.put("user", "ADMIN");
    info.put("password", "KYLIN");
    Connection conn = driver.connect("jdbc:kylin:// localhost:7070/kylin_project_
name", info);
    Statement state = conn.createStatement();
    ResultSet resultSet = state.executeQuery("select * from test_table");
    while (resultSet.next()) {
        assertEquals("foo", resultSet.getString(1));
        assertEquals("bar", resultSet.getString(2));
        assertEquals("tool", resultSet.getString(3));
    }
```

以下是Kylin JDBC基于PreparedStatement的Query示例代码：

```
    Driver driver = (Driver) Class.forName("org.apache.kylin.jdbc.Driver").
newInstance();
    Properties info = new Properties();
    info.put("user", "ADMIN");
    info.put("password", "KYLIN");
    Connection conn = driver.connect("jdbc:kylin:// localhost:7070/kylin_project_
name", info);
    PreparedStatement state = conn.prepareStatement("select * from test_table where
id=?");
    state.setInt(1, 10);
    ResultSet resultSet = state.executeQuery();
    while (resultSet.next()) {
        assertEquals("foo", resultSet.getString(1));
        assertEquals("bar", resultSet.getString(2));
        assertEquals("tool", resultSet.getString(3));
    }
```

5.4.4 获取元数据信息

Kylin JDBC驱动支持获取元数据信息，我们可以基于SQL的一些过滤表达式（比如%）列出Catalog、Schema、表和列信息，下面是如何获取元数据信息的示例代码：

```
    Driver driver = (Driver) Class.forName("org.apache.kylin.jdbc.Driver").
newInstance();
    Properties info = new Properties();
    info.put("user", "ADMIN");
    info.put("password", "KYLIN");
    Connection conn = driver.connect("jdbc:kylin:// localhost:7070/kylin_project_
name", info);
    Statement state = conn.createStatement();
    ResultSet resultSet = state.executeQuery("select * from test_table");
    ResultSet tables = conn.getMetaData().getTables(null, null, "dummy", null);
    while (tables.next()) {
        for (int i = 0; i < 10; i++) {
            assertEquals("dummy", tables.getString(i + 1));
        }
    }
```

5.5 通过Tableau访问Kylin

Tableau是一款应用比较广泛的商业智能工具软件，有着很好的交互体验，可基于拖曳式生成各种可视化图表，相信很多读者已经了解或使用过该产品。本节会讲解如何使用Tableau访问Apache Kylin的数据。基于Apache Kylin提供的ODBC驱动，Tableau可以很好地对接大数据，让用户以更友好的方式对大数据进行交互式的分析。

本文基于Tableau9.1版本讲解，在使用Tableau之前，请确保您已经安装了ODBC驱动。

5.5.1 连接Kylin数据源

通过驱动连接Kylin数据源的方式为：启动Tableau9.1桌面版，单击左边面板中的“Other Database（ODBC）”，在弹出的窗口中选择“KylinODBCDriver”，如图5-12所示。

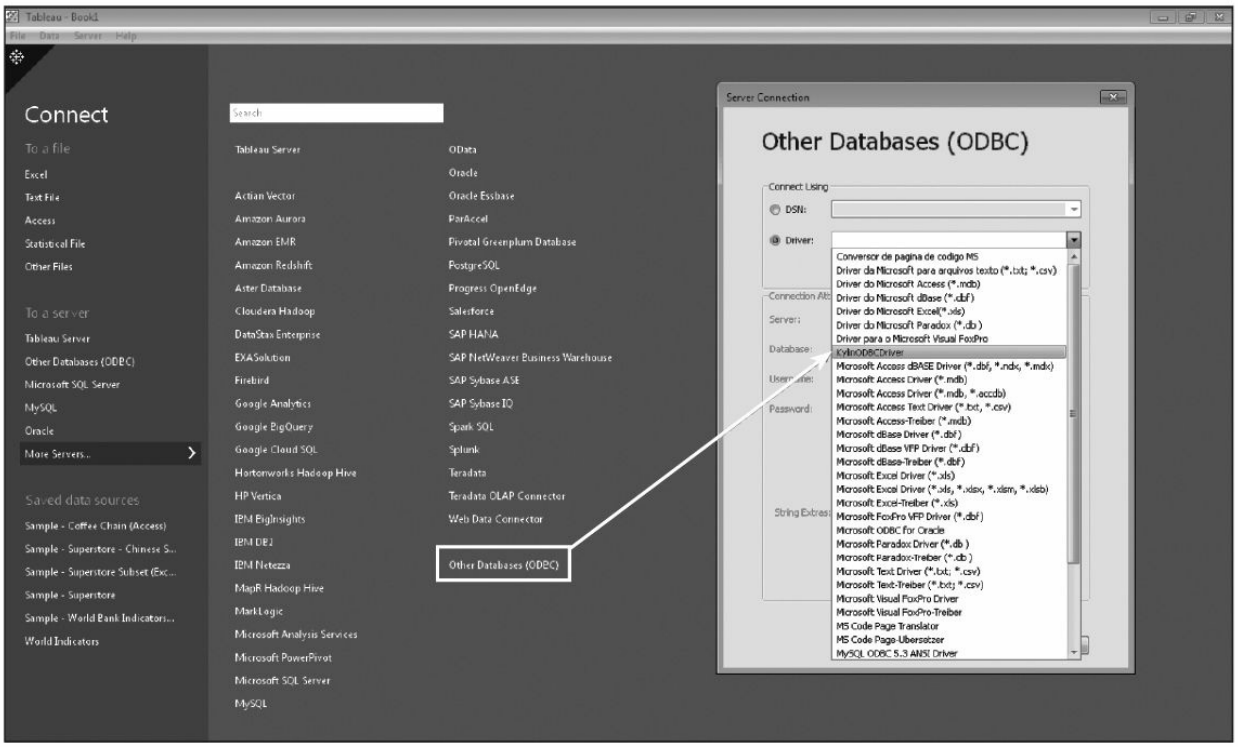


图5-12 在Tableau中选择Apache Kylin ODBC驱动

在弹出的驱动连接窗口中填写服务器、认证、项目，单击“Connect”按钮，你将会看到所有你有权限访问的项目，如图5-13所示。

5.5.2 设计数据模型

在Tableau客户端的左面板中，选择“defaultCatalog”作为数据库，在搜索框中单击“Search”将会列出所有的表，可通过拖曳的方式把表拖到右边的面板中，给这些表设置正确的连接方式，如图5-14所示。

5.5.3 通过Live方式连接

模型设计完成之后，我们需要选择Tableau与后端交互的连接方式，如图5-15所示。Tableau支持两种连接方式，分别为Live和Extract。Extract模式会把全部数据加载到系统内存，查询的时候直接从内存中获取数据，是非常不适合大数据处理的一种方式，因为大数据无法被全部驻留在内存中。Live模式会实时发送请求到服务器查询，配合Apache Kylin亚秒级的查询速度，能够很好地实现交互式的大数据可视化分析。请总是选择Live为连接Apache Kylin的连接方式。

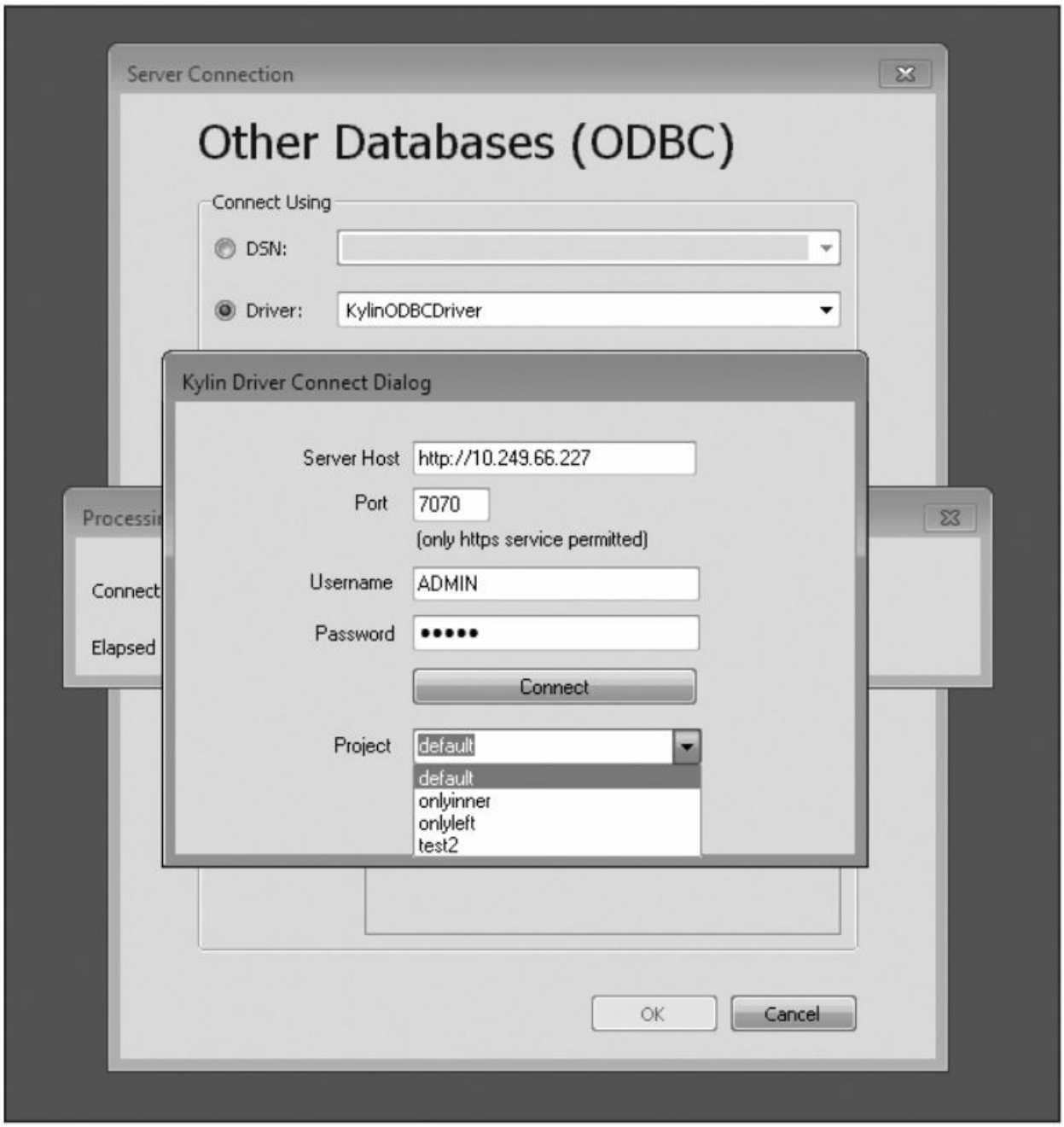


图5-13 Apache Kylin连接信息

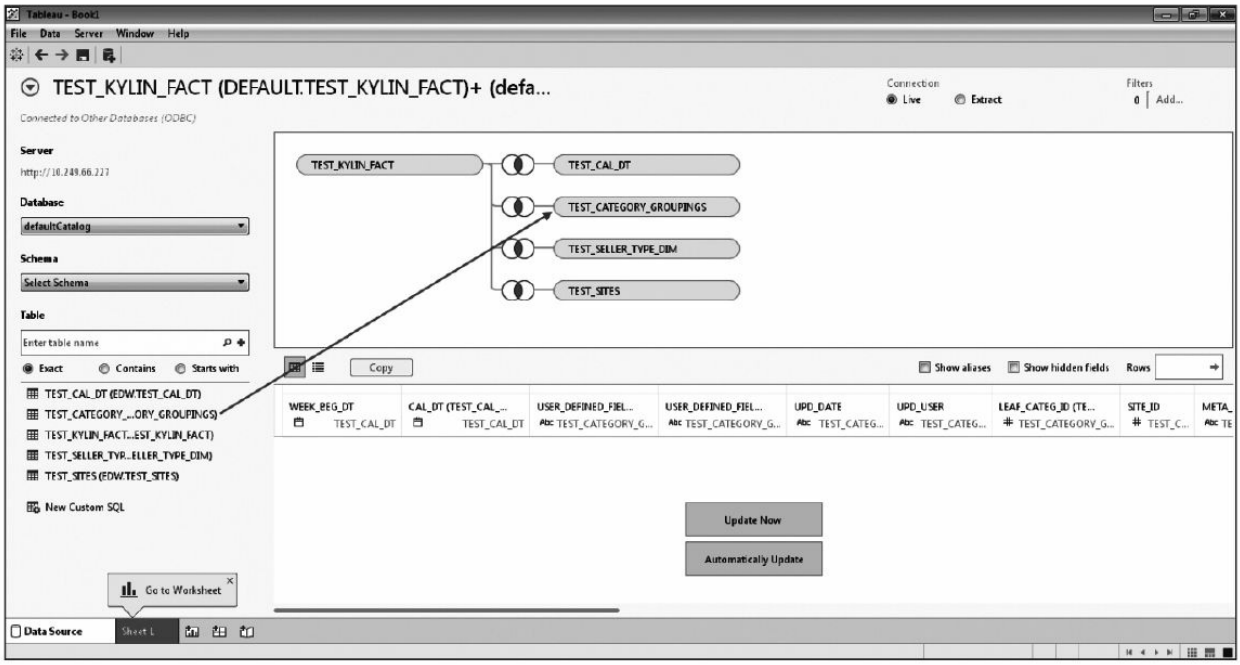


图5-14 在Tableau中设计数据模型

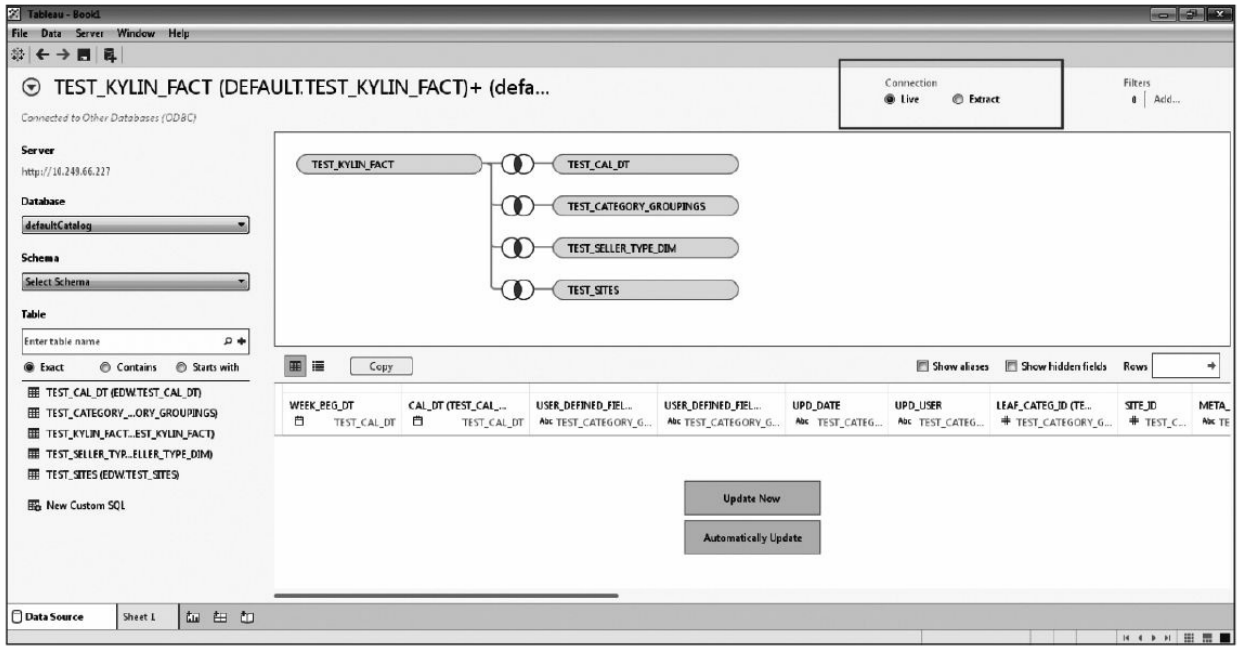


图5-15 选择连接方式

5.5.4 自定义SQL

如果用户想通过自定义SQL进行交互，可以单击图5-16左下角的“New Custom SQL”，在弹出的对话框中输入SQL即可实现。

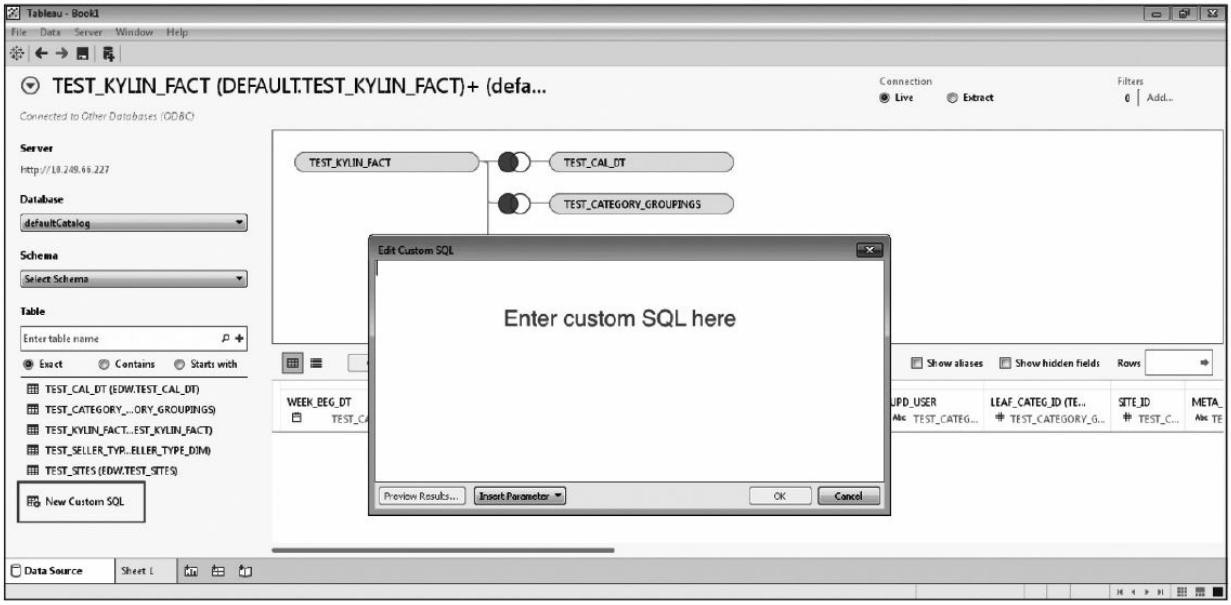


图5-16 “New Custom SQL”对话框

5.5.5 可视化

在Tableau右侧面板中，我们可以看到有列框（Columns）和行框（Rows），把度量拖到列框中，把维度拖到行框中，就可以生成自己的图表了，如图5-17所示。

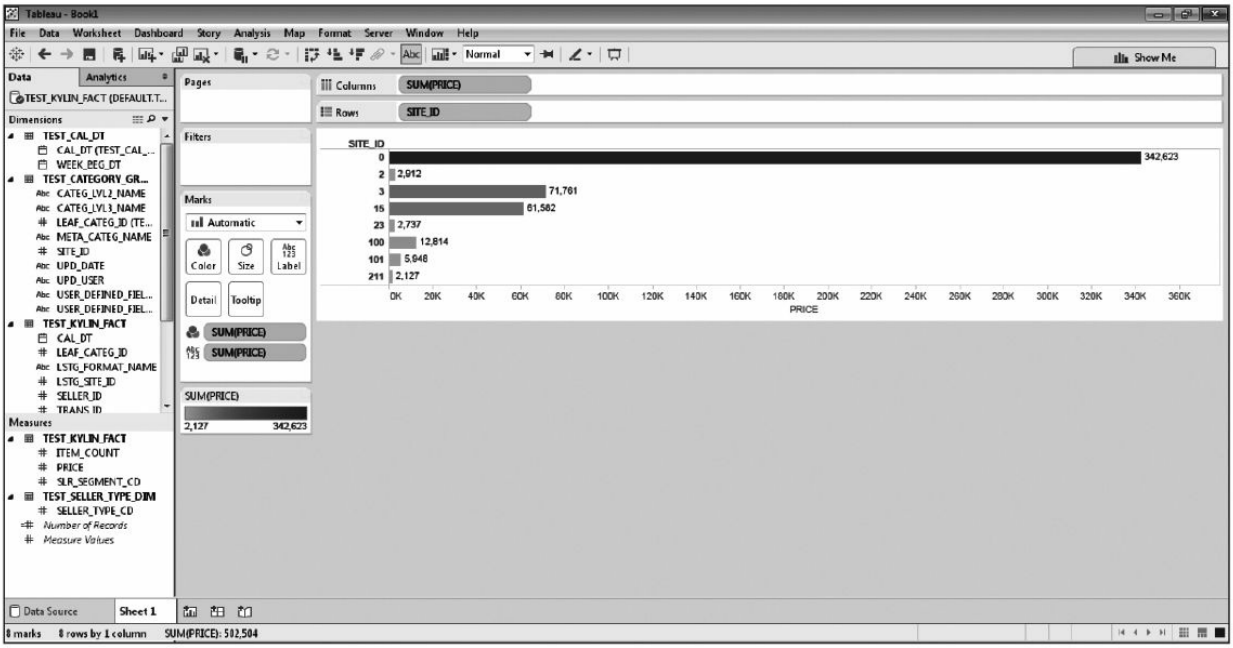


图5-17 在Tableau中拖曳列行框展示数据

5.5.6 发布到Tableau Server

如果想将本地Dashboard发布到Tableau Server，则展开右上角的“Server”按钮，然后单击“Publish Workbook”即可，如图5-18所示。

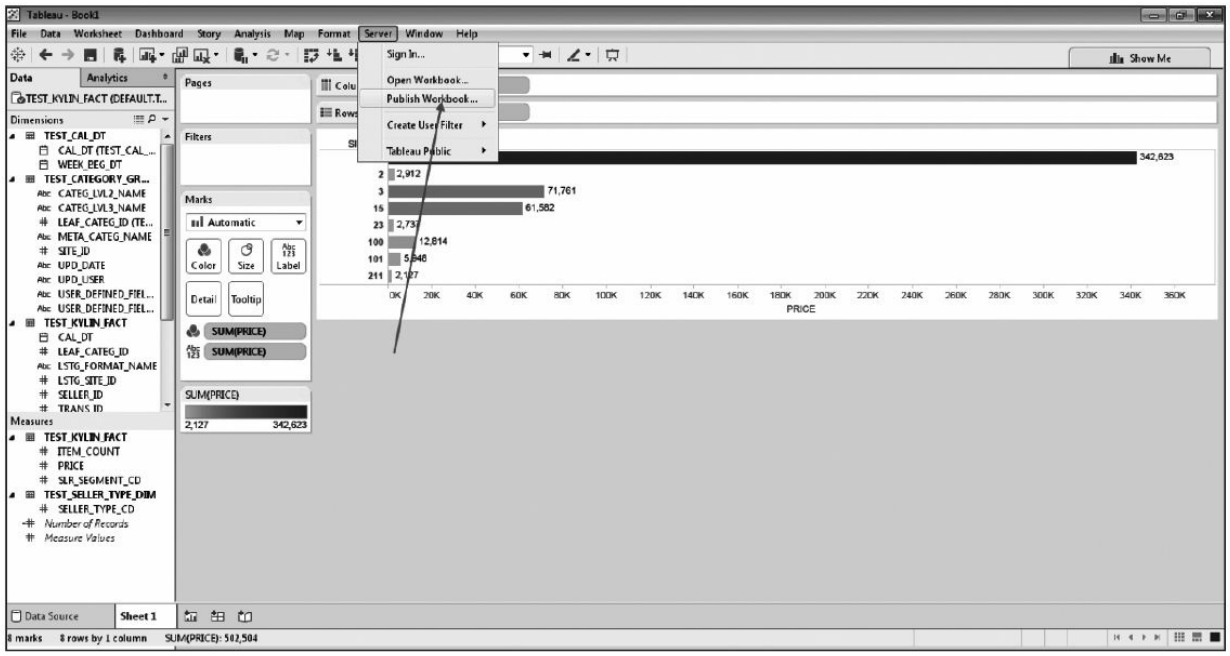


图5-18 发布到Tableau Server

5.6 Zeppelin集成

Apache Zeppelin是一个开源的数据分析平台，是Apache的顶级项目。Zeppelin后端以插件形式支持多种数据处理引擎，如Spark、Flink、Lens等，同时还提供了Notebook式的UI进行可视化相关的操作。为此，Apache Kylin开发了对应的Zeppelin模块，现在已经合并到Zeppelin主分支中，在Zeppelin0.5.6及后续版本中都可以对接使用Kylin，从而实现通过Zeppelin访问Kylin的数据。

5.6.1 Zeppelin架构简介

如图5-19所示，Zeppelin客户端通过HTTP Rest和Websocket两种方式与服务端进行交互。在服务端，Zeppelin支持可插拔的Interpreter（解释器）。以Apache Kylin为例，只需要开发Kylin的Interpreter，并将其集成至Zeppelin便可以基于Zeppelin客户端与Kylin服务端进行通信，高速访问Kylin中的大量数据了。

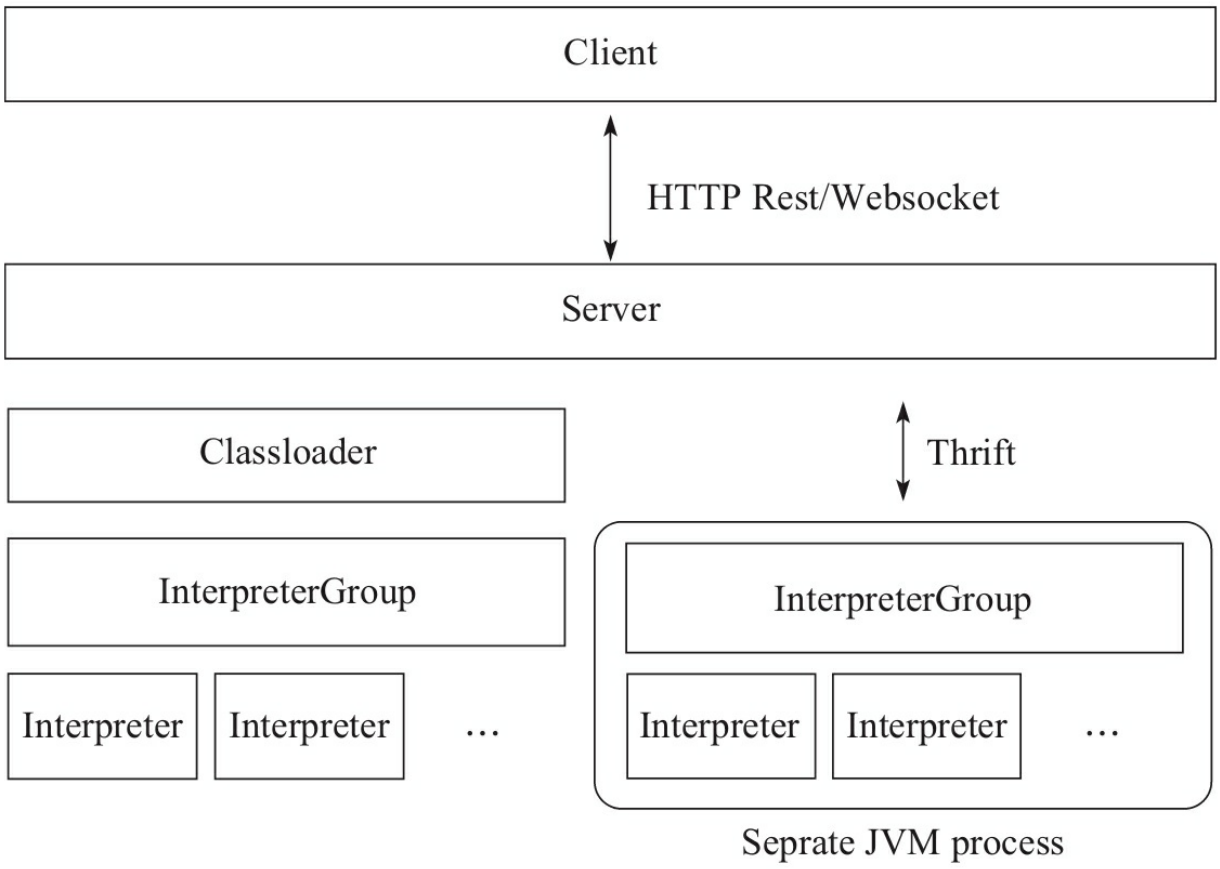


图5-19 Zeppelin架构

5.6.2 KylinInterpreter的工作原理

KylinInterpreter是Zeppelin的一个Interpreter插件，可用来连接Apache Kylin数据。它是构建在Apache Kylin的Rest API之上的，也是一种典型的使用Kylin API的场景。Kylin-Interpreter读取Zeppelin前端针对Kylin配置的URL、User、Password等连接信息，再结合每次查询的SQL、project、limit、offset和isPartial等参数，就可以生成Rest API的请求，通过HTTP POST方式发送到Apache Kylin服务器以获取数据。

下面是KylinInterpreter的部分代码，结合注释可以明白KylinInterpreter是如何访问Apache Kylin API的。

```
    public HttpResponse prepareRequest(String sql) throws IOException {
        String KYLIN_PROJECT = getProperty(KYLIN_QUERY_PROJECT);
    ......
    ......
        //从配置项中读取账号密码，并基于 Base64 进行编码
        byte[] encodeBytes = Base64.encodeBase64(new String(getProperty(KYLIN_USERNAME)
            + ":" + getProperty(KYLIN_PASSWORD)).getBytes("UTF-8"));
        //设置请求参数
        String postContent = new String("{\"project\":" + "\"" + KYLIN_PROJECT + "\""
            + "," + "\"sql\":" + "\"" + sql + "\""
            + "," + "\"acceptPartial\":" + "\"" + getProperty(KYLIN_QUERY_ACCEPT_PARTIAL)
+ "\""
            + "," + "\"offset\":" + "\"" + getProperty(KYLIN_QUERY_OFFSET) + "\""
            + "," + "\"limit\":" + "\"" + getProperty(KYLIN_QUERY_LIMIT) + "\"" + "}");
        postContent = postContent.replaceAll("[\u0000-\u001f]", " ");
        StringEntity entity = new StringEntity(postContent, "UTF-8");
        entity.setContentType("application/json; charset=UTF-8");
        HttpPost postRequest = new HttpPost(getProperty(KYLIN_QUERY_API_URL));
    postRequest.setEntity(entity);
        //设置请求头信息，加上 Basic Authentication 信息
        postRequest.addHeader("Authorization", "Basic " + new String(encodeBytes));
        postRequest.addHeader("Accept-Encoding", "UTF-8");
```

```
    HttpClient httpClient = HttpClientBuilder.create().build();
    return httpClient.execute(postRequest);
}
```

Zeppelin的前端有自己的数据分装格式，所以KylinInterpreter需要把Kylin返回的数据进行适当的转换以让Zeppelin前端能够理解。所以KylinInterpreter的主要任务就是拼接参数，完成向Kylin服务端发起的HTTP请求，然后格式化返回的结果，交给Zeppelin前端展现。

5.6.3 如何使用Zeppelin访问Kylin

首先读者需要到Zeppelin官网下载0.5.6或之后版本的二进制包，然后按照官网提示配置启动，从而打开Zeppelin前端页面（官网有非常详细的介绍，这里就不再赘述了）。

1.配置Interpreter

打开Zeppelin配置页面，单击Interpreter页面，创建针对Kylin某个项目的Interpreter配置，如图5-20所示。

2.查询

打开Notebook创建一个新的Note，在Note中输入SQL。注意，针对Kylin的查询需要在SQL前面加上“%kylin”，Zeppelin后端需要知道用哪个对应的Interpreter去处理查询。效果如图5-21所示，可以拖曳维度和度量灵活获取自己想要的结果。

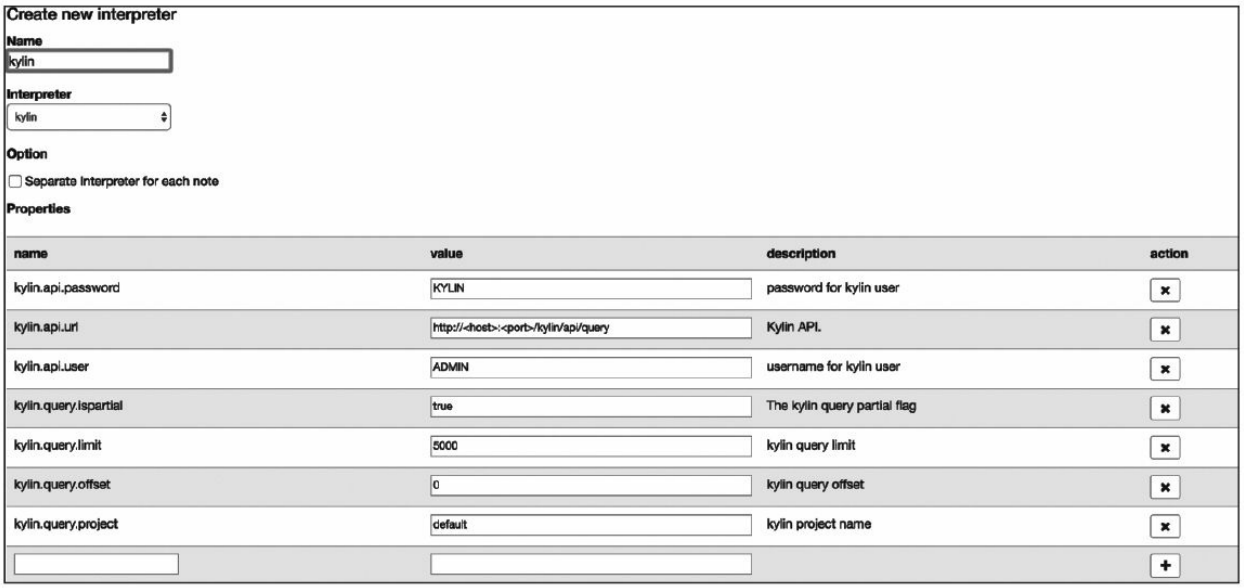


图5-20 创建Kylin Intepreter配置

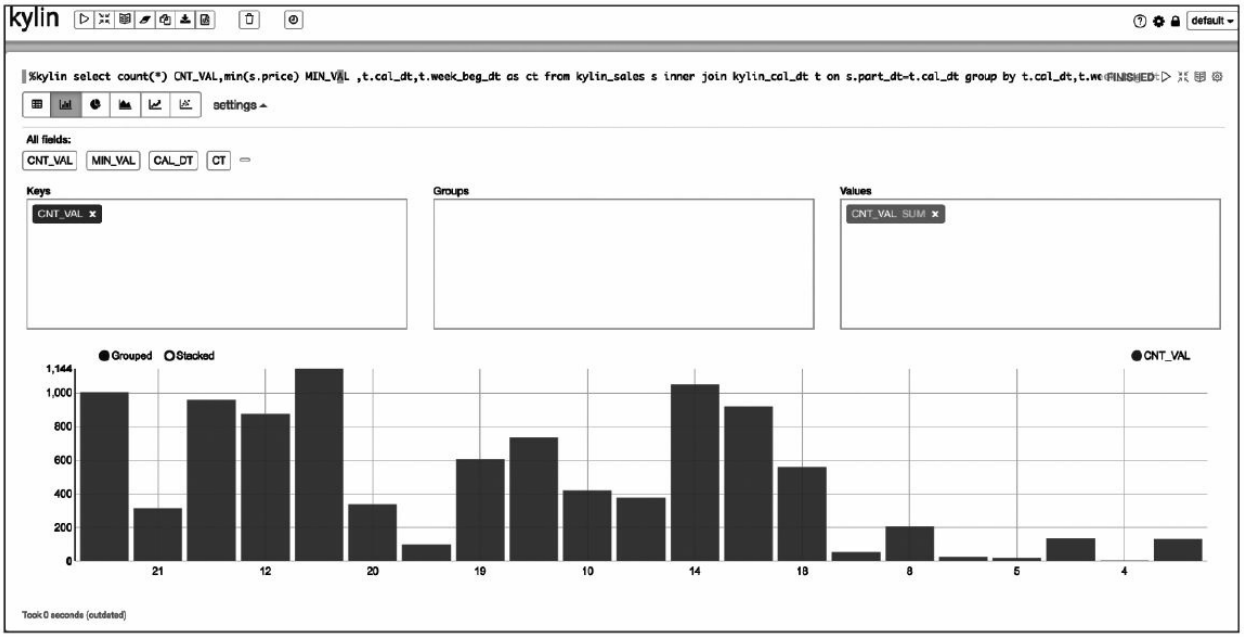


图5-21 Zeppelin展现Apache Kylin的返回数据

3.Zeppelin的发布功能

对于Zeppelin中的任何一个查询，你都可以创建一个链接，并且将该

链接分享给其他人，从而分享你的分析工作成果。感兴趣的读者可以到Zeppelin官网http://zeppelin.apache.org/了解更多特性。

5.7 小结

Apache Kylin提供了灵活的前端连接方式，包括Rest API、JDBC和ODBC。用户可以根据需要使用已有的BI工具（比如Tableau）查询Apache Kylin，也可以开发定制的应用程序，通过这些API访问Apache Kylin。

此外通过Rest API，用户还可以读取元数据，触发Cube构建，查询构建进度，甚至实现自动创建Cube等高级功能。有兴趣的读者可以参考http://kylin.apache.org/docs15/howto/howto_use_restapi.html。

第6章 Cube优化

Apache Kylin的核心思想是根据用户的数据模型和查询样式对数据进行预计算，并在查询时直接利用预计算结果返回查询结果。相比普通的大规模并行处理的解决方案，Kylin具有响应时间快、查询时资源需求小、吞吐量大等优点。用户的数据模型包括维度、度量、分割时间列等基本信息，也包括用户通过Cube优化工具赋予的额外的模型信息。例如，层级（Hierarchy）是一种用来描述若干个维度之间存在层级关系的优化工具，提供层级信息有助于帮助预计算跳过多余的预计算，从而减少预计算的工作量，并且最终减少存储引擎所需要存储的Cube数据大小。数据模型是数据固有的属性，除此之外，查询的样式如果相对固定，也可以用来帮助Cube的优化。例如，如果我们知道客户端的查询总是会带有某个维度上的过滤（Filter）条件，或者总是会按照这个维度进行聚合（Group By），那么所有的不带这个维度的场景下的预计算都可以被跳过，因为即使为这些场景进行了预计算，这些预计算结果也从来不会被用到。

总的来说，在构建Cube之前，Cube的优化手段提供了更多与数据模型或查询样式相关的信息，用于指导构建出体积更小、查询速度更快的Cube。可以看到Cube的优化目的始终有两个：空间优化和查询时间优化。

6.1 Cuboid剪枝优化

6.1.1 维度的诅咒

从之前章节的介绍可以知道，在没有采取任何优化措施的情况下，Kylin会对每一种维度的组合进行预计算，每种维度的组合的预计算结果被称为Cuboid。假设有4个维度，结合简单的数学知识，我们可能最终会有2^4 =16个Cuboid需要计算。

但在现实情况中，用户的维度数量一般远远大于4个。假设用户有10个维度，那么没有经过任何优化的Cube就会存在2^{10} =1024个Cuboid；而如果用户有20个维度，那么Cube中总共会存在2^{20} =1048576个Cuboid。虽然每个Cuboid的大小存在很大的差异，但是单单想到Cuboid的数量就足以让人想象到这样的Cube对构建引擎、存储引擎来说压力有多么巨大。因此，在构建维度数量较多的Cube时，尤其要注意Cube的剪枝优化。

6.1.2 检查Cuboid数量

Apache Kylin提供了一个简单的工具，供用户检查Cube中哪些Cuboid最终被预计算了，我们称其为被物化（Materialized）的Cuboid。同时，这种方法还能给出每个Cuboid所占空间的估计值。由于该工具需要在对数据进行一定阶段的处理之后才能估算Cuboid的大小，因此一般来说只能在Cube构建完毕之后再使用该工具。目前关于这一点也是该工具的一大不足，后面将在https://issues.apache.org/jira/browse/KYLIN-1743中试图解决这一问题。

由于同一个Cube的不同Segment之间仅是输入数据不同，模型信息和优化策略都是共享的，所以不同Segment中哪些Cuboid被物化哪些没有被物化都是一样的。因此只要Cube中至少有一个Segment，那么就能使用如下的命令行工具去检查这个Cube中的Cuboid状态：

```
bin/kylin.sh org.apache.kylin.engine.mr.common.CubeStatsReader CUBE_NAME
CUBE_NAME 想要查看的 Cube 的名字
```

该命令的输出如图6-1所示。

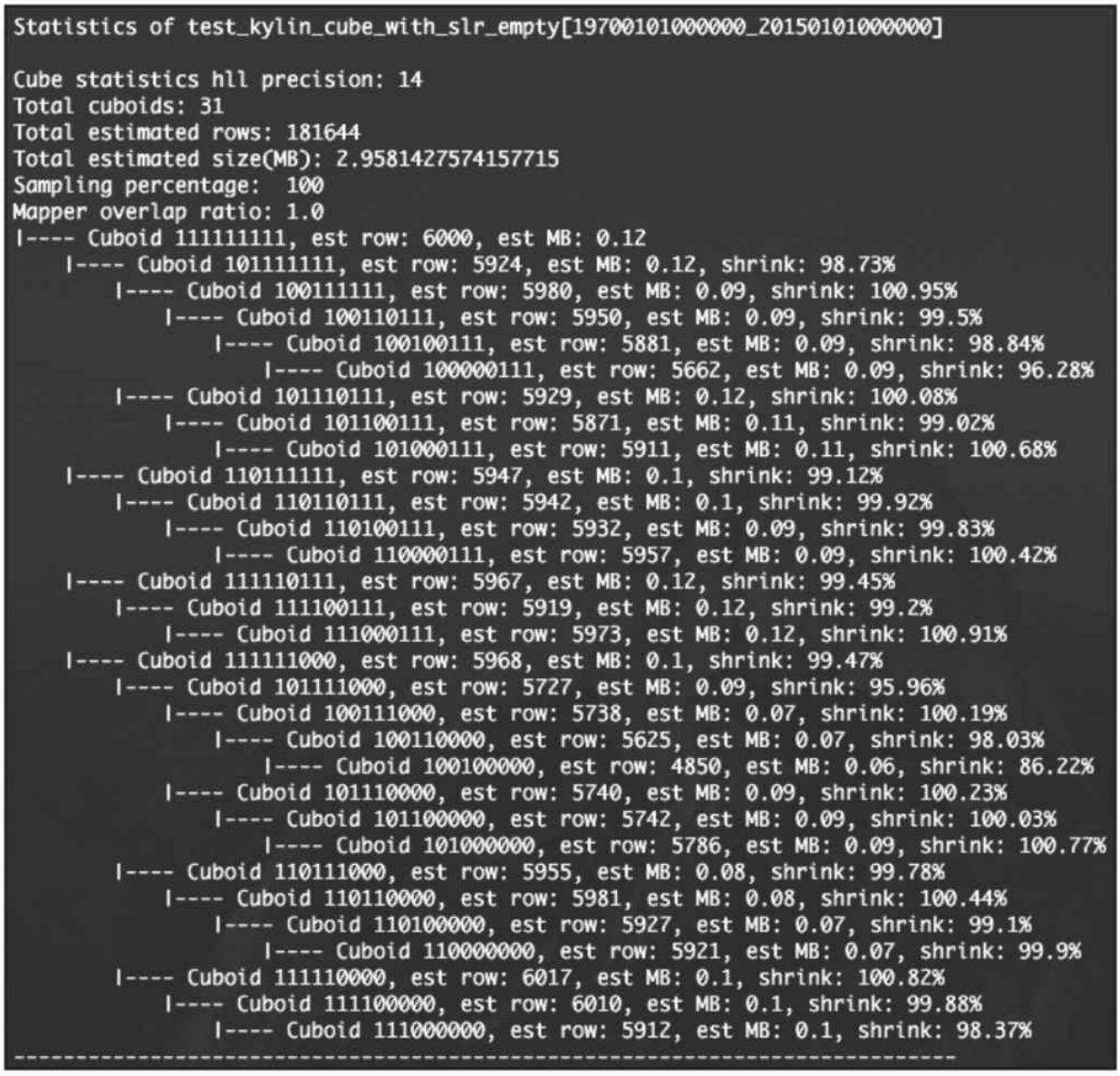


图6-1 CubeStatsReader的输出

在该命令的输出中，可依次看到对每个Segment的分析结果，不同的Segment的分析结果基本趋同。在上面的例子中Cube只有一个Segment，因此只有一份分析结果。对于该结果，从上往下看，首先能看到Segment的一些整体信息，如估计Cuboid大小的精度（Hll Precision）、总共的Cuboid数量、Segment的总行数估计、Segment的大小估计等。Segment的大小估计

是构建引擎自身用来指导后续子步骤的依据，如决定mapper reducer的数量、数据分片数量等，虽然有的时候对大小的估计存在误差（因为存储引擎对最后的Cube数据进行了编码或压缩，所以数据大小无法精确预估），但是从整体来说，对于不同的Cuboid的大小估计值可以给出一个比较直观的判断。由于没有编码或压缩时的不确定性因素，因此Segment中的行数估计会比大小估计更加精确一些。

从分析结果的下半部分可以看到，所有的Cuboid及它的分析结果都以树状的形式打印了出来。在这棵树中，每个节点代表一个Cuboid，每个Cuboid都由一连串1或0的数字组成，数字串的长度等于有效维度的数量，从左到右的每个数字依次代表Rowkeys设置中的各个维度。如果数字为0，则代表这个Cuboid中不存在相应的维度；如果数字为1，则代表这个Cuboid中存在相应的维度。除了最顶端的Cuboid之外，每个Cuboid都有一个父亲Cuboid，且都比父亲Cuboid少了一个“1”。其意义是这个Cuboid就是由它的父亲节点减少一个维度聚合而来的（上卷）。最顶端的Cuboid称为Base Cuboid，它直接由源数据计算而来。Base Cuboid中包含所有的维度，因此它的数字串中所有的数字均为1。

每行Cuboid的输出中除了0和1的数字串以外，后面还有每个Cuboid的具体信息，包括该Cuboid行数的估计值、该Cuboid大小的估计值，以及这个Cuboid的行数与父亲节点的对比（Shrink值）。所有Cuboid行数的估计值之和应该等于Segment的行数估计值，同理，所有Cuboid的大小估计值

应该等于该Segment的大小估计值。每个Cuboid都是在它的父亲节点的基础上进一步聚合而成的，因此从理论上说每个Cuboid无论是行数还是大小都应该小于它的父亲。但是，由于这些数值都是估计值，因此偶尔能够看到有些Cuboid的行数反而还超过了父亲节点，即Shrink值大于100%的情况。在这棵树中，我们可以观察每个节点的Shrink值，如果该值接近100%，则说明这个Cuboid虽然比它的父亲Cuboid少了一个维度，但是并没有比它的父亲Cuboid少很多行数据。换而言之，即使没有这个Cuboid，我们在查询时使用它的父亲Cuboid，也不会有太大的代价。关于这方面的详细内容将在6.1.4节中详细展开。

6.1.3 检查Cube大小

还有一种更为简单的方法可以帮助我们判断Cube是否已经足够优化。在Web GUI的Model页面选择一个READY状态的Cube，当我们把光标移到该Cube的Cube Size列时，Web GUI会提示Cube的源数据大小，以及当前Cube的大小除以源数据大小的比例，称为膨胀率（Expansion Rate），如图6-2所示。

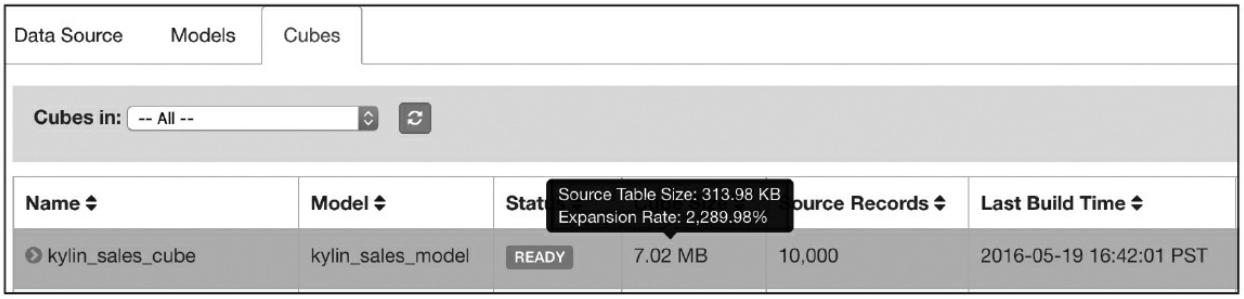


图6-2 查看Cube的膨胀率

一般来说，Cube的膨胀率应该在0%~1000%之间，如果一个Cube的膨胀率超过1000%，那么Cube管理员应当开始挖掘其中的原因。通常，膨胀率高有以下几个方面的原因。

·Cube中的维度数量较多，且没有进行很好的Cuboid剪枝优化，导致Cuboid数量极多。

·Cube中存在较高基数的维度，导致包含这类维度的每一个Cuboid占

用的空间都很大，这些Cuboid累积造成整体Cube体积变大。

·存在比较占用空间的度量，例如Count Distinct，因此需要在Cuboid的每一行中都为其保存一个较大的寄存器，最坏的情况将会导致Cuboid中每一行都有数十KB，从而造成整个Cube的体积变大。

……

因此，对于Cube膨胀率居高不下的情况，管理员需要结合实际数据进行分析，可灵活地运用本章接下来介绍的优化方法对Cube进行优化。

6.1.4 空间与时间的平衡

理论上所有能用Cuboid处理的查询请求都可以使用Base Cuboid来处理，就好像所有能用Base Cuboid处理的查询请求都能够通过直接读取源数据的方式来处理一样。但是Kylin之所以在Cube中物化这么多的Cuboid，就是因为不同的Cuboid有各自擅长的查询场景。面对一个特定的查询，使用精确匹配的Cuboid就好像是走了一条捷径，能帮助Kylin最快地返回查询结果，因为这个精确匹配的Cuboid已经为此查询做了最大努力的预先聚合，查询引擎中只需要做很少的运行时聚合就能返回结果。每个Cuboid其实都代表着一种查询的样式，如果每种样式都要做好精确的匹配，则会变得很奢侈，那么我们有必要考虑牺牲一部分查询样式的精确匹配Cuboid。这个不精确匹配的Cuboid可能是6.1.2节中提到的Cuboid的父亲Cuboid，甚至如果它的父亲Cuboid也被牺牲了，Kylin可能会一路追溯到Base Cuboid来回答查询请求。使用不精确匹配的Cuboid比起使用精确匹配的Cuboid，需要做更多查询时的聚合计算；但是如果Cube优化得当，那么查询时的聚合计算的开销就会没有想象中的那么恐怖。以6.1.2节中Shrink值接近100%的Cuboid为例，假设我们牺牲了这样的Cuboid，那么只要它的父亲Cuboid被物化，使用它的父亲Cuboid的开销就没那么大，因为父亲Cuboid并没有比它多很多行的记录。

从以上角度来说，Kylin的核心优势在于使用额外的空间存储预计算

的结果，以换取查询时间的缩减。而Cube的剪枝优化则是一种试图减少额外空间占用的方法，这种方法的前提是不会明显影响查询时间的缩减。在做剪枝优化的时候，需要选择跳过那些“多余”的Cuboid：有的Cuboid因为查询样式的原因永远不会被查询到，因此显得多余；有的Cuboid的能力和其他Cuboid接近，因此显得多余。但是Cube管理员无法提前甄别每一个Cuboid是否多余，因此Kylin提供了一系列简单的工具来帮助他们完成Cube的剪枝优化。

6.2 剪枝优化的工具

6.2.1 使用衍生维度

首先让我们观察以下这个维度表(Lookup Table)，如图6-3所示。

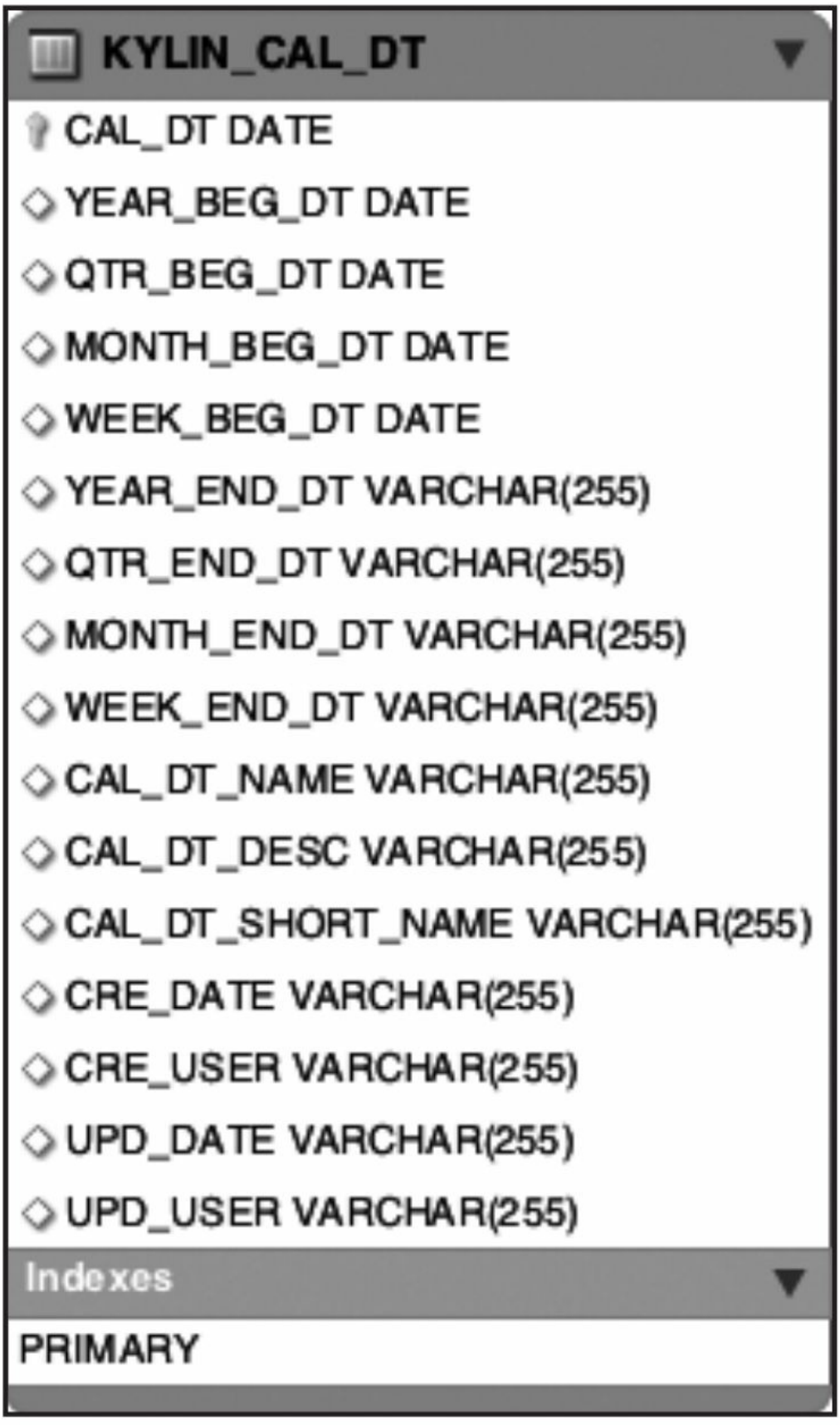

KYLIN_CAL_DT
CAL_DT DATE
YEAR_BEG_DT DATE
QTR_BEG_DT DATE
MONTH_BEG_DT DATE
WEEK_BEG_DT DATE
YEAR_END_DT VARCHAR(255)
QTR_END_DT VARCHAR(255)
MONTH_END_DT VARCHAR(255)
WEEK_END_DT VARCHAR(255)
CAL_DT_NAME VARCHAR(255)
CAL_DT_DESC VARCHAR(255)
CAL_DT_SHORT_NAME VARCHAR(255)
CRE_DATE VARCHAR(255)
CRE_USER VARCHAR(255)
UPD_DATE VARCHAR(255)
UPD_USER VARCHAR(255)
Indexes
PRIMARY

图6-3 一个维度表

这是一个常见的时间维度表，里面充斥着用途各异的时间维度，例如每个日期所处的星期、每个日期所处的月份等。这些维度可以被分析师灵活地用来进行各个时间粒度上的聚合分析，而不需要进行额外的上卷（Roll Up）操作。但是为了这个目的一下子引入这么多个维度，导致Cube中总共的Cuboid数量呈现爆炸式的增长往往是得不偿失的，所以在维度中只放入了这个维度表的主键（在底层实现中，我们更偏向使用事实表上的外键，因为在left joint的情况下事实表外键是维度表主键的超集），也就是只物化按日聚合的Cuboid。当用户需要以更高的粒度（比如按周、按月）来聚合时，如果在查询时获取按日聚合的Cuboid数据，并在查询引擎中实时地进行上卷操作，那么就达到了使用牺牲一部分运行时性能来节省Cube空间占用的目的。

Kylin将这样的理念包装成为了一个简单的优化工具——衍生维度。衍生维度用于在有效维度内将维度表上的非主键维度排除掉，并使用维度表的主键（其实是事实表上相应的外键）来替代它们。Kylin会在底层记录维度表主键与维度表其他维度之间的映射关系，以便在查询时能够动态地将维度表的主键“翻译”成这些非主键维度，并进行实时聚合。虽然听起来有些复杂，但是使用起来其实非常简单，在创建Cube的Cube designer的第二步，即添加维度的时候，要单击“Add Dimension”中的Derived而非Normal，如图6-4所示。

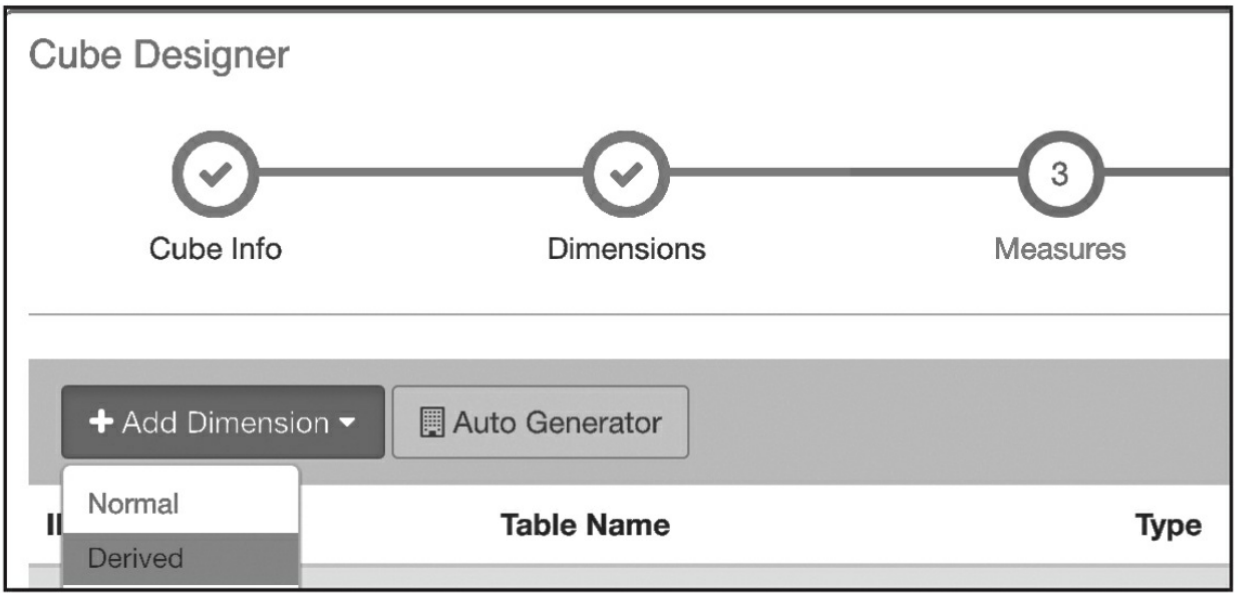


图6-4 创建维度

系统会弹出新的对话框让用户填写这批衍生维度的细节，一批衍生维度对应一个需要做衍生维度处理的维度表。在这里首先为本批衍生维度命名，例如“derived_date”，然后在下面选择相应的维度表，并且选择维度表上的非主键维度（如图6-5所示）。如果维度表上的某个非主键维度没有通过“+New Derived”选中，那么在查询时就不能包含它，因为系统不会保存维度表主键（CAL_DT）与这个维度之间的映射关系。例如，假设没有选择QTR_BEG_DT，那么带有QTR_BEG_DT的查询就会失败。

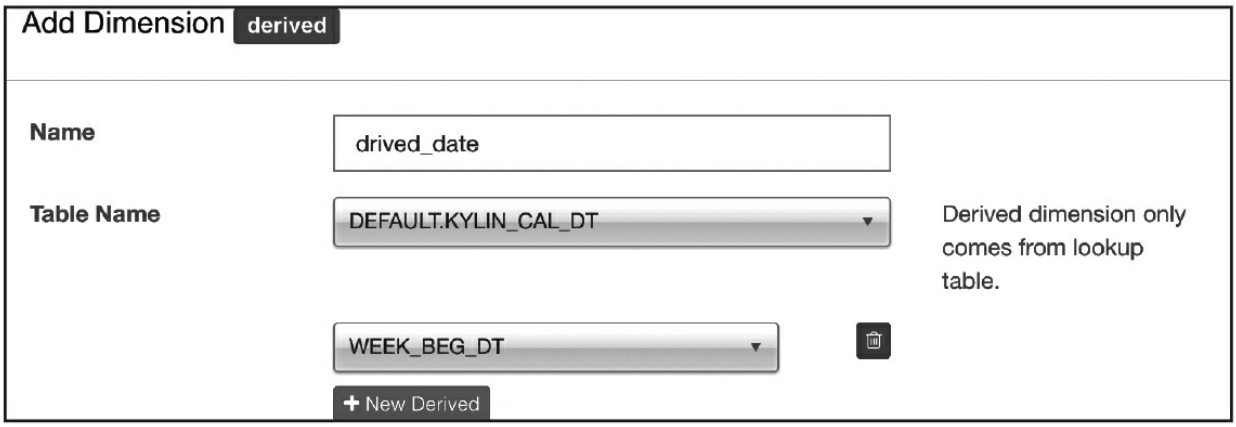


图6-5 添加衍生维度

选中的衍生维度在Cube中并没有直接存在，事实上如果我们前往Cube Designer的Advanced Setting，在Aggregation Groups和Rowkeys部分完全看不到这些衍生维度，我们甚至还会找不到这个维度表KYLIN_CAL_DT的主键，因为就如之前所描述的，我们实际上都是在用事实表上的外键作为这些衍生维度背后真正有效的维度，在前面的例子中，事实表与KYLIN_CAL_DT是通过以下的方式来连接的：

```
    Join Condition:
    DEFAULT.KYLIN_SALES.PART_DT = DEFAULT.
KYLIN_CAL_DT.CAL_DT
```

因此在Advanced Setting的Rowkeys部分就会看到PART_DT而看不到CAL_DT，更看不到那些KYLIN_CAL_DT上的衍生维度（如图6-6所示）。

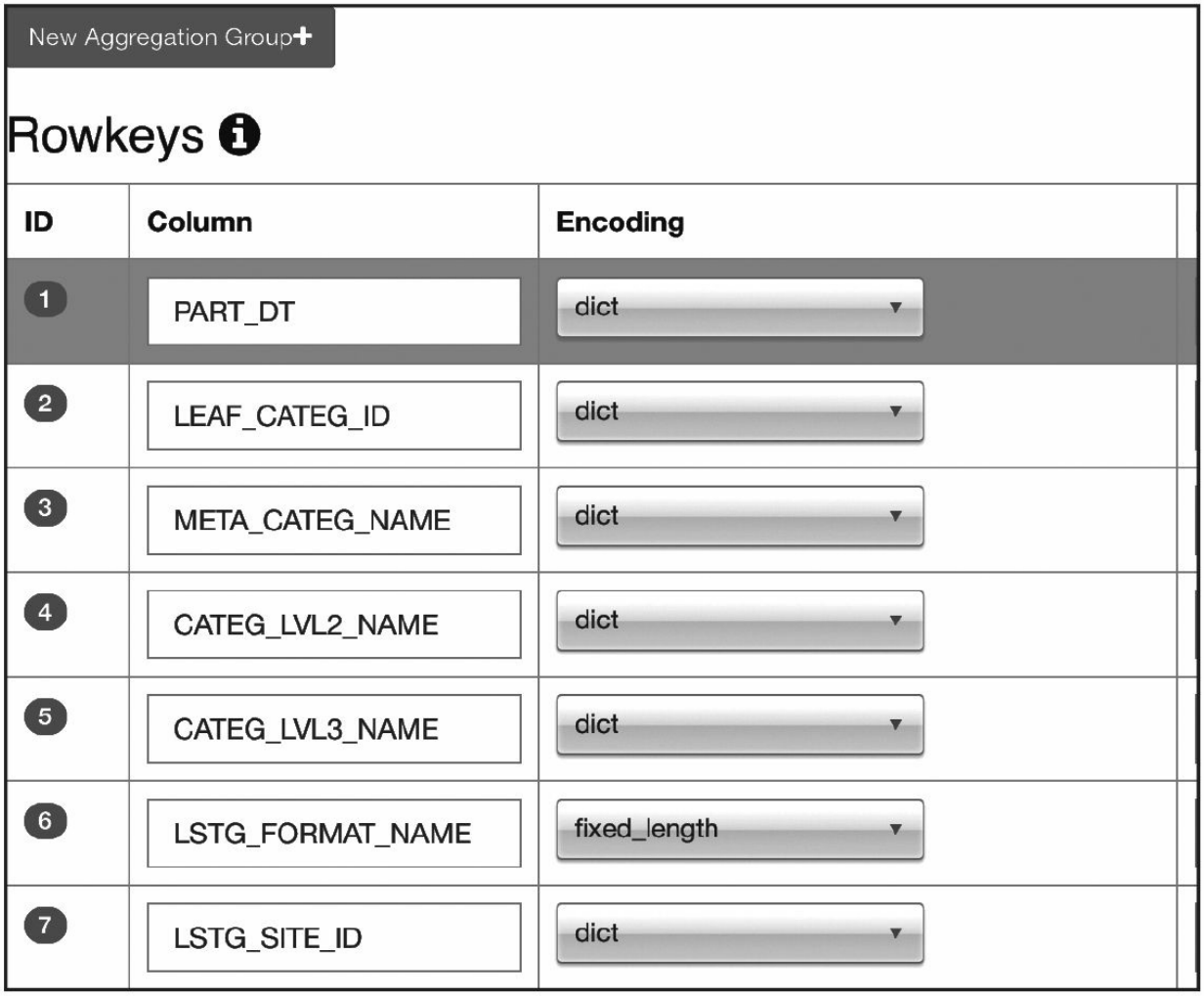


图6-6 Advanced Setting中的PART_DT外键

虽然衍生维度具有非常大的吸引力，但这也并不是说所有维度表上的维度都得变成衍生维度，如果从维度表主键到某个维度表维度所需要的聚合工作量非常大，例如从CAT_DT到YEAR_BEG_DT基本上需要365:1的聚合量，那么将YERR_BEG_DT作为一个普通的维度，而不是衍生维度可能是一种更好的选择。

·说明 衍生维度目前只用于衍生维度表上的维度，如果想要实现事

实表上的衍生，例如从事实表的user id维度衍生出user name列，那么请参考https://issues.apache.org/jira/browse/KYLIN-1313。

6.2.2 使用聚合组

聚合组（Aggregation Group）是一种更为强大的剪枝工具。聚合组假设一个Cube的所有维度均可以根据业务需求划分成若干组（当然也可以是一个组），由于同一个组内的维度更可能同时被同一个查询用到，因此会表现出更加紧密的内在关联。每个分组的维度集合均是Cube所有维度的一个子集，不同的分组各自拥有一套维度集合，它们可能与其他分组有相同的维度，也可能没有相同的维度。每个分组各自独立地根据自身的规则贡献出一批需要被物化的Cuboid，所有分组贡献的Cuboid的并集就成为了当前Cube中所有需要物化的Cuboid的集合。不同的分组有可能会贡献出相同的Cuboid，构建引擎会察觉到这点，并且保证每一个Cuboid无论在多少个分组中出现，它都只会被物化一次（如图6-7所示）。

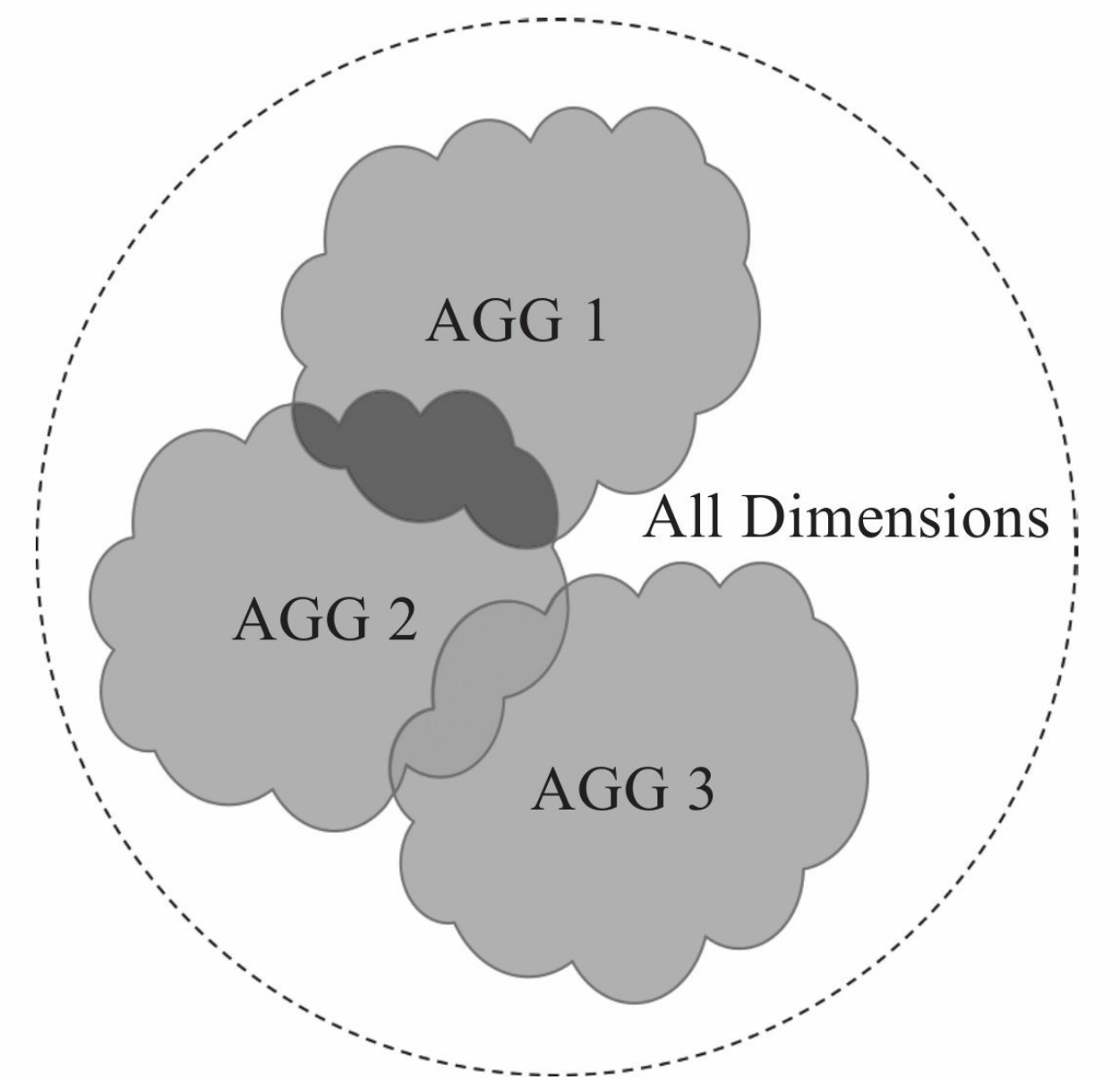


图6-7 聚合组重叠示意图

对于每个分组内部的维度，用户可以使用如下三种可选的方式定义它们之间的关系，具体如下。

·强制维度（Mandatory），如果一个维度被定义为强制维度，那么这个

分组产生的所有Cuboid中每一个Cuboid都会包含该维度。每个分组中都可以有0个、1个或多个强制维度。如果根据这个分组的业务逻辑，则相关的查询一定会在过滤条件或分组条件中，因此可以在该分组中把该维度设置为强制维度。

·层级维度(Hierarchy)，每个层级包含两个或更多个维度。假设一个层级中包含D1，D2…Dn这n个维度，那么在该分组产生的任何Cuboid中，这n个维度只会以()，(D1)，(D1，D2)…(D1，D2…Dn)这n+1种形式中的一种出现。每个分组中可以有0个、1个或多个层级，不同的层级之间不应当有共享的维度。如果根据这个分组的业务逻辑，则多个维度直接存在层级关系，因此可以在该分组中把这些维度设置为层级维度。

·联合维度(Joint)，每个联合中包含两个或更多个维度，如果某些列形成一个联合，那么在该分组产生的任何Cuboid中，这些联合维度要么一起出现，要么都不出现。每个分组中可以有0个或多个联合，但是不同的联合之间不应当有共享的维度(否则它们可以合并成一个联合)。如果根据这个分组的业务逻辑，多个维度在查询中总是同时出现，则可以在该分组中把这些维度设置为联合维度。

这些操作可以在Cube Designer的Advanced Setting中的Aggregation Groups区域完成，如图6-8所示。

图6-8 Advanced Setting中的Aggregation Groups

从图6-8中可以看到目前Cube中只有一个分组，单击左下角的“New Aggregation Group”可以添加一个新的分组。在某一个分组内，我们首先需要制定这个分组包含（Include）哪些维度，然后就可以进行强制维度、层级维度和联合维度的创建。除了Include选项，其他的三项都是可选的。

聚合组的设计非常灵活，甚至可以用来描述一些极端的设计。假设我们的业务需求非常单一，只需要某些特定的Cuboid，那么可以创建多个聚合组，每个聚合组代表一个Cuboid。具体的方法是在聚合组中先包含某个Cuboid所需的所有维度，然后把这些维度都设置为强制维度。这样当前的聚合组就只能产生我们想要的那一个Cuboid了。

再比如，有的时候我们的Cube中有一些基数非常大的维度，如果不做特殊处理，它就会和其他的维度进行各种组合，从而产生一大堆包含它的Cuboid。包含高基数维度的Cuboid在行数和体积上往往非常庞大，这

会导致整个Cube的膨胀率变大。如果根据业务需求知道这个高基数的维度只会与若干个维度(而不是所有维度)同时被查询到，那么就可以通过聚合组对这个高基数维度做一定的“隔离”。我们把这个高基数的维度放入一个单独的聚合组，再把所有可能会与这个高基数维度一起被查询到的其他维度也放进来。这样，这个高基数的维度就被“隔离”在一个聚合组中了，所有不会与它一起被查询到的维度都没有和它一起出现在任何一个分组中，因此也就不会有多余的Cuboid产生。这点也大大减少了包含该高基数维度的Cuboid的数量，可以有效地控制Cube的膨胀率。

6.3 并发粒度优化

当Segment中某一个Cuboid的大小超出一定的阈值时，系统会将该Cuboid的数据分片到多个分区中，以实现Cuboid数据读取的并行化，从而优化Cube的查询速度。具体的实现方式如下：构建引擎根据Segment估计的大小，以及参数“kylin.hbase.region.cut”的设置决定Segment在存储引擎中总共需要几个分区来存储，如果存储引擎是HBase，那么分区的数量就对应于HBase中的Region数量。kylin.hbase.region.cut的默认值是5.0，单位是GB，也就是说对于一个大小估计是50GB的Segment，构建引擎会给它分配10个分区。用户还可以通过设置kylin.hbase.region.count.min（默认为1）和kylin.hbase.region.count.max（默认为500）两个配置来决定每个Segment最少或最多被划分成多少个分区。

由于每个Cube的并发粒度控制不尽相同，因此建议在Cube Designer的Conf？iguration Overwrites中为每个Cube量身定制控制并发粒度的参数。在以下的例子中，将把当前Cube的kylin.hbase.region.count.min设置为2，kylin.hbase.region.count.max设置为100（如图6-9所示）。这样无论Segment的大小如何变化，它的分区数量最小都不会低于2，最大都不会超过100。相应地，这个Segment背后的存储引擎（HBase）为了存储这个Segment，也不会使用小于两个或超过100个的分区。我们还调整了默认的kylin.hbase.region.cut，这样50GB的Segment基本上会被分配到50个分区，

相比默认设置，我们的Cuboid可能最多会获得5倍的并发量。

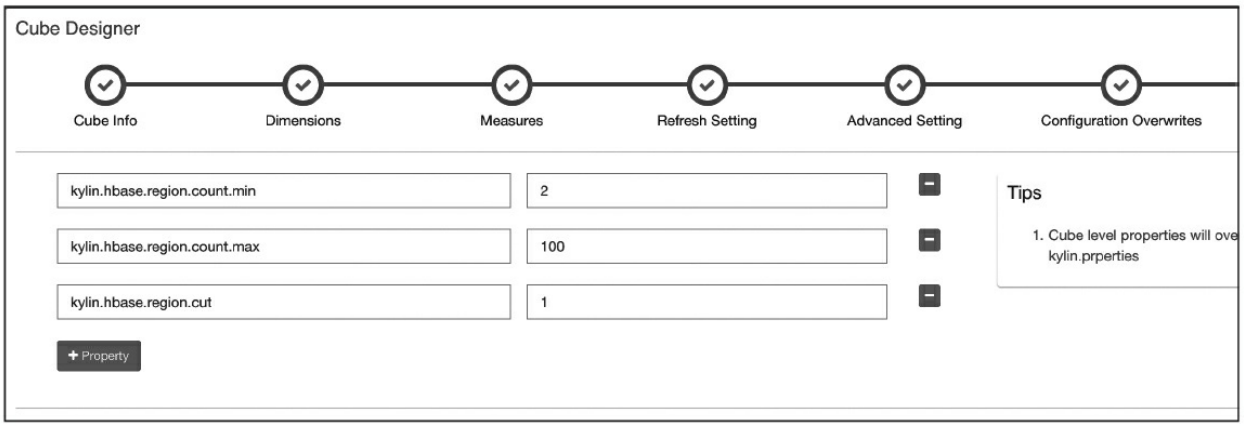


图6-9 设置Cube的并发度

6.4 Rowkeys优化

前面章节的侧重点是减少Cube中Cuboid的数量，尤其是减少那些包含高基数维度的Cuboid的数量，因为这类Cuboid的占用空间往往比其他Cuboid大很多。我们将以减少Cuboid数量为目的的Cuboid优化统称为Cuboid剪枝。本节将侧重通过Cube的Rowkeys方面的设置来优化Cube的查询性能。

Cube的每个Cuboid中都包含了大量的行，每个行又分为Rowkeys和Measure部分。每行Cuboid数据中的Rowkeys都包含当前Cuboid中所有维度值的组合。Rowkeys中的各个维度按照Cube Designer→Advanced Setting→RowKeys中定义的次序和编码进行组织（如图6-10所示）。

Rowkeys

ID	Column	Encoding	Length	Shard By
1	PART_DT	dict	0	false
2	LEAF_CATEG_ID	dict	0	false
3	META_CATEG_NAME	dict	0	false
4	CATEG_LVL2_NAME	dict	0	false
5	CATEG_LVL3_NAME	dict	0	false
6	LSTG_FORMAT_NAME	fixed_length	12	false
7	LSTG_SITE_ID	dict	0	false

图6-10 定义Rowkeys的次序

在Rowkeys设置区域中，每个维度都有几项关键的配置，下面将逐一

道来。

6.4.1 编码

编码(Encoding)代表了该维度的值应使用何种方式进行编码，合适的编码能够减少维度对空间的占用，例如，我们可以把所有的日期都用三个字节进行编码，相比于字符串存储，或者是使用长整数形式存储的方法，我们的编码方式能够大大减少每行Cube数据的体积。而Cube中可能存在数以亿计的行数，使用编码节约的空间累加起来将是一个非常巨大的数字。

目前Kylin支持的编码方式有以下几种。

·Date编码：将日期类型的数据使用三个字节进行编码，其支持从0000-01-01到9999-01-01中的每一个日期。

·Time编码：仅支持表示从1970-01-0100：00：00到2038-01-1903：14：07的时间，且Time-stamp类型的维度经过编码和反编码之后，会失去毫秒信息，所以说Time编码仅仅支持到秒。但是Time编码的优势是每个维度仅仅使用4个字节，这相比普通的长整数编码节约了一半。如果能够接受秒级的时间精度，请选择Time编码来代表时间的维度。

·Integer编码：Integer编码需要提供一个额外的参数“Length”来代表需要多少个字节。Length的长度为1~8。如果用来编码int32类型的整数，可以

将Length设为4；如果用来编码int64类型的整数，可以将Length设为8。在更多情况下，如果知道一个整数类型维度的可能值都很小，那么就能使用Length为2甚至是1的int编码来存储，这将能够有效避免存储空间的浪费。

·Dict编码：对于使用该种编码的维度，每个Segment在构建的时候都会为这个维度所有可能的值创建一个字典，然后使用字典中每个值的编号来编码。Dict的优势是产生的编码非常紧凑，尤其在维度值的基数较小且长度较大的情况下，特别节约空间。由于产生的字典是在查询时加载入构建引擎和查询引擎的，所以在维度的基数大、长度也大的情况下，容易造成构建引擎或查询引擎的内存溢出。

·Fixed_length编码：编码需要提供一个额外的参数“Length”来代表需要多少个字节。该编码可以看作Dict编码的一种补充。对于基数大、长度也大的维度来说，使用Dict可能不能正常工作，于是可以采用一段固定长度的字节来存储代表维度值的字节数组，该数组为字符串形式的维度值的UTF-8字节。如果维度值的长度大于预设的Length，那么超出的部分将会被截断。

在未来，Kylin还有可能为特定场景、特定类型的维度量身定制特别的编码方式，例如在很多行业，身份证号码可能就是一个重要的维度，但是身份证号码由于其具有特殊性而不能使用整数类型的编码（身份证最后一位可能是X），其高基数的特点也决定了不能使用dict编码，在目前的版本中只能使用f？ixed_length编码，但是显然f？ixed_length不能充分利

用身份证号码中大部分字节是数字的特性来进行深度编码，因此存在一定程度的浪费。

6.4.2 按维度分片

6.3节已经提到了系统可能会对Cuboid的数据进行分片处理。但是默认情况下Cuboid的分片策略是随机的，也就是说，我们无法控制Cuboid的哪些行会被分到同一个分片中。这种默认的方法固然能够提高读取的并发程度，但是它仍然有优化的空间。按维度分片（Shard by Dimension）提供了一种更加高效的分片策略，那就是按照某个特定维度进行分片。简单地说，如果Cuboid中某两个行的Shard by Dimension的值相同，那么无论这个Cuboid最终会被划分成多少个分片，这两行数据必然会被分配到同一个分片中。

这种分片策略对查询有着极大的好处。我们知道，Cuboid的每个分片都会被分配到存储引擎的不同物理机器上。Kylin在读取Cuboid数据的时候会向存储引擎的若干机器发送所读取的RPC请求。在RPC请求接收端，存储引擎会读取本机的分片数据，并在进行一定的预处理后再发送RPC回应（如图6-11所示）。以HBase存储引擎为例，不同的Region代表不同的Cuboid分片，在读取Cuboid数据的时候，HBase会为每个Region开启一个Coprocessor实例来处理查询引擎的请求。查询引擎将查询条件和分组条件作为请求参数的一部分发送到Coprocessor中，Coprocessor就能够在返回结果之前先对当前分片的数据做一定的预聚合（这里的预聚合不是Cube构建的预聚合，而是针对特定查询深度的预聚合）。

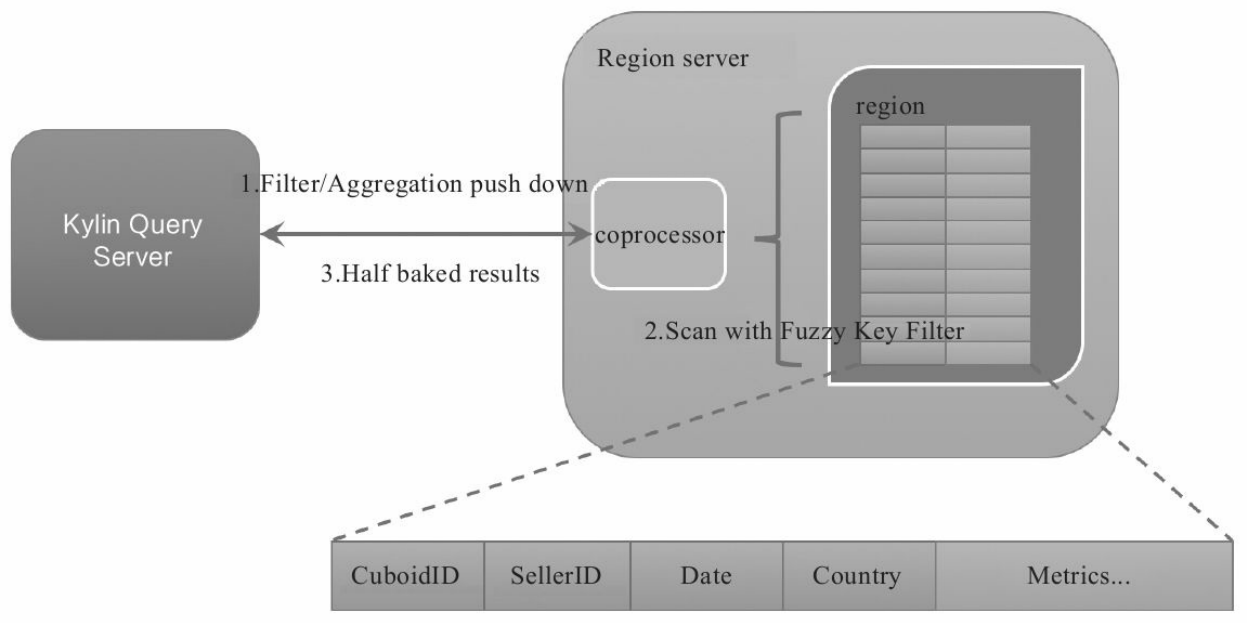


图6-11 存储引擎执行RPC查询

如果按照维度划分分片，假设按照一个基数比较高的维度seller_id进行分片，那么在这种情况下，每个分片将会承担一部分的seller_id，且各个分片不会有相同的seller_id。所有按照seller_id分组（Group by seller_id）的查询都会变得更加高效，因为每个分区预聚合的结果都会更加专注于某一些seller_id之上，使得分片返回的结果数量大大减少，查询引擎端也无需对各个分片的结果做分片间的聚合。按维度分片也能让过滤条件的执行更加高效，因为是按维度分片，所以每个分片的数据都会更加“整洁”，更方便查找和索引。

6.4.3 调整Rowkeys顺序

在Cube Designer→Advanced Setting→Rowkeys部分，我们可以上下拖动每一个维度来调节维度在Rowkeys中的顺序。这种顺序对于查询非常重要，因为在目前的实现中，Kylin会把所有的维度按照顺序黏合成一个完整的Rowkeys，并且按照这个Rowkeys升序排列Cuboid中所有的行（如图6-12所示）。

不难发现，如果在一个比较靠后的维度上有过滤条件，那么这个过滤条件的执行就会非常复杂。以目前的HBase存储引擎为例，Rowkeys对应HBase中的Rowkeys，是一段字节数组。目前没有创建单独的每个维度上的倒排索引，因此对于在比较靠后的维度上的过滤条件，只能依靠HBase的FuzzyKeyFilter来执行。尽管HBase做了大量相应的优化，但是因为是在对靠后的字节运用FuzzyKeyFilter，因此一旦前面维度的基数很大，那么FuzzyKeyFilter的寻找代价就会很高，执行效率就会变差。所以，在调整Rowkeys的顺序时需要遵守以下几个原则。

Dimensions					Metrics		
D1	D2	D3	D4	D5	M1	M2	M2
a1	b1	c1	d1	e1	100	200	300
a2	b2	c2	d2	e2	200	400	600
a1	xxx	c1	yyy	e1	1	1	1

Logical table for cuboid 00010101

Dimensions			Metrics		
D1	D3	D5	M1	M2	M2
a1	c1	e1	101	201	301
a2	c2	e2	200	400	600

Row Key	
00010101	a1,c1,e1
00010101	a2,c2,e2

Metric
101,201,301
200,400,600

HBase schema

图6-12 HBase中的Rowkeys存储

·在查询中被用作过滤条件的维度有可能放在其他维度的前面。

·将经常出现在查询中的维度放在不经常出现的维度的前面。

·对于基数较高的维度，如果查询会有这个维度上的过滤条件，那么

将它往前调整；如果没有，则向后调整。最好是结合6.2.2节的聚合组一起使用。

6.5 其他优化

6.5.1 降低度量精度

有一些度量具有多种可选精度，但是精度越高的度量往往越会存在一定的代价，它意味着更大的占用空间和运行时开销。以近似值的Count Distinct度量为例，Kylin提供了多种可选精度，现挑选其中的几种进行对比，见表6-1。

表6-1 Count Distinct精度和占用空间列表

Count Distinct 类型	精确度	每行占用空间
HLLC 10	< 9.75%	1KB
HLLC 12	< 4.88%	4KB
HLLC 14	< 2.44%	16KB
HLLC 15	< 1.72%	32KB
HLLC 16	< 1.22%	64KB

可以看到，精度最大的类型比精度最小的类型多出64倍的空间占用，而即使精度最小的Count Distinct度量也已经非常占用空间了。因此，当业务可以接受较低一些的精度时，用户应当考虑到Cube空间方面的影响，尽量选择小精度的度量。

6.5.2 及时清理无用的Segment

在第3章中已经提到过，随着增量构建出来的Segment的慢慢累积，Cube的查询性能将会变差，因为每次跨Segment的查询都需要从存储引擎中读取每一个Segment的数据，并且在查询引擎中对不同Segment的数据做进一步的聚合，这对于查询引擎和存储引擎来说都是巨大的压力。从这个角度来说，及时地使用第3章介绍的Segment碎片清理方法，有助于提高Cube的使用效率。

6.6 小结

本章从多个角度介绍了Cube优化的方法：从Cuboid剪枝的角度，从并发粒度控制的角度，从Rowkeys设计的角度，还有从度量精度选择的角度。总的来说，Cube优化需要Cube管理员对Kylin有较为深刻的理解和认识，这也在无形中提高了使用和管理Kylin的门槛。为此，我们希望在将来的版本中能够通过机器学习，以及对数据分布和查询样式的分析方法，自动化一部分优化操作，帮助用户更加容易地管理Kylin中的数据。

第7章　应用案例分析

前面的章节已经介绍了Apache Kylin的基本工作原理，包括如何在Hive或Kafka中准备数据源，如何根据业务需求设计星形数据模型，如何基于数据模型设计和优化Cube，以及不同的Cube构建算法、SQL查询和可视化的多种接口。本章将结合应用案例，介绍Apache Kylin在真实业务场景中的详细使用过程和优化方案，以期读者在实现自己的业务需求时能够有所帮助。案例中所用到的数据集和工具都包含在Apache Kylin的二进制包中，感兴趣的读者可以自行下载，并参照本书的介绍动手试验。

7.1 基本多维分析

在Apache Kylin官方提供的二进制包中，包括一份基于销售业务分析应用场景的样例数据，可帮助用户快速体验Apache Kylin的功能。本节将基于这一数据集，为读者介绍从零开始到最终实现SQL快速查询的全部过程。

7.1.1 数据集

Apache Kylin官方提供的这一份样例数据集规模并不大，总共仅有1MB左右，共计3张表，其中事实表有10？000条数据。因为数据规模较小，所以在虚拟机中进行快速实践和操作很方便。读者可以自行搭建Hadoop Sandbox的虚拟机并快速部署Apache Kylin，然后导入该数据集进行试验。有关Apache Kylin快速安装部署的方法，请参考本书10.2节的内容。

在前面的章节中已经提到过，Apache Kylin仅支持星形的数据模型。这里用到的样例数据集就是一个规范的星形模型结构，它总共包含了如下3个数据表。

（1）KYLIN_SALES

该表是事实表，保存了销售订单的明细信息。每一列均保存了卖家、商品分类、订单金额、商品数量等信息，每一行对应着一笔交易订单。

（2）KYLIN_CATEGORY_GROUPINGS

该表是维表，保存了商品分类的详细介绍，例如商品分类名称等。

（3）KYLIN_CAL_DT

该表是维表，保存了时间的扩展信息。如单个日期所在的年始、月始、周始、年份、月份等。

这三张表一起构成了整个星形模型的结构，图7-1是实例-关系图（因篇幅有限，图7-1中未列出表上的所有列）。

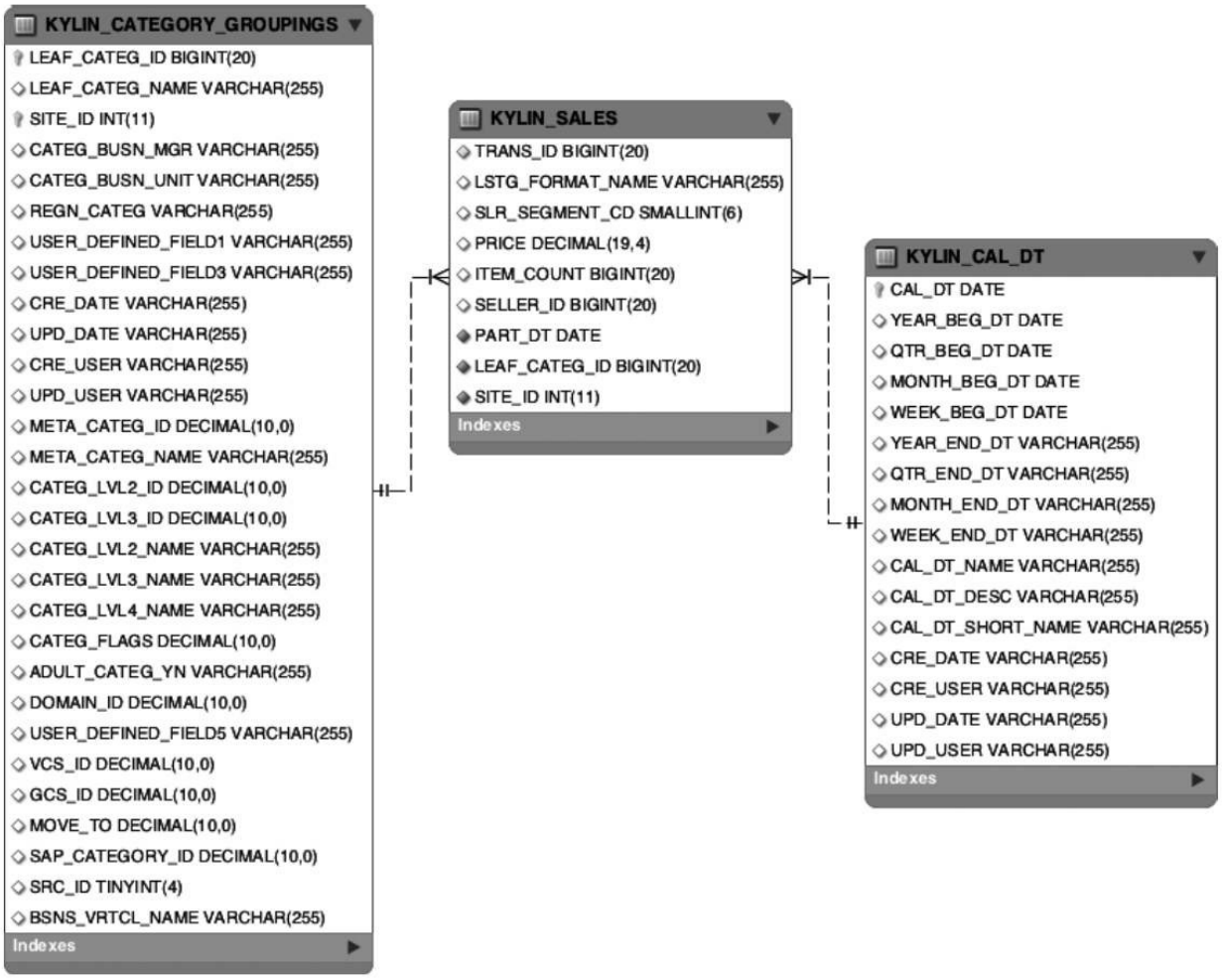


图7-1 数据模型E-R图

在介绍完三张表之间的关系之后，再对表中的一些关键字段进行介绍，见表7-1。

表7-1 数据表字段介绍

表	字　段	意　义
KYLIN_SALES	PART_DT	订单日期
KYLIN_SALES	LEAF_CATEG_ID	商品分类 ID

（续）

表	字　段	意　义
KYLIN_SALES	SELLER_ID	卖家 ID
KYLIN_SALES	PRICE	订单金额
KYLIN_SALES	ITEM_COUNT	购买商品个数
KYLIN_SALES	LSTG_FORMAT_NAME	订单交易类型
KYLIN_CATEGORY_GROUPINGS	USER_DEFINED_FIELD1	用户定义字段 1
KYLIN_CATEGORY_GROUPINGS	USER_DEFINED_FIELD3	用户定义字段 3
KYLIN_CATEGORY_GROUPINGS	UPD_DATE	更新日期
KYLIN_CATEGORY_GROUPINGS	UPD_USER	更新负责人
KYLIN_CATEGORY_GROUPINGS	META_CATEG_NAME	一级分类
KYLIN_CATEGORY_GROUPINGS	CATEG_LVL2_NAME	二级分类
KYLIN_CATEGORY_GROUPINGS	CATEG_LVL3_NAME	三级分类
KYLIN_CAL_DT	CAL_DT	日期
KYLIN_CAL_DT	WEEK_BEG_DT	周始日期

到这里，读者对案例所用的数据集应该有一个详细的了解了。接下来，我们开始使用Apache Kylin对这些数据进行真实的操作吧！

7.1.2 数据导入

首先，把数据导入Hive，作为Hive表。部署好Apache Kylin实例之后，在Apache Kylin安装目录的bin文件夹中，有一个可执行脚本，用户可以调用它把样例数据导入Hive中，如下：

```
$KYLIN_HOME/bin/sample.sh
```

脚本执行成功之后，建议读者执行Hive命令行，以确认这些数据已经导入成功，命令如下：

```
hive
hive> show tables;
OK
kylin_cal_dt
kylin_category_groupings
kylin_sales
Time taken: 0.127 seconds, Fetched: 3 row(s)
hive> select count(*) from kylin_sales;
Query ID = root_20160707221515_b040318d-1f08-44ab-b337-d1f858c46d7d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
    set hive.exec.reducers.bytes.per.reducer=<number>
```

```
    In order to limit the maximum number of reducers:
        set hive.exec.reducers.max=<number>
    In order to set a constant number of reducers:
        set mapreduce.job.reduces=<number>
    Starting Job = job_1467288198207_0129, Tracking URL = http://sandbox.
hortonworks.com:8088/proxy/application_1467288198207_0129/
    Kill Command = /usr/hdp/2.2.4.2-2/hadoop/bin/hadoop job  -kill
job_1467288198207_0129
    Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
    2016-07-07 22:15:11,897 Stage-1 map = 0%,  reduce = 0%
    2016-07-07 22:15:17,502 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.64 sec
    2016-07-07 22:15:25,039 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.37 sec
    MapReduce Total cumulative CPU time: 3 seconds 370 msec
    Ended Job = job_1467288198207_0129
    MapReduce Jobs Launched:
    Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.37 sec   HDFS Read: 505033
HDFS Write: 6 SUCCESS
    Total MapReduce CPU Time Spent: 3 seconds 370 msec
    OK
    10000
    Time taken: 24.966 seconds, Fetched: 1 row(s)
```

然后，打开Apache Kylin的Web UI，在如图7-2所示的页面中创建一个新的项目（Project），并命名为Kylin_Sample_1。

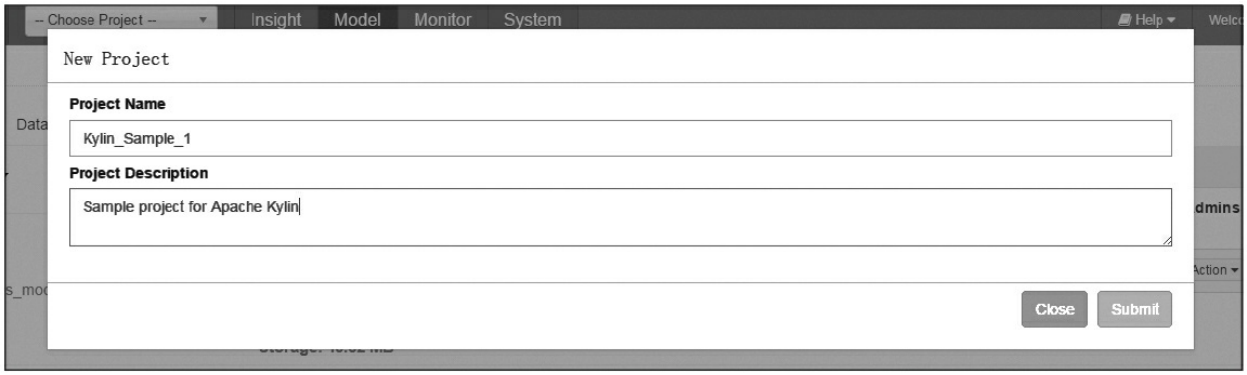


图7-2 创建新项目

在Web UI的左上角选择刚刚创建的项目（如图7-3所示），表示接下来的全部操作都在这个项目中，并且在当前项目中进行的操作不会对其他项目产生影响。

图7-3 选择项目

最后，把Hive数据表同步到Kylin当中，为了方便操作，我们通过“Load Hive Table Metadata From Tree”的方式进行同步（如图7-4所示）。

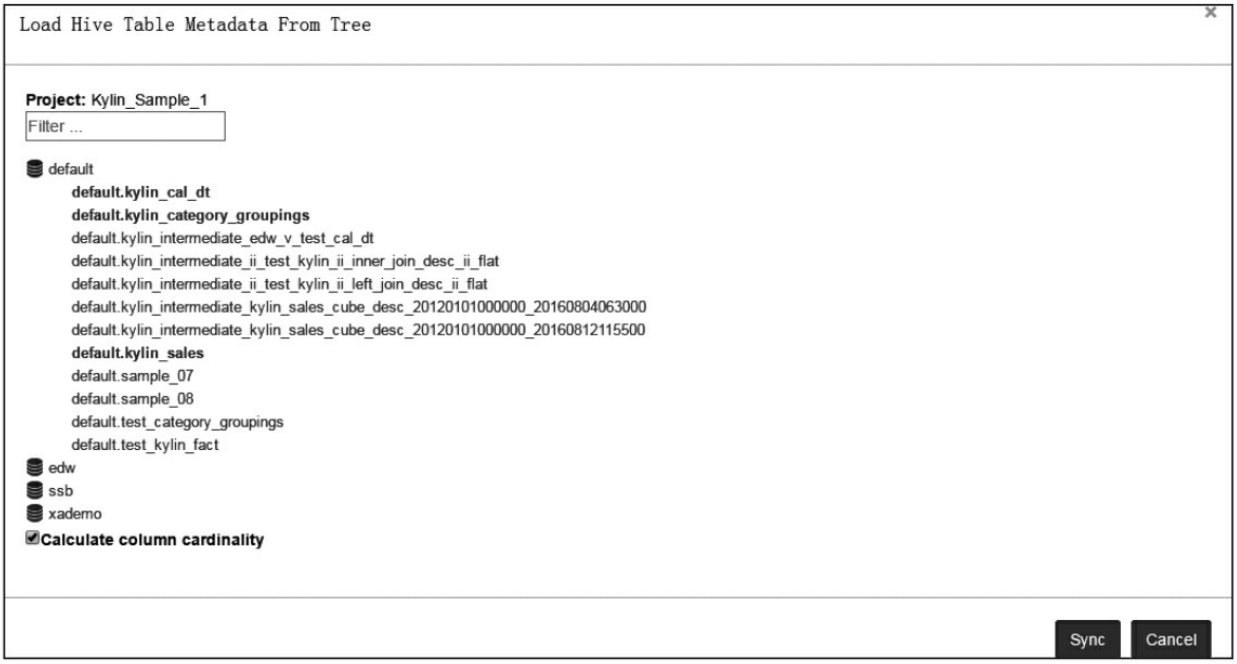


图7-4 同步Hive表

导入后系统会自动计算各表各列的维数，以掌握数据的基本情况。

稍等几分钟后，就可以通过数据源表的详情页查看这些信息了（如图7-5所示）。

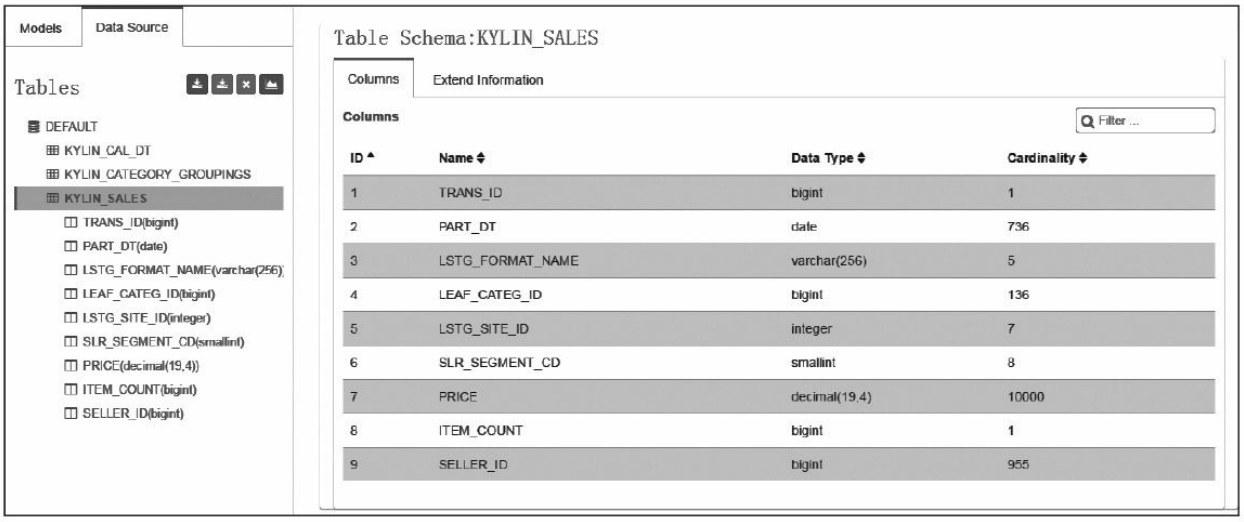


图7-5 查看Hive表信息

7.1.3 创建数据模型

在数据源就绪的基础之上，现在可以开始创建数据模型了。从图7-1所示的表间关系可以看出，该模型包含1个事实表和2个维表，表间通过外键进行关联。实际上，并不是表上所有的字段都有被分析的需要，因此我们可以有目的地仅选择所需要的字段添加到数据模型中；然后，根据分析师用户的具体分析场景，把这些字段设置为维度或度量。

下面来看一下在Apache Kylin中是如何创建数据模型的。操作前从Web UI的左上角项目列表中选择刚刚创建的Kylin_Sample_1项目，然后进入Models页面（如图7-6所示）。

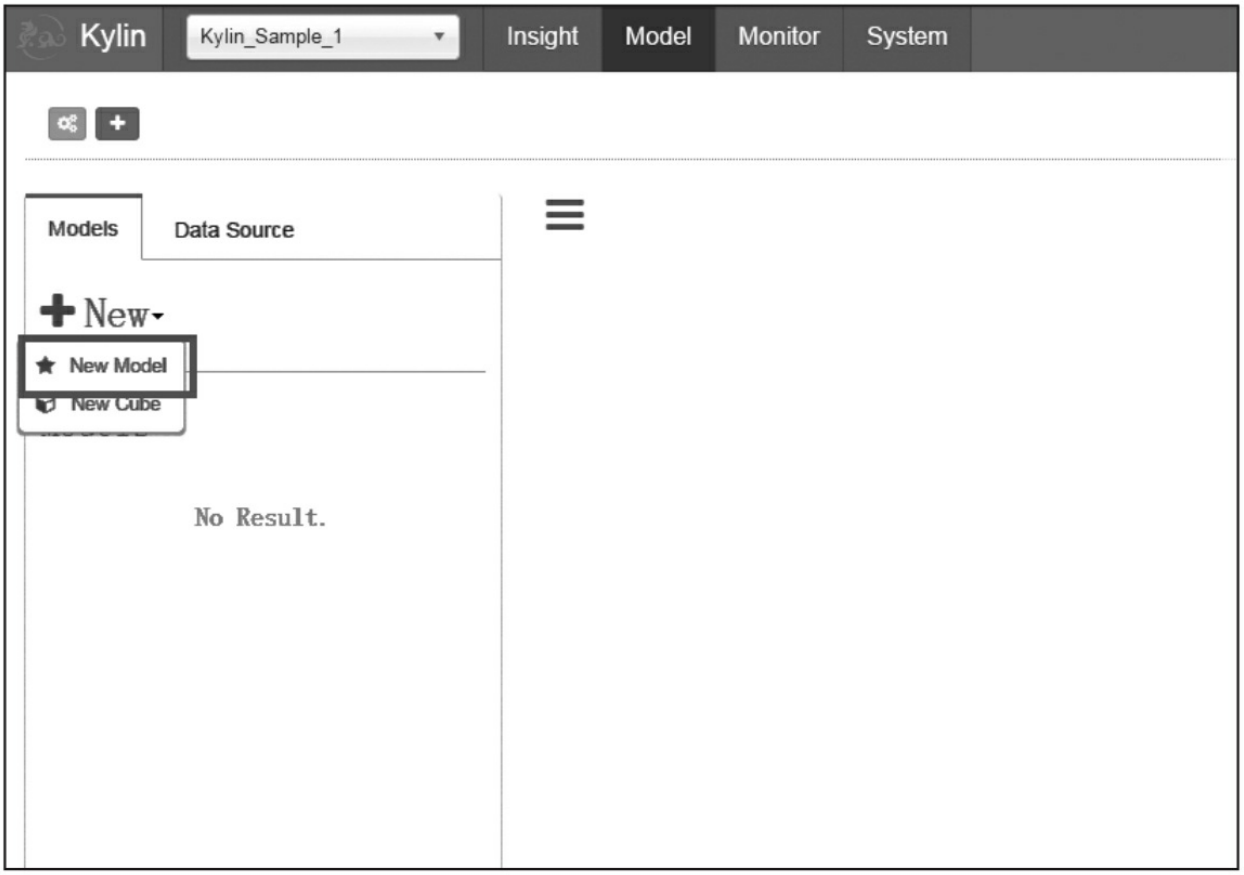


图7-6 新建数据模型(1)

第一步，创建一个数据模型(Data Model)，并命名为Sample_Model_1，然后单击下一步(如图7-7所示)。

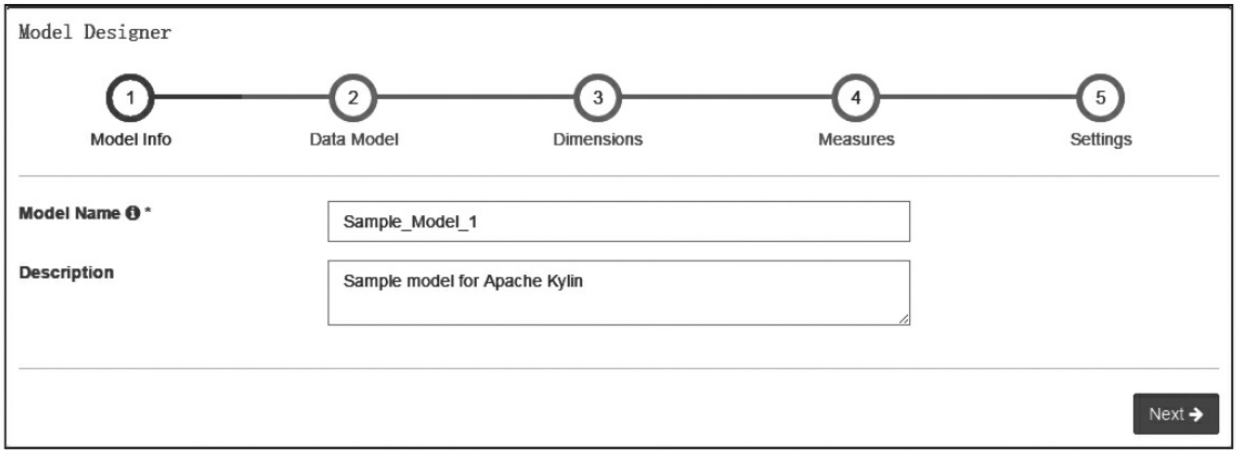


图7-7 新建数据模型(2)

第二步，为模型选择事实表(Fact Table)和维表(Lookup Table)。根据星形模型结构，选择KYLIN_SALES为事实表，然后添加KYLIN_CAL_DT和KYLIN_CATEGORY_GROUPINGS作为维表，并设置好连接条件。

(1)KYLIN_CAL_DT

连接类型:Inner

连接条件:
DEFAULT.KYLIN_SALES.PART_DT=DEFAULT.KYLIN_CAL_DT.CAL_DT

(2)KYLIN_CATEGORY_GROUPINGS

连接类型:Inner

连接条件:
DEFAULT.KYLIN_SALES.LEAF_CATEG_ID=DEFAULT.KYLIN_CATEGOR

图7-8是设置好之后的界面，在此显示出以供参考。

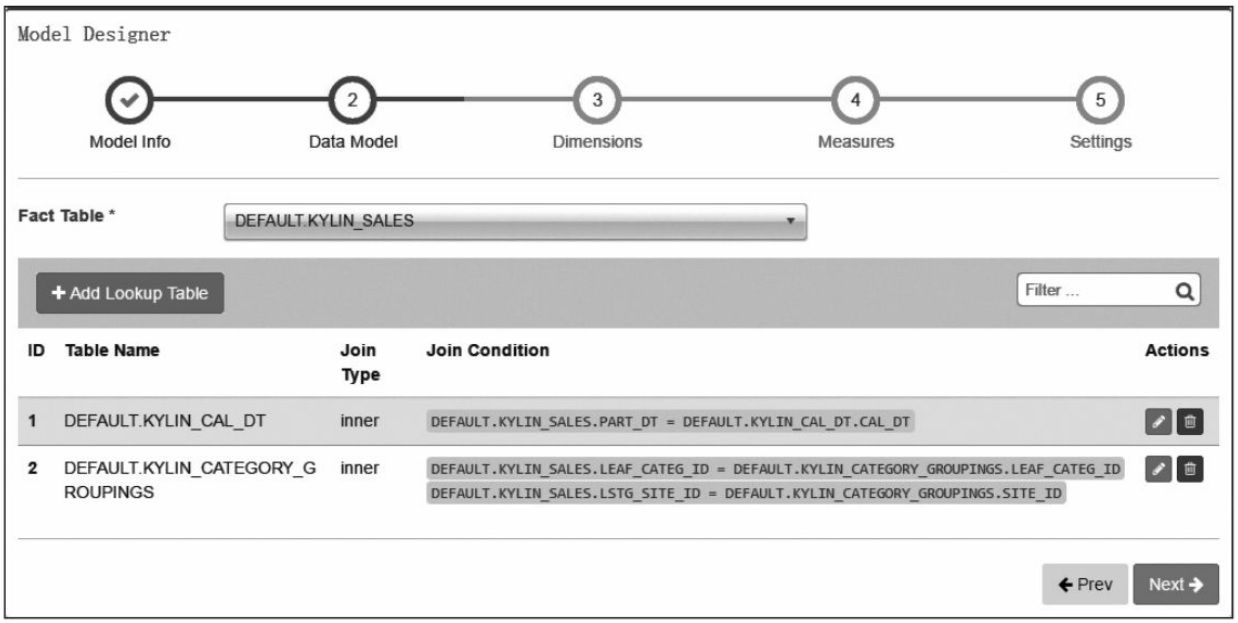


图7-8 新建数据模型(3)

第三步，从上一步添加的事实表和维表中选择需要作为维度的字段。一般会把时间当作过滤条件，所以这里首先添加时间字段。此外，再添加商品分类、卖家ID等字段作为维度，具体情况如图7-9所示。

第四步，根据业务需要，从事实表上选择衡量指标的字段作为度量。例如，PRICE字段用来衡量销售额，ITEM_COUNT字段用来衡量商品销量，SELLER_ID用来衡量卖家的销售能力等，最终结果如图7-10所示。

第五步，设置时间分段。一般来说，销售数据都是与日俱增的，每天都会有新数据通过ETL到达Hive中，因此需要选择通过增量构建的方式构建Cube，故而选择用于分段的时间字段DEFAULT.KYLIN_SALES.PART_DT作为时间分割列。根据样例数据可以

看到，这一列时间的格式是yyyy-MM-dd，所以选择对应的日期格式。此外，我们既不需要设置单独的时间分区列，也不需要添加固定的过滤条件，设置效果如图7-11所示。

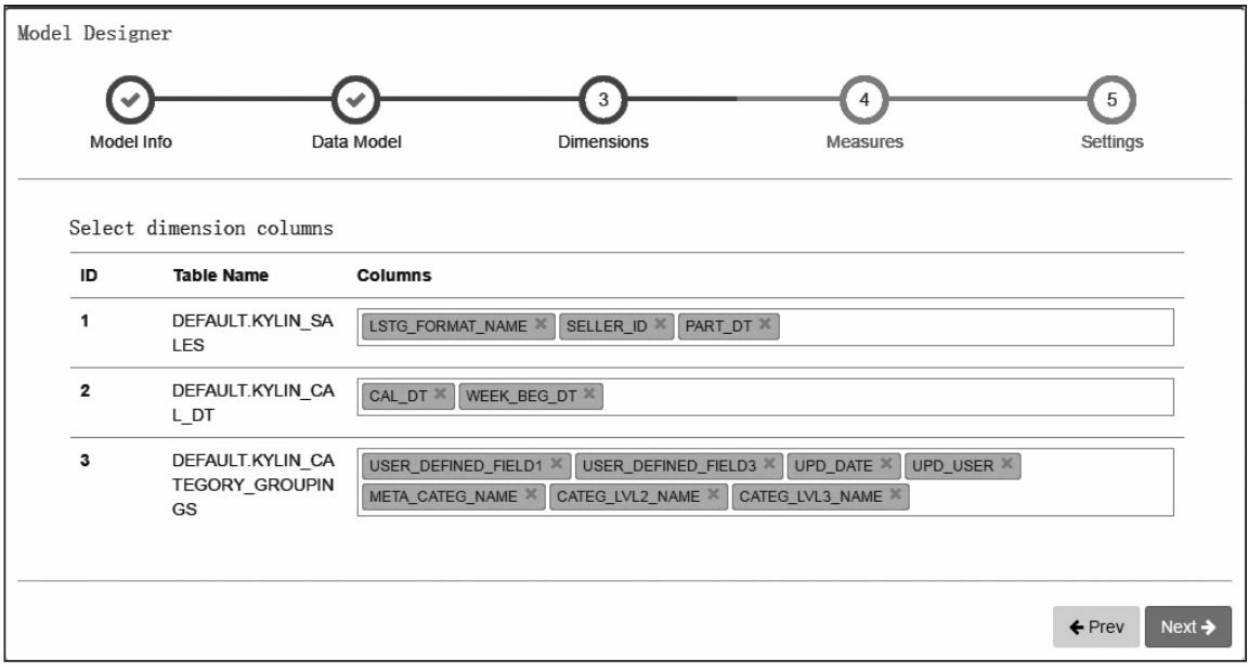


图7-9 新建数据模型(4)

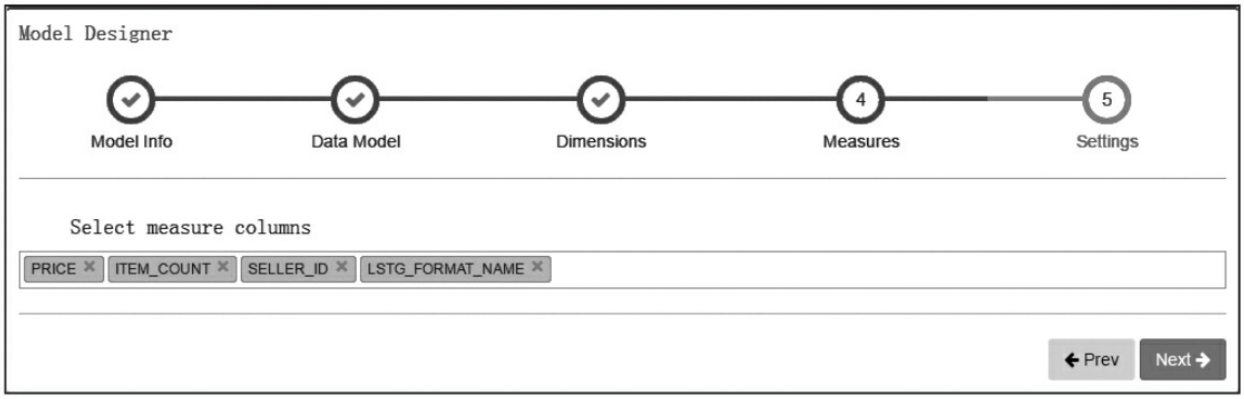


图7-10 新建数据模型(5)

最后，单击保存(Save)按钮，到此数据模型就创建成功了。

7.1.4 创建Cube

在创建好数据模型之后，还需要根据查询需求定义度量的预计算类型、维度的组合等，这个过程就是Cube的设计过程。

在Apache Kylin的Web UI中，首先需要选择Kylin_Sample_1项目，跳转到Models页面，然后按照图7-12所示的内容创建一个Cube。

第一步，在“Model Name”中选择上一过程创建好的数据模型Kylin_Sample_Model_1，输入新建Cube的名称Kylin_Sample_Cube_1，其余字段保持默认，然后单击“Next”（如图7-13所示）。

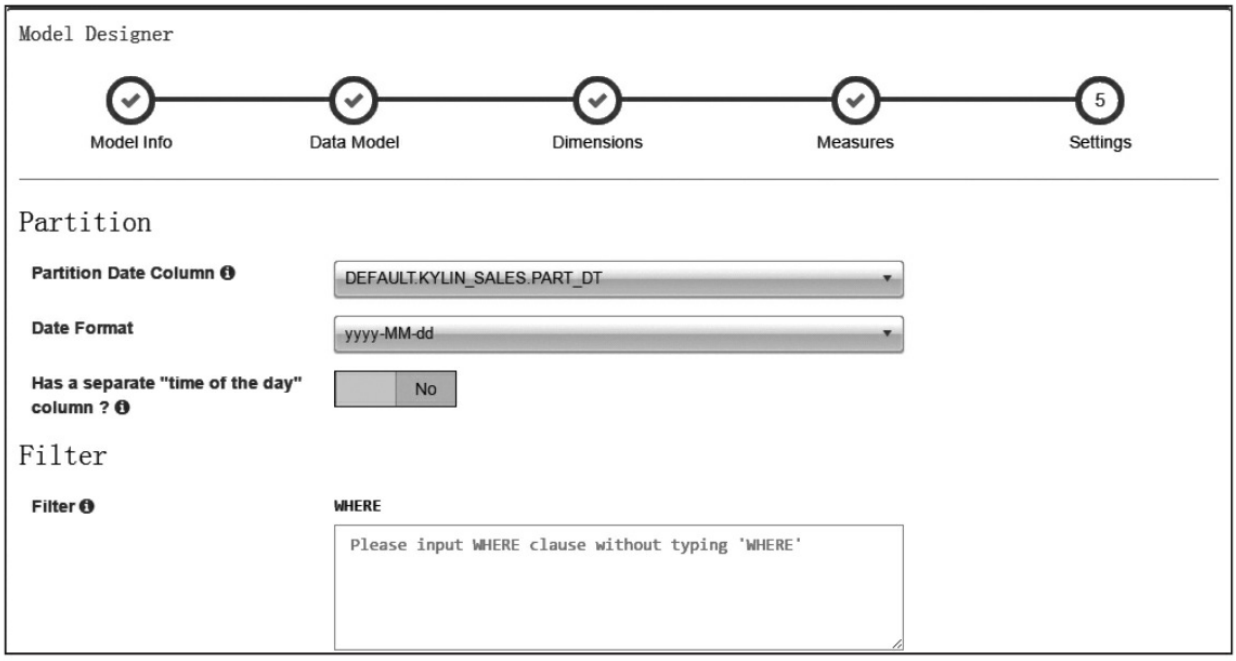


图7-11 新建数据模型(6)

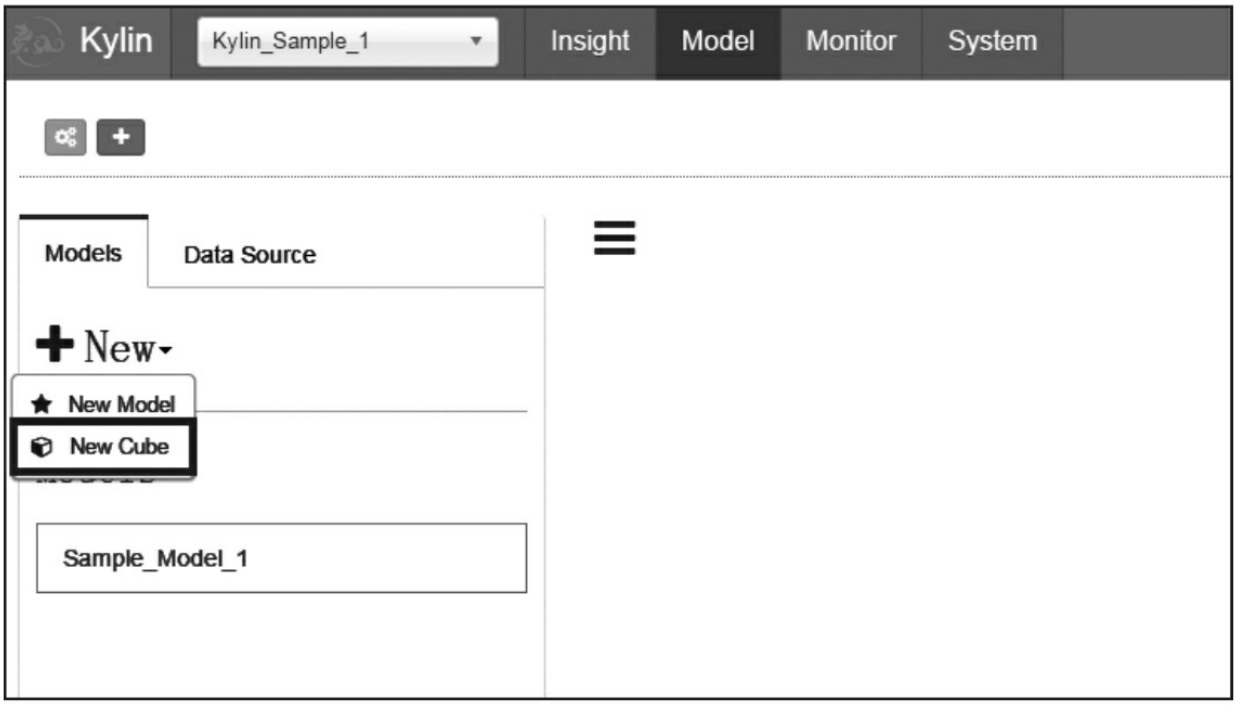


图7-12 新建Cube(1)

第二步，从数据模型的维度中选择一些列作为Cube的维度。这里的设置会影响到生成的Cuboid数量，进而影响Cube的数据量大小。

在KYLIN_CATEGORY_GROUPINGS表里，和商品分类相关的三个字段(META_CATEG_NAME、CATEG_LVL2_NAME、CATEG_LVL3_NAME)都可能出现在过滤条件中，我们先把它们添加为普通类型维度(Normal Dimension)。因为从维表上添加普通维度不能通过自动生成器(Auto Generator)生成，因此这里采用手动添加的方式，具体过程如下。

1)单击添加维度按钮(Add Dimension)，然后选择普通类型

(Normal)。

2)针对每一个维度字段，首先在Name输入框中输入维度名称，在Table Name中选择KYLIN_CATEGORY_GROUPINGS表，然后在Column Name中选择相应的列名。

此外，在查询中经常会把时间作为过滤或聚合的条件，如按周过滤、按周聚合等。这里以按周过滤为例，需要用到KYLIN_CAL_DT中的WEEK_BEG_DT字段，但是该字段实际上可以由PART_DT字段来决定，即根据每一个PART_DT值对应出一个WEEK_BEG_DT字段，因此，我们添加WEEK_BEG_DT字段为衍生维度(Derived)。

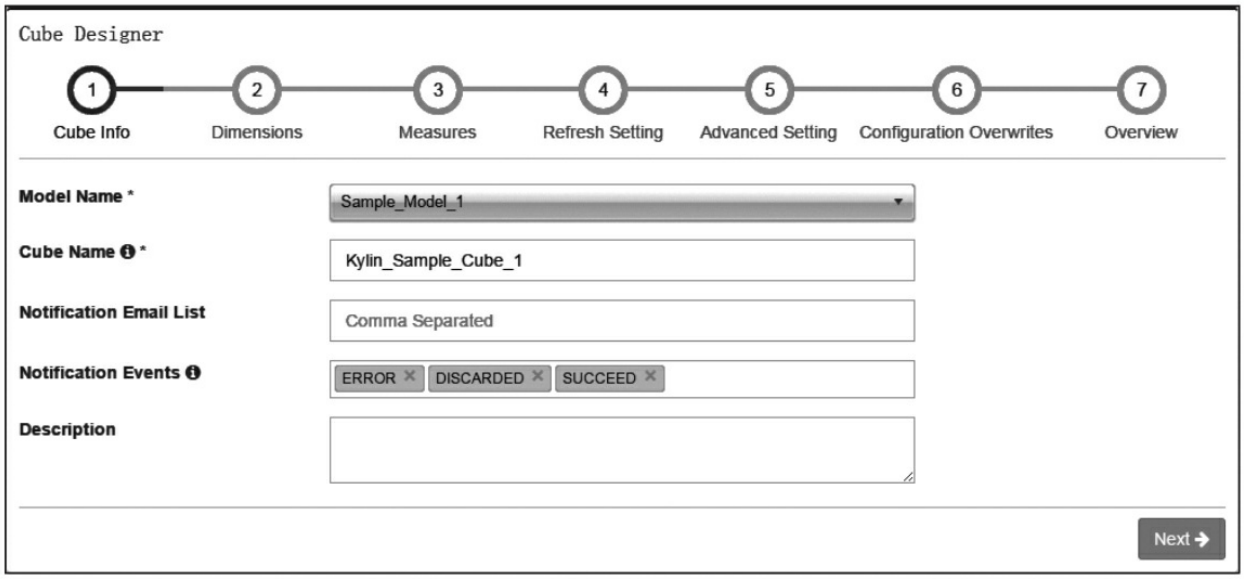


图7-13 新建Cube(2)

同样的，DEFAULT.KYLIN_CATEGORY_GROUPINGS表中还有一些可作为衍生维度的字段，如USER_DEFINED_FIELD1、

USER_DEFINED_FIELD3、UPD_DATE、UPD_USER等。

在事实表上，表征交易类型的LSTG_FORMAT_NAME字段也会用于过滤或聚合条件，因此，我们再添加LSTG_FORMAT_NAME字段作为普通维度。

最终，维度的设置结果如图7-14所示。

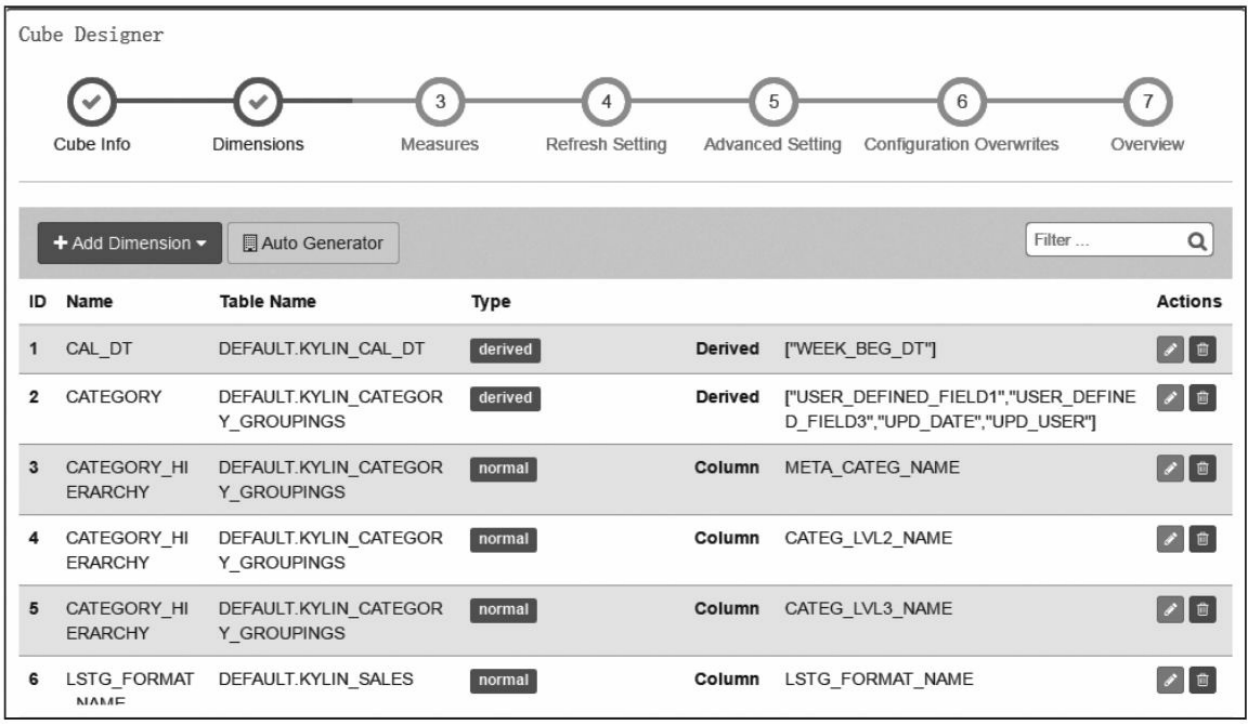


图7-14 新建Cube（3）

第三步，根据数据分析中的聚合需求，为Cube定义度量的聚合形式。默认情况下，系统会自动创建一个COUNT()聚合，用于考量交易订单的数量。在这个案例中，还需要通过PRICE的不同聚合形式考量销售额，如总销售额为SUM(PRICE)、最高订单金额为MAX(PRICE)、最低订单金额

为MIN(PRICE)。因此，我们手动创建三个度量，分别选择聚合表达式为SUM、MIN、MAX，并选择PRICE列作为目标列(如图7-15所示)。

其次，我们还需要通过COUNT(DISTINCT SELLER_ID)考量卖家个数。根据前面章节的介绍，Apache Kylin默认使用HyperLogLog算法进行COUNT_DISTINCT的计算，该算法是个近似算法，在创建度量时需要选择一个近似度，本案例对精确性的要求不高，为了提升查询性能，这里选择精度较低的“Error Rate<9.75%”。同样，再创建一个COUNT(DISTINCT LSTG_FORMAT_NAME)的度量考量不同条件下的交易类型(如图7-16所示)。

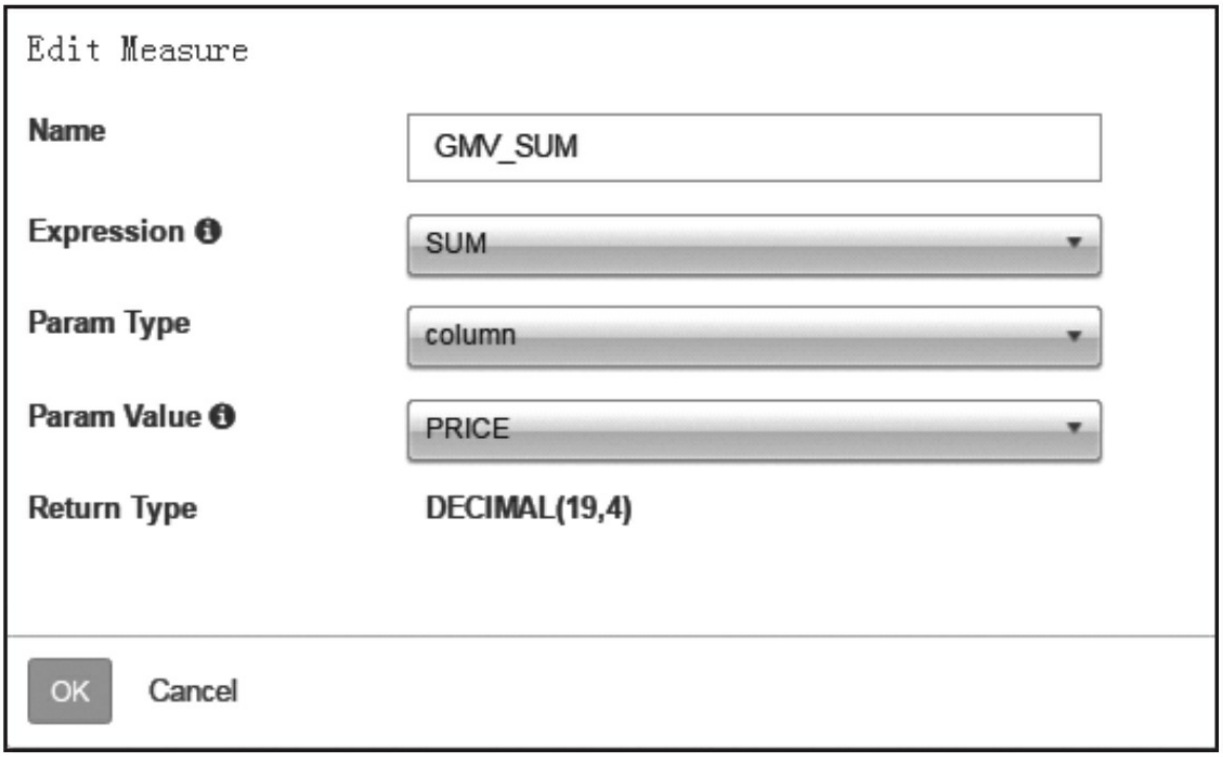


图7-15 新建Cube(4)

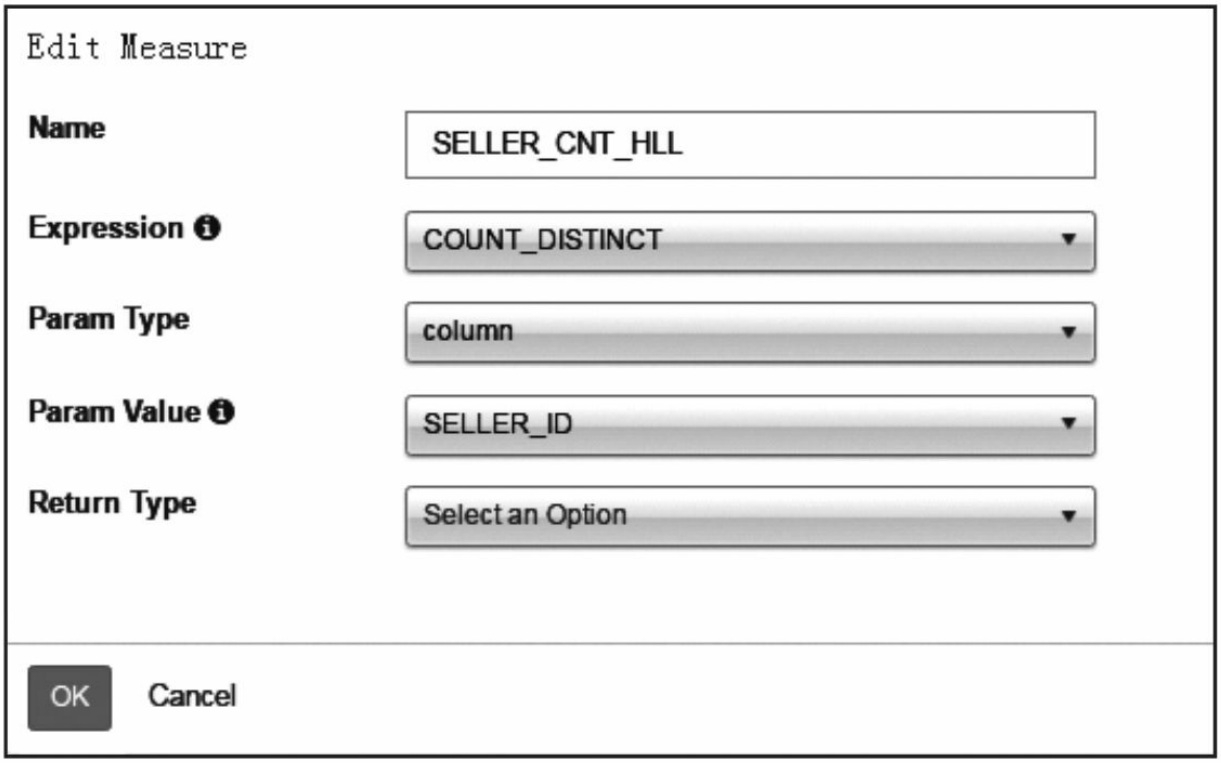


图7-16 新建Cube(5)

在销售业务分析的场景中，往往需要挑选出销售业绩最好的商家，这时候就需要用到TOP-N的度量了。在这个例子中，我们会选出SUM(PRICE)最高的一些SELLER_ID，实际上就是执行如下的SQL语句：

```
SELECT SELLER_ID, SUM(PRICE) FROM KYLIN_SALES
GROUP BY SELLER_ID
ORDER BY SUM(PRICE) DESC
```

因此，我们创建一个TOP-N的度量，选择PRICE字段作为SUM/ODER BY字段，选择SELLER_ID字段作为GROUP BY字段，并选择TOPN(100)作为度量的精度(如图7-17所示)。

最终添加的度量如图7-18所示。

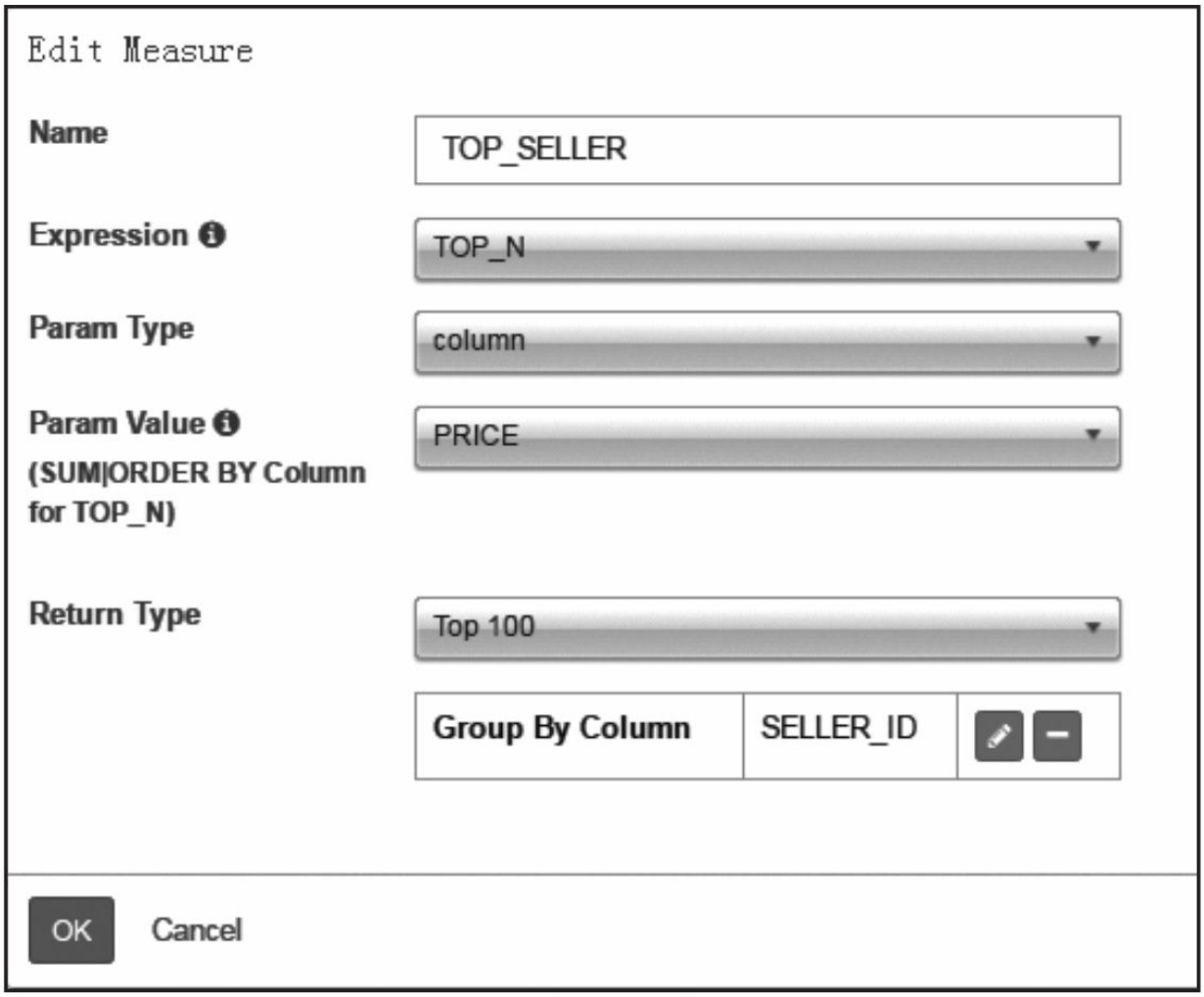


图7-17 新建Cube(6)

第四步，对Cube的构建和维护进行配置。一般情况下，一个销售统计的SQL查询往往会按月、周进行过滤和聚合，所以我们可以设置Cube自动按周、月进行自动合并，即每7天进行一次合并，每4周(28天)进行一次合并，设置自动合并阈值(Auto Merge Thresholds)，如图7-19所示。

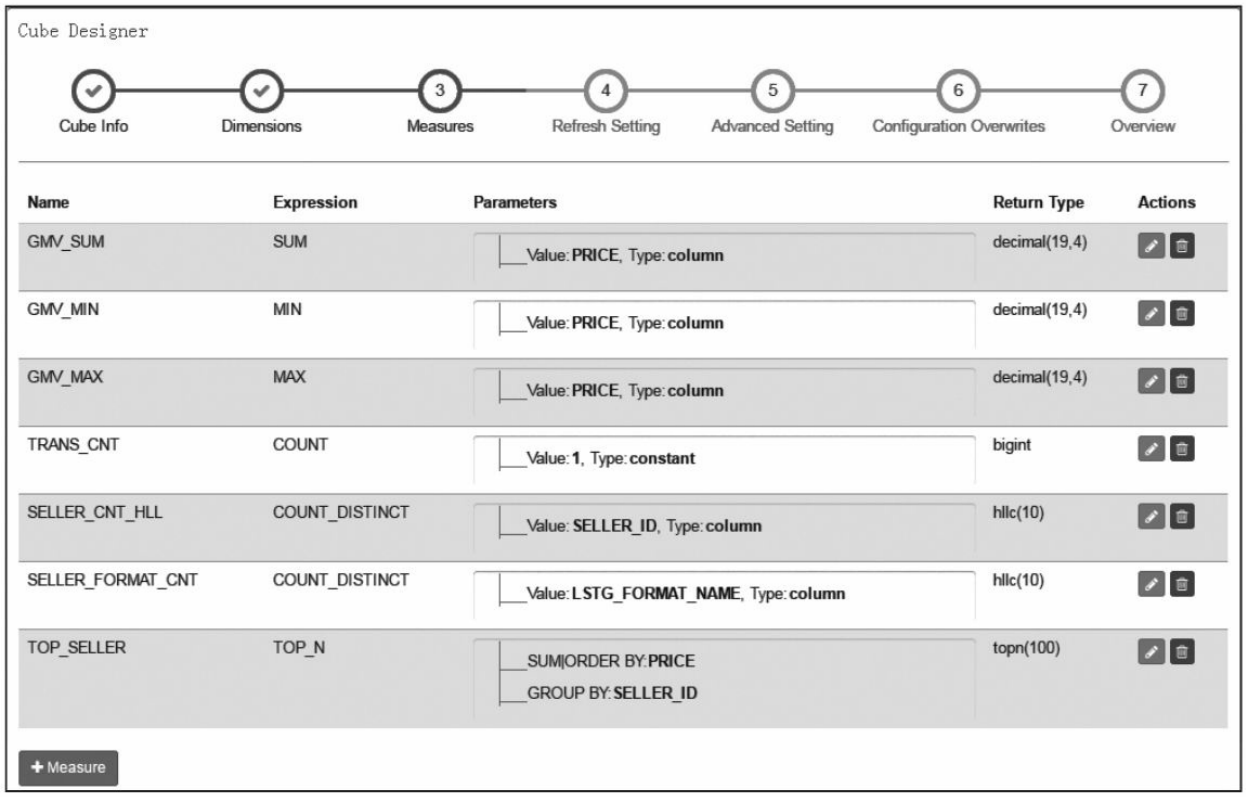


图7-18 新建Cube(7)

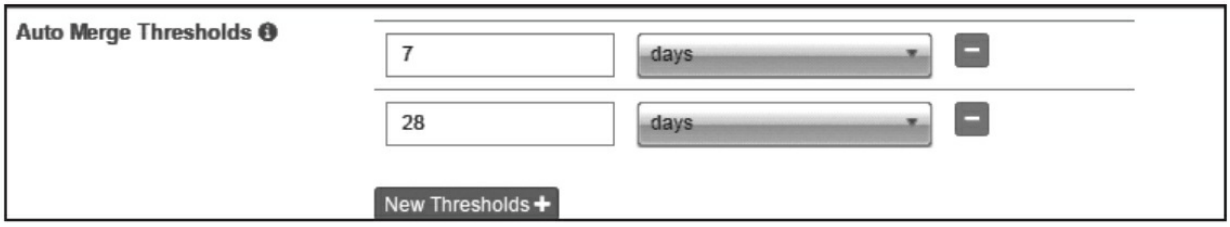


图7-19 新建Cube(9)

因为存在对于历史订单的查询需求，我们在此不对Cube做自动清理，所以需要设置保留时间阈值(Retention Threshold)为0。

在创建数据模型的时候我们曾提到过，希望采用增量构建的方式对Cube进行构建，并选择了PART_DT字段作为分段时间列(Partition Date

Column)。在创建Cube时，需要指定Cube构建的起始时间，在这个例子中，根据样例数据中的时间条件，我们选择2012-01-0100:00:00作为分段起始时间(Partition Start Date)。

第五步，通过对Cube进行高级设置优化Cube的存储大小和查询速度，主要包括聚合组和Rowkey。在第6章曾提到过，添加聚合组可以利用字段间的层级关系和包含关系有效地降低Cuboid的数量。在这个案例当中，与商品分类相关的三个字段(META_CATEG_NAME、CATEG_LVL2_NAME、CATEG_LVL3_NAME)实际上具有层级关系，如一级类别(META_CATEG_NAME)包含多个二级类别(CATEG_LVL2_NAME)，二级类别又包含多个三级类别(CATEG_LVL3_NAME)，所以，我们可以为它们创建层级结构的组合(Hierarchy Dimensions)。最终，聚合组的设计如图7-20所示。

图7-20 新建Cube（9）

在第6章中还提到过，参与Cuboid生成的维度都会作为Rowkeys，因此需要把这些列添加到Rowkeys当中。在这个案例中，总共需要添加7个Rowkeys。在每个Rowkeys上，还需要为列值设置编码方法。在这个案例中，除了把LSTG_FORMAT_NAME设置为Fixed_length类型（长度为12）外，还要将其余的Rowkeys都设置为Dict编码。

Rowkeys的顺序对于查询性能来说至关重要，正如第6章所讲的，一般把最常出现在过滤条件中的列放置在Rowkeys的前面，在这个案例中，是把PART_DT放在Rowkeys的第一位，后面按照层级来排列商品分类的字段。最终，Rowkeys的设置如图7-21所示。

图7-21 新建Cube（10）

第六步，设置Cube的配置覆盖。在这里添加的配置项可以在Cube级别覆盖从kylin.properties配置文件读取出来的全局配置。在这个案例中，可以直接采用默认配置，不做任何修改。

第七步，对Cube的信息进行概览。请读者仔细确认这些基本信息，包括数据模型名称、事实表及维度和度量的个数。确认无误后单击“Save”按钮，并在弹出的确认提示框中选择“Yes”。

最终，Cube的创建就完成了。这时刷新Model页面，在Cube列表中就可以看到新创建的Cube了。因为新创建的Cube没有被构建过，是不能被查询的，所以状态仍然是“禁用（DISABLED）”（如图7-22所示）。

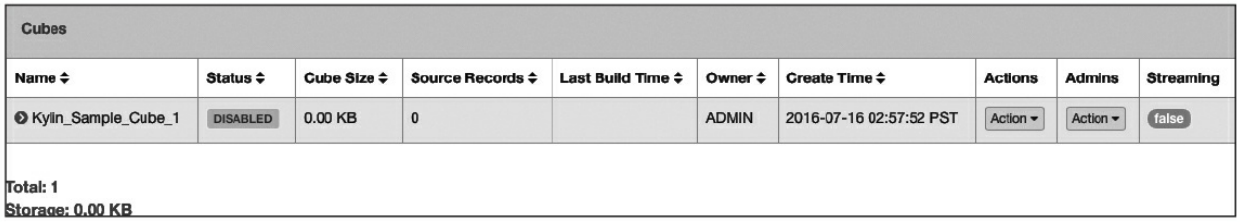

Cubes
Name
Status
Cube Size
Source Records
Last Build Time
Owner
Create Time
Actions
Admins
Streaming
Kylin_Sample_Cube_1
DISABLED
0.00 KB
0
ADMIN
2016-07-16 02:57:52 PST
Action
Action
false
Total: 1
Storage: 0.00 KB

图7-22 Cube列表

7.1.5 构建Cube

在创建好Cube之后，只有对Cube进行构建，才能利用它执行SQL查询。本节将一起完成一些Cube构建相关的任务。

·提示　在第3章中提到过，Cube的构建分为增量构建和全量构建两种方式，7.1.4节创建的Cube采用了增量构建的方式。

1.初次构建

首先打开Apache Kylin的Web UI，并选择刚刚创建的Kylin_Sample_1项目，然后跳转到Model页面，找到Cube列表。

第一步，在Cube列表中找到刚刚创建好的Cube——Kylin_Sample_Cube_1。单击右侧的Action按钮，在弹出的菜单中选择Build（如图7-23所示）。

第二步，在弹出的Cube构建确认对话框中确认Cube的分段时间列（Partition Date Column）是DEFAULT.KYLIN_SALES.PART_DT，以及起始时间是2012-01-0100:00:00。正如第3章所介绍的，一次构建会为Cube产生一个新的Segment，每次的SQL查询都会访问一个或多个符合条件的Segment；为了尽可能地让一个Segment更好地适用于查询条件，因此选择按年构建，即每个年份构建一个Segment。在这个例子中，输入结束日期

(End Date)为2013-01-0100:00:00。设置完成后单击Submit按钮(如图7-24所示)。

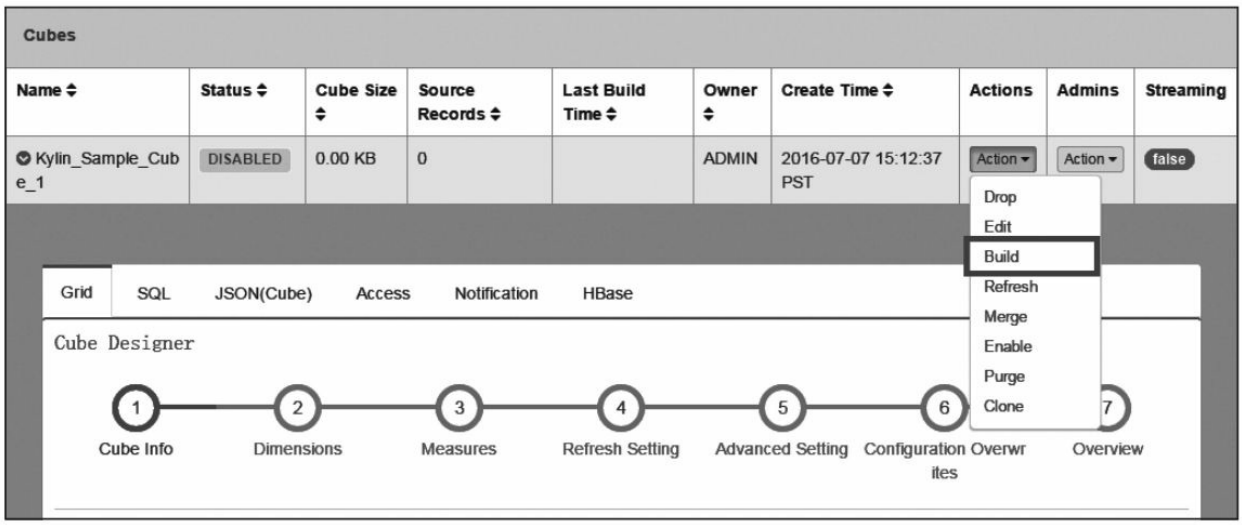


图7-23 构建Cube(1)

·注意 增量构建具体是按年构建还是按月构建应该根据实际的业务需求、ETL时效及数据量大小而定。如果一次构建的数据量过大,则可能导致构建时间过长,或者出现内存溢出等异常。在当前的样例数据中,数据量较小,按年构建是可以顺利完成的。

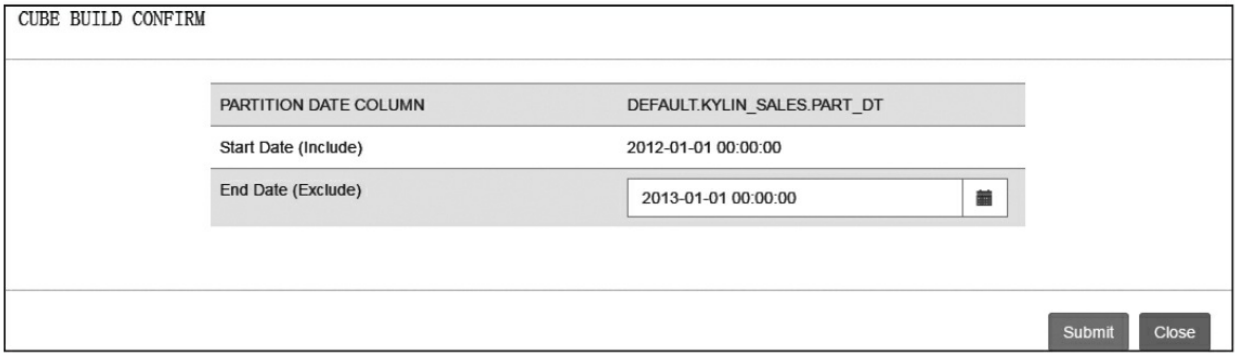


图7-24 构建Cube(2)

当任务成功提交之后，切换到Monitor页面，这里会列出所有的任务列表。我们找到列表最上面的一个任务（名称是：Kylin_Sample_Cube_1-20120101000000_20130101000000），这就是我们刚刚提交的任务。在这一行双击或单击右侧的箭头图标，页面的右侧会显示当前任务的详细信息。有关每一个步骤的具体内容，用户可参考第3章的相关内容，此处不再赘述。

待构建任务完成之后，读者可以在Monitor页面看到该任务状态已被置为完成（Finished）。这时候，第一个Segment就构建完成了。前往Cube列表中查看，会发现该Cube的状态已被置为“就绪（Ready）”了。

2.增量构建

因为选择了增量构建的方式，所以在第一个Segment构建完成之后，就要开始构建第二个Segment。首先在Model页面的Cube列表中找到该Cube，单击右侧的Actions按钮，然后选择“Build”，打开Cube构建确认对话框。

在这个对话框中，首先要确认起始时间（Start Date）是2013-01-0100:00:00，因为这是上次构建的结束日期，为保障所构建数据的连续性，Apache Kylin自动将新一次构建的起始时间更新为上次构建的结束日期。同样的，在结束时间（End Date）里输入2014-01-0100:00:00，然后单击Submit按钮，开始构建下一年的Segment（如图7-25所示）。

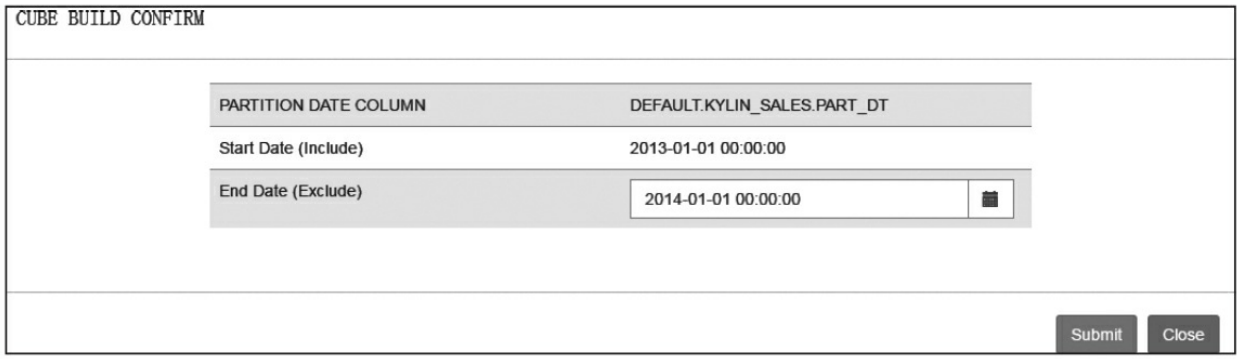


图7-25 构建Cube（3）

与第一次构建一样，读者可以前往Monitor页面监控和查询该任务的状态。待构建完成之后，在Cube的详情页中查看，会发现Cube的两个Segment都已就绪（如图7-26所示）。

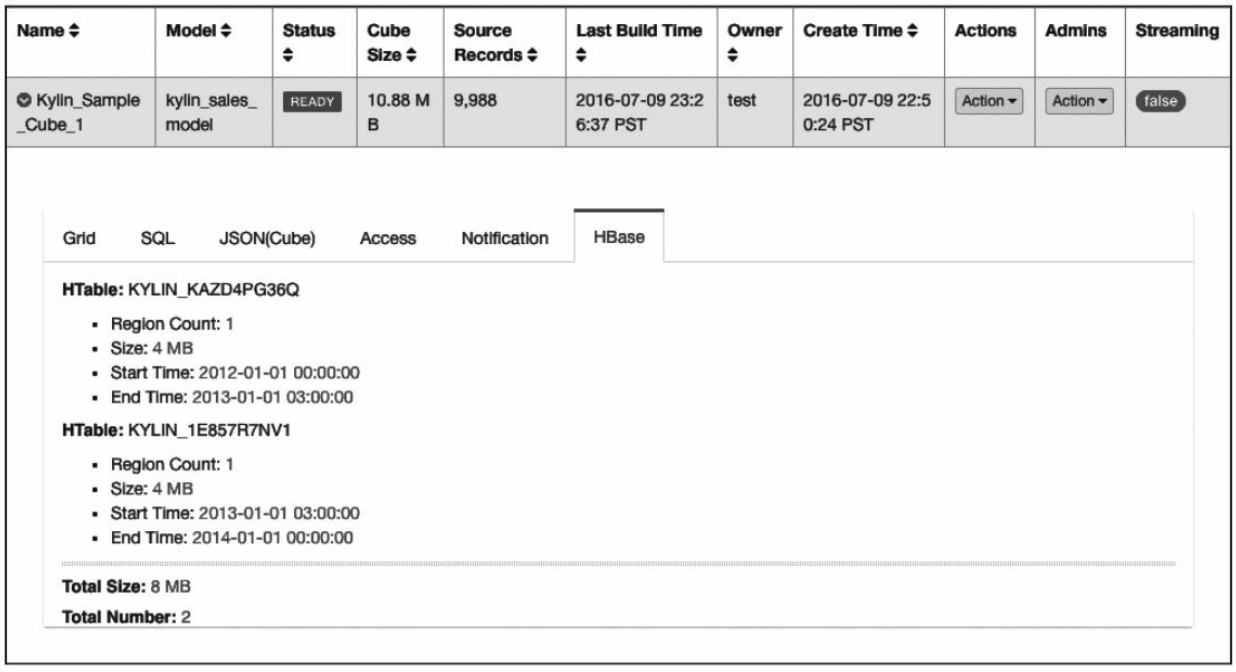


图7-26 构建Cube（4）

7.1.6 SQL查询

当Cube构建任务完成之后，系统一般会自动把Cube的状态切换为就绪(Ready)。接下来，就可以利用该Cube进行SQL查询了。在这个案例中，前面已经构建好了两个Segment，现在可以跳转到Insight页面开始执行SQL查询了。

首先，在Web UI上选择本案例所用的Kylin_Sample_1项目。然后根据Cube上维度和度量的设计，在查询输入框中输入SQL语句，然后单击Submit按钮。下面将给出SQL查询的例子和相应的结果说明。

例1：单表行数统计

```
SELECT COUNT(*) FROM KYLIN_SALES
```

这条SQL语句可用于查询KYLIN_SALES表中的总行数，读者可以同时在Hive中执行同样的查询进行性能对比。在笔者的对比中，Hive查询耗时29.385秒，Apache Kylin查询耗时0.18秒。

例2：多表连接

```
    SELECT
    KYLIN_SALES.PART_DT
    ,KYLIN_SALES.LEAF_CATEG_ID
    ,KYLIN_SALES.LSTG_SITE_ID
    ,KYLIN_CATEGORY_GROUPINGS.META_CATEG_NAME
    ,KYLIN_CATEGORY_GROUPINGS.CATEG_LVL2_NAME
    ,KYLIN_CATEGORY_GROUPINGS.CATEG_LVL3_NAME
    ,KYLIN_SALES.LSTG_FORMAT_NAME
    ,SUM(KYLIN_SALES.PRICE)
    ,COUNT(DISTINCT KYLIN_SALES.SELLER_ID)
    FROM KYLIN_SALES as KYLIN_SALES
    INNER JOIN KYLIN_CAL_DT as KYLIN_CAL_DT
    ON KYLIN_SALES.PART_DT = KYLIN_CAL_DT.CAL_DT
    INNER JOIN KYLIN_CATEGORY_GROUPINGS as KYLIN_CATEGORY_GROUPINGS
    ON  KYLIN_SALES.LEAF_CATEG_ID  =  KYLIN_CATEGORY_GROUPINGS.LEAF_CATEG_ID  AND
KYLIN_SALES.LSTG_SITE_ID = KYLIN_CATEGORY_GROUPINGS.SITE_ID
    GROUP BY
    KYLIN_SALES.PART_DT
    ,KYLIN_SALES.LEAF_CATEG_ID
    ,KYLIN_SALES.LSTG_SITE_ID
    ,KYLIN_CATEGORY_GROUPINGS.META_CATEG_NAME
    ,KYLIN_CATEGORY_GROUPINGS.CATEG_LVL2_NAME
    ,KYLIN_CATEGORY_GROUPINGS.CATEG_LVL3_NAME
    ,KYLIN_SALES.LSTG_FORMAT_NAME
```

这里的SQL语句将事实表KYLIN_SALES和两张维表根据外键进行了内部连接。在笔者的对比试验中，Hive查询耗时34.361秒，Apache Kylin查询耗时0.33秒。

例3：维度列COUNT_DISTINCT

```
    SELECT
    COUNT(DISTINCT KYLIN_SALES.PART_DT)
    FROM KYLIN_SALES as KYLIN_SALES
    INNER JOIN KYLIN_CAL_DT as KYLIN_CAL_DT
    ON KYLIN_SALES.PART_DT = KYLIN_CAL_DT.CAL_DT
    INNER JOIN KYLIN_CATEGORY_GROUPINGS as KYLIN_CATEGORY_GROUPINGS
    ON  KYLIN_SALES.LEAF_CATEG_ID  =  KYLIN_CATEGORY_GROUPINGS.LEAF_CATEG_ID  AND
KYLIN_SALES.LSTG_SITE_ID = KYLIN_CATEGORY_GROUPINGS.SITE_ID
```

这条SQL语句对PART_DT字段进行了COUNT_DISTINCT查询，但是

该字段并没有被添加为COUNT_DISTINCT的度量。这个功能是从1.5.2版本开始支持的，即对于未定义的维度列，可以执行COUNT_DISTINCT的查询。在笔者的对比试验中，Hive查询耗时44.911秒，Apache Kylin查询耗时0.12秒。

例4：全表查询

```
SELECT * FROM KYLIN_SALES
```

默认情况下，Apache Kylin并不会对原始数据的明细进行保存，因此并不支持形如SELECT*的SQL查询。但是，用户经常希望通过执行SELECT*获取部分样例数据；因此Kylin对这种SQL会返回不精确的查询结果。如果读者希望Apache Kylin支持原始数据的保存和查询，可以在Cube中定义RAW类型的度量。

7.2 流式分析

一般来说，对离线数据进行分析时，其时效性往往是受限的。虽然用户可以在ETL结束之后立即触发Cube增量构建，但由于ETL具有延时性，因此数据分析师通常难以及时掌握到最新的数据。为了解决这一问题，Apache Kylin从1.5.0版本开始支持流式构建Cube，关于流式构建的详细内容，读者可以仔细阅读第4章的相关章节。

本节将基于HDP2.2.4.2Sandbox的环境，利用模拟数据，从一个实际的零售业务的使用场景出发，为用户详细介绍流式数据创建、构建Cube，以及执行SQL查询的详细过程。关于在Sandbox环境中部署Apache Kylin服务的相关内容，请参考第10章。在这个例子中，作为数据源的Kafka版本是0.8.1。

7.2.1 Kafka数据源

Apache Kylin的二进制包中提供了一个Kafka的模拟数据生成器。这个生成器作为Kafka的生产者(Producer)会不断地随机生成结构化的数据并推送到Kafka队列当中。下面是创建模拟数据的具体步骤。

1)在Ambari中启动Kafka。

2)创建一个Kafka的Topic，并命名为kylin_demo。

```
    /usr/hdp/current/kafka-broker/bin/kafka-topics.sh --create --zookeeper
localhost:2181 --replication-factor 1 --partitions 1 --topic kylin_demo
    Created topic "kylin_demo".
```

3)调用Apache Kylin二进制包中自带的数据生成器向Kafka中推送数据。

```
    export KYLIN_HOME='/root/apache-kylin-1.5.2-bin'
    cd $KYLINN_HOME
    ./bin/kylin.sh org.apache.kylin.source.kafka.util.KafkaSampleProducer --topic
kylin_demo --broker sandbox:6667 —delay 0
```

这个工具每2秒钟将会生成一条数据，并发送到Kafka队列中。

4)使用Kafka的控制台消费者(Console Consumer)查看队列中的数据，以确保数据按照需要生成。

```
    /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --zookeeper
sandbox.hortonworks.com:2181 --topic kylin_demo --from-beginning
    {"amount":4.036149489293339,"category":"ELECTRONIC","order_time":1462465632689,
"device":"Windows","qty":4,"currency":"USD","country":"AUSTRALIA"}
    {"amount":83.74699855368388,"category":"CLOTH","order_time":1462465635214,"devi
ce":"iOS","qty":8,"currency":"USD","country":"INDIA"}
```

从消费者的输出中可以看到，这个模拟数据是一个零售业务的订单数据流。每条数据的结构都是JSON格式，不包含层级关系，可以直接作为Apache Kylin流式构建的输入源。表7-2对各个字段都进行了介绍。

表7-2 流式数据结构介绍

字段名	字段类型	意义
ORDER_TIME	bigint	下单时间
AMOUNT	double	订单金额
CATEGORY	string	订单分类
DEVICE	string	下单设备
CURRENCY	string	交易货币种类
COUNTRY	string	交易所在国家

7.2.2 创建数据表

在准备好输入数据源的基础之上，下面正式开始在Apache Kylin中进行操作。关于如何在Sandbox环境中快速部署和启动Apache Kylin的相关内容，请参见第10章。

第一步，选择先前创建好的Kylin_Sample_1项目（参见第7.1.2节）；然后在Model页面单击“Add Streaming Table”按钮，如图7-27所示。

第二步，在左侧的JSON框中输入一条流式数据中的样例数据，用于自动获取数据结构。在这个例子中，输入如下的数据，并单击页面中部的“» ”按钮。

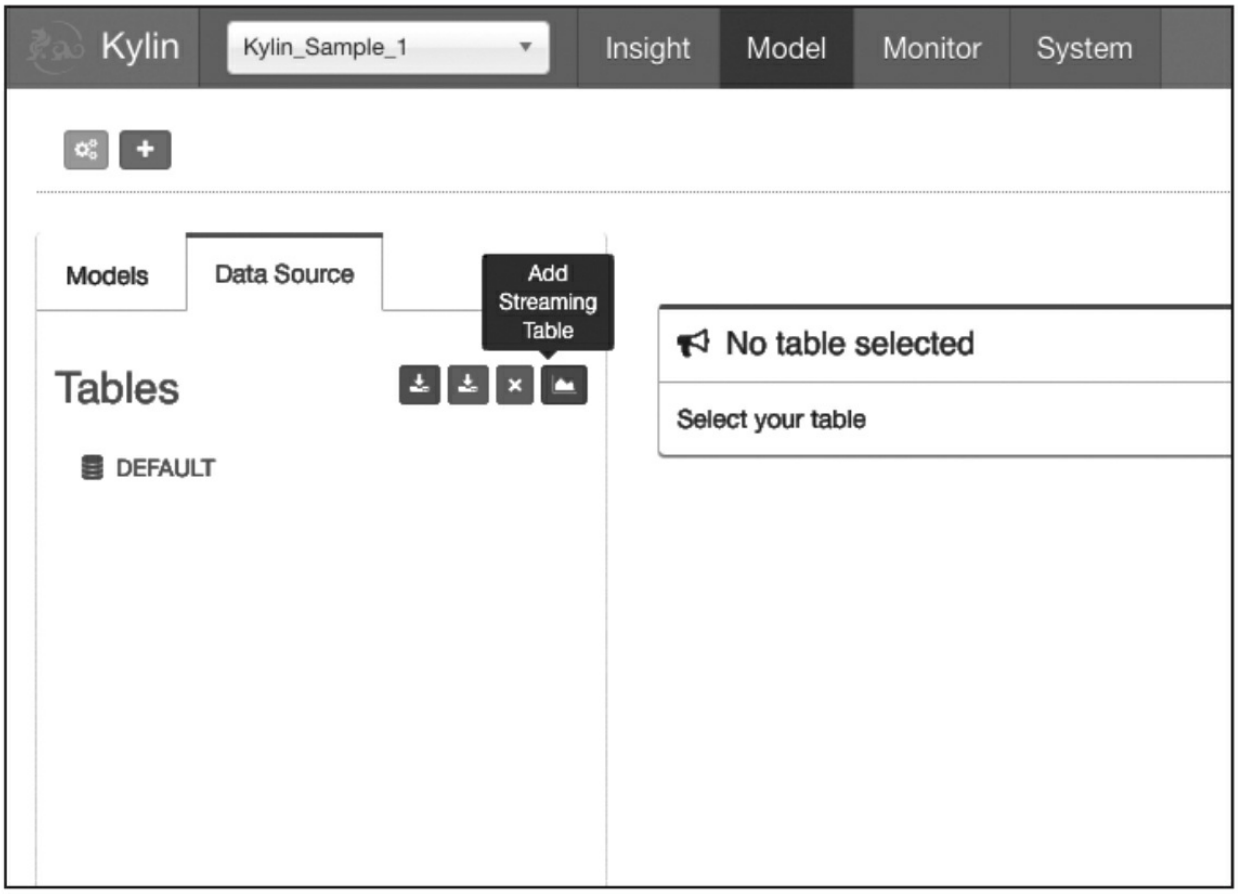


图7-27 创建数据源

```
    {"amount":4.036149489293339,"category":"ELECTRONIC","order_time":1462465632689,
"device":"Windows","qty":4,"currency":"USD","country":"AUSTRALIA"}
```

第三步，Apache Kylin会自动对左侧的JSON输入进行解析，并将解析出来的字段及其类型显示在页面右侧的列表中。我们先在右侧的“Table Name”输入框中输入一个表名：STREAMING_SALES_TABLE，这个表名也将在实际的SQL查询中被用到。然后选择列出的所有字段，即把所有列都添加到Cube当中。同时，可以看到，在order_time列的右侧有一个“timestamp”的标志，代表这一列将作为表征数据流的时间戳；在列表的最下端，有一些自动添加的字段，如“year_start”和“quarter_start”，这些时

间扩展列将在构建和查询时提供更好的灵活性。设置完成之后(如图7-28所示)，单击“Next”按钮。

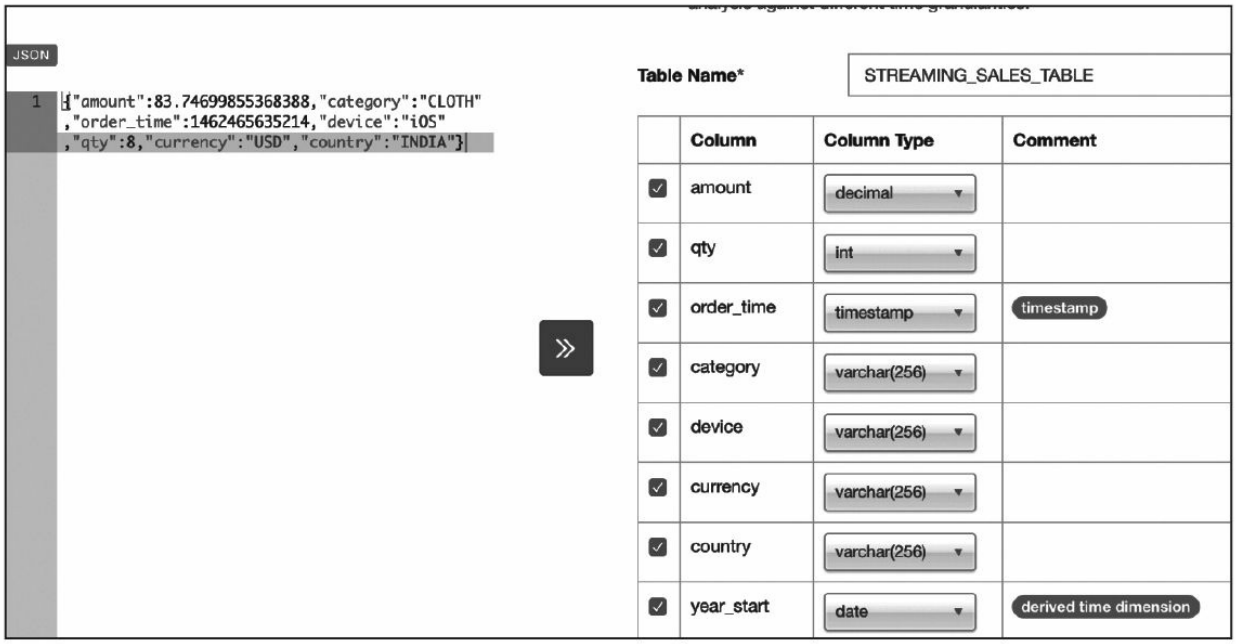


图7-28 创建数据表

第四步，设置Kafka的Topic信息。首先，在Topic输入框中输入先前创建的Kafka Topic名称：kylin_demo；接着在Cluster选项卡中添加一个Broker，并按图7-29所示的信息进行填写；其余保持默认。

图7-29 配置Kafka

第五步，确认高级设置项，按图7-30进行设置。关于各项的具体介绍请参见第4章。

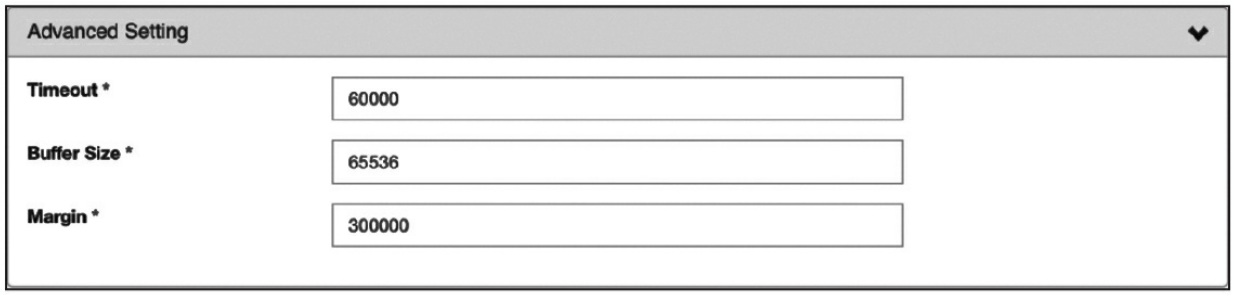


图7-30 Kafka高级设置

第六步，确认解析器（Parser）设置。请选择order_time作为Parser Timestamp Column（如图7-31所示）。

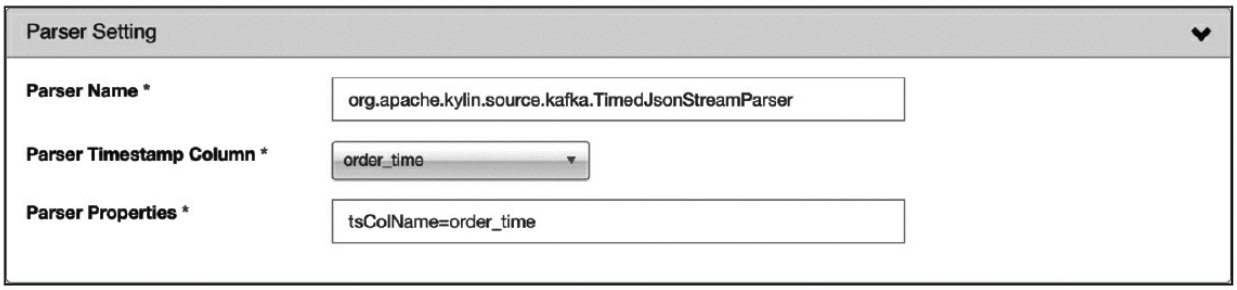


图7-31 解析器设置

最终，单击Submit按钮，一个流式构建的数据源表就创建成功了。

7.2.3 创建数据模型

和增量构建的流程一样，我们也要为流式构建的Cube创建数据模型。关于创建数据模型的一些细节在7.1节已经有过相关介绍，在此不再赘述，只介绍该案例的特别配置。

第一步，创建一个数据模型，并取名为Kylin_Sample_Model_2（如图7-32所示）。

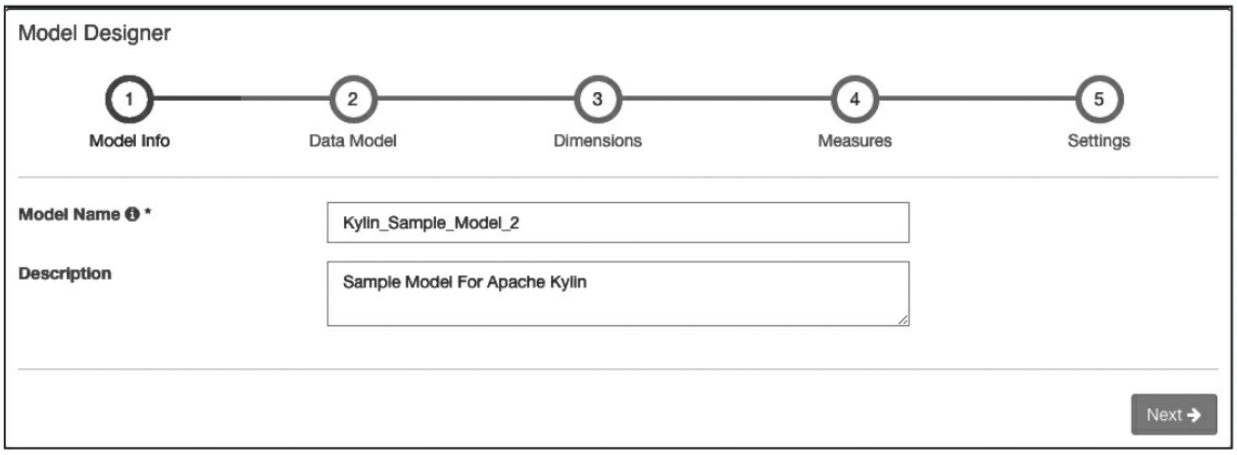


图7-32 创建数据模型（1）

第二步，选择上一步创建好的STREAMING_SAMPLE_TABLE_1表作为事实表。

第三步，选择图7-33中的8列作为数据模型的维度列。

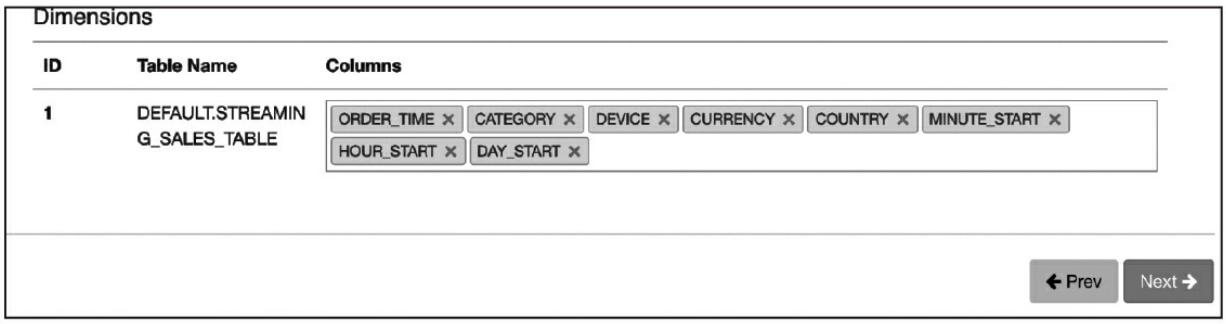


图7-33 创建数据模型(2)

第四步，选择图7-34中的2列作为数据模型的度量列。

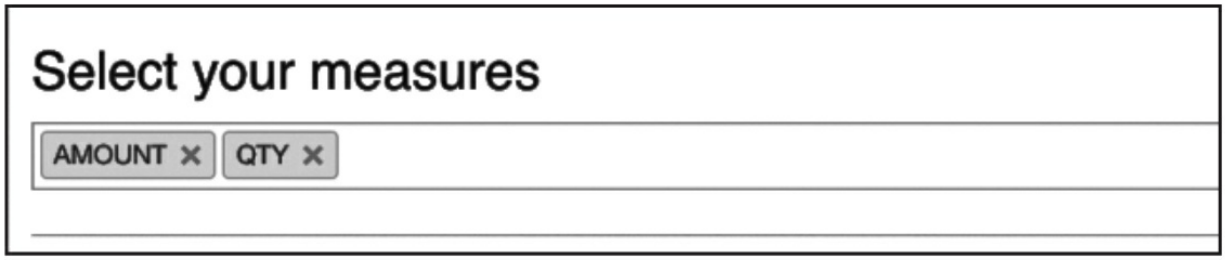


图7-34 创建数据模型(3)

第五步，选择MONTH_START列作为分段时间列，因此我们可以对Cube进行分钟级的增量构建，其余的保持默认，如图7-35所示。

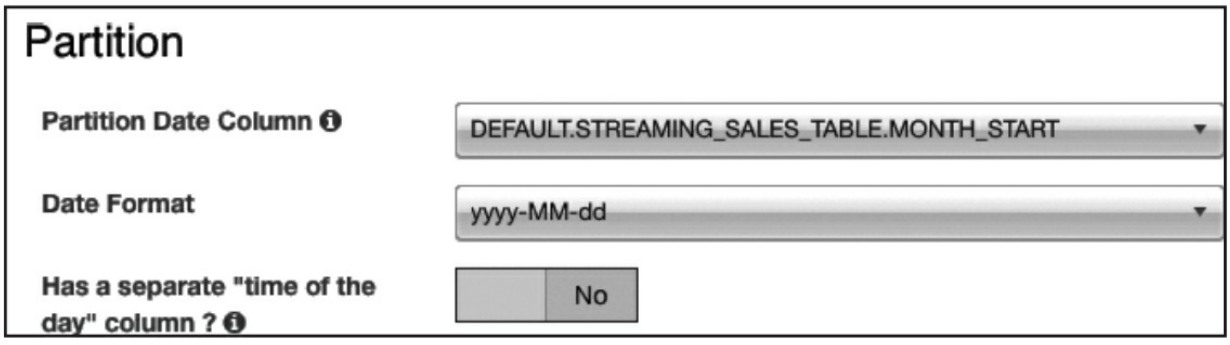


图7-35 选择分区时间列

最终，单击“Save”按钮，保存所创建的数据模型。当看到成功提示时，数据模型就创建成功了。

7.2.4 创建Cube

接下来，基于创建好的数据模型开始在Apache Kylin中创建流式构建的Cube。

第一步，在Model页面创建一个新的Cube，基于前一节已经创建好的数据模型Kylin_Sample_Model_2来完成，并取名为Kylin_Sample_Cube_2。

第二步，为Cube添加如下的维度，因为只有一个事实表，因此所有的维度都是普通类型（Normal）（如图7-36所示）。

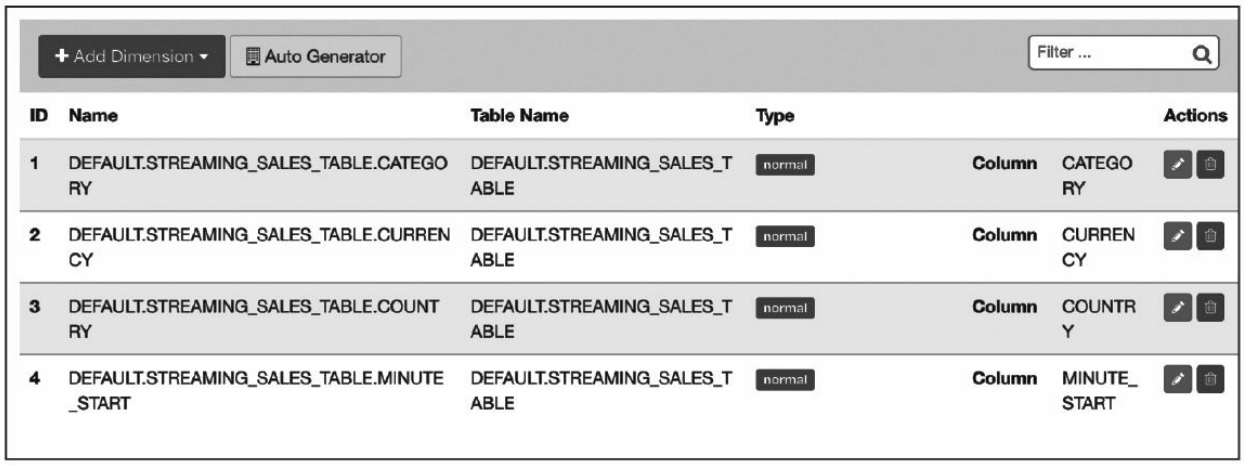


图7-36 创建Cube（1）

第三步，根据数据模型中的度量列为Cube添加度量的预计算模型，设置如图7-37所示。

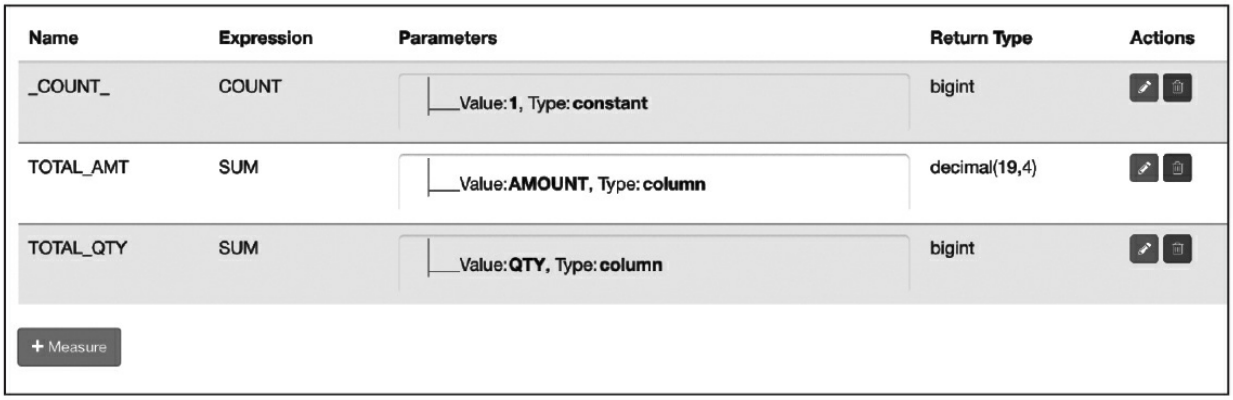


图7-37 创建Cube(2)

第四步，设置Cube的自动合并时间。因为流式构建需要频繁地构建较小的Segment，为了不对HBase存储器造成过大的压力，同时获取较好的查询性能，所以需要通过自动合并将已有的多个小Segment合并成一个较大的Segment。这里我们设置一个梯度的自动合并时间：0.5小时、4小时、1天、7天、28天。此外，设置保留时间(Retention Range)为30天(如图7-38所示)。

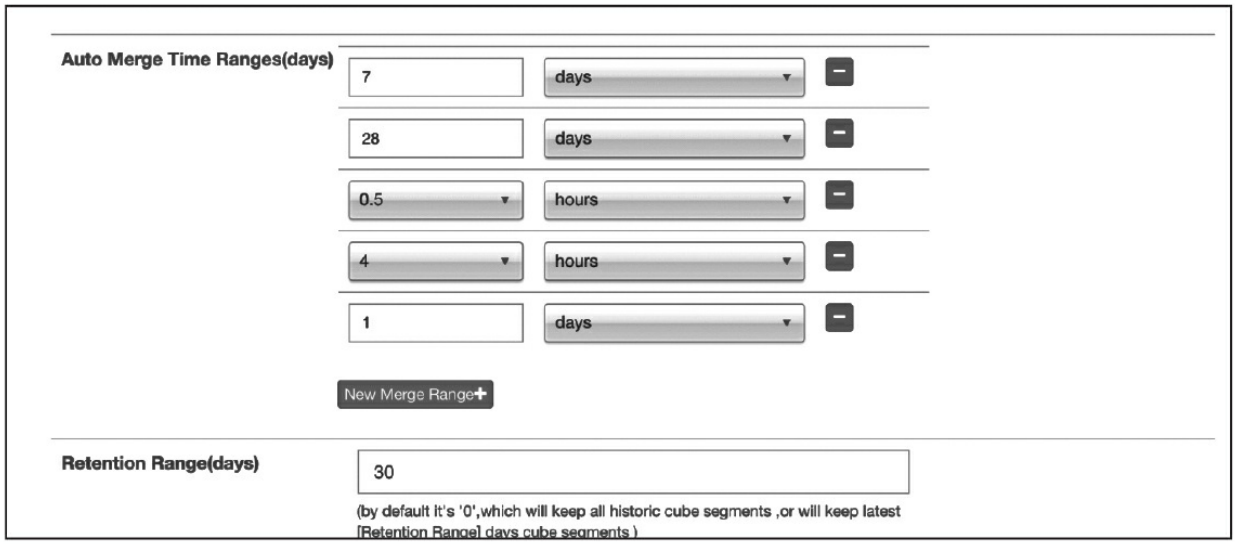


图7-38 创建Cube(3)

第五步，调整Rowkeys的排列顺序，将最容易出现在查询条件中的时间字段放在最前面，如图7-39所示。

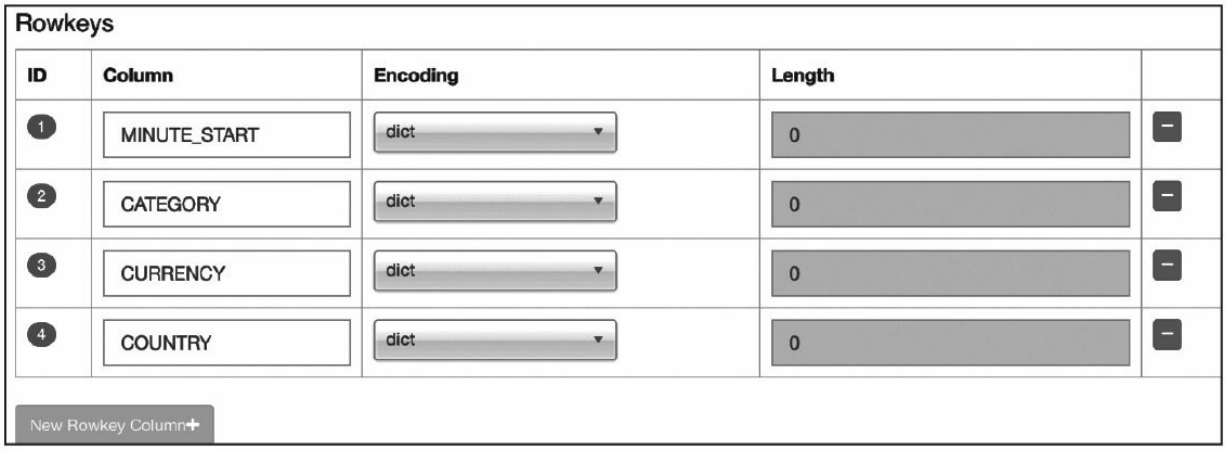


图7-39 创建Cube(4)

最终，单击“Save”保存Cube，当看到成功提示时，一个流式构建的Cube就创建完成了。

7.2.5 构建Cube

从第4章的介绍可以得知，流式构建和普通的增量构建的执行方式是不同的。我们需要使用一个命令行工具来触发流式构建的执行。在这个例子中，请执行下面的命令：

```
$KYLIN_HOME/bin/streaming_build.sh STREAMING_CUBE 300000 0
streaming started name: STREAMING_CUBE id: 1462471500000_1462471800000
```

在这个命令中，我们设置了5分钟的边界位移。构建任务的日志保存在$KYLIN_HOME/logs目录下，以JOB ID命名，如streaming_STREAMING_CUBE_1462471500000_1462471800000.log。当任务完成之后，可以在Monitor页面查看执行的结果。

当任务执行成功之后，读者需要手动启用该Cube。即在Cube列表中找到该Cube，单击右侧的Actions按钮，并选择Enable。

7.2.6 SQL查询

待Cube构建并启用成功之后，我们就可以进行SQL查询了。查询的过程和普通的增量构建是一样的，读者只需要根据左侧列出的表、列信息并结合Cube上维度和度量的定义编写SQL语句即可。

这里给出一个SQL语句的例子，读者可以自行在SQL输入框中进行执行和测试。

```
    select minute_start, count(*), sum(amount), sum(qty) from streaming_sales_table
group by minute_start
```

图7-40是在笔者环境中执行的结果，执行时间是0.16秒。

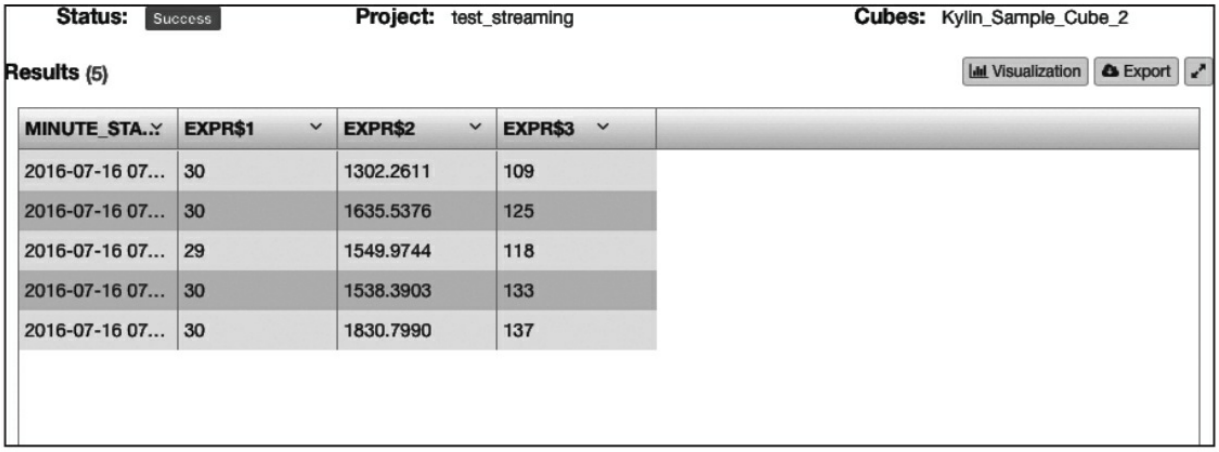


图7-40 流式构建查询结果

7.3 小结

本章通过具体的真实案例详细讨论了Apache Kylin建模和构建Cube的整个过程。其中的各项设置，尤其是Cube的高级设置，如聚合组和Rowkeys等配置，值得参考。示例数据源于真实的案例，稍有简化，但仍然非常有代表性，希望能给读者带来帮助。

第8章　扩展Apache Kylin

Apache Kylin有着卓越的可扩展架构。总体架构上的三大依赖——数据源、计算引擎和存储引擎——都有清晰的接口，保证Apache Kylin可以随时接入最新的数据和计算存储技术，并随着Hadoop生态圈一起演进。此外，作为Cube核心的聚合类型也可以扩展，用户可以定制业务领域的特殊聚合，在Apache Kylin上直接实现业务逻辑。维度编码也可以为特定数据扩展实现最高效的数据压缩。

本章的所有内容都是基于Apache Kylin v1.5.2.1的。新版本在细节上可能会略有不同。

8.1 可扩展式架构

Apache Kylin在v1.3版本之前极度依赖Hive、MapReduce和HBase。尽管适应了大多数用户的部署环境，但从设计的角度来看，Apache Kylin与Hadoop是紧耦合的关系，不利于扩展。随着Apache Kylin的推广及Hadoop世界的多样化发展，越来越多的问题在架构层涌现。比如“可不可以直接从Oracle导入数据”、“为什么不使用HBase作为存储”、“能不能用Spark构建Cube”，等等。

Apache Kylin v1.5版本从系统层面针对可扩展性进行了重构（见第1章图1-4），将系统的三大依赖（数据源、计算、存储）进行了剥离，定义了清晰的接口。这保证了Apache Kylin可以更容易地根据需要新增数据源、替换计算框架和存储系统，在日新月异的技术潮流中始终保持领先地位。

此外Apache Kylin还允许多个计算引擎和存储引擎并存，保证了极大的灵活性。假设用户希望从MapReduce引擎过渡到Spark引擎，那么可以分批次逐一升级每个Cube的计算引擎。例如，首先升级部分Cube并进行试验，确认Spark引擎的稳定性和先进性，之后再按计划升级其余的Cube，管理升级过程中的不确定性风险。

8.1.1 工作原理

下面从设计的角度详细介绍可扩展架构的工作原理。

每一个Cube都可以设定自己的数据源、计算引擎和存储引擎，这些设定信息均保存在Cube元数据中。在构建Cube时，首先由工厂类创建数据源、计算引擎和存储引擎对象。这三个对象独立创建，相互之间没有关联（如图8-1所示）。

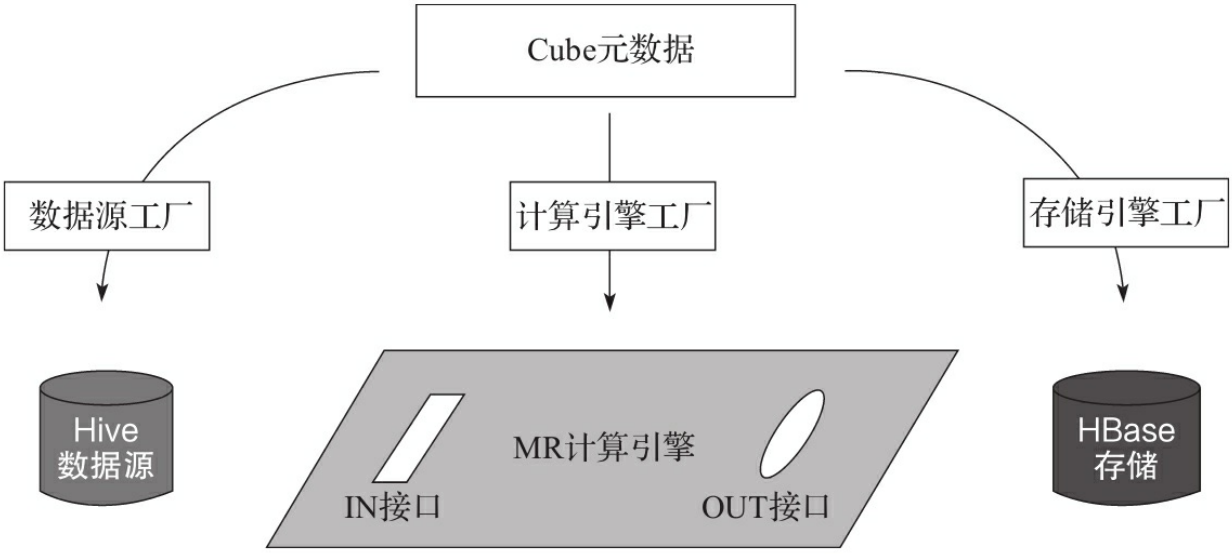


图8-1 工厂创建数据源和引擎对象

要把它们串联起来，使用的是适配器设计模式。计算引擎好比是一块主板，主控整个Cube的构建过程。它以数据源为输入，以存储为Cube的输出，因此也定义了IN和OUT两个接口。数据源和存储引擎则需要适配

IN和OUT，提供相应的接口实现，把自己接入计算引擎，适配过程参见图8-2。适配完成之后，数据源和存储引擎即可被计算引擎调用。三大引擎连通，就能协同完成Cube构建。

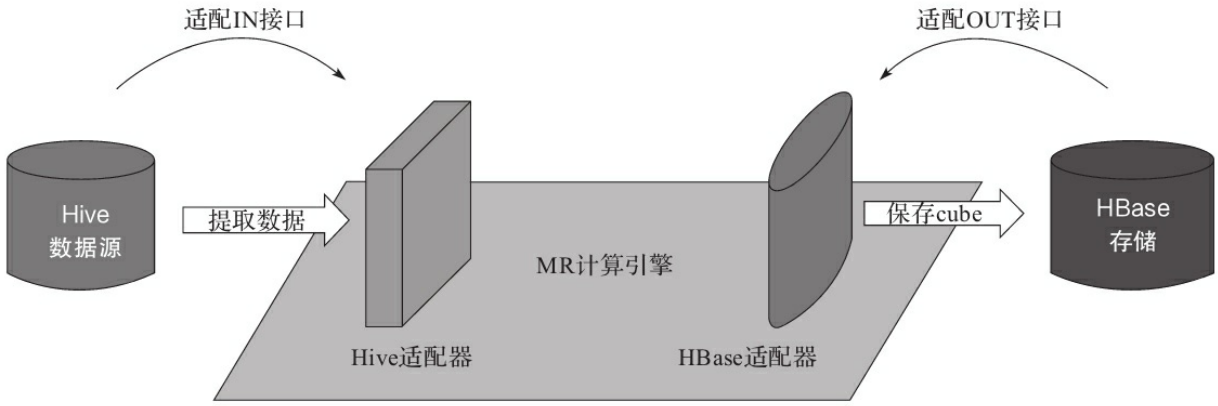


图8-2 数据源和存储引擎适配IN/OUT接口

由图8-2可以得知，计算引擎只提出接口需求，每个接口都可以有多种实现，也就是能接入多种不同的数据源和存储。类似的，每个数据源和存储也可以实现多个接口，适配到多种不同的计算引擎上。三者之间是多对多的关系，可以任意组合，十分灵活。

8.1.2 三大主要接口

本节将从代码的层面再进一步加深对数据源、计算引擎、存储引擎三大主要接口的理解。

先来看数据源接口，代码如下：

```
public interface ISource {

    //适配指定的构建引擎接口。返回一个对象，实现指定的 IN 接口
    public <I> I adaptToBuildEngine(Class<I> engineInterface);

    //返回一个 ReadableTable，用来顺序读取一个表
    public ReadableTable createReadableTable(TableDesc tableDesc);
}
```

这个简单的接口只包含以下两个方法。

·adaptToBuildEngine：适配指定的构建引擎接口。返回一个对象，实现指定的IN接口。该接口主要由计算引擎调用，要求数据源向计算引擎适配。如果数据源无法提供指定接口的实现，则适配失败，Cube构建将无法进行。

·createReadableTable：返回一个ReadableTable，用来顺序读取一个表。除了计算引擎之外，有时也会调用此方法顺序访问数据维表的内容，用来创建维度字典或维表快照。

再来看存储引擎接口，代码如下：

```
public interface IStorage {

    //适配指定的构建引擎接口。返回一个对象，实现指定的 OUT 接口
    public <I> I adaptToBuildEngine(Class<I> engineInterface);

    //创建一个查询对象 IStorageQuery，用来查询给定的 IRealization
    public IStorageQuery createQuery(IRealization realization);
}
```

存储引擎接口也只包含两个方法。

·adaptToBuildEngine：适配指定的构建引擎接口。返回一个对象，实现指定的OUT接口。该接口主要由计算引擎调用，要求存储引擎向计算引擎适配。如果存储引擎无法提供指定接口的实现，则适配失败，Cube构建将无法进行。

·createQuery：创建一个查询对象IStorageQuery，用来查询给定的IRealization。简单来说，就是返回一个能够查询指定Cube的对象。IRealization是在Cube之上的一个抽象。其主要的实现就是Cube，此外还有被称为Hybrid的联合Cube。

最后考察的是计算引擎接口。目前的计算引擎在设计上都是批处理的模式，因此称为IBatchCubingEngine。流式处理是另一类数据处理的模式，在实时计算和实时分析中有很多应用，不排除Apache Kylin在将来也会加入针对流式处理的专用构建接口：

```
public interface IBatchCubingEngine {

    //返回一个工作流计划，用以构建指定的 CubeSegment
    public DefaultChainedExecutable createBatchCubingJob(CubeSegment newSegment,
String submitter);

    //返回一个工作流计划，用以合并制定的 CubeSegment
    public DefaultChainedExecutable createBatchMergeJob(CubeSegment mergeSegment,
String submitter);

    //指明该计算引擎的 IN 接口
    public Class<?> getSourceInterface();

    //指明该计算引擎的 OUT 接口
    public Class<?> getStorageInterface();
}
```

该接口定义了以下4个方法。

·createBatchCubingJob：返回一个工作流计划，用以构建指定的CubeSegment。这里的CubeSegment是一个刚完成初始化，但还不包含数据的CubeSegment。返回的DefaultChainedExecutable是一个工作流的描述对象。它将被保存并由工作流引擎在稍后调度执行，从完成Cube的构建。

·createBatchMergeJob：返回一个工作流计划，用以合并指定的CubeSegment。这里的CubeSegment是一个待合并的CubeSegment，它的区间横跨了多个现有的CubeSegment。返回的工作流计划一样会在稍后被调度执行，执行的过程会将多个现有的CubeSegment合并为一个，从而降低Cube的碎片化程度。

·getSourceInterface：指明该计算引擎的IN接口。

·getStorageInterface：指明该计算引擎的OUT接口。

将上面所述的内容串联起来，即从代码的角度重现了可扩展架构三大引擎之间的互动，该过程具体描述如下。

1）Rest API接受到构建（合并）CubeSegment的请求。

2）EngineFactory根据Cube元数据的定义，创建IBatchCubingEngine对象，并调用其上的createBatchCubingJob（或者createBatchMergeJob）方法。

3）IBatchCubingEngine根据Cube元数据的定义，通过SourceFactory和StorageFactory创建出相应的数据源ISource和存储IStorage对象。

4）IBatchCubingEngine调用ISource上的adaptToBuildEngine方法，传入IN接口，要求数据源向自己适配。

5）IBatchCubingEngine调用IStorage上的adaptToBuildEngine方法，传入OUT接口，要求存储引擎向自己适配。

6）适配成功后，计算引擎协同数据源和存储引擎计划Cube构建的具体步骤，将结果以工作流（DefaultChainedExecutable）的形式返回。

7）执行引擎将在稍后执行工作流，完成Cube构建。

8.2 计算引擎扩展

本节将介绍Apache Kylin现有的MapReduce构建引擎，并以它为范例进一步说明如何扩展或创建一个新的Cube计算引擎。（注意，“构建引擎”和“计算引擎”是同义词，可以互换。）

8.2.1 EngineFactory

每一个构建引擎必须实现接口IBatchCubingEngine，并在EngineFactory中注册实现类。只有这样才能在Cube元数据中引用该引擎，否则会在构建Cube时出现“找不到实现”的错误。

注册的方法是通过配置$KYLIN_HOME/conf/kylin.properties来完成的。在其中添加一行构建引擎的声明。比如：

```
kylin.job.engine.2=org.apache.kylin.engine.mr.MRBatchCubingEngine2
```

EngineFactory在启动时会读取kylin.properties，列出所有注册的构建引擎，建立标识号到实现类之间的映射。这样今后就可以用标识号2来代表org.apache.kylin.engine.mr.MRBatchCubingEngine2这个引擎了，并且可在Cube元数据中引用它。

Apache Kylin v1.5版本中有两个内置的构建引擎：

```
kylin.job.engine.0=org.apache.kylin.engine.mr.MRBatchCubingEngine
kylin.job.engine.2=org.apache.kylin.engine.mr.MRBatchCubingEngine2
```

其中0号引擎是从v1.3之前延续下来的老版本。保留它主要是为了版本升级时能向前兼容。由v1.3版本创建的Cube在升级到v1.5后无需修改仍然可以继续使用。标识号为2的引擎是v1.5引入的新引擎，也是新Cube的默认引擎。它包含了两种Cube构建算法（逐层构建算法和快速构建算

法），会在运行时根据数据的分布情况自动选择较优的算法，以提供更快的构建速度。

多个引擎可以同时并存，并由不同的Cube使用，每个Cube必须且只能选择一个构建引擎。从设计上来说，Apache Kylin不保证不同构建引擎之间的兼容性。也就是说若要切换构建引擎，唯一可靠的方法就是创建一个新Cube，并选用新引擎。直接修改元数据更改构建引擎的方法是没有保障的，会导致任意可能的错误。

8.2.2 MRBatchCubingEngine2

从v1.5版本开始，默认的构建引擎实现类为org.apache.kylin.engine.mr.MRBatchCubing Engine2。该类实现了IBatchCubingEngine接口。通过分析和理解该类，我们可以学习如何开发一个构建引擎：

```
public class MRBatchCubingEngine2 implements IBatchCubingEngine {

    //返回一个工作流计划，用以构建指定的 CubeSegment
    public DefaultChainedExecutable createBatchCubingJob(CubeSegment newSegment,
String submitter) {
        return new BatchCubingJobBuilder2(newSegment, submitter).build();
    }

    //返回一个工作流计划，用以合并指定的 CubeSegment
    public DefaultChainedExecutable createBatchMergeJob(CubeSegment mergeSegment,
String submitter) {
        return new BatchMergeJobBuilder2(mergeSegment, submitter).build();
    }

    //指明该计算引擎的 IN 接口
    public Class<?> getSourceInterface() {
        return IMRInput.class;
    }

    //指明该计算引擎的 OUT 接口
    public Class<?> getStorageInterface() {
        return IMROutput2.class;
    }
}
```

很容易看出这只是一个入口类，构建Cube的主要逻辑都封装在BatchCubingJobBuilder2和BatchMergeJobBuilder2中。将复杂的逻辑分而治之，分解成多个更简单更小的类然后组装，是一种良好的设计习惯。

这里简要说明一下DefaultChainedExecutable，顾名思义，它代表了一种可执行的对象，其中包含了很多子任务。它执行的过程就是依次串行执行每一个子任务，直到所有子任务都完成。Apache Kylin的Cube构建比较复杂，要执行很多步骤，步骤之间有直接的依赖性和顺序性。DefaultChainedExecutable很好地抽象了这种连续依次执行的模型，可以用来表示Cube构建的工作流。

另外，重要的输入输出接口也在这里进行声明。IMRInput是IN接口，由数据源适配实现；IMROutput2是OUT接口，由存储引擎适配实现。

8.2.3 BatchCubingJobBuilder2

Cube构建与合并的逻辑分别封装在BatchCubingJobBuilder2和BatchMergeJobBuilder2中。这两个类大同小异，这里就以BatchCubingJobBuilder2为例进行说明。至于BatchMergeJobBuilder2，则留给有兴趣的读者自行研习。

BatchCubingJobBuilder2的主体函数build()如下所示：

```
public class BatchCubingJobBuilder2 extends JobBuilderSupport {
    ......

    private final IMRBatchCubingInputSide inputSide;
    private final IMRBatchCubingOutputSide2 outputSide;
    ......

    public CubingJob build() {
        logger.info("MR_V2 new job to BUILD segment " + seg);

        final CubingJob result = CubingJob.createBuildJob((CubeSegment) seg, submitter,
config);
        final String jobId = result.getId();
        final String cuboidRootPath = getCuboidRootPath(jobId);

        // Phase 1: Create Flat Table & Materialize Hive View in Lookup Tables
        inputSide.addStepPhase1_CreateFlatTable(result);

        // Phase 2: Build Dictionary
        result.addTask(createFactDistinctColumnsStepWithStats(jobId));
        result.addTask(createBuildDictionaryStep(jobId));
        result.addTask(createSaveStatisticsStep(jobId));
        outputSide.addStepPhase2_BuildDictionary(result);
```

```
            // Phase 3: Build Cube
             addLayerCubingSteps(result, jobId, cuboidRootPath); // layer cubing, only
selected algorithm will execute
              result.addTask(createInMemCubingStep(jobId, cuboidRootPath)); // inmem
cubing, only selected algorithm will execute
            outputSide.addStepPhase3_BuildCube(result, cuboidRootPath);

            // Phase 4: Update Metadata & Cleanup
            result.addTask(createUpdateCubeInfoAfterBuildStep(jobId));
            inputSide.addStepPhase4_Cleanup(result);
            outputSide.addStepPhase4_Cleanup(result);

            return result;
        }

        ......
}
```

先来看IMRBatchCubingInputSide inputSide和IMRBatchCubingOutputSide2outputSide这两个成员变量。它们分别来自数据源接口IMRInput和存储接口IMROutput2，分别代表输入和输出两端参与创建工作流。其中具体的内容将在下面的章节详细介绍，目前我们只需要知道它们代表着数据源的输入和存储输出即可。

下面再来看build()函数。从代码注释中可以清晰地看到，整个构建过程是一个子任务依次串行执行的过程，这些子任务又被分为4个阶段。

第一阶段：创建平表。

这一阶段的主要任务是预计算连接运算符，把事实表和维表连接为一张大表，也称为平表。这部分工作可通过调用数据源接口来完成，因为

数据源一般有现成的计算表连接方法，高效且方便，没有必要在计算引擎中重复实现。

第二阶段：创建字典。

创建字典由三个子任务完成，由MR引擎完成，分别是抽取列值、创建字典和保存统计信息。是否使用字典是构建引擎的选择，使用字典的好处是有很好的数据压缩率，可降低存储空间，同时也提升存储读取的速度。缺点是构建字典需要较多的内存资源，创建维度基数超过千万的容易造成内存溢出。虽然可以通过调换外部存储来解决，但也以是降低速度为代价的。

第三阶段：构建Cube。

第二版MR引擎带有两种构建Cube的算法，分别是分层构建和快速构建。对于不同的数据分布来说它们各有优劣，区别主要在于数据通过网络洗牌的策略不同。由于网络是大多数Hadoop集群的瓶颈，因此不同的洗牌策略往往决定了构建的速度。两种算法的子任务将被全部加入工作流计划中，在执行时会根据源数据的统计信息自动选择一种算法，未被选择的算法的子任务将被自动跳过。在构建Cube的最后还将调用存储引擎的接口，存储引擎负责将计算完的Cube放入存储。

第四阶段：更新元数据和清理。

最后阶段，Cube已经构建完毕，MR引擎将首先添加子任务更新Cube元数据，然后分别调用数据源接口和存储引擎接口对临时数据进行清理。

可以看到整个构建过程是由构建引擎来主导的，由它负责调度数据源和存储引擎。除了计算Cube的主要任务是由构建引擎完成的以外，前期的创建平表和数据导入等操作则是由数据源完成的，Cube保存则由存储引擎完成。三者协同，缺一不可。

扩展构建引擎的要点在上面的代码中已有体现。即首先要有清晰的职能划分，哪些功能由构建引擎负责，哪些由数据源和存储引擎负责，都要有清楚的设计。其次是接口的定义，数据源和构建引擎的接口应当符合松耦合高内聚的原则，最小化的接口应使引擎之间的对接变得尽量简单。最后是构建引擎的串联，将构建分步骤交由三大组件逐一完成，制定工作流计划并返回。

8.2.4 IMRInput

在对MR构建引擎的主体已经有了了解之后，再来仔细看一下IMRInput接口，这是MRBatchCubingEngine2对数据源的要求。所有希望接入MRBatchCubingEngine2的数据源都必须实现该接口。

先看IMRInput的上半部分：

```
public interface IMRInput {

    // 返回一个 IMRTableInputFormat 对象，用以从数据源中读取指定的关系表
    public IMRTableInputFormat getTableInputFormat(TableDesc table);

    // IMRTableInputFormat 是一个辅助接口，用来帮助 Mapper 读取数据源中的一张表
    public interface IMRTableInputFormat {

        // 配置给定 MapReduce 任务的 InputFormat
        public void configureJob(Job job);

        // 解析 Mapper 的输入对象，返回关系表的一行
        public String[] parseMapperInput(Object mapperInput);
    }
    ......
```

第一部分是IMRTableInputFormat的定义。这个辅助接口用于帮助MapReduce任务读取数据源中的一张表。为了适应MapReduce编程接口，其中又分为两个方法。方法conf？igureJob(Job)在启动MR任务之前被调用，负责配置所需的InputFormat，连接数据源中的关系表。由于不同的InputFormat所读入的对象类型各不相同，为了使得构建引擎能够统一处

理，因此又引入了第二个方法parseMapperInput(Object)，对Mapper的每一行输入都会调用该方法一次。该方法的输入是Mapper的输入，具体类型取决于InputFormat，输出为统一的字符串数组，每列为一个元素。整体表示关系表中的一行。这样Mapper就能遍历数据源中的表了。

再看下半部分：

```
public interface IMRInput {
      ......
    //返回一个辅助对象（接口就在下面），参与创建一个 CubeSegment 的构建工作流
    public IMRBatchCubingInputSide getBatchCubingInputSide(CubeSegment seg);

    //本辅助接口代表数据输入端参与创建构建 CubeSegment 的工作流
    //主要负责从数据源提取数据并创建一张临时平表（第一阶段）
    //然后在工作流的末尾清除这张临时表（第四阶段）
    public interface IMRBatchCubingInputSide {

        //返回一个 IMRTableInputFormat，帮助 MR 任务读取之前创建的平表
        public IMRTableInputFormat getFlatTableInputFormat();

        //由构建引擎调用，要求数据源在工作流中添加步骤完成平表的创建
        public void addStepPhase1_CreateFlatTable(DefaultChainedExecutable jobFlow);

        //清理收尾，清除已经没用的平表和其他临时对象
        public void addStepPhase4_Cleanup(DefaultChainedExecutable jobFlow);
    }
}
```

IMRBatchCubingInputSide接口代表数据源配合构建引擎创建工作流计划，这在8.2.3节中已有提及。下面来具体看一下该接口的内容。

·addStepPhase1_CreateFlatTable：由构建引擎调用，要求数据源在工作流中添加步骤完成平表的创建。

·getFlatTableInputFormat:返回一个IMRTableInputFormat，帮助MR任务读取之前创建的平表。

·addStepPhase4_Cleanup:清理收尾，清除已经没用的平表和其他临时对象。

这三个方法将由构建引擎依次调用。

8.2.5 IMROutput2

下面再来看一下IMROutput2接口，所有希望接入MRBatchCubingEngine2的存储都必须实现该接口。这是MRBatchCubingEngine2对存储引擎的要求。

IMROutput2包含IMRBatchCubingOutputSide2和IMRBatchMergeOutputSide2这两个子接口。两者大同小异，分别参与CubeSegment初次构建的工作流和CubeSegment合并时的工作流。这里只介绍前者，后者可以参考Apache Kylin的源代码自行学习：

```
public interface IMROutput2 {

    // 返回一个 IMRBatchCubingOutputSide2 对象，参与创建指定 CubeSegment 的工作流
    public IMRBatchCubingOutputSide2 getBatchCubingOutputSide(CubeSegment seg);

    // 本辅助接口代表数据输出端参与创建构建 CubeSegment 的工作流
    // 包含有三个方法，由构建引擎分别在字典创建后、Cube 计算完成后和清尾阶段调用
    public interface IMRBatchCubingOutputSide2 {

        // 构建引擎在字典创建后调用，存储引擎可以在这里完成预备存储的初始化工作
        public void addStepPhase2_BuildDictionary(DefaultChainedExecutable jobFlow);

        // 构建引擎在 Cube 计算完成之后调用，存储引擎保存 Cube 数据
        public void addStepPhase3_BuildCube(DefaultChainedExecutable jobFlow, String
cuboidRootPath);

        // 构建引擎在收尾阶段调用，清理存储端的任何垃圾
        public void addStepPhase4_Cleanup(DefaultChainedExecutable jobFlow);
    }

    ......
}
```

IMRBatchCubingOutputSide2则代表存储引擎配合构建引擎创建工作

流计划，这在8.2.3节中也有提及。下面来具体看一下该接口的内容。

·addStepPhase2_BuildDictionary：由构建引擎在字典创建后调用。存储引擎可以借此机会在工作流中添加步骤完成存储端的初始化或准备工作。

·addStepPhase3_BuildCube：由构建引擎在Cube计算完毕之后调用，通知存储引擎保存CubeSegment的内容。每个构建引擎计算Cube的方法和结果的存储格式可能都会有所不同。存储引擎必须依照数据接口的协议读取CubeSegment的内容，并加以保存。

·addStepPhase4_Cleanup：由构建引擎在最后清理阶段调用，给存储引擎清理临时垃圾和回收资源的机会。

现在，简单回顾一下，本节主要介绍了Apache Kylin现有的MapReduce构建引擎的设计和原理。目的是通过它来展现数据源、构建引擎和存储引擎这三者之间的依赖和协作关系。此外，还从代码的层面说明了如何使用良好的接口设计和隔离这三者，使它们在协作的同时保持独立性和灵活性，并能够被单独地替换实现。

不论是扩展现有的MapReduce构建引擎，还是设计一个全新的构建引擎，下面的一些基本原则应当都适用。

·构建引擎驱动整体构建的过程，数据源和存储引擎分别从输入和输

出两端加以辅佐。

·构建引擎定义所需的输入和输出接口，数据源和存储引擎提供实现。

·构建引擎在构建过程中，通过(且仅通过)接口调用数据源和存储引擎，以保证三大引擎的独立性和可扩展性。

8.3 数据源扩展

本节将介绍Hive数据源，并以它为范例说明如何为Apache Kylin的MapReduce引擎增添一种数据源。请注意，由于数据源的实现依赖构建引擎对输入接口的定义，因此本节的具体内容只适用于MapReduce引擎。如果要为其他的构建引擎做扩展，请仔细阅读构建引擎的相关文档和代码。

实现数据源扩展之前，首先要对构建引擎有足够的了解。前文已经介绍了MapReduce构建引擎的工作流程和其对数据输入端的接口定义（详情见8.2.3节和8.2.4节）。

实现数据源首先要实现ISource接口。例如HiveSource的主要实现如下：

```
public class HiveSource implements ISource {

    @Override
    public <I> I adaptToBuildEngine(Class<I> engineInterface) {
        if (engineInterface == IMRInput.class) {
            return (I) new HiveMRInput();
        } else {
            throw new RuntimeException("Cannot adapt to " + engineInterface);
        }
    }

    @Override
    public ReadableTable createReadableTable(TableDesc tableDesc) {
        return new HiveTable(tableDesc);
    }
}
```

上面的代码包含两个非常简单的方法。方法adaptToBuildEngine()只能适配IMRInput，返回HiveMRInput实例，也就是暂时只能与MapReduce引擎协作。由于MR v1引擎和MR v2引擎都以IMRInput为输入接口，因此这个实现可以兼容两个版本的MR引擎。另一个方法createReadableTable()返回一个ReadableTable对象，提供读取一张Hive表的能力。

再来看一下HiveMRInput。由于代码较长，且内容较为直观，这里就不再一一赘述了，只做整体上的介绍。

根据IMRInput的定义，HiveMRInput的实现主要分为两部分。一是HiveTableInputFormat对IMRTableInputFormat接口的实现。主要使用了HCatInputFormat作为MapReduce的输入格式，用通用的方式读取所有类型的Hive表。Mapper输入对象为DefaultHCatRecord，统一转换为String[]后交由构建引擎处理。

二是BatchCubingInputSide对IMRBatchCubingInputSide的实现。主要实现了在构建的第一阶段创建平表的步骤。首先用count(*)查询获取Hive平表的总行数，然后用第二句HQL创建Hive平表，同时添加参数根据总行数分配Reducer数目。合理地分配Reducer数目非常重要。它不仅会影响HQL的并发度和执行速度，同时还会影响下一轮构建Cube的Mapper输入个数。该数目不能太大，不然会导致Reducer和Mapper数目过多，MR系统执行单位不够，排长队等待执行。该数目也不能太小，不然Reducer和Mapper数目太少，并发度不够，执行缓慢。

具体细节请参阅Apache Kylin源代码。

8.4 存储扩展

本节将介绍HBase存储引擎，并以它为范例说明如何为Apache Kylin的MapReduce引擎增添一种存储引擎。请注意，由于存储引擎的实现依赖构建引擎对输出接口的定义，因此本节的具体内容只适用于MapReduce引擎。如果要为其他的构建引擎做扩展，请仔细阅读构建引擎的相关文档和代码。

实现存储扩展之前，首先要对构建引擎有足够的了解。前文已经介绍了MapReduce构建引擎的工作流程和其对数据输出端的接口定义（详情见8.2.3节和8.2.4节）。

实现存储引擎的入口在于对IStorage接口的实现。比如HBaseStorage的代码摘要如下：

```
public class HBaseStorage implements IStorage {
    ......

    @Override
    public <I> I adaptToBuildEngine(Class<I> engineInterface) {
        if (engineInterface == IMROutput.class) {
            return (I) new HBaseMROutput();
        } else if (engineInterface == IMROutput2.class) {
            return (I) new HBaseMROutput2Transition();
        } else {
            throw new RuntimeException("Cannot adapt to " + engineInterface);
        }
    }

    ......

    @Override
    public IStorageQuery createQuery(IRealization realization) {

        if (realization.getType() == RealizationType.CUBE) {
            ......
            return ret;
        } else {
                throw new IllegalArgumentException("Unknown realization type " + realization.
getType());
            }
        }
    }
```

首先是adaptToBuildEngine()方法，能够适配IMROutput和IMROutput2两个版本的输出接口，适配MR v1和MR v2两代引擎。其次是createQuery()方法，返回对指定IRealization(数据索引实现)的一个查询对象。因为HBase存储是为Cube定制的，所以只支持Cube类型的数据索引。具体的IStorageQuery实现应根据存储引擎的版本而有所不同。主要是因为从Apache Kylin v1.5开始引入了分片存储和并行扫描，以致底层的HBase存储格式会有所区别，因此查询的实现也有了差别。

再来简单介绍一下HBaseMROutput2Transition对IMROutput2接口的实现：

```
public class HBaseMROutput2Transition implements IMROutput2 {

    @Override
    public IMRBatchCubingOutputSide2 getBatchCubingOutputSide(CubeSegment seg) {
        return new IMRBatchCubingOutputSide2() {
            HBaseMRSteps steps = new HBaseMRSteps(seg);

            @Override
            public void addStepPhase2_BuildDictionary(DefaultChainedExecutable jobFlow) {
                jobFlow.addTask(steps.createCreateHTableStepWithStats(jobFlow.getId()));
            }

            @Override
            public void addStepPhase3_BuildCube(DefaultChainedExecutable jobFlow, String cuboidRootPath) {
                jobFlow.addTask(steps.createConvertCuboidToHfileStep(cuboidRootPath, jobFlow.getId()));
                jobFlow.addTask(steps.createBulkLoadStep(jobFlow.getId()));
            }

            @Override
            public void addStepPhase4_Cleanup(DefaultChainedExecutable jobFlow) {
                //nothing to do
            }
        };
    }
    ......
```

观察IMRBatchCubingOutputSide2的实现。它在两个时间点参与Cube构建的工作流。一是在字典创建之后（Cube构造之前），在addStepPhase2_BuildDictionary()中添加了“创建HTable”这一步，估算最终CubeSegment的大小，并以此来切分HTable Regions，创建HTable。

第二个插入点是在Cube计算完毕之后，由构建引擎调用addStepPhase3_BuildCube()。这里要将Cube保存为HTable，实现分为“转

换HFile”和“批量导入HTable”两步。因为直接插入HTable比较缓慢，为了最快速地将数据导入到HTable，采取了Bulk Load的方法。先用一轮MapReduce将Cube数据转换为HBase的存储文件格式HFile，然后就可以直接将HFile导入空的HTable中，完成数据导入。

最后一个插入点addStepPhase4_Cleanup()是空实现，对于HBase存储来说没有需要清理的资源。

8.5 聚合类型扩展

Apache Kylin的核心思想是预聚合，用预先计算来代替查询时计算。聚合类型代表了系统的关键能力。处处为扩展性和灵活性设计的Apache Kylin在这里也没有令人失望。开发者完全可以定制新的聚合类型，以满足行业和领域的特殊需要。

本节将以基于HyperLogLog算法的去重计数为例，讲解Apache Kylin聚合类型的扩展接口和实现方法。

8.5.1 聚合的JSON定义

要了解聚合类型的工作原理和扩展方式，先要从聚合在Cube元数据中的定义开始。下面是基于HyperLogLog实现去重基数的一个度量在Cube中的定义：

```
/* 来自 test_kylin_cube_with_slr_left_join_desc.json */
{
    "uuid": "bbbba905-1fc6-4f67-985c-38fa5aeafd92",
    "name": "test_kylin_cube_with_slr_left_join_desc",
    ......
    "measures": [
    ......
        {
            "name": "SELLER_CNT_HLL",
            "function": {
                "expression": "COUNT_DISTINCT",          /* 聚合函数 */
                "parameter": {
                    "type": "column",
                    "value": "SELLER_ID",
                    "next_parameter": null
                },
                "returntype": "hllc(12)"                  /* 聚合数据类型 */
            }
        }
  ......
```

注意定义一个聚合类型的关键信息：

·聚合函数，这里是“COUNT_DISTINCT”。

·聚合数据类型，这里是“hllc(12)”（注意区别，这是聚合数据类型，而不是聚合类型）。

一种聚合函数可以有多种实现，因此单单靠聚合函数并不能确定一种聚合类型的实现。比如COUNT_DISTINCT就有基于HyperLogLog的近似算法实现和基于BitMap的精确实现。聚合函数加上聚合数据类型才能唯一确定一种聚合类型。

这里可根据函数“COUNT_DISTINCT”和类型“hllc(12)”来确定用户定义的是基于HyperLogLog的精度为12的去重计数度量。

在Cube中引用的每一种聚合类型都需要有具体的实现才能工作。提供聚合类型实现的方法将在下文介绍。

8.5.2 聚合类型工厂

前面已经提到过需要根据“聚合函数”和“聚合数据类型”来唯一确定一个“聚合类型”。聚合类型工厂（MeasureTypeFactory）就是聚合类型的工厂类。其定义的主体如下：

```
//类型 T 是聚合数据的类型
abstract public class MeasureTypeFactory<T> {

    //创造一个 MeasureType 实例，根据指定的聚合函数和聚合数据类型
    abstract public MeasureType<T> createMeasureType(String funcName, DataType dataType);

    //返回支持的聚合函数，比如 "COUNT_DISTINCT"
    abstract public String getAggrFunctionName();

    //返回支持的聚合数据类型，比如 "hllc"
    abstract public String getAggrDataTypeName();

    //返回聚合数据类型的序列化器，注意序列化器的实现必须线程安全
    abstract public Class<? extends DataTypeSerializer<T>> getAggrDataTypeSerializer();
    ......
```

每一个聚合类型都必须有对应的工厂类来提供实例。注册聚合类型工厂的方式是通过修改kylin.properties来实现的。比如要添加一种新的聚合类型MyAggrType，可以在kylin.properties中添加一行：

```
kylin.cube.measure.customMeasureType.FUNC_NAME=some.package.MyAggrTypeFactory
```

这里“FUNC_NAME”必须是要扩展的聚合函数的名称，“some.package.MyAggrTypeFactory”为聚合类型的工厂类全名。

在启动时，系统会扫描kylin.properties，将所有前缀

为“kylin.cube.measure.customMeasureType.”配置项读出，注册为扩展聚合类型工厂。在注册过程中，工厂的getAggrFunctionName()和getAggrDataTypeName()会被调用，以确认工厂所支持的聚合函数和聚合数据类型。

在保存Cube时，系统会校验所有度量所引用的聚合类型是否都有对应的实现注册。如果有未知的聚合类型，系统将会报错。

有了工厂类，系统就会在Cube计算和查询的各个阶段调用createMeasureType()方法创建聚合类型实例，再通过它聚合数据。下面将进行详细介绍。

8.5.3 聚合类型的实现

根据聚合函数和聚合数据类型，MeasureType由MeasureTypeFactory创造。Measure Type中包含了聚合从定义到计算，从查询到存储的全部逻辑，是一个比较大的接口。下面将分多个段落来逐一介绍。

先来看定义相关的部分：

```
// 类型 T 是聚合数据的类型
abstract public class MeasureType<T> {

    // 检查用户定义的 FunctionDesc 是否有效
    public void validate(FunctionDesc functionDesc) throws IllegalArgumentException {
        return;
    }

    // 该聚合数据类型是否需要较大的内存
    public boolean isMemoryHungry() {
        return false;
    }

    // 聚合是否只应用在 Base Cuboid 上
    public boolean onlyAggrInBaseCuboid() {
        return false;
    }
    ......
```

上述代码的说明如下。

·MeasureType的泛型参数T代表聚合数据类型。比如，以HyperLogLog为例，它的聚合数据类型是HyperLogLogPlusCounter。

·validate()方法校验传入的FunctionDesc（即度量定义中聚合函数的

部分）是否合法。在创建Cube的过程中这个方法会被多次调用，检查用户定义的度量是否正确，比如数据的精度是否在有效范围内，等等。如果校验失败，那么该方法应该抛出IllegalArgumentException。

·isMemoryHungry()报告该聚合数据在运算时是否需要较多的内存。一些基本的聚合函数比如SUM和COUNT在计算时只需要几个字节，然而类似HyperLogLogPlusCounter的大型数据结构可能一个就需要10KB乃至100KB的内存。需要在内存分配上给予特别对待。

·onlyAggrInBaseCuboid()定义该聚合运算是否只发生在Base Cuboid上。如果是，那么在其他Cuboid上该聚合函数将被跳过。

下面是MeasureType中关于计算的接口定义：

```
abstract public class MeasureType<T> {
    ......

    //返回一个 MeasureIngester 用于初始化一个聚合数据对象
    abstract public MeasureIngester<T> newIngester();

    //返回一个 MeasureAggregator 用来聚合数据
    abstract public MeasureAggregator<T> newAggregator();

    //返回聚合函数中是否需要用到字典
    public List<TblColRef> getColumnsNeedDictionary(FunctionDesc functionDesc) {
        return Collections.emptyList();
    }
    ......
```

上述代码说明如下。

·newIngester()返回一个MeasureIngester对象。MeasureIngester也是一

个抽象类，需要实现。其中主要的方法是valueOf()，它能根据一行原始记录（也就是数据源的一行中输入String[]，详见8.3）初始化一个聚合数据对象。例如对HyperLogLog来说，所谓的初始化就是创造一个HyperLogLogPlusCounter，然后将原始记录中的被计数字段加入其中。

·newAggregator()返回一个MeasureAggregator对象。MeasureAggregator也是一个抽象类，其上需要实现的方法主要有aggregate()和getState()，用来聚合由MeasureIngester产生的聚合数据对象。

·getColumnsNeedDictionary()是一个比较特殊的方法，用来声明一个或多个字段需要用到的字典。如果有声明，那么构建引擎将在构建过程中创建声明字段的字典，并提交给MeasureIngester使用。

下一个方法是关于Cube的选择。我们知道在查询过程中，用户输入的是SQL语句，其中用到了字段和聚合函数。一个Cube必须满足SQL中所有的字段和聚合函数，才能被选中来回答这条查询语句：

```
    abstract public class MeasureType<T> {
        ......

        //判断一个度量是否能满足未匹配的维度和聚合函数
        public CapabilityInfluence influenceCapabilityCheck(Collection<TblColRef> unmatched
Dimensions, Collection<FunctionDesc> unmatchedAggregations, SQLDigest digest, MeasureDesc
measureDesc) {
            return null;

}

......
```

对于基本的如SUM和COUNT_DISTINCT之类的聚合函数，系统能够自行判断是否与查询匹配。然而对于扩展聚合函数，用户可能希望定制匹配规则。比如TopNMeasureType和RawMeasureType就能匹配字段，而不像普通聚合类型那样只能匹配聚合函数。

在Cube匹配的过程中，上述inf？luenceCapabilityCheck()将为每个自定义聚合度量调用一次，以传入未匹配的字段和聚合函数。若（自定义）度量能匹配部分字段或聚合函数，则应当修改传入的集合，去掉已匹配的部分，同时返回一个CapabilityInf？luence对象，标记自己对匹配过程的影响。只有匹配过程完毕时所有的字段和聚合函数都被匹配，查询才能继续，否则，系统将报告没有匹配的Cube，并异常退出该查询。

如果存在多个Cube都能满足的一个查询，这时候Cost（开销）较小的Cube会被选中。所有参与了匹配过程的，且对匹配有贡献的聚合类型都有机会调整所在Cube的Cost，调整是通过CapabilityInf？luence对象上的suggestCostMultiplier()方法来完成的。比如一个定义了TopN的Cube和一个普通的Cube都能满足“今日销量前10”这个查询，区别在于前者有TopN度量做了预计算，而后者是通过查询时聚合，然后排序取前10来完成的。这时TopN聚合类型就会通过CapabilityInf？luence.suggestCostMultiplier()返回一个小于1的修正乘数，使修正后的Cost远远小于普通的Cube，从而保证TopN在回应查询时更具优势。

下面两个方法与查询组件Apache Calcite有关。Apache Calcite是流行

的数据管理组件，具有SQL解析、优化和处理的能力。其内容丰富，已经超出了本书的范围，在此不做详细介绍。

```
abstract public class MeasureType<T> {
    ......

    //是否需要重写 Calcite 层的聚合运算
    abstract public boolean needRewrite();

    //返回 Calcite 聚合函数的实现类
    abstract public Class<?> getRewriteCalciteAggrFunctionClass();
    ......
```

代码说明如下：

·needRewrite()返回该聚合函数是否需要重写Calcite层。因为自定义函数基本上都是SQL语句的扩展，Calcite不可能包含相关实现，因此这里一般要返回“是”。

·getRewriteCalciteAggrFunctionClass()，如果上面返回“是”，那么这个方法会被调用来获取一个实现了Calcite聚合函数接口的实现类。其内容与MeasureAggregator大致类似，只是接口的形式略有不同。

更多关于Apache Calcite的内容，请查阅Apache Calcite的官方文档。

最后一部分是查询时存储的读取和Tuple的填入。所谓的Tuple是关系运算术语，表示关系表上的一行，Tuple填入指将预聚合的度量值填入关系表，返回查询结果的过程：

```
abstract public class MeasureType<T> {
    ......

    //返回是否启用高级的 Tuple 填入
    public boolean needAdvancedTupleFilling() {
        return false;
    }

    //简单的 Tuple 填入实现
    public void fillTupleSimply(Tuple tuple, int indexInTuple, Object measureValue) {
        tuple.setMeasureValue(indexInTuple, measureValue);
    }

    //返回一个高级 Tuple 填入实现
    public IAdvMeasureFiller getAdvancedTupleFiller(FunctionDesc function, TupleInfo
returnTupleInfo, Map<TblColRef, Dictionary<String>> dictionaryMap) {
        throw new UnsupportedOperationException();
    }

    //高级 Tuple 填入接口
    public static interface IAdvMeasureFiller {

            //读入一个度量值
            void reload(Object measureValue);

            //返回能继续填入的行数
            int getNumOfRows();

            //填入内容到下一个 Tuple
            void fillTuple(Tuple tuple, int row);
        }
}
```

代码说明如下。

·needAdvancedTupleFilling()返回是否启用高级的Tuple填入。Tuple填入分为简单和高级两种模式。在简单模式下，默认一个度量值对应一条关系记录，这也是默认的实现。高级模式允许一个度量值被分解并填入多条关系记录。

·fillTupleSimply()为简单填入模式的实现。传入参数包含度量值和要填入的Tuple对象。

·getAdvancedTupleFiller()在高级填入模式下启用，返回一个IAdvMeasureFiller对象，其包含了更多的方法可实现度量值到Tuple的一对多填入。

最后IAdvMeasureFiller接口包含3个方法：

1）reload()为每个度量值被调用一次。每次的度量值将被用于填写后续的Tuple，直到其内容耗尽为止。那时reload()将被再次调用，填充新的度量值。

2）getNumOfRows()返回最近填充的度量值还能填写多少个Tuple。

3）fillTuple()用于填充下一个Tuple。

以上是MeasureType接口的全部内容，包含聚合的定义、计算、查询和存储各方面。要实现一个全新的聚合类型是相当复杂的工作。好在Apache Kylin所有的内置聚合函数也是从MeasureType继承而来的，本身也是很好的范例，推荐参阅HLLCMeasureType和Top-NMeasureType的源代码加深对这部分的理解。

现在简单回顾一下聚合类型的扩展步骤。要添加一种新的聚合类型，首先要确定“聚合函数”和“聚合数据类型”。然后实现相应的

MeasureTypeFactory并在kylin.properties中注册。接着在Cube定义中就可以引用该聚合类型了。MeasureType会在运行时通过MeasureTypeFactory创建，接管聚合的定义、计算、查询、存储等一系列过程。

8.6 维度编码扩展

Apache Kylin对维度的保存也采用了编码的机制。通过编码可以极大地提高压缩率，用更小的空间保存更多的数据，同时也能提高读写的速度。默认维度编码主要有“字典”和“定长”两种。除此之外，开发者还可以订制新的维度编码。本节将介绍如何扩展Apache Kylin，增添新的维度编码。

8.6.1 维度编码的JSON定义

先来看一下维度编码在Cube定义中是如何被使用的，从而对其有一个感性的认识：

```
/* 来自 test_kylin_cube_with_slr_left_join_desc.json */
{
    "uuid": "bbbba905-1fc6-4f67-985c-38fa5aeafd92",
    "name": "test_kylin_cube_with_slr_left_join_desc",
    ......
    "rowkey": {
        "rowkey_columns": [
            ......
            {
                "column": "lstg_format_name",
                "encoding": "fixed_length:12"
            },
            {
                "column": "lstg_site_id",
                  "encoding": "dict"
              },
              {
                  "column": "slr_segment_cd",
                  "encoding": "dict"
              }
        ]
    },
    ......
```

从定义中可以看到维度编码其实是定义在“rowkey”段落中的，也就是只有被保存在Cube中的那些维度才需要编码，对于不需要在Cube中存储的维度，比如衍生维度，是不需要编码的。

上例中出现了两种内置编码，分别是“f？ixed_length:12”和“dict”。维度编码除了有类别的区分之外，比如dict和f？ixed_length，还可以带有参数，比如“f？ixed_length:12”中的长度12。

8.6.2 维度编码工厂

那么dict和f？ixed_length编码又是如何映射到相关的实现中的呢？这就要需要介绍维度编码工厂——DimensionEncodingFactory了。

维度编码工厂负责维度编码的注册和实例的创建，主要的接口如下：

```
public abstract class DimensionEncodingFactory {
    ......
    //返回所支持的编码名称
    abstract public String getSupportedEncodingName();

    //返回一个新的维度编码实例
    abstract public DimensionEncoding createDimensionEncoding(String encodingName,
String[] args);
}
```

代码说明如下：

·getSupportedEncodingName()返回所支持的编码名称，在注册编码时使用。

·createDimensionEncoding()创建一个新的编码实例，注意传入的编码名称和参数，传入的名称与所支持的编码名称一定要相同。

要想添加一种新的维度编码，首先必须在系统中注册其编码工厂。方法是修改kylin.properties文件，添加kylin.cube.dimension.customEncodingFactories参数。比如：

```
kylin.cube.dimension.customEncodingFactories=some.package.MyEncodingFactory
```

在系统初始化阶段，将会读取kylin.cube.dimension.customEncodingFactories参数。它是一个逗号分隔的字符串，其中每个单位都是一个编码工厂类的全名。通过反射创建工厂类，然后调用getSupportedEncodingName()获得所支持的编码名称(比如dict和f？ixed_length)，注册到系统中。

注册后的编码就可以被Cube使用了。如果Cube引用了不存在的维度编码，那么系统将会在加载Cube元数据时出错，相关的Cube会被忽略，错误将会出现在日志中。

8.6.3 维度编码的实现

注册了维度编码工厂，通过createDimensionEncoding()，维度编码就能在需要的时候被创建，接管维度值到代码的编码和解码工作。

Apache Kylin对维度编码有如下的基本要求。

·等长：编码后，所有维度值的代码长度相同。

·双向：编码和原来的值可以双向转换。

·保序：编码二进制大小与原值的大小保持一致。

接下来看一下DimensionEncoding接口的具体定义：

```
//注意维度编码是可以序列化的
public abstract class DimensionEncoding implements Externalizable {
    ......

    //判断指定的代码是否代表NULL
    public static boolean isNull(byte[] bytes, int offset, int length) {
        ......
    }

    //获得固定的代码长度
    abstract public int getLengthOfEncoding();

    //将给定的维度值(以byte形式表示的字符串)转换为编码
    abstract public void encode(byte[] value, int valueLen, byte[] output, int output
Offset);

    //将给定的编码转换为维度值(以String形式返回)
    abstract public String decode(byte[] bytes, int offset, int len);

    //返回一个DataTypeSerializer,以序列化器的接口实现同样的编码解码功能
    abstract public DataTypeSerializer<Object> asDataTypeSerializer();
}
```

首先可以看到维度编码必须是Externalizable，这保证了编码能被序列化传递到分布式系统的任何地方。

·isNull()静态方法用来判断一个编码是否为NULL。这也是编码系统的一个约定，以全0xff代码代表NULL。

·getLengthOfEncoding()返回编码的固定长度，也就是代码的二进制字节数。

·encode()和decode()方法是双向编码解码的实现。注意接口略有不对称，在encode()中维度值将以UTF-8编码的byte数组形式给出，而在decode()时将以String形式返回。不管如何表示，维度值的本质就是一个字符串，这是确定的。将来的版本可能会重构这对接口，使其更加对称和

美观。

·asDataTypeSerializer()会以序列化器的形式封装编码解码的过程。因为编码解码也可以看作维度值到字节流的序列化和反序列化过程，在不少代码中以序列化的接口来调用会更加自然。因此追加了这个方法，将编码解码逻辑包装成序列化器的形式返回。

实现了上述的DimensionEncoding，一个维度编码就完成了。Apache Kylin内置的DictionaryDimEnc和FixedLenDimEnc也可以作为实现的参考。

现在简单回顾一下维度编码的扩展步骤。要添加一种新的维度编码，首先要实现DimensionEncodingFactory并在kylin.properties中注册。这样新的编码名称就可以在Cube定义中使用了。运行时系统会通过DimensionEncodingFactory创建DimensionEncoding，进行编码解码工作。

实现的过程中尤其要注意Apache Kylin对维度编码的基本要求，等长、双向、保序。还有特殊的0xff保留编码表示的NULL。如果这些约定被破坏，那么在查询和构建过程中将出现结果错误和其他异常。应该尽早用单元测试覆盖这些需求和边界情况，以确保在集成环境中不会因为编码出错而产生不可知的异常。

8.7 小结

本章讲述了Apache Kylin各方面的可扩展性，包括总体的可扩展架构和数据源、构建引擎、存储引擎三大部件，以及聚合类型和维度编码。可以看到，可扩展性贯穿了Apache Kylin的所有关键功能，是系统的核心设计理念之一。这保证了Apache Kylin能够更快速地适应新的技术趋势，在澎湃的技术进化浪潮中始终保持领先的地位。

最后，Apache Kylin的开发非常活跃，有上百位开发者在随时修改和提交代码。本书撰写时Apache Kylin的版本为v1.5.2.1，因为图书发行有时间延迟，如果本章涉及的具体代码细节与最新的Apache Kylin不符，作者在这里表示歉意。虽然具体代码多变，但抽象和设计仍然相对稳定，相信本章一定能为Apache Kylin的开发爱好者带来帮助。

第9章　Apache Kylin的企业级功能

Apache Kylin是为满足企业的大数据分析需求而诞生的，基于此，从一开始就考虑了企业对数据软件各方面的要求，如安全验证、权限控制、高可用、可扩展等。这些功能在后来的企业应用中被证明是非常有必要的。本章将介绍Kylin的这些企业级功能。

9.1 身份验证

身份验证模块可为Kylin的Web界面和RESTful Service提供安全验证，它检查用户提供的用户名和密码以决定是否让其登录或调用API。

Kylin的Web模块使用Spring框架构建，在安全实现上选择了Spring Security。Spring Security是Spring项目组中用来提供安全认证服务的框架，它广泛支持各种身份验证模式，这些验证模型大多由第三方提供，Spring Security也提供了自己的一套验证功能。

·注意　下文将假设读者熟悉Java Web Application和Spring框架。这些框块内容丰富，但不在本书范围之内。互联网上有大量相关的资料，请有需要的读者自行查阅。参考资料地址为：

http://docs.oracle.com/javaee/7/tutorial/partwebtier.htm#BNADP

http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

下面介绍Kylin是如何配置使用Spring Security的。首先，在Web模块的主配置文件web.xml中，可以看到安全配置文件kylinSecurity.xml被加入Spring配置文件列表，同时声明了相应的Listener，具体代码如下：

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:applicationContext.xml
        classpath:kylinSecurity.xml
        classpath*:kylin-*-plugin.xml
    </param-value>
</context-param>

<filter>
    <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

然后，在安全配置文件kylinSecurity.xml中，告知Spring Security框架应如何构造authentication-manager的对象。构造authentication-manager对象需要一个或多个“authentication-provider”对象；而authentication-provider又需要user-service对象来提供用户的信息。

在kylinSecurity.xml里，Kylin提供了三个配置prof？ile：“testing”、“ldap”和“saml”，依次对应于三种用户验证方式：自定义验证、LDAP验证和单点登录验证。以下三个小节将对它们分别进行介绍。

9.1.1 自定义验证

自定义验证是基于配置文件的一种简单验证方式；由于它对外依赖少，开箱即用，所以是Kylin默认的用户验证方式；同时由于它缺乏灵活性、安全性低，因此建议仅在测试阶段使用。下面是自定义验证（也就是“testing”prof？ile）在kylinSecurity.xml中相关的配置：

```
    <beans profile="testing">
        <!-- user auth -->
        <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.
BCryptPasswordEncoder" />

        <scr:authentication-manager alias="testingAuthenticationManager">
            <scr:authentication-provider>
                <scr:user-service>
                    <scr:user name="MODELER" password="$2a$10$Le5ernTeGNIARwMJsY0WaOLioN
Qdb0QD11DwjeyNqqNRp5NaDo2FG" authorities="ROLE_MODELER" />
                    <scr:user name="ANALYST" password="$2a$10$s4INO3XHjPP5Vm2xH027Ce9
QeXWdrfq5pvzuGr9z/lQmHqi0rsbNi" authorities="ROLE_ANALYST" />
                    <scr:user name="ADMIN" password="$2a$10$o3ktIWsGYxXNuUWQiYlZXOW5hW
cqyNAFQsSSCSEWoC/BRVMAUjL32" authorities="ROLE_MODELER, ROLE_ANALYST, ROLE_ADMIN" />
                </scr:user-service>
                <scr:password-encoder ref="passwordEncoder" />
            </scr:authentication-provider>
        </scr:authentication-manager>
    </beans>
```

可以看到，这个“userService”是基于配置实现的，它默认只有三个用户记录：MODELER（具有ROLE_MODLER角色）、ANALYST（具有ROLE_ANALYST角色）和ADMIN（具有ROLE_MODLER、ROLE_ANALYST及ROLE_ADMIN角色）。关于这三个角色，将在9.2节“授权”中详细介绍。密码则使用“passwordEncoder”，也就是BCrypt加密的形

式。要添加、删除或修改某个用户信息，只需要修改这里的内容就可以了。

9.1.2 LDAP验证

LDAP（Lightweight Directory Access Protocol，轻量级目录访问协议）用于提供被称为目录服务的信息服务。目录以树状的层次结构来存储数据，可以存储包括组织信息、个人信息、Web链接、JPEG图像等各种信息。

支持LDAP协议的目录服务器产品有很多，大多数企业也都使用LDAP服务器来存储和管理公司的组织和人员结构；集成LDAP服务器在完成用户验证时，不仅可以避免重复创建用户、管理群组等繁琐的管理步骤，还可以提供更高的便捷性和安全性。Apache Kylin连接LDAP验证的流程如图9-1所示。

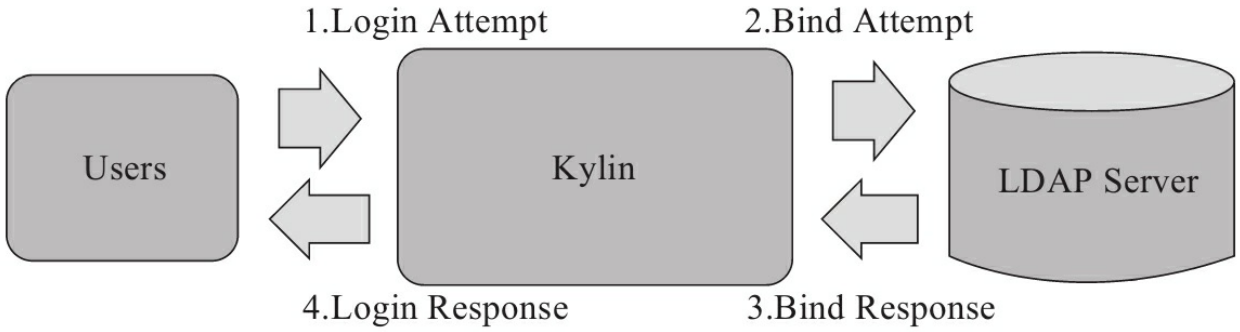


图9-1 LDAP验证流程

Kylin的LDAP验证是基于Spring Security提供的LDAP验证器实现的，在其上略有扩展。下面是kylinSecurity.xml中“ldap”的配置片段：

```
    <beans profile="ldap">
        <scr:authentication-manager alias="ldapAuthenticationManager">
            <!-- do user ldap auth -->
            <scr:authentication-provider ref="kylinUserAuthProvider"></scr:authentication-
provider>

            <!-- do service account ldap auth -->
            <scr:authentication-provider ref="kylinServiceAccountAuthProvider"></scr:
authentication-provider>
        </scr:authentication-manager>

    </beans>
```

LDAP的authentication-manager使用了两个authentication-provider：一个名为“kylinUserAuth Provider”，另一个名为“kylinServiceAccountAuthProvider”。这两个provider都会去LDAP服务器中查询信息，都是类org.apache.kylin.rest.security.KylinAuthenticationProvider的实例，只是查询LDAP的属性（searchBase、searchPattern）不同。这样设计的目的是，访问Kylin的用户通常有两种类型：一种是以人的身份登录来做各种操作，另一种是以API的方式调用Kylin的各种服务，通常被称为服务账户；在LDAP中，服务账户与普通账户（User Account）往往是分开管理的。分开这两种类型，可以让Kylin管理员根据自己的环境做灵活的设定。

接下来以“kylinUserAuthProvider”为例介绍更多的细节，请看如下的配置代码：

```
    <bean id="ldapSource" class="org.springframework.security.ldap.DefaultSpringSec
urityContextSource">
        <constructor-arg value="${ldap.server}" />
        <property name="userDn" value="${ldap.username}" />
        <property name="password" value="${ldap.password}" />
    </bean>

    <bean id="kylinUserAuthProvider" class="org.apache.kylin.rest.security.
KylinAuthenticationProvider">
        <constructor-arg>
        <bean id="ldapUserAuthenticationProvider" class="org.springframework.security.
ldap.authentication.LdapAuthenticationProvider">
        <constructor-arg>
            <bean class="org.springframework.security.ldap.authentication.BindAuthenticator">
                <constructor-arg ref="ldapSource" />
                <property name="userSearch">
                    <bean id="userSearch" class="org.springframework.security.ldap.search.
FilterBasedLdapUserSearch">
                        <constructor-arg index="0" value="${ldap.user.searchBase}" />
                        <constructor-arg index="1" value="${ldap.user.searchPattern}" />
                        <constructor-arg index="2" ref="ldapSource" />
                    </bean>
                </property>
            </bean>
          </constructor-arg>
          <constructor-arg>
              <bean class="org.apache.kylin.rest.security.AuthoritiesPopulator">
              <constructor-arg index="0" ref="ldapSource" />
              <constructor-arg index="1" value="${ldap.user.groupSearchBase}" />
              <constructor-arg index="2" value="${acl.adminRole}" />
              <constructor-arg index="3" value="${acl.defaultRole}" />
              </bean>
      </constructor-arg>
      </bean>
      </constructor-arg>
 </bean>
```

在上述代码片段中，“kylinUserAuthProvider”使用了两个构造器对象，“ldapUserAuth entication Provider”和“org.apache.kylin.rest.security.AuthoritiesPopulator”。“ldapUserAticationProvider”使用配置的信息（ldap.user.searchBase、

ldap.user.searchPattern、ldap.server、ldap.username、ldap.password）连接LDAP服务器做bind操作，查询并获取用户信息。接下来，“org.apache.kylin.rest.security.AuthoritiesPopulator”会使用查询获得的用户信息，进一步查取此用户所属的群组（Group），然后群组将根据配置的属性（acl.adminRole）来生成用户的角色（Role）信息，从而完成登录验证。

·注意　org.apache.kylin.rest.security.KylinAuthenticationProvider类里封装了一个authenti cation Provider实例，这个authenticationProvider才是真正的验证器。KylinAuthenticationProvider在其上做了缓存：如果用户登录的信息之前已经被缓存，那么就直接返回结果而不去查询真正的验证器如LDAP服务器。这样做的好处是，一来可以避免对LDAP服务器造成访问压力，二来可以提高验证的效率，主要是为高频率的API调用而考虑的。

以上介绍了Kylin里基于LDAP用户验证的原理，接下来介绍一下如何启用LDAP验证。在正常情况下，只需要安装好LDAP服务器，创建用户和群组，然后在Kylin的conf/kylin.properties里配置相应的属性即可。

在conf/kylin.properties里有如下与LDAP相关的配置属性：

```
kylin.security.profile=testing

# default roles and admin roles in LDAP, for ldap and saml
acl.defaultRole=ROLE_ANALYST,ROLE_MODELER
acl.adminRole=ROLE_ADMIN

#LDAP authentication configuration
ldap.server=ldap:// ldap_server:389
ldap.username=
ldap.password=

#LDAP user account directory;
ldap.user.searchBase=
ldap.user.searchPattern=
ldap.user.groupSearchBase=

#LDAP service account directory
ldap.service.searchBase=
ldap.service.searchPattern=
ldap.service.groupSearchBase=
```

首先，要设置kylin.security.prof？ile为“ldap”，并配置“ldap.server”“ldap.username”和“ldap.password”的值，提供LDAP服务器的地址和认证方式（如果需要认证的话）。请注意这里“ldap.password”的值需要加密（加密方法为AES）。下载任意版本的Apache Kylin的源代码，在IDE里运行org.apache.kylin.rest.security.PasswordPlaceholderConf？igure，传入“AES<your_password>”作为参数（替换<your_password>为非加密的原始密码），可以获得改密码的加密值。

其次，设置“ldap.user.searchBase”“ldap.user.searchPattern”和“ldap.user.groupSearchB

在LDAP目录结构中开始查询用户的基础节点；“ldap.user.searchPattern”是查找一个用户记录的模式，如（cn={0}）是指检查某个记录的cn（Common Name）是否跟用户提供的名称相当，匹配成功即使用此用户记录。管理员也可以在这里加入更复杂的过滤条件。如果用户没有匹配成功，那么系统会报UsernameNotFoundException的错误。

属性“ldap.user.groupSearchBase”配置了在LDAP中查找群组的基础节点。当找到用户记录时，Kylin会从此节点往下查找用户所属的群组信息，群组会被映射成角色（Role）。前面提到过，Kylin里有三种用户角色：ROLE_ANALYST、ROLE_MODELER和ROLE_ADMIN。其中前两个是默认赋予的角色，第三个是管理员角色，需要跟某个LDAP的群组相关联（通过acl.adminRole配置）。如果用户在LDAP的群组A中，登录成功后会获得ROLE_A的角色；如果要将群组A做为Kylin管理员群，那么就要配置acl.adminRole＝ROLE_A。

下面是一个使用了LDAP认证的配置示例：

```
kylin.security.profile=ldap

# default roles and admin roles in LDAP, for ldap and saml
acl.defaultRole=ROLE_ANALYST,ROLE_MODELER
acl.adminRole=ROLE_KYLIN-ADMIN


#LDAP authentication configuration
ldap.server=ldap://10.0.0.123:389
ldap.username=cn=Manager,dc=example,dc=com
ldap.password=<password_hash>

#LDAP user account directory;
ldap.user.searchBase=ou=People,dc=example,dc=com
ldap.user.searchPattern=(&(cn={0}))
ldap.user.groupSearchBase=OU=Groups,DC=example,DC=com


#LDAP service account directory
ldap.service.searchBase=ou=Service,dc=example,dc=com
ldap.service.searchPattern=(&(cn={0}))
ldap.service.groupSearchBase=OU=Groups,DC=example,DC=com
```

9.1.3 单点登录

单点登录(Single Sign On, SSO)是一种高级的企业级认证服务。用户只需要登录一次，就可以访问所有相互信任的应用系统；它具有一个账户多处使用、避免频繁登录、降低泄漏风险等优点。

安全断言标记语言(Security Assertion Markup Language, SAML)是一个基于XML的标准，用于在不同的安全域(Security Domain)之间交换认证和授权数据。SAML是实现SSO的一种标准化技术，是由国际标准化组织OASIS制定和发布的。

为了满足企业对安全的更高要求，Kylin提供了对标准SAML的单点登录验证服务，其中就采用了Spring Security SAML Extension。关于Spring Security SAML Extension的使用可以参考Spring网站的文档 http://docs.spring.io/autorepo/docs/spring-security-saml/1.0.x-SNAPSHOT/reference/htmlsingle/。

下面简要介绍一下在Kylin里启用SSO的步骤，以便读者有一个总体的了解。

1)生成IDP元数据的配置文件：联系IDP(ID Provider)，也就是提供SSO服务的供应商，生成SSO metadata f？ile。这是一个XML文件，其中包

含了IDP的服务信息、当前应用的回调URL、加密证书等必要信息。生成的配置文件，需要安装在Kylin Server的classpath上。

2）生成JKS的keystore：Kylin需要加密SSO请求，故需要将含有加密的秘钥和公钥导入到一个keystore中，然后将keystore的信息配置在Kylin的kylinSecurity.xml中。

3）激活更长加密（Higer Ciphers）：检查并确认已经下载安装了Java Cryptography Extension（JCE）Unlimited Strength Jurisdiction Policy Files；如果没有，则下载并复制local_policy.jar和US_export_policy.jar到$JAVA_HOME/jre/lib/security目录下。

4）部署IDP配置文件和keystore到Kylin：将IDP配置文件命名为sso_metadata.xml，然后复制到Kylin的classpath中，如$KYLIN_HOME/tomcat/webapps/kylin/WEB-INF/classes；将生成的keystore文件命名为samlKeystore.jks，然后复制到Kylin的classpath中。

5）配置其他属性，如saml.metadata.entityBaseURL，saml.context.serverName，使用正确的机器名。

6）最后，设置kylin.security.prof？ile=saml且重启Kylin以使所有的saml配置生效。

启用SSO后，当用户初次在浏览器中访问Kylin的时候，Kylin会将用

户转向SSO提供的登录页面，用户在SSO登录页面验证成功后，页面将自动跳转回Kylin，这个时候Kylin会解析到SAML中的信息，获取验证后的用户名，再查询LDAP以获取用户群组信息，赋予用户权限，完成验证登录。

注意　对于API的调用，也就是请求URL为/kylin/api/*的请求，Kylin会继续使用LDAP完成验证，而不是重定向到SSO，这样做主要基于以下几点考虑。

·API的调用一般来自于应用程序或脚本，无法完成浏览器跳转等一系列操作。

·SSO服务器可能只支持用户账户的验证（如启用了2FA等更高级的方式），而不支持服务账户。

·SSO服务器只完成用户验证，而不提供用户的群组和角色信息。

所以在启用了SSO后Kylin依旧保留LDAP服务器的配置，以完成对所有用户的验证。

9.2 授权

用户的授权发生在登录验证之后。授权决定了此用户在系统中的角色和所能采取的动作。Apache Kylin中的授权是角色加访问控制（Access Control Level，ACL）的授权。9.1节提到Kylin有三种用户角色，下面是对这三种角色的详细解释。

·ROLE_ANALYST：分析师角色；具有该角色的用户，将能够查询与操作系统内具有相应ACL权限的Cube。

·ROLE_MODELER：建模人员角色；具有该角色的用户，将能够创建和修改数据模型及Cube，对自己创建的Cube可以管理其ACL；并操作其他具有相应ACL的Cube。

·ROLE_ADMIN：管理员角色；具有该角色的用户，将能够执行系统管理相关的操作，如导入Hive表、开启/关闭缓存、重载元数据等；并具有对所有项目和Cube的创建、修改、查询和删除的权力。

在当前版本（v1.5.2）中，当用户登录Kylin成功时，将默认具有ROLE_ANALYST和ROLE_MODELER角色（由“acl.defaultRole”配置）；Kylin视该用户是否属于相应的管理员群组（由“acl.adminRole”配置），来决定是否授予用户ROLE_ADMIN的角色。

Kylin提供了项目级别和Cube级别的访问控制(Access Control Level, ACL),以实现细粒度的授权。若用户创建了项目或Cube,那么该用户便是此实例的所有者,具有了对此实例的控制权限;其他用户如果想访问该实例,需要现有的所有者对其授权。

访问控制权限分为如下4种类型。

·QUERY:可以查询Cube的内容,但不能操作和修改,通常将此权限赋予像数据分析师之类的只需要查询Cube的人员或服务账号。

·OPERATION:可以对Cube进行操作,如构建、刷新等,但是不能修改Cube的定义;一般将此权限赋予需要构建Cube的人员或服务账号。

·EDIT:可以修改Cube的定义;一般将此权限赋予非创建者的其他建模人员等。

·ADMIN:对Cube有管理权限,包含了以上三种权限,还可以进行删除操作;一般将此权限赋予系统管理人员。

管理Cube级别的权限,选择并展开要管理的Cube,再单击“Access”标签页,会显示已有的ACL,如果要添加,则单击“Grant”按钮,如图9-2所示。

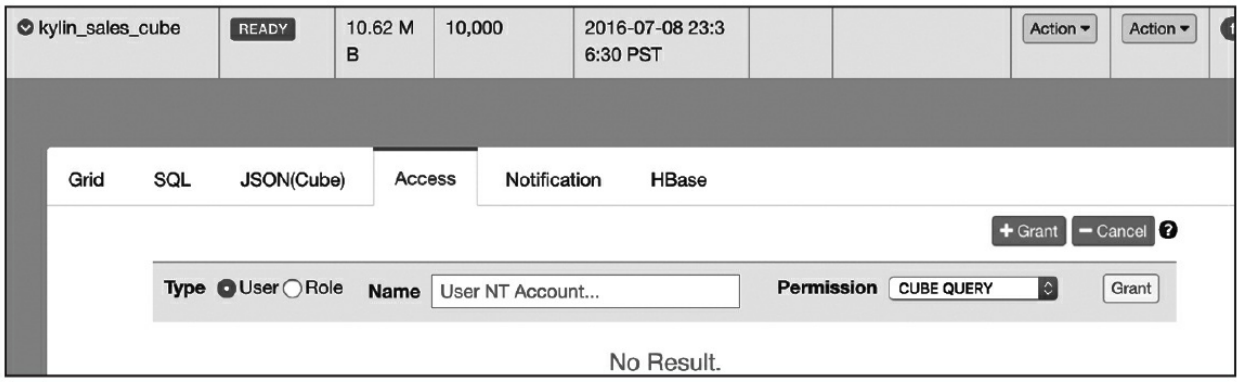


图9-2 管理Cube ACL

管理项目级别的权限，需要单击页面左上角的“Manage Project”，选择并展开想要管理的项目，再单击“Access”标签页，如图9-3所示。

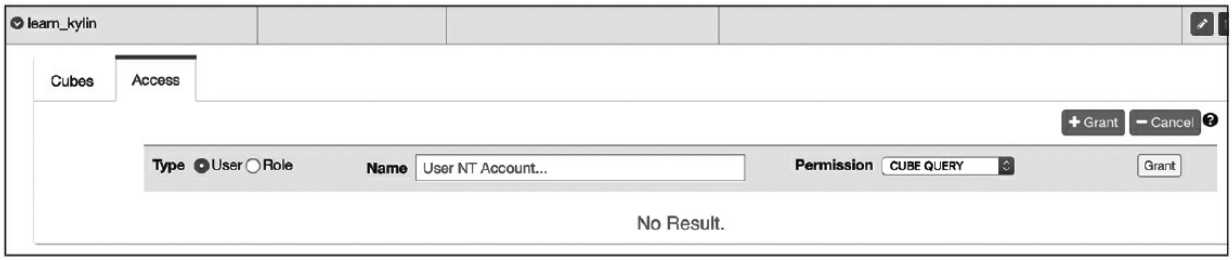


图9-3 管理项目ACL

ACL可以按用户来授予，或者按角色来批量授予。当选择“User”的时候，在“Name”输入框中输入用户名；当选择是“Role”类型的时候，从下拉框中选择一个角色；然后单击“Grant”按钮添加，如图9-4所示。

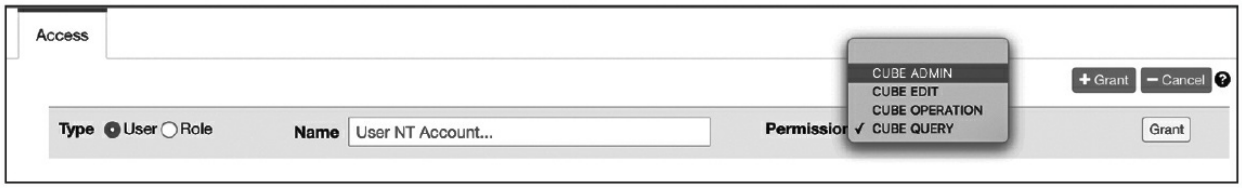


图9-4 授予ACL

授权后的ACL会显示在列表中，用户可以修改或收回（使用Revoke按钮），如图9-5所示。

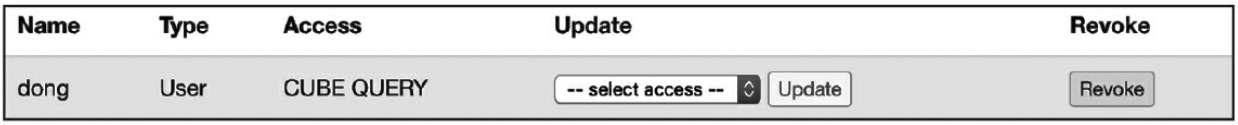


图9-5 修改或收回ACL

当用户执行某个操作的时候，如果发现其不具备某个权限，Kylin会报Access Denied的错误。

9.3 小结

本章介绍了Apache Kylin的企业级功能，主要是其安全验证和授权方面的功能。尤其是身份验证功能，基于Spring Security，具有非常好的可扩展性和灵活性。可以与所有主流的企业用户管理平台对接，实现单点登录（SSO）。除此之外，还有更多企业级功能有待开发，例如更强的按行列管理的安全性、企业级的监控能力，等等。Apache Kylin社区会在这些方面持续努力。

另外还有企业级运行和维护的相关内容，将在第10章详细介绍。

第10章　运维管理

第9章中介绍了Apache Kylin作为企业数据平台组件的一些重要功能。本章将首先介绍Apache Kylin的快速安装方法和基本配置，帮助读者在Hadoop环境中迅速启动Apache Kylin服务；然后着重介绍Apache Kylin在实际的企业环境中常用的分布式部署方案，以及如何达到高并发、高性能、高可用的目的；此外，本章还将介绍Apache Kylin运维管理中需要注意的事项和常用的工具，以及如何从Apache Kylin开源社区获取帮助，使Apache Kylin运维管理人员能够明确运维任务、快速定位故障并找到解决途径。

10.1 安装和配置

在Apache Kylin中，对Cube的构建和存储需要依赖于Hadoop和HBase集群，因此在部署Apache Kylin之前需要事先准备好所需要的软硬件环境。此外，为了使Apache Kylin更加充分地利用集群资源，还需要调整Apache Kylin的配置参数以实现功能适配和性能优化。本节将对这些内容做详细介绍。

10.1.1 必备条件

本节将介绍运行Apache Kylin所需要的必备条件，如果希望搭建一套新的测试或生产环境运行Apache Kylin，那么建议参考以下要讲到的必备条件。如果已有的生产环境不满足这些条件，如使用了API不兼容的Hadoop版本，那么可能会造成运行时的问题。

1.硬件需求

Apache Kylin的运行需要一定的硬件配置，如进行并发查询时需要在内存中进行大量的聚合操作，所以建议给Apache Kylin服务器保证一定的CPU和内存资源，这里给出的是能够顺利运行Apache Kylin所需的最小硬件配置。

·内存：8GB内存以上。

·CPU：核数在4核以上。

·硬盘：40GB存储空间以上。

2.Hadoop环境

Apache Kylin需要运行在Hadoop环境当中，作为Hadoop、Hive、HBase的客户端节点。根据Apache Kylin社区的文档来看，目前Apache

Kylin支持的这些Hadoop组件版本如表10-1所示。

表10-1　Kylin支持的Hadoop环境版本

组件依赖	版本
Hadoop	2.4-2.7
Hive	0.13-1.2.1
HBase	0.98-0.99, 1.x
JDK	1.7+

Apache Kylin的核心代码是使用Java语言编写并编译的，语言规范和API均是基于JDK1.7版本的，所以需要在运行Apache Kylin的服务器上提前预装与JDK1.7兼容的Java环境。

·说明　一般来说，Hadoop、HBase、Hive等组件的Java API在不同版本或商业分发中可能存在差异，而Apache Kylin是根据上述版本的Apache开源版本的标准API进行编译和测试的。因此，如果读者使用的是其他版本或商业发行版，可能会因为API不兼容导致运行时出错。

3.权限需求

为了能够在Hadoop环境中顺利运行Apache Kylin，用户必须在Apache Kylin服务器上安装相应的组件，并保证运行Apache Kylin的用户拥有相关权限。在构建Cube的时候需要向Hadoop提交MapReduce任务，并把中间结果保存在HDFS上，所以运行Apache Kylin的用户必须拥有提交MapReduce任务和读写HDFS的权限；建议把Apache Kylin部署在Hadoop集群的客户端节点上，可以使Apache Kylin获得更低的网络延迟

和更好的伸缩性。

为了从Hive中获取数据，Apache Kylin服务器上必须安装Hive客户端，运行Apache Kylin的用户也必须拥有运行Hive命令行（如hive或beeline）和管理、读写Hive表的权限。

Apache Kylin的存储依赖于HBase，因此Apache Kylin服务器上必须安装HBase客户端，运行Apache Kylin的用户则必须拥有运行HBase命令行（如hbase shell）和管理、读写HBase表（HTable）的权限。

4.第一次使用Apache Kylin

如果是第一次接触Apache Kylin，并希望在最短的时间内部署和试用Apache Kylin，那么可以使用Sandbox进行快速部署。多数Hadoop发行商（如Hortonworks、Cloudera、MapR等）都推出了All-In-One的虚拟机镜像文件，其中包含一个预先配置好的单节点的Hadoop/HBase环境，用户可以在VirtualBox等虚拟机软件中进行快速部署，以满足Apache Kylin运行的基本条件。

这里以Hortonworks Sandbox（HDP2.2.4）为例，用户可以在Hortonworks官网上下载Sandbox的镜像文件，并导入VirtualBox作为一个虚拟机。为保障Sandbox能够稳定运行，并顺利执行Cube构建任务，请给虚拟机的硬件分配足够的硬件资源。

·内存:8GB或以上。

·硬盘:40GB或以上。

·CPU:2核或以上。

虚拟机启动完毕后,再手动启动Ambari服务,方便在图形界面中管理集群服务。在虚拟机的命令行中输入以下命令就可启动Ambari了:

```
ambari-server start
ambari-agent start
```

当看到如下提示,就说明Ambari启动成功了:

```
[root@sandbox ~]# ambari-server start
Using python  /usr/bin/python2.6
Starting ambari-server
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start.....................
Ambari Server 'start' completed successfully.
[root@sandbox ~]# ambari-agent start
Verifying Python version compatibility...
Using python  /usr/bin/python2.6
Checking for previously running Ambari Agent...
Starting ambari-agent
Verifying ambari-agent process status...
Ambari Agent successfully started
Agent PID at: /var/run/ambari-agent/ambari-agent.pid
Agent out at: /var/log/ambari-agent/ambari-agent.out
Agent log at: /var/log/ambari-agent/ambari-agent.log
```

Ambari启动成功后，打开网页浏览器，登录Ambari的Web UI（地址是http://虚拟机IP:8080，登录账号:admin，登录密码:admin）。

默认情况下，HBase服务是关闭的，需要手动开启（如图10-1所示），开启步骤如下：

1）在左侧服务列表中选中HBase。

2）展开右侧“Service Actions”菜单，单击Start。

3）等待启动进度条完成，左侧服务列表中的HBase图标变成绿色。

图10-1　Ambari控制台页面

到这里，运行Apache Kylin所需的软硬件环境就准备好了。

10.1.2 快速启动Apache Kylin

准备好Apache Kylin的运行环境之后，就可以下载Apache Kylin的二进制包并进行安装部署了。Kylin既支持单点部署，也支持分布式部署以实现负载均衡。本节首先介绍单节点上的快速部署。

1.下载二进制包

Apache Kylin官网上提供了各个版本Apache Kylin的二进制包下载（http://kylin.apache.org/download/），目前最新的版本是1.5.2.1，该版本针对不同的Apache Hadoop/HBase版本提供了3种二进制包下载。

（1）Apache Kylin1.5.2.1二进制包for HBase0.98/0.99

该二进制包基于Apache HBase0.98的API进行编译，如果读者的Apache HBase版本是0.98或0.99，那么请下载这个二进制包。

（2）Apache Kylin1.5.2.1二进制包for HBase1.x

该二进制包基于Apache HBase1.1.3的API进行编译，如果读者的Apache HBase版本是1.x，那么请下载这个二进制包。

（3）Apache Kylin1.5.2.1二进制包for CDH5.7

因为CDH（Cloudera Hadoop Distribution）的Apache Hadoop环境对API做过一些调整，所以Apache Kylin社区专门为CDH5.7的Hadoop/HBase的API编译了二进制包，如果读者使用了CDH5.7的Apache Hadoop环境，那么请下载这个二进制包。

·说明　Apache软件基金会为每个Apache开源项目提供了一个归档下载站点，如果用户希望下载Apache Kylin的历史版本，可以登录该站点进行下载；用户也可以在Apache Kylin的下载页面找到该站点的入口。该站点的地址是：https://archive.apache.org/dist/kylin/ 。

2.验证签名

为了保证所下载的二进制包未经篡改，建议用户先对下载好的二进制包进行签名验证。用户可以在Apache Kylin归档下载站点下载与二进制包对应的gpg签名文件（如Apache Kylin1.5.2.1版本对应的签名文件是apache-kylin-1.5.2.1-bin.tar.gz.asc），然后使用gpg命令进行校验：

```
gpg --verify apache-kylin-1.5.2.1-bin.tar.gz.asc apache-kylin-1.5.2.1-bin.tar.gz
```

如果校验时看不到public key的报错，那么可以到Apache Kylin的github（https://github.com/apache/kylin/blob/master/KEYS）下载public keys文件并使用gpg命令导入：

```
gpg --import KEYS.txt
```

如果校验结果给出如下提示，则说明校验成功：

```
gpg: Signature made Sat Jun  4 19:48:47 2016 CST using RSA key ID XXXXXXXX
gpg: Good signature from "[NAME] <email@example.org>"
```

3.快速部署

在一个节点上安装Apache Kylin十分简单，把下载的二进制包解压到本地目录中即可（如/usr/local或/opt目录）。例如，在命令行中输入如下命令，把Apache Kylin解压到/usr/local目录中：

```
cd /usr/local
tar -zxvf apache-kylin-x.y.z-bin.tar.gz
```

解压完成之后，需要设置环境变量KYLIN_HOME到解压目录，如：

```
export $KYLIN_HOME=/usr/local/apache-kylin-x.y.z-bin
```

到此为止，Apache Kylin的安装过程就完成了。浏览Apache Kylin的安装目录，可以看到，安装目录中主要包含如下5个子目录。

·bin目录：包含了所有运行和维护Apache Kylin的可执行文件，如启动脚本、Streaming控制脚本、依赖查询脚本等。

·conf目录：包含了Apache Kylin所有的配置文件，如日志配置、服务配置、任务配置等。

·logs目录：该目录为Apache Kylin启动时自动创建的目录，是默认的

日志路径。包含Apache Kylin运行日志、标准输出、JVM垃圾回收日志等。

·tomcat目录：该目录是一个内嵌的tomcat二进制包，是整个Apache Kylin服务的载体，同时服务于Rest API和Web UI。

·sample目录：该目录包含了一个小数据集的样例数据，用于帮助用户快速体验和测试Kylin的功能。

4.启动Apache Kylin

在第一次启动Apache Kylin之前，可以先运行下面的命令检查Apache Kylin所需要的环境、权限是否就绪。如果该命令没有通过，那么读者需要根据相应的提示对环境进行调整。检查命令如下：

```
$KYLIN_HOME/bin/check-env.sh
```

对环境的检查通过之后，可以直接在命令行中输入如下命令启动Apache Kylin服务：

```
$KYLIN_HOME/bin/kylin.sh start
```

一般的情况下，当命令执行结束并有如下提示时，Apache Kylin服务就启动好了，读者可以在浏览器中开启Apache Kylin的Web UI了：

```
A new Kylin instance is started by root, stop it using "kylin.sh stop"
Please visit http://<ip>:7070/kylin
You can check the log at bin/../logs/kylin.log
```

成功启动Apache Kylin之后，如果读者想尽快体验一下，可以导入Apache Kylin官方提供的样例数据。在命令行中执行如下命令，即可导入一份小型数据集：

```
$KYLIN_HOME/bin/sample.sh
```

稍等几分钟，当看到如下提示时，Apache Kylin的样例数据就导入完毕了：

```
    Sample cube is created successfully in project 'learn_kylin'; Restart Kylin server or reload the metadata from web UI to see the change.
```

如果用户在Apache Kylin的Web UI上仍未看到导入的Cube，那么可能是因为缓存的原因，需要用户单击System页面的“Reload Metadata”按钮以重新加载元数据，或者直接重启Kylin服务。

如果需要停止Kylin服务，则可以在命令行中执行如下命令：

```
$KYLIN_HOME/bin/kylin.sh stop
```

如果KYLIN_HOME目录中的pid文件丢失，则可能导致上面的命令无法找到Apache Kylin进程。这时用户可以简单地通过Linux命令进行查找：

```
ps -ef | grep kylin
```

找到Kylin服务的进程号，并确认无误后，使用kill命令关闭该进程：

```
kill -KILL <KYLIN_PID>
```

10.1.3 配置Apache Kylin

把Apache Kylin安装到集群节点之后，往往还需要对Apache Kylin进行配置，一方面将Apache Kylin接入现有的Apache Hadoop、Apache HBase、Apache Hive环境，另一方面可以根据实际环境条件对Apache Kylin进行性能优化。

1.配置文件

在Apache Kylin安装目录的conf目录里，保存了对Apache Kylin进行配置的所有配置文件。读者可以修改配置文件中的参数，以达到环境适配、性能调优等目的。conf目录下默认存在以下配置文件。

（1）kylin.properties

该文件是Apache Kylin服务所用的全局配置文件，和Apache Kylin有关的配置项都在此文件中。具体配置项在下文会有详细讲解。

（2）kylin_hive_conf.xml

该文件包含了Apache Hive任务的配置项。在构建Cube的第一步通过Hive生成中间表时，会根据该文件的设置调整Hive的配置参数。

（3）kylin_job_conf_inmem.xml

该文件包含了MapReduce任务的配置项。当Cube构建算法是Fast Cubing时，会根据该文件的设置来调整构建任务中的MapReduce参数。

(4)kylin_job_conf.xml

该文件包含了MapReduce任务的配置项。当kylin_job_conf_inmem.xml不存在，或者Cube构建算法是Layer Cubing时，可用来调整构建任务中的MapReduce参数。

2.重要配置项

这些配置文件中，最重要就是kylin.properties了。本节将对一些最常用的配置项进行详细介绍。

(1)kylin.metadata.url

指定Apache Kylin元数据库的URL。默认为HBase中kylin_metadata表，用户可以手动修改表名以使用HBase中的其他表保存元数据。在同一个HBase集群上部署多个Apache Kylin服务时，可以为每个Apache Kylin服务配置一个元数据库URL，以实现多个Apache Kylin服务间的隔离。例如，Production实例设置该值为kylin_metadata_prod，Staging实例设置该值为kylin_metadata_staging，在Staging实例中的操作不会对Production环境产生影响。

(2)kylin.hdfs.working.dir

指定Apache Kylin服务所用的HDFS路径，默认在HDFS上/kylin的目录下，以元数据库URL中的HTable表名为子目录。例如，如果元数据库URL设置为kylin_metadata@hbase，那么该HDFS路径的默认值就是/kylin/kylin_metadata。请预先确保启动Kylin的用户有读写该目录的权限。

（3）kylin.server.mode

指定Apache Kylin服务的运行模式，值可以是“all”、“job”、“query”中的一个，默认是“all”。Job模式指该服务仅用于Cube任务调度，而不用于SQL查询。Query模式表示该服务仅用于SQL查询，而不用于Cube构建任务的调度。All模式指该服务同时用于任务调度和SQL查询。

（4）kylin.job.hive.database.for.intermediatetable

指定Hive中间表保存在哪个Hive数据库中，默认是default。如果执行Kylin的用户没有操作default数据库的权限，那么可以修改此参数以使用其他数据库。

（5）kylin.hbase.default.compression.codec

Apache Kylin创建的HTable所采用的压缩算法，配置文件中默认使用了snappy。如果实际环境不支持snappy压缩，则可以修改该参数以使用其他压缩算法，如lzo、gzip、lz4等，删除该配置项即不启动任何压缩算法。

(6)kylin.security.prof？ile

指定Apache Kylin服务启用的安全方案，可以是“ldap”、“saml”、“testing”。默认值是testing，即使用固定的测试账号进行登录。用户可以修改此参数以接入已有的企业级认证体系，如ldap、saml。具体设置可以参考其他章节。

(7)kylin.rest.timezone

指定Apache Kylin的Rest服务所使用的时区，默认是PST。用户可以根据具体应用的需要修改此参数。

(8)kylin.hive.client

指定Hive命令行类型，可以使用cli或beeline。默认是cli，即hive cli。如果实际系统只支持beeline作为Hive命令行，那么可以修改此配置为beeline。

(9)kylin.coprocessor.local.jar

指定Apache Kylin在HTable上部署的HBase协处理jar包，默认是$KYLIN_HOME/lib目录下以kylin-coprocessor开头的jar包。

(10)kylin.job.jar

指定构建Cube时提交MapReduce任务所用的jar包。默认是

$KYLIN_HOME/lib目录下以kylin-job开头的jar包。

（11）deploy.env

指定Apache Kylin部署的用途，可以是DEV、PROD、QA。默认是DEV，在DEV模式下一些开发者功能将被启用。

·说明　关于其他的配置文件，如kylin_hive_conf.xml和kylin_job_conf.xml，它们的配置项都是Hive和Hadoop规定的配置项，不属于Kylin的范畴，用户可以阅读关于Hadoop和Hive的官方文档或其他相关图书进行了解，在此将不再赘述。

10.1.4 企业部署

上文介绍了对Apache Kylin进行快速部署及进行基本配置的方法。但是，如果想在企业中部署Apache Kylin作为线上系统以支持多业务多用户的使用场景，则必须考虑对于高并发、高可用、高性能的支持。本节将介绍在企业集群中如何更好地部署Apache Kylin，以应对复杂的线上场景。

1.集群部署

Apache Kylin支持线性的水平伸缩，即通过集群部署的方式实现高并发和高可用。

图10-2所示的就是集群部署的架构图，多个Apache Kylin服务可以通过负载均衡将任务和查询进行分散，缓解单点带来的故障隐患和性能瓶颈，同时提高查询的并发能力。

部署方法具体如下。

1）添加更多的Apache Kylin节点，在每个Apache Kylin实例的配置文件kylin.properties中设置相同的元数据库URL，如kylin.metadata.url=kylin_metadata_cluster@hbase。

2）设置其中一个节点的kylin.server.mode为all（或者job），其余节点为

query。即一个节点做任务调度，其余节点做查询处理。

3）修改配置文件中的kylin.rest.servers参数，如：

```
kylin.rest.servers=host1:7070,host2:7070,host3:7070
```

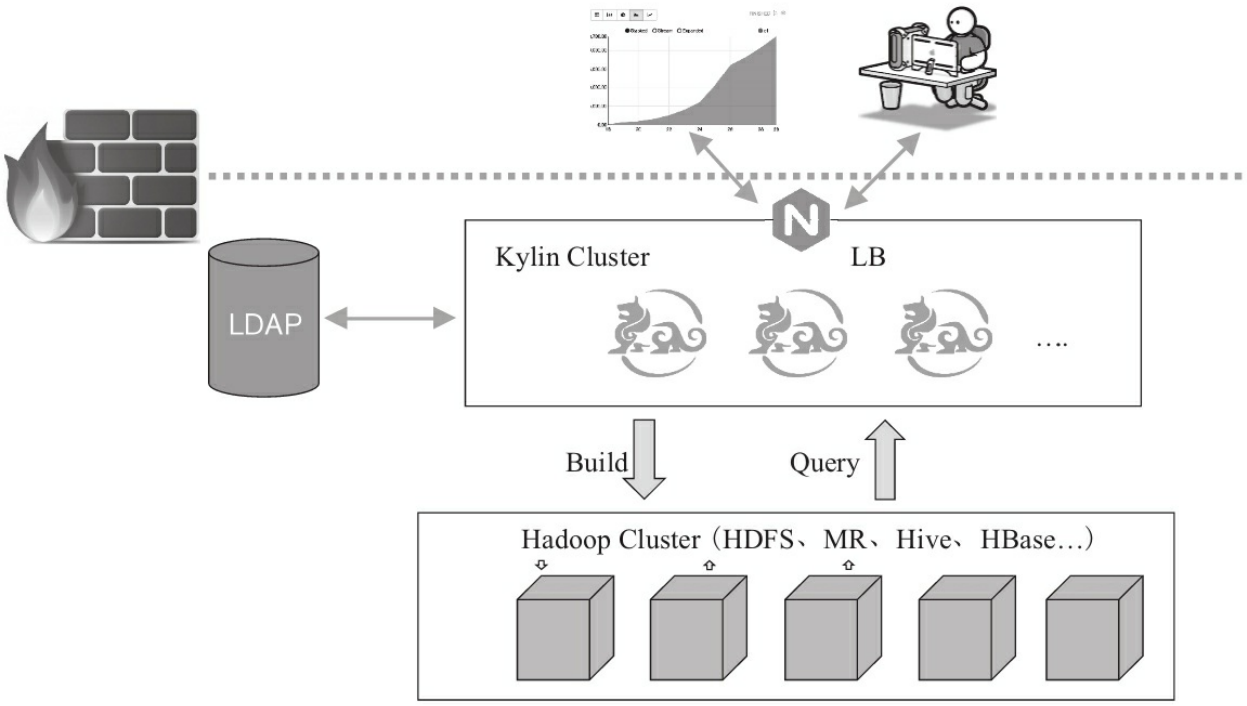


图10-2 Kylin集群部署架构图

4）部署一个负载均衡器（如Nginx），将请求分发至Apache Kylin集群。

2.读写分离

通常来说，一个Cube构建任务可能需要花费数十分钟的时间，并占用Hadoop集群资源。如果HBase和Hadoop共享同一集群，由于Region Server的硬件资源被MapReduce任务占用，在执行Kylin查询时性能会有

所降低。因此，可以在集群部署的基础上，将HBase和Hadoop集群隔离，实现Kylin的读写分离，最大程度地降低构建任务和查询的相互影响。

图10-3是读写分离部署的架构图，从图10-3中可以看出，HBase集群是一个独立于Hadoop的集群，拥有独占的HDFS。Cube构建仅发生在Hadoop集群，不会对HBase的查询产生影响；Cube文件会在构建成功后转移到HBase集群。在这种模式中，Apache Kylin节点需要靠近HBase集群，旨在查询时获取最好的网络性能。

以集群部署的方案为基础，读写分离的具体部署方法如下。

1）把Apache Kylin节点配置为Hadoop集群的客户端节点，并修改HBase客户端配置文件，把HBase服务器指向HBase集群。

2）修改Apache Kylin配置文件，设置kylin.hbase.cluster.fs=hdfs://hbase-cluster:8020。注意，这个值要和HBase Master节点上“root.dir”的Namenode地址保持一致。

3.Staging/Prod部署

一般来说，Apache Kylin用户创建出的Cube要经过多次优化和调试才能拥有较好的性能，调试的过程往往需要多次修改元数据，并执行重建Cube等的操作，这一方面需要操作者拥有较高的权限，另一方面也会密集占用集群资源。而线上的生产系统对稳定性和响应时间有SLA的需求，

往往不希望用户做频繁的更改。因此，可以采用Staging/Prod模式部署Kylin服务，将Staging和线上环境进行区分。

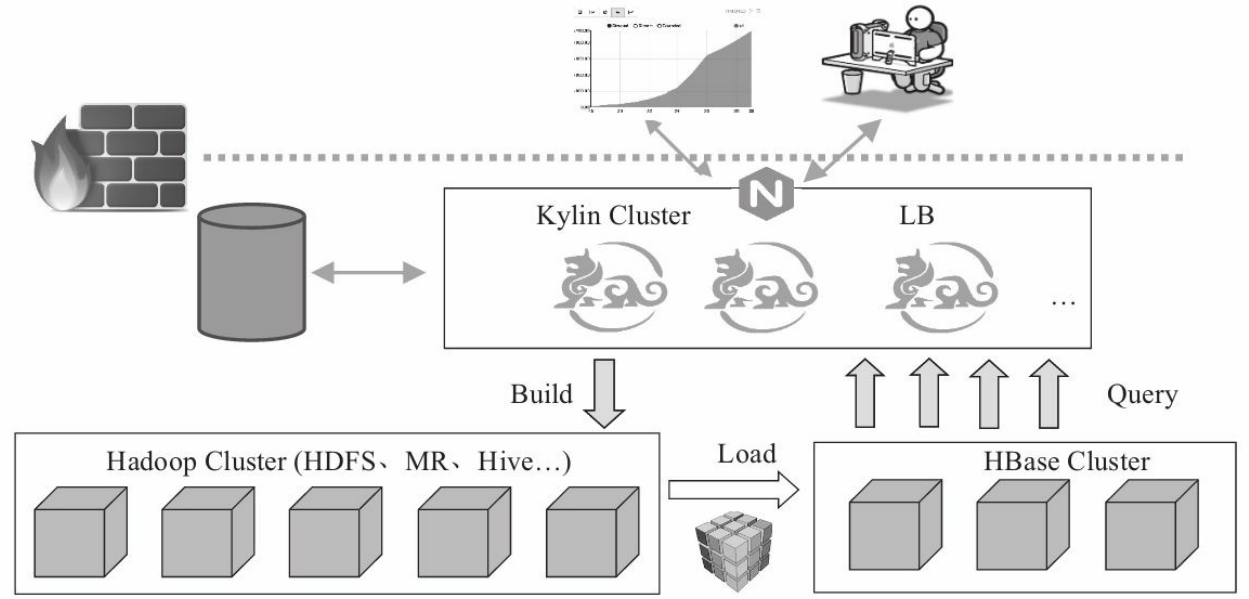


图10-3 Kylin读写分离部署架构图

(1)部署架构

图10-4是Staging/Prod模式的架构图。即在同样的Hadoop/HBase集群中部署两个Apache Kylin集群，一个作为Staging环境，另一个作为线上Prod环境。Staging给了用户较高的写权限，当新用户开始试用Apache Kylin时，先在Staging环境中创建、测试和调试Cube，当Cube调试完毕并被Apache Kylin管理员检验通过后，再由管理员把Cube发布到Prod环境中。相反的，如果一个Cube需要从Prod环境中退役，也可以将其从Prod环境迁移到Staging环境中留存一段时间。

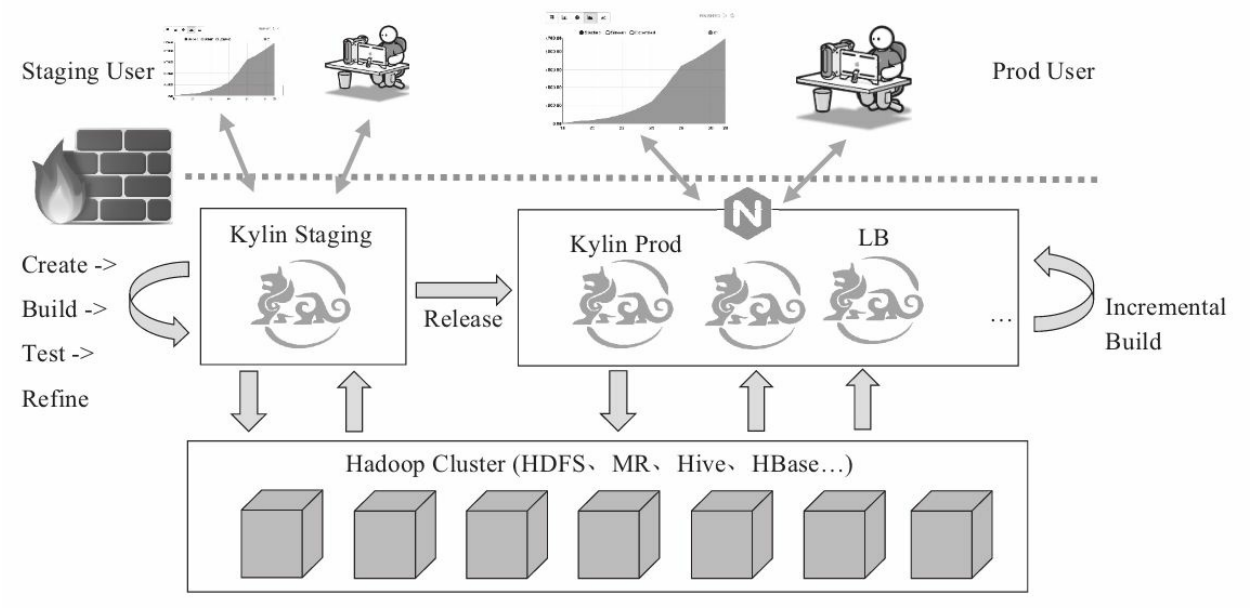


图10-4 Kylin Staging/Prod部署架构图

Prod环境的使用不同于Staging环境的元数据库，它只允许用户进行查询和构建。管理员发布Cube的过程只涉及元数据的转移，所以仅花费数分钟即可完成。此外，还可以为Staging和Prod环境配置不同的MapReduce任务队列以隔离计算资源。

(2)元数据迁移

Apache Kylin中提供了一个发布Cube的工具，可以方便地把Cube从Staging环境迁移到Prod环境中，命令如下：

```
$KYLIN_HOME/bin/org.apache.kylin.storage.hbase.util.CubeMigrationCLI srcKylinConfigUri
dstKylinConfigUri cubeName projectName copyAclOrNot purgeOrNot overwriteIfExists realExecute
```

其中共有8个参数，具体见表10-2。

表10-2 元数据迁移命令参数

参数	描述
srcKylinConfigUri	源 Kylin 的配置文件路径
dstKylinConfigUri	目的 Kylin 的配置文件路径
cubeName	迁移 Cube 的名称
projectName	目标项目的名称
copyAclOrNot	是否复制权限控制列表，true 或 false
purgeOrNot	是否把 Cube 数据清空，true 或 false
overwriteIfExists	如果目标系统中已存在，那么是否覆盖，true 或 false
realExecute	是否真正执行，true 或 false

用户需要根据自己的实际情况设置这些参数，第一次执行时可以将realExecute设置为false，以测试整个过程是否会出错。确认无误后，再将realExecute参数设置为true，然后真正执行迁移操作。由于Cube的迁移只涉及元数据的移动，因此不会占用过多的时间。

10.2 监控和诊断

对于Apache Kylin的运维人员来说，通过日志和报警对Apache Kylin进行监控、了解Apache Kylin整体的运行情况都是重要的工作职责之一。那么，如何收集和阅读Apache Kylin的日志呢？如何设置任务报警以便及时采取措施呢？如何开启系统仪表盘了解Apache Kylin的运行情况呢？本节将针对这些问题进行详细解答。

10.2.1 日志

当Apache Kylin顺利启动之后，默认会在Apache Kylin安装目录下产生一个logs目录，该目录将保存Apache Kylin运行过程中产生的所有日志文件。Apache Kylin的日志文件包含了很多信息，一些是运行时的环境信息，一些是任务流的过程参数，还有一些可能是警告和错误信息。其中，某些报错只是描述一个任务或Apache Kylin服务的暂时情况，并不意味Apache Kylin服务遇到了致命的问题。

Apache Kylin的日志主要包含以下3个文件。

（1）kylin.log

该文件是主要的日志文件，所有的logger默认写入该文件，其中与Apache Kylin相关的日志级别默认是DEBUG。日志随日期轮转，即每天0点时将前一天的日志存放到以日期为后缀的文件中（如kylin.log.2014-01-01），并把新一天的日志保存到全新的kylin.log文件中。

（2）kylin.out

该文件是标准输出的重定向文件，一些非Apache Kylin产生的标准输出（如Tomcat启动输出、Hive命令行输出等）将被重定向到该文件。

(3)kylin.gc

该文件是Apache Kylin的Java进程记录的GC日志。为避免多次启动覆盖旧文件，该日志使用了进程号作为文件名的后缀（如kylin.gc.9188）。

在一些较老版本的Apache Kylin中，该文件夹下还保存了另外两个日志文件。它们的内容是kylin.log的子集，只包含特定功能相关的日志。

(1)kylin_job.log

该文件保存了Cube构建相关的日志。默认级别是DEBUG。

(2)kylin_query.log

该文件保存了查询相关的日志。默认级别是DEBUG。

·说明　日志是快速了解Apache Kylin服务运行状况最直接的方式之一。当运维人员遇到故障问题或执行运维操作时，应当首先查看日志。例如，在需要重启Apache Kylin服务时，最好先查看日志，以确认暂时没有用户进行查询或正在构建Cube，在服务空闲的时候执行重启、升级等维护操作。

下面将以查询为例，简单介绍一下如何在日志中获取查询的更多信息。首先在Web UI中执行一个查询，然后马上到kylin.log文件尾部查找相关日志。当查询结束后，我们会看到如下的记录片段：

```
2016-06-10 10:03:03,800 INFO  [http-bio-7070-exec-10] service.QueryService:251 :
==========================[QUERY]===============================
SQL: select * from kylin_sales
User: ADMIN
Success: true
Duration: 2.831
Project: learn_kylin
Realization Names: [kylin_sales_cube]
Cuboid Ids: [99]
Total scan count: 9840
Result row count: 9840
Accept Partial: true
Is Partial Result: false
Hit Exception Cache: false
Storage cache used: false
Message: null
==========================[QUERY]===============================
```

表10-3是对上述片段中主要字段的介绍。

表10-3 日志中Query相关信息及介绍

字段	介绍
SQL	查询所执行的 SQL 语句
User	执行查询的用户名
Success	该查询是否成功
Duration	该查询所用时间（单位：秒）
Project	该查询所在项目
Realization Names	该查询所击中的 Cube 名称

因为篇幅原因本书不再做过多介绍，对日志感兴趣的读者可以根据日志记录中的类名和行号阅读相关源码。

显然，这些日志的路径可以通过修改配置文件进行调整。从最新的Apache Kylin代码中可以看出，新版本的Apache Kylin会把日志的配置文

件kylin-server-log4j.properties放置到conf目录下，如果用户有需求调整日志级别、修改日志路径等，都可以修改此文件中的内容，并重启Kylin服务。

当Apache Kylin出现故障后，第一要务就是查看日志。一般情况下，故障的发生大多数是由于程序中出现了异常（Exception），根据故障发生的时间查找日志中的异常或ERROR记录，能够快速定位问题出现的根本原因。有时候，故障的发生是由于系统中存在缺陷（Bug），用户可以在JIRA（详见10.5.2节）中提交Bug时附上相关的日志片段，以便于社区开发者快速定位和重现问题。

10.2.2 任务报警

在Apache Kylin中，构建一个Cube往往至少需要花费几十分钟的时间。因此，当一个Cube构建任务完成或失败时，运维人员常常希望可以在第一时间得到通知，以便进行下一步的增量构建或故障排查。于是，Apache Kylin中提供了一个邮件通知的功能，可以在Cube状态发生改变时，向Cube管理员发送电子邮件。

想要通过电子邮件实现任务报警，需要首先在配置文件kylin.properties中进行设置。

·将mail.enabled设置为true，即可启动邮件通知功能。

·将mail.host设置为邮件的SMTP服务器地址。

·将mail.username设置为邮件的SMTP登录用户名。

·将mail.password设置为邮件的SMTP登录密码。

·将mail.sender设置为邮件的发送邮箱地址。

设置完毕后，重新启动Apache Kylin服务，这些配置即可生效。

接下来，需要对Cube进行配置。首先设置Cube的邮件联系人，作为邮

件通知的收件人。如图10-5所示。

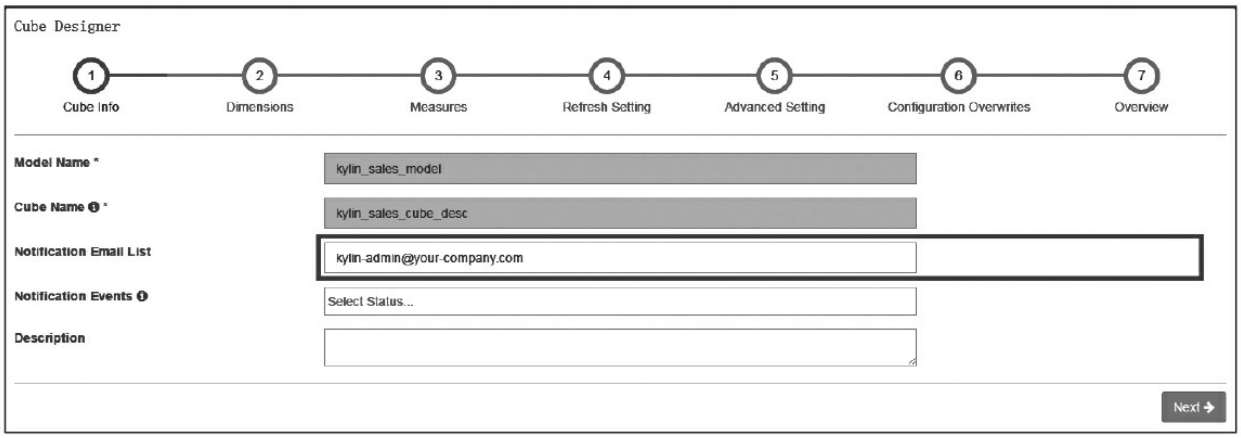


图10-5 邮件通知设置(1)

然后，选择一些状态作为邮件通知的触发条件。即当Cube构建任务切换到这些状态时，就给用户发送邮件通知(如图10-6所示)。

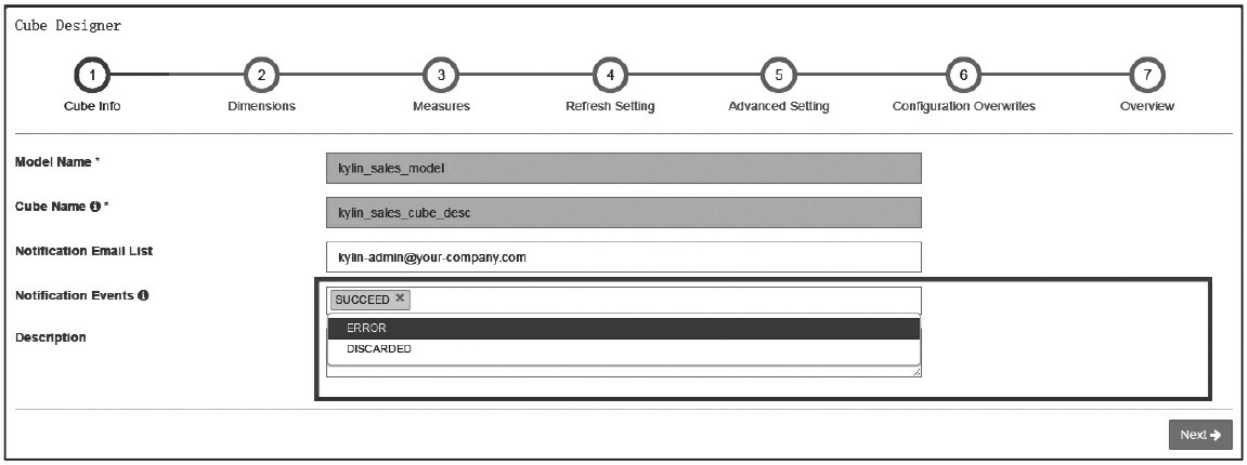


图10-6 邮件通知设置(2)

除了邮件通知，用户有时还会希望把Apache Kylin的监控接入到现有的一站式监控平台当中。为了实现这一目的，用户可以基于Apache Kylin

的Rest API进行二次开发，在第三方系统中实时掌握Apache Kylin的最新状态。关于Rest API的详细信息，请参考第5章。

10.2.3 诊断工具

在Kylin1.5.2之后的版本中，在Web UI上提供了一个“诊断”功能。该功能的入口总共有两处：项目诊断和任务诊断。

1.项目诊断

用户经常会遇到一些棘手的问题，例如Cube创建失败、SQL查询失败，或者SQL查询时间过长等。运维人员需要抓取相关信息并进行分析，以找出问题的根本所在。这时候，可以选择错误所在的项目，然后单击System页面下的Diagnosis按钮，系统会自动生成一个zip格式的压缩包，其包含了该项目下所有的有用信息，可以帮助运维人员缩小问题排查的范围，并为问题的快速定位提供了方向（如图10-7所示）。

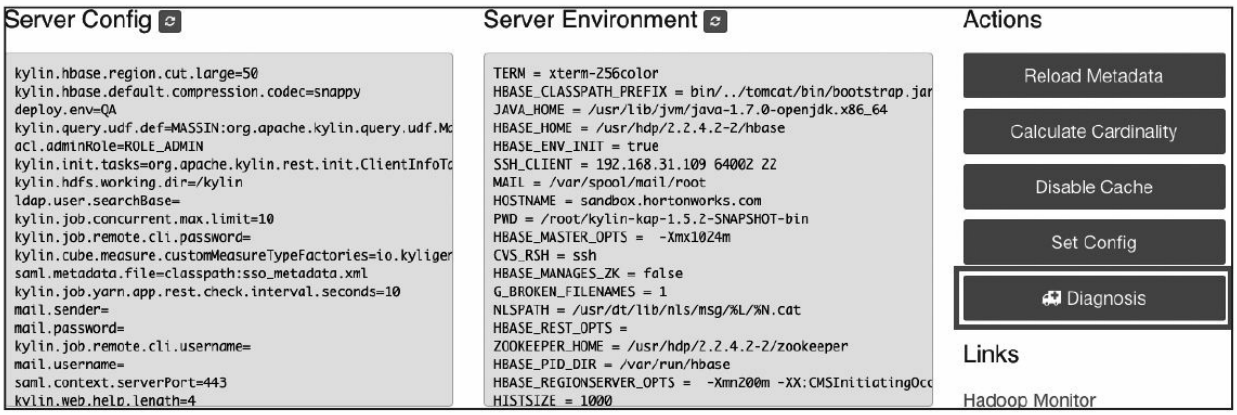


图10-7 项目诊断

2.任务诊断

若一个Cube构建任务执行失败或时间过长，运维人员还可以单击Job下的Diagnosis菜单项（如图10-8所示），以生成一个专门针对该任务的zip压缩包，帮助运维人员快速分析任务的失败原因。

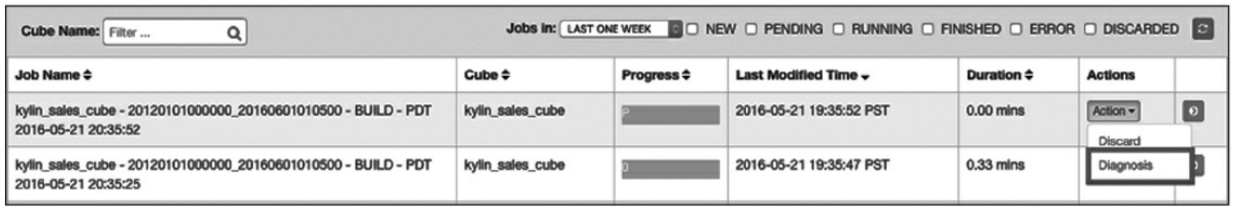


图10-8 任务诊断

10.3 日常维护

Apache Kylin服务器每天都会接受不同用户提交的多个Cube构建任务，有时会因为Cube设计不当或集群环境异常等原因，导致Cube构建失败或时间过长，这时就需要运维人员提高警惕；此外，Kylin服务运行一段时间之后，一些存在于HBase或HDFS上的数据会成为无用数据，需要定时对存储器进行垃圾清理。本节将介绍这些日常维护中常见的问题和相应的对策。

10.3.1 基本运维

作为Apache Kylin的运维人员，工作中需要做到以下几点。

·确保Apache Kylin服务运行正常。

·确保Apache Kylin对集群资源的利用正常。

·确保Cube构建任务正常。

·实现灾难备份和恢复。

……

常言道“工欲善其事，必先利其器”，运维人员需要熟练地运用本章提到的各种工具，对Apache Kylin服务的每日运行状况进行监控，此外，也方便在遇到问题时找到合理的解决途径。

为了保障Apache Kylin服务的正常运行，运维人员可以对Apache Kylin的日志进行监控，确保Apache Kylin进程的稳定运行。为了确保Apache Kylin对集群资源的正常利用，运维人员需要经常查看MapReduce任务队列的闲忙程度，以及HBase存储的利用率（如HTable数量、Region数量等）。为了确保Cube构建任务的正常，请根据邮件通知或Web UI的Monitor页面对任务进行监控。

10.3.2 元数据备份

元数据是Apache Kylin中最重要的数据之一，备份元数据是运维工作中一个至关重要的环节。只有这样，在由于误操作导致整个Apache Kylin服务或某个Cube异常时，才能将Apache Kylin快速从备份中恢复出来。

一般情况下，在每次进行故障恢复或系统升级之前，对元数据进行备份是一个良好的习惯，这可以保证Apache Kylin服务在系统更新失败后依然有回滚的可能，在最坏的情况下依然保持系统的鲁棒性。

Apache Kylin提供了一个工具用于备份Kylin的元数据，具体用法是：

```
$KYLIN_HOME/bin/metastore.sh backup
```

当看到如下提示时即为备份成功：

```
metadata store backed up to /usr/local/kylin/meta_backups/meta_2016_06_10_20_24_50
```

上面的示例命令会把Apache Kylin用到的所有元数据以文件的形式下载到本地目录当中（如/usr/local/kylin/meta_backups/meta_2016_06_10_20_24_50）。目录结构如表10-4所示。

表10-4 元数据备份目录介绍

目录名	介绍
project	包含了项目的基本信息，项目所包含其他元数据类型的声明
model_desc	包含了描述数据模型的基本信息、结构的定义
cube_desc	包含了描述 Cube 模型的基本信息、结构的定义
cube	包含了 Cube 实例的基本信息，以及下属 Cube Segment 的信息
cube_statistics	包含了 Cube 实例的统计信息
table	包含了表的基本信息，如 Hive 信息
table_exd	包含了表的扩展信息，如维度
table_snapshot	包含了 Lookup 表的镜像
dict	包含了使用字典列的字典
execute	包含了 Cube 构建任务的步骤信息
execute_output	包含了 Cube 构建任务的步骤输出

此外，元数据备份也是故障查找的一个工具，当系统出现故障导致前端频繁报错时，通过该工具下载元数据并查看文件，往往能对确定元数据是否存在问题提供很大帮助。

10.3.3 元数据恢复

有了元数据备份，用户往往还需要从备份中恢复元数据。和备份类似，Apache Kylin中提供了一个元数据恢复的工具，具体用法是：

```
$KYLIN_HOME/bin/metastore.sh restore /path/to/backup
```

如果从10.3.2节元数据备份的例子中进行恢复，需要在命令行中执行：

```
cd $KYLIN_HOME
bin/metastore.sh restore meta_backups/meta_2016_06_10_20_24_50
```

恢复成功后，用户可以在Web UI上单击“Reload Metadata”按钮对元数据缓存进行刷新，即可看到最新的元数据。

此外，Apache Kylin中的metastore.sh脚本还有很多功能，如remove、fetch、cat等，用户可以通过执行如下命令来查看其具体的使用方法：

```
$KYLIN_HOME/bin/metastore.sh
```

10.3.4 系统升级

从Apache Kylin项目开源并加入Apache软件基金会至今，已进行过多次版本发布，每次发布都会引入很多新的功能特性，修复旧版本的功能缺陷，或者重构代码结构。这就可能导致一个问题：元数据或Cube数据在不同版本间不兼容。对于已经部署了老版本的Apache Kylin用户来说，为了利用到新版本所添加的功能，必须对现有的Apache Kylin实例进行升级。本节将介绍如何从一些使用较为广泛的老版本进行升级。

1.从1.5.1升级到1.5.2

Apache Kylin1.5.2与Apache Kylin1.5.1版本的元数据、Cube数据都是兼容的，不需要对数据进行升级。但是因为HBase协处理的通信协议发生了改动，而已有的HTable仍然绑定了老版本的HBase协处理器Jar包，因此必须为现有的HTable重新部署HBase协处理器Jar包。在命令行中执行如下命令即可为所有的HTable重新部署HBase协处理器jar包：

```
    $KYLIN_HOME/bin/kylin.sh org.apache.kylin.storage.hbase.util.DeployCoprocessorCLI
$KYLIN_HOME/lib/kylin-coprocessor-*.jar all
```

除了HBase协处理器，在Apache Kylin1.5.2版本中还对kylin.properties配置文件进行了更新，升级人员在覆盖配置文件的时候需要多加注意，因为有三个在Apache Kylin1.5.1及之前版本中存在的HBase Region划分相

关的选项被弃用了，它们分别是：

·kylin.hbase.region.cut.small=5

·kylin.hbase.region.cut.medium=10

·kylin.hbase.region.cut.large=50

在Apache Kylin1.5.2的新版本中，如果需要对不同的Cube设置不同的HBase Region划分尺度，可以在Web UI上为单个Cube重写kylin.hbase.region.cut和kylin.hbase.hf？ile.size.gb这两个配置，如图10-9所示。

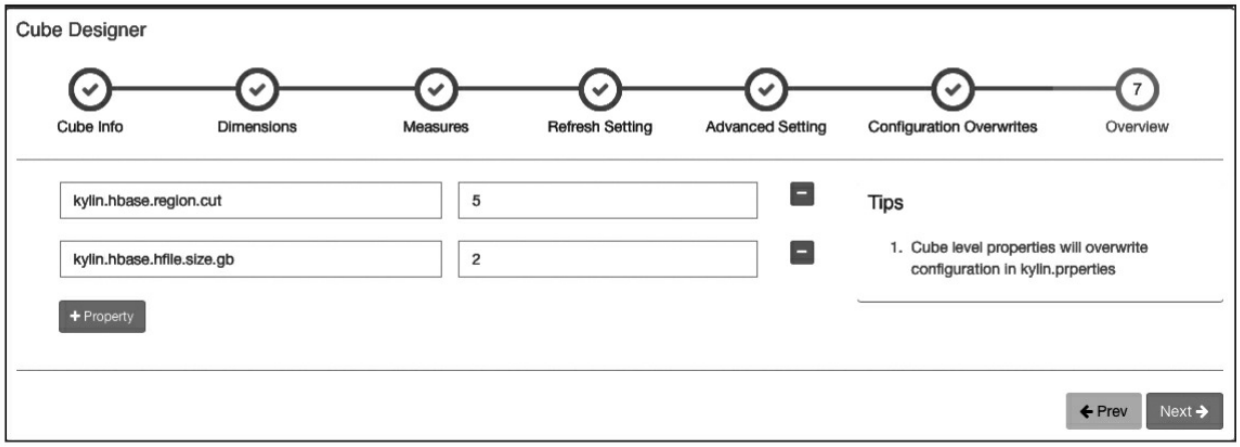


图10-9 为Cube配置region切分方式

此外，在Apache Kylin1.5.2版本中，Apache Kylin为采用Fast Cubing算法进行构建的Cube安排了一个单独的配置文件，用于配置Map Reduce任务的参数，配置文件保存在conf/kylin_job_conf_inmem.xml中。同时，下列

在Apache Kylin1.5.1及之前版本中存在的配置文件也被弃用：

·kylin_job_conf_small.xml

·kylin_job_conf_medium.xml

·kylin_job_conf_large.xml

2.从Apache Kylin1.5.0及之前版本升级到Apache Kylin1.5.1

Apache Kylin1.5.1和Apache Kylin1.5.0及之前版本的元数据格式并不兼容，用户需要对元数据进行升级。在升级后，原有的已构建好的Cube仍然可以继续进行查询。以下是升级的具体过程。

1）停止正在运行的老版本Apache Kylin实例。

2）备份元数据。根据上文介绍的元数据备份方法，在执行升级操作前对元数据进行备份，以便于在升级失败后进行回滚。

3）备份配置文件。将老版本Apache Kylin安装目录下的conf目录备份至其他目录。

4）安装Apache Kylin1.5.1。前往Apache Kylin官网下载1.5.1版本的二进制包，并解压到一个新的目录中，并把上一步备份好的conf目录复制到该安装目录。需要特别注意的是，新版本可能引入了一些新的配置参数，用户需要对新老版本的配置文件进行手动比较和合并。

5）执行升级。把新的安装目录设置为KYLIN_HOME，然后将第一步的元数据备份复制到该目录。接下来在命令行中执行如下命令，对复制进来的元数据文件进行升级。注意，请不要直接在第2步的元数据备份目录下执行升级，因为升级命令会直接修改元数据文件，这样会因为丧失老版本的元数据文件而丧失回滚的能力。

元数据升级代码如下：

```
export KYLIN_HOME="<path_of_1_5_1_installation>"
$KYLIN_HOME/bin/kylin.sh  org.apache.kylin.cube.upgrade.entry.CubeMetadataUpgradeEntry_
v_1_5_1 < COPIED_METADATA_DIR>
```

6）上传元数据。利用上文介绍的元数据恢复工具，将升级后的元数据恢复到Apache Kylin实例的元数据仓库中。如果这新老版本实例的元数据仓库使用了同一个元数据库（见kylin.metadata.url属性），则需要先对元数据仓库进行重置。可以参考如下命令：

```
$KYLIN_HOME/bin/metastore.sh reset
$KYLIN_HOME/bin/metastore.sh restore <UPGRADED_METADARA_DIR>
```

7）重新部署HBase协处理器Jar包。与Apache Kylin1.5.1升级到Apache Kylin1.5.2版本一样，此处也需要对HBase协处理器进行重新部署，新版本的HBase协处理器Jar包放置在安装目录的lib目录中。具体部署方法可以参考前文，在此不再赘述。

8）启动新的Apache Kylin实例。

3.升级失败后回滚

如果升级任务失败，导致生产系统长时间无法运行，往往会对业务系统产生严重的影响。因此，为了保证生产系统的正常运行，必须立即在升级失败后进行回滚，将系统恢复到升级之前的状态。以下是回滚的具体操作步骤。

1）停止新Apache Kylin实例。

2）恢复元数据。正如上文所述，升级前必须对元数据进行备份。实现这一步需要从升级前备份的元数据目录中对元数据仓库进行恢复。如果新老Apache Kylin实例的元数据仓库使用了同一个HTable，则需要先对元数据进行重置。请参考以下命令：

```
export KYLIN_HOME="<path_of_old_installation>"
$KYLIN_HOME/bin/metastore.sh reset
$KYLIN_HOME/bin/metastore.sh restore <path_of_BACKUP_FOLDER>
```

3）重新部署HBase协处理器。因为升级过程中给HTable部署了新版本的HBase协处理器Jar包，所以回滚时需要把老版本的HBase协处理Jar包部署到HTable中。老版本的HBase协处理器Jar包放置于Apache Kylin安装目录的lib目录中。具体部署方法可以参考前文，在此不再赘述。

4）启动老版本的Apache Kylin实例。

10.3.5 垃圾清理

如前文所述，在Apache Kylin运行一段时间之后，会有很多数据因为不再使用而变成了垃圾数据，这些数据占据着大量的HDFS、HBase等资源，当积累到一定规模时它们会对集群性能产生影响。

这些垃圾数据主要包括：

·Purge之后原Cube的数据。

·Cube合并之后原Cube Segment的数据。

·任务中未被正常清理的临时文件。

·很久之前Cube构建的日志和任务历史。

为了对这些垃圾数据进行清理，Apache Kylin提供了两个常用的工具。请特别注意，数据一经删除将彻底无法恢复！建议使用前先进行元数据备份，并对目标资源进行谨慎核对。

1.清理元数据

第一个是元数据清理工具，该工具有一个delete参数，默认是false。只有当delete参数为true时，工具才会真正对无效的元数据进行删除。该工

具的执行方式如下：

```
$KYLIN_HOMEbin/metastore.sh cleanup [--delete true]
```

第一次执行该工具时建议省去delete参数，这样一来，就只会列出所有可以被清理的资源供用户核对，而并不实际进行删除操作。当用户确认无误后，再添加delete参数并执行命令，才会进行实际的删除操作。

默认情况下，该工具会清理的资源列表如下：

·2天前创建的已经无效的Lookup表镜像、字典、Cube统计信息。

·30天前结束的Cube构建任务的步骤信息、步骤输出。

2.清理存储器

第二个工具是存储器清理工具。顾名思义，就是对HBase和HDFS上的资源进行清理。同样的，该工具也有一个delete参数，默认是false。当且仅当delete参数的值是true时，工具才会对存储器中的资源真正执行删除操作。该工具的执行方式如下：

```
$KYLIN_HOME/bin/kylin.sh storage cleanup [--delete true]
```

第一次执行该工具时建议省去delete参数，这样只会列出所有可以被清理的资源供用户核对，而并不实际进行删除操作。当用户确认无误后，再添加delete参数并执行命令，才会进行实际的删除操作。

默认情况下，该工具会清理的资源列表如下：

·创建时间在2天前，且已无效的HTable。

·在Cube构建时创建的但未被正常清理的的Hive中间表、HDFS临时文件。

10.4 常见问题和修复

在运维过程中，有些问题是经常会遇到的。我们收集了社区中出现频率较高的一些问题，并在此给出常见的解决方法。

问题1：升级Apache Kylin版本后所有查询报错Timeout visiting cube！

这个错误的原因是HBase协处理器中出现错误，建议用户查看日志，并详细查看该报错周边是否存在其他异常。常见的原因是由于新老版本的协处理器通信协议不兼容，解决方法是重新部署HBase协处理器。

Apache Kylin中提供了一个命令用于对某些HTable的协处理器进行动态更新：

```
$KYLIN_HOME/bin/kylin.sh org.apache.kylin.storage.hbase.util.DeployCoprocessorCLI
$KYLIN_HOME/lib/kylin-coprocessor-*.jar [all|-cube <cube name>|-table <htable name>]
```

该命令最后的参数用于指定HTable的范围。如果是all，那么该Apache Kylin服务中所有的HTable都将被更新；如果是-cube<cube name>，则被指定的Cube下所有的HTable都将被更新；如果是-table<htable name>，则仅仅更新被指定名称的HTable。

问题2：构建Cube的MapReduce总是无法启动，原因是yarn无法分配足够的内存。

在构建Cube时有两种算法：Fast Cubing和Layer Cubing，而Fast Cubing对内存的要求较高。因此在Apache Kylin配置文件kylin_job_conf.xml和kylin_job_conf_inmem.xml中对Mapper的内存定义了默认值，见参数mapreduce.map.memory.mb和mapreduce.map.java.opts。这些值超过了yarn中定义的container内存限制。建议对yarn配置和Kylin配置文件中Mapper内存配置进行核对，然后做响应的调整。

问题3：在Fast Cubing构建Cube时，Mapper抛出OutOfMemoryException异常。

造成这种问题的原因，一方面有可能是Mapper的内存设置过少，可以调整kylin_job_conf.xml（Apache Kylin1.5.2版本之前）或kylin_job_conf_inmem.xml（Apache Kylin1.5.2版本之后）中对Mapper内存的设置，参数是mapreduce.map.memory.mb和mapreduce.map.java.opts。另一方面，有可能是一个Mapper处理了过多的数据，可以缩小kylin_hive_conf.xml中的dfs.block.size参数，使Hive输出更多小文件，从而增大Mapper数量，并减小单个Mapper处理的数据量。

问题4：在构建Cube时报错说Cube Signature不一致。

Cube Signature是对Cube结构一致性的保护措施，造成Signature不一致的原因主要有两个：一是手动修改了JSON格式的元数据信息并覆盖了metastore，在这种情况下请再次核对修改是否有必要，如果确有必要，建

议Purge这个Cube后在Web UI上修改元数据结构；二是Apache Kylin升级后由于版本不兼容导致的，这种情况出现的概率较小，建议先到社区求助。对于了解Cube Signature的高级用户，可以使用下面这个工具强制刷新所有的Cube Signature：

```
$KYLIN_HOME/bin/kylin.sh org.apache.kylin.cube.cli.CubeSignatureRefresher [cube names]
```

在这个命令中，Cube名字参数可选，若为空则更新所有的Cube。这个工具会重新计算并覆盖Cube的Signature，如果手动修改了就绪状态的Cube元信息，并执行此工具则可能导致Cube的不同Segment间的结构不一致，请慎用！

10.5 获得社区帮助

目前，Apache Kylin是Apache软件基金会的顶级项目，拥有相当活跃的开源社区。因此，用户有任何问题都可以向Apache Kylin社区进行讨论。Apache Kylin社区推荐的两种交流方式是邮件列表和JIRA。

10.5.1 邮件列表

邮件列表是Apache Kylin社区进行技术讨论和用户交流最活跃的渠道。当用户遇到任何技术问题，首先可以到Apache Kylin邮件列表的归档中进行历史检索，以查看先前是否存在相关的讨论。如果找不到有用的信息，用户可以在订阅Apache Kylin的邮件列表之后，用个人邮箱向Apache Kylin邮件列表发送邮件，所有订阅了邮件列表的用户都会看到此邮件，并回复邮件以发表自己的见解。

Apache Kylin主要有3个邮件列表，分别是dev、user、issues。dev列表主要讨论Kylin的开发及新版本发布，user列表主要讨论用户使用过程中遇到的问题，issues主要用于追踪Kylin项目管理工具（JIRA）的更新动态。以dev为例，用户必须先订阅才能收发邮件列表中的邮件，订阅的方法具体如下。

1）发送邮件到dev-subscribe@kylin.apache.org。

2）收到确认邮件后，按照邮件提示，给指定邮箱发送确认信息。

3）订阅成功。

正因为Apache Kylin社区是开源社区，因此所有用户和Committer都是志愿进行贡献的，所有的讨论和求助是没有SLA（Service Level

Agreement)的。为了提高讨论效率、规范提问，建议用户在撰写邮件时详细描述问题的出错情况、重现过程、安装版本等，并且最好能提供相关的出错日志。

10.5.2 JIRA

JIRA是Apache项目用于项目管理和缺陷跟踪的系统。当用户遇到问题，并怀疑是Apache Kylin的缺陷所导致的时，或者有给Apache Kylin添加新特性的想法时，都可以登录Apache Kylin的JIRA系统提交Ticket。同样的，为了提高沟通效率，建议在JIRA Ticket的描述中详细叙述遇到的问题和期望的解决方案。

Apache Kylin JIRA的地址是：https://issues.apache.org/jira/browse/KYLIN。用户首次登录需要用电子邮箱注册一个账号，接下来就可以创建Ticket或对现有的Ticket进行评论了。

10.6 小结

本章汇总了Apache Kylin从安装到配置，从监控到维护的各方面内容。需要注意的是Apache Kylin本质上是一个Hadoop应用程序，它的稳定和健康在很大程度上依赖Hadoop集群服务的稳定和健康。在实际使用中，很多常见的Apache Kylin问题追溯源头都会归因于Hadoop集群的异常。因此要用好Apache Kylin，有一个强健的Hadoop集群和运维团队至关重要。

第11章　参与开源

Apache Kylin（截稿时）是唯一一个由中国团队贡献到Apache软件基金会（ASF）的顶级开源项目，在这之前，曾经也有过几个项目加入Apache孵化器项目，但最终都失败了。而Apache Kylin在短短的11个月之内，从无到有；从最初的几个人发展到几百人的活跃社区；从初加入的懵懂到毕业时候的成熟；从最初基金会及导师们的都不太看好，到最终毕业时给予极高的褒奖；Kylin社区证明了中国开发者也可以毫无障碍地参与到国际顶级社区，引导大数据技术发展。Apache孵化器副总裁Ted Dunning在Apache Kylin毕业成为顶级项目的官方新闻中评价道："Apache Kylin在技术方面当然是振奋人心的，但同样令人兴奋的是Kylin代表了亚洲国家，特别是中国，在开源社区中越来越高的参与度"。

作为Apache软件基金会项目的开拓者，我们也非常希望能够将经验分享给每一位有志于贡献到开源社区的朋友，来一起壮大开源世界的力量，并进一步扩大来自中国的开发者的影响力。

11.1 Apache Kylin的开源历程

2013年年中，Kylin项目在eBay内部启动。

2014年10月，Kylin平台在eBay内部正式上线，并正式在github.com上开源，短短几天即获得业界专家的认可，他们也鼓励eBay的Kylin核心团队加入Apache软件基金会，与Hadoop、Spark、Kafka等知名大数据项目一起构建大数据生态社区。

2014年11月，正式加入Apache孵化器项目，并改名为Apache Kylin。

2015年11月，在经过严格的社区讨论和投票之后，Apache Kylin正式毕业成为ASF顶级项目，为首个，也是目前唯一的来自中国的Apache顶级项目（TLP）。

2016年3月，Kyligence公司成立，成为推动社区发展的新动力。

Apache Kylin从刚开始开源时不到10个人的社区，逐步发展到了几百人的全球活跃社区，并赢得了众多使用者，在eBay、美团、百度、网易、京东、微软、中国移动等著名公司被作为核心大数据分析系统。此外，还发展了非常多的贡献者（Contributor）、核心代码提交者（Committer）及项目管理委员会成员（PMC Member），其中还包括来自京东、美团、网易等的工程师，社区正在一天天逐步壮大。

11.2 为什么参与开源

作为个人，参与开源已经成为越来越多朋友的共识。在今天，特别是大数据领域，绝大部分技术都是开源的，包括Hadoop、Spark、Kafka、Kylin等，个人在开源社区的成长和能力提升显而易见。在有兴趣的领域内，不断参与和贡献，赢得社区的认可，甚至进一步成为Committer或PMC Member，这对个人的技术能力，对职业的发展等都会带来极大的价值。

作为公司，参与开源也已经成为一个趋势，几乎各个互联网公司都在通过开源项目来彰显自己的技术实力，构建技术品牌。好的开源项目可以极大地提升公司的形象，特别是在工程师群体中的认知度。这里典型的案例是LinkedIn，从Kakfa开始，LinkedIn不断开源了很多优秀的项目，特别是Kafka，很好地推动了整个行业的发展，几乎成为了事实标准。LinkedIn也由此收获了无数的赞誉。

11.3 Apache开源社区简介

11.3.1 简介

Apache软件基金会(Apache Software Foundation, ASF)[1]，是依据美国国内税法(Internal Revenue Code, IRC)的501(c)3条款在1999年建立的非营利性组织，具有如下特点。

·为开放、协作的软件开发项目提供基础，包括硬件、交流及业务基础架构等。

·作为一个独立的法律实体以使得公司及个人能够捐献资源并保证这些资源为公益所使用。

·为个人志愿者提供针对基金会项目法律诉讼的庇护方式。

·保护Apache品牌及其软件产品，避免被其他组织滥用。

在其所支持的Apache项目与子项目中，所发行的软件产品都遵循Apache许可证(Apache License)协议，目前协议版本为2.0，即常说的Apache License2.0[2]。

Apache软件基金会正式创建于1999年，在这之前，为了维护Apache

HTTP服务器，在Apache HTTP服务器的作者Rob MaCool不再维护这个项目时，由一群这个服务器软件的爱好者和使用者自发成立了一个兴趣小组，通过邮件列表的方式进行交流和软件维护。这些开发者、使用者和爱好者逐渐将这个小组命名为“Apache组织”。这个命名来自北美当地的一支印第安部落，该部落以超高的军事素养和超人的耐力著称，19世纪后半期对入侵者进行了反抗。为了对这支印第安部落表示敬仰之意，取其部落名称（Apache）作为服务器名，并逐渐演化成了今天的Apache软件基金会组织。直到今天，ASF社区依然保持了使用邮件列表作为主要沟通方式和参与项目方式的传统。

自1999年成立以来，这个全志愿者基金会见证了超过350个领先的开源项目，包括Apache HTTP服务器——全球最流行的Web服务器软件。通过被称为“The Apache Way”的ASF精英管理过程，超过550位个人成员和4700位代码提交者成功合作建立了可免费使用的企业级软件，使全球数以百万级的用户受益：数千软件解决方案在Apache许可证下发布；社区积极参与ASF邮件列表，指导项目，并举办该基金会的正式用户会议、培训和ApacheCon。ASF是美国501（c）（3）慈善组织，由个人捐赠和包括Bloomberg、Budget Direct、Cerner、Citrix、Cloudera Comcast、Facebook、Google、Hortonworks、HP、Huawei、IBM、InMotion Hosting、iSigma、LeaseWeb、Matt Mullenweg、Microsoft、PhoenixNAP、Pivotal、Private、Internet Access、Produban、Red Hat、Serenata Flowers、WANdisco和Yahoo等企业赞助商提供的资金以维持运营。

[1] 参考https://zh.wikipedia.org/wiki/Apache软件基金会。

[2] 参考https://www.apache.org/licenses/LICENSE-2.0。

11.3.2 组织构成与运作模式

Apache软件基金会是由Apache会员（ASF Member）组成的志愿者组织并运营和管理所有项目，由不同的实体来治理。

董事会（Board of Directors，Board）：基金会下设董事会来管理和监督ASF的日程事务、商务合作、捐赠等事宜，确保整个社区按照章程正常运作，董事会由会员（ASF Member）组成。董事会每月会举行一次会议，审议相关顶级项目的报告，各个副总裁，包括基础架构、品牌、媒体、法律等报告，审议支出等信息，所有信息会员都有权利查看并质疑。并可以进一步发起讨论以要求相关项目的进一步解释和改正。

项目管理委员会（Project Management Committee，PMC）：由董事会依据决议建立，管理一个或多个社区，其由项目贡献者组成（注：每一个会员，即ASF Member，都是贡献者），每个PMC至少都有一个ASF的官员，其为主席，也可能会有一个或多个ASF会员。PMC主席由董事会任命，并作为ASF的官员（副总裁）。主席向董事会负主要责任，并有权建立规则和管理社区的日常工作流程。ASF章程定义了PMC及主席职位。从基金会的角度，PMC的角色是监管，主要职责不是代码及编程，而是确保所有的法律问题被解决，流程被遵守，以及由整个社区来完成每一次发布。其次的职责是整个社区的长远发展和健康。ASF的角色是分配给你个人，而不是

你现在的工作或是雇佣者或是公司。尽管如此，PMC也保持着很高的标准，特别是主席，是董事会的眼睛和耳朵，董事会信任并依靠主席来提供法律监管。董事会有依据决议随时终止PMC的权利。

官员（一般称为副总裁）：由董事会任命，在特定领域内设置基金会级别的规则，包括法律、品牌、募捐等，各官员由董事会选举任命。

11.3.3 项目角色

每一个独立的Apache项目社区的精英管理体制通常是由不同的角色组成的[1]。

用户(User):用户指使用该项目的人,他们对Apache项目的贡献是向开发者提交反馈,包括Bug报告、特性建议等。用户以在邮件列表及用户支持论坛中帮助其他用户来参与Apache社区。

开发者(Developer):开发者为特定的项目贡献代码或文档。他们会更为积极地在开发者邮件列表中参与讨论,提交Patch、撰写文档、提供建议、评论等。开发者也被称为贡献者(Contributors)。

提交者(Committer):提交者是被赋予代码库写权限及书面签署了CLA(贡献者许可协议)的开发者。他们拥有apache.org邮箱地址。他们可以自行决定项目的短期决议,不需要别人允许而提交Patch。项目管理委员会(PMC)可以同意和批准(批准了的称为最终决议),或者拒绝。请注意是由项目管理委员会作出项目相关的决议,而不是单个的提交者。

项目管理委员会成员(PMC Member):项目管理委员会成员是被社区选举的开发者或提交者以贡献到项目的演进及示范相应的承诺。他们拥有代码库写权限、apache.org邮箱、社区相关决定的投票权及提名活跃用

户成为贡献者的权利。项目管理委员会作为一个整体控制整个项目。更具体的，项目管理委员会必须为任何一个他们项目的软件产品的正式的发布进行投票。

项目管理委员会主席（PMC Chair）：项目管理委员会主席是由董事会从项目管理委员会成员中任命的。项目管理委员会作为整体控制和领导整个项目。主席是项目和董事会之间的接口，主席有特定的责任。

Apache软件基金会会员（ASF Member）：Apache软件基金会会员是由现有会员提名并被选举的个人以贡献到基金会的演进和发展中。会员关心ASF本身。这通常通过顶级项目及跨项目相关的活动来体现。法律上，一个会员是一个基金会的“股东”，主人之一。他们拥有选举董事会，作为董事会选取的候选人及提议新成员的权利。他们同时也拥有建议新的项目成为孵化器项目的权利。会员通过邮件列表及年会来协调他们的活动。

[1] 参考ASF官方文档：How it works, https://www.apache.org/foundation/how-it-works.html#roles。

11.3.4 孵化项目及顶级项目

每一个希望加入Apache软件基金会的项目，都需要先提交并获得投票通过才能成为孵化器项目（Apache Incubation Project），在一定的时间内满足社区的标准，并通过投票才能成为顶级项目。孵化的流程可用图11-1来描述。

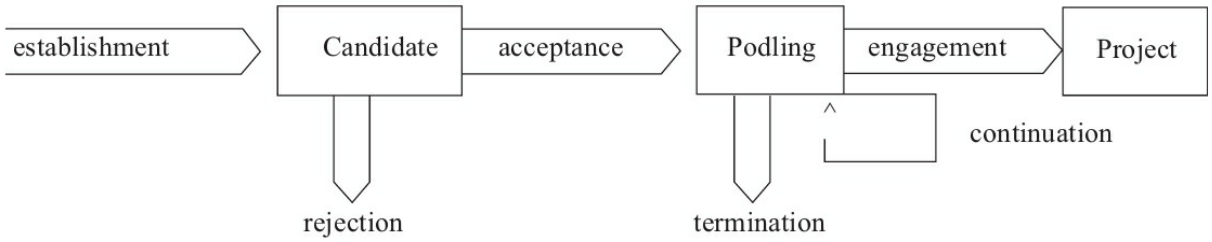


图11-1 Apache项目孵化流程

孵化流程大致分为以下几个阶段。

创立：一个候选的项目需要确定一个Champion的导师来帮助该项目，之后撰写相关的项目建议书，并提交到Apache孵化项目管理委员会（Incubator PMC，IPMC）。需要遵守社区的规范，并积极听取社区的意见和反馈。之后IPMC会进行相关的投票以决定是否接受该候选项目加入孵化器。

接受：是否接受一个项目是由投票来决定的，通常投票的形式依赖于具体的问题，如果你的项目中有项目管理委员会的成员将会带来很大的

帮助，特别是能够更好地得到来自社区的真实发生的反馈。一般提交到孵化器的候选项目只要总数有三票以上的＋1投票（即同意票），即使有-1投票（即不同意，每一个-1需要一个+1来抵消，因此最后需要计算所有剩下的＋1票），便可获得成为孵化器项目的资格，但一般要建议相关项目负责人等需要就相关的-1采取进一步的解释或行动以符合社区的相关规范等。

一旦被投票通过则该候选项目即被接受成为孵化器项目，以特有的名词——Podling——来称呼，以区别于Apache顶级项目（Top Level Project）。每一个Podling都需要导师，包括初始的Champion等。

评审：随着项目的的发展和逐渐稳定（基础设施、社区、决策等），不可避免地将迎来孵化项目管理委员会的评估以决定项目如何从孵化器中退出，请注意这里的退出可以为好的结果也可以为不好的结果。积极的退出是毕业成为顶级项目，或者子项目；另外一方面，项目的终止也是退出的一种方式。

终止或退休：结束孵化有两种较为负面的终结方式，即终止或退休。如果你的项目接收到终止指令，则意味着你的项目存在很大的问题。例如法律问题，或者IPMC在项目上分歧很大而不能在合理的时间内解决。你可以向董事会或支持者上诉终止决定，但你必须意识到一些董事会成员也是IPMC成员，上诉一般是不太会成功。退休一般是来自项目管理委员会内部的决定，因多种理由而由该项目委员会或IPMC来决定关闭该项

目。退休后不再由Apache软件基金会来开发和承担相关的责任。这并不意味着不可以使用相关的代码，只是不再拥有社区。

毕业：每一个Podling都期望能够毕业成为Apache顶级项目（Top Level Project，TLP）。这将由两次或多次投票来完成。先由该Podling的社区在合适的时候（一般可以征询导师的意见）发起毕业投票，需要满足特定的条件，包括活跃度、多样性、定期发布、公开决定，等等，此时有效的投票来自PMC member及导师。社区投票通过后再撰写相应的项目建议书至孵化项目管理委员会（IPMC）再进行投票，此时该Podling将被更加仔细和严格地被审阅，有效的投票来自IPMC member、ASF member等。需要特别指出的是，毕业投票必须是最终一致投票，意味着即使只有一票不同意该Podling也不能毕业，必须修复相关的问题，使得社区最终一致同意才能毕业成为顶级项目。

11.4 如何贡献到开源社区

11.4.1 什么是贡献

参与开源社区，贡献到开源社区，是很多工程师的梦想，如果自己的Patch能够被著名的项目所接受，那将是一件非常令人兴奋的事情。但很多国人开发者有一个误区，认为只有提交代码才算是贡献者。这里必须澄清的是，开源社区，特别是Apache社区，非常欢迎代码之外的贡献，包括文档、测试、报Bug、修Bug、宣传、文章、博客、线下活动等，都非常欢迎。提交代码只是诸多贡献中的一种。

11.4.2 如何贡献

以Apache Kylin为例，社区从最初的不到十个人壮大到今天几百人的规模，其中贡献者就非常多（贡献者不能按照Github上的“Contributor”计算，有很多贡献并不会体现在代码的提交中，在Apache社区的定义中，只要帮助到了项目的发展，从代码、文档、宣传到宣讲等都是贡献）。有的贡献者在最初只是问问题，报告各种环境下碰到的问题和Bug等，甚至帮忙修改了一些拼写错误等。由于开源项目所具有的特点，社区不可能完全设想到所有的应用场景、部署模式及使用方式，因此很多时候都需要每个使用者、爱好者尽量地将他们碰到的问题报告给社区，这样核心开发团队及整个社区才可以进一步来分析和找出解决方案，整个项目就能往前发展了。当然，如果能够提出自己的解法及Patch，则会有更高的认知度。如果对一些特定的场景、应用需求等提出了自己的功能需求并实现之，则会影响和贡献到整个项目的进一步发展。举例来说，在初期，Apache Kylin仅支持将HBase和Hadoop部署在一个集群，但美团公司的技术人员在业务需求的驱动下，开发了一个读写分离的新功能特性，支持了写入不同集群HBase的能力，并最终贡献到了社区，成为目前Kylin非常重要的一个特性。而越来越多的贡献正在从不同的公司和团队中提出，这些可以从社区的邮件列表和JIRA中看到。

11.5 礼仪与文化

在鼓励读者参与和贡献到开源中的时候，也必须要说明一下，社区很多时候都是虚拟的，不是面对面的交流，更多的时候是邮件、JIRA或Github Issues等方式的交流。这里，由于东西方文化的区别，有很多工程师吃了亏。因此特别需要大家注意以下几个方面。

尊重社区和贡献者。很多工程师在使用开源软件的时候，特别是碰到问题，在进行提问、报告Bug等的时候表现出一些非常不成熟的行为。例如经常碰到有人有了问题，在社区提问，经常急哄哄地希望即刻被解决，而很多时候因为解答不及时或没有响应的时候就采用各种刷屏等过激方式，甚至对社区进行指责。但是要知道开源项目作者将项目贡献出来，已经是最大的帮助了，整个开源社区的运营依靠的是每个人的志愿行为，类似的行为不但不会带来解答，很多时候只会带来负面效果，特别是在西方人为主的社区中。另一种行为是不做深入研究，甚至不会搜索，碰到非常基础的问题不管三七二十一就提问，这种问题很多时候简单地Google一下就可以找到解决方案，特别是在刚使用一个开源项目的时候。希望大家能够尊重社区和贡献者，这样整个社区也才会尊重每一个人，接纳更多来自中国的贡献者。

注意用词与语气。作为非英语母语的工程师，在西方社区中进行交流确实没有其他母语是英语国家的工程师便利。在社区中与人交流的时

候一定要多注意一些这方面的差异。不管是提问、解答问题，还是讨论功能等的时候，多多组织一下语言，现在各种工具非常丰富，足以帮助大家无障碍地进行沟通。我们在和ASF基金会董事、其他顶级项目负责人交流的时候经常被反应来自中国的工程师不太注意这方面的问题，有些时候甚至还很粗鲁，例如使用命令的口气和方式等，这可能会导致难以与社区的其他成员建立信任。对于我们来说，英语是一个挑战，但更多的时候是一个机会，参与国际社区是学英语的最好机会！

Speak Loudly。另外一个有趣的现象是中国工程师贡献了很多，特别是补丁、代码甚至特性等，但却很少有中国工程师成为Committer和PMC member。这在很大程度上制约了我们自身的发展，很难去影响项目的演进。一方面是由于使用英语带来了一些挑战，另外则是咱们国人比较含蓄，不太看重自己的贡献，或者将自己的贡献说出来，这方面也需要大家更多的努力。

11.6 如何参与Apache Kylin

参与Apache Kylin社区，首先要做的是订阅相关的邮件列表。

·开发者邮件列表：dev@kylin.apache.org

·使用者邮件列表：user@kylin.apache.org

使用你常用的邮件地址，发送一封邮件（内容为空即可）到dev-subscribe@kylin.apache.org或user-subscribe@kylin.apache.org。之后你会收到一封询问邮件，单击回复该邮件即可确认你的订阅。之后你会收到一封确认邮件，后续相关邮件列表里的讨论就会被接收下来。由于社区讨论非常频繁，建议设立相关的邮件规则来过滤和归档。

在使用Apache Kylin过程中碰到任何问题，都可以向相关的邮件列表发送邮件，特别提醒一下，提供更多的信息、日志等内容，将有助于志愿者及时地分析和解答相关的问题。也可以在Apache Kylin的JIRA系统中提交相关的Bug等信息。

最后，如果对Apache Kylin的开发感兴趣，可以下载源代码来进行进一步的研究，特别是在实际生产环境中碰到的一些问题的解决方案和想法等都可以提交到社区，贡献更多的场景以丰富和完善整个Apache Kylin项目和社区。

11.7 小结

本章介绍了Apache Kylin的开源历程和Apache基金会的组织架构和工作方式。希望通过这些内容让更多国人了解到开源的文化和魅力，激励更多人投入到开源软件的事业中来。

第12章　Apache Kylin的未来

大数据上的多维分析是非常活跃的领域。作为目前处于领先地位的开源项目，Apache Kylin在未来有着广阔的发展空间和无限的可能。

12.1 大规模流式构建

实时或近实时的数据分析是一类刚性需求。现有的Kylin流式Cube构建在很大程度上满足了这类需求，但也存在着如下的一些缺陷。

·架构上存在单点故障缺陷。流式计算在一个单一节点上完成。一旦出现，故障就需要人工干预，无法自动修复。

·配置较为复杂，负责流式构建的计算节点需要人工配制，使用不便。

·吞吐数据量限制。受到单一节点限制，单位时间内能处理的数据有理论限制。

·可能丢失迟到的数据。因为当前流式构建是按时间切分数据的，当数据源不能保证记录按时间严格排序的时候，迟到的数据将无法被Cube捕捉。

基于上面的原因，社区正在积极研发第二代流式构建技术。新的设计将利用Hadoop/Spark集群来扩展计算能力，解决单点失效隐患，可以横向扩展提升吞吐量，支持大规模流式处理。同时使用数据源偏移量（Offset）来记录数据的位置，确保不遗漏任何记录。

更进一步，可以在Cube的基础上再添加实时节点，将最后几分钟的

数据缓存在内存中。将实时节点联合Cube的内容就能构成实时查询的Lambda架构，通过一个SQL接口同时查询历史和实时数据，真正做到秒级别延迟的实时大数据分析。

12.2 拥抱Spark技术栈

Spark是继MapReduce之后的新一代分布式计算架构。尽管目前的成熟度和普及度还不如MapReduce，但因为其出色的性能和友好的开发接口广受各界欢迎。有人甚至将Spark誉为MapReduce终结者，称其最终将取代MapReduce。

Apache Kylin的可扩展架构可以很好地适应不同的计算技术。核心开发团队曾尝试基于Spark开发一款新的Kylin构建引擎。尽管初步实验结果并没有取得非常明显的速度提升，但随着Spark技术群的持续升温，对Spark新引擎的呼声将会越来越高。新的构建引擎可以和老引擎并存，所以并不存在二选一的问题。用户可以很安全地在同一个Kylin环境里同时启用两个引擎，选择一些项目试用新引擎，满意之后再慢慢切换。

支持SparkSQL作为Hive之外的另一种数据源是另一种角度的Spark集成。随着越来越多的数据格式被连接到Spark平台上，SparkSQL大有取代Hive成为新一代大数据仓库的态势。让Kylin从SparkSQL获取数据构建Cube的需求也就呼之欲出。目前社区已经将SparkSQL数据接入作为中期目标，提上开发日程。

12.3 更快的存储和查询

更快的查询和更高效的存储是永恒的主题。Kylin作为高速OLAP引擎，速度永远是最重要的技术指标。

当前的存储引擎HBase有着比较明显的速度短板（因为其要同时兼顾读写和弱类型），其他更快的存储技术，比如Kudu或Parquet，甚至有人建议Cassandra或ElasticSearch，只读性能都成倍高于HBase。因此在HBase之外支持其他更快速的存储引擎也是一个重要的发展方向。回报可能是查询速度成倍的提升。

12.4 前端展现及与BI工具的整合

有了高速OLAP引擎之后，用户很自然就会希望有相应的前端可视化。目前这里是Kylin的空白。通过JDBC/ODBC接口，Kylin可以和不少开源OLAP前端集成，比如Saiku、Caravel和Zeppelin，但使用起来需要额外的安装和配置，不是最为便捷的方式。

为多个流行的开源展现及BI工具提供原生的连接器，无缝地接入现有系统已成为用户的迫切需求，通过不断提高各个工具的整合能力，Apache Kylin将会为分析师提供更为友好的用户体验。

12.5 高级OLAP函数

Apache Kylin支持标准SQL，但对于高级OLAP函数还缺乏完整的支持，特别是在一些不太常用的函数如财物分析类函数等领域尚未完全实现。在Kylin被越来越多地试用到更多应用场景中的时候，这些函数将成为需要支持的重要特性。在社区中，我们提供了按需开发的能力，在有相关需求的时候会进一步增强这方面的开发。

另外，SQL2003等更新的标准中，对OLAP函数等也做了相应的规范，为了能够更好地支持丰富的分析需求，这类函数也将被逐步实现。

12.6 展望

今天，Apache Kylin提供的OLAP on Hadoop技术已经可以解决超大规模数据集上的亚秒级交互查询分析需求，在未来，用户将不仅仅希望能在Kylin上进行查询，包括对数据的增加、删除、修改等也希望能通过同一个平台来实现，即完整的数据仓库能力。对于社区不断发展的未来，我们相信Kylin完全会朝着完整的数据仓库解决方案的方向发展。