# 论架构风格在图书馆数据中台的实践

## 一、摘要

2019年11月，本人所在信息技术部承接了馆内数据中台的建设，该项目为期半年。该项目于 2020 年 5 月正式上线， 2020 年 6 月通过最终验收。通过数据中台，实现了我馆数据分层与水平解耦，沉淀了公共基础数据。其中通过数据建模实现跨域数据整合和知识沉淀，通过数据服务实现数据的封装和开放，通过数据开发工具满足个性化数据需求，为全市文旅数据融合工作提供了有力的数据支撑。该项目时间紧任务重，涉及人员组织多，涉及馆内 10 余个部门，外部配合协作 10 多个厂商团队。本人在此项目中担任系统架构设计师，主要完成技术方案评估与实现，项目立项论证等工作。本文以馆内大数据基础平台为例，讨论软件架构风格在该项目中的具体应用，包括设计层次的划分、各个层次的实现以及在实际场景中遇到的问题。

## 二、正文

近年来，随着我馆服务质量的不断提升，大量软硬件系统在我馆得到部署。硬件设备包括自助借还机、交互机器人、电子书借阅机、人脸识别门禁、人流量计数器，软件系统包括图书管理系统、微信小程序、移动端APP。软硬件系统的增加带来了大量用户数据的流入，如读者借阅数据、读者信息、阅读偏好、入馆行为等海量数据，但是这些数据缺乏统一的接入和分析。因此，需要通过数据技术，对海量数据进行采集、计算、存储、加工，同时统一标准和口径，并在此基础之上，对数据进行归纳分析，对外提供统一的、科学的、精准的馆情数据服务。笔者带领团队于2019年11月承接馆内数据中台的建设，并担任系统架构设计师。主要负责项目整体技术方案评估实现、立项论证以及项目管理工作。该项目的架构工作于同年12月完成，整个项目耗时6个月，于2020年5月上线测试。于2020年6月完成验收。

在实际开发阶段，笔者意识到架构风格是描述系统组织方式的惯用模式。可以为项目提供架构级的解决方案。这种架构级的软件重用可以极大提高系统建设进程。软件系统开发中常用的软件架构风格有数据流风格，调用/返回风格，独立构件风格，虚拟机风格，仓库风格。数据流风格包括批处理序列与管道-过滤器，其每一步处理都是独立，顺序执行的，适用于简单的线性流程。调用/返回风格包括主程序/子程序风格，面向对象风格，层次结构风格，主要通过分治思想进一步降低系统耦合度。独立构件风格包括进程通信，事件驱动风格（隐式调用），为软件重用提供了支持。虚拟机风格包括解释器风格，基于规则的系统风格，其具有良好灵活性，可自定义规则。仓库风格以数据为中心，包括数据库系统风格，黑板系统风格和超文本系统风格。

针对数据中台的应用场景，为尽可能达到软件复用与业务水平扩展的目的，笔者采用层次化的架构风格来进行系统架构。与一般的业务系统不同，数据中台本质上是面向数据的，笔者根据自己的实践经验，从数据处理的维度上对数据中台进行划分，将其分成数据接入层、数据模型层、数据服务层，数据开发层，各个层次只与相邻的两层发生交互。在不同的层次上，用户对数据利用的维度也从基础维度到跨域维度甚至是自定义维度，层层递进。下面笔者就从这四个层次入手，分析各个层次的具体作用。

在数据接入方面。在实际接入过程中，笔者除了考虑常规的WEB类应用比如APP、小程序、Web等，还要众多硬件设备的数据接入。这些硬件设备由于某种原因往往只支持基于TCP的自定义应用协议。由于工期进度等等原因，在实际项目推进中，让对方再提供一套HTTP的协议进行接入往往不太现实，因此再数据接入支持方面，协议也不局限于HTTP，也有基于TCP 的图书馆自有协议SIP2和厂家自定义协议。为此在实现过程中，笔者将接入服务集群分为两大类,一类是常规的HTTP服务集群,另一类是走TCP自定义协议的服务集群。这两个集群由同一个NGINX服务器做负载均衡与反向代理，用端口加以区分。由于NGINX1.9以上才支持四层的反向代理，因此在实际部署中特地使用第三方模块重新编译NGINX 来对反代TCP协议进行支持。

在数据模型方面。数据模型主要是在数据采集的基础上进行规范、整合和深度挖掘。因此笔者也将其概括为三层：分别是基础模型、融合模型和挖掘模型。其中基础模型一般是关系建模，主要实现数据的标准化，这也叫作“书同文、车同轨”。通过设计好的数据标准将接入的数据进行入库管理。其次是融合模型。融合模型一般是维度建模，主要实现跨域数据的整合，整合的形式可以是汇总、关联，也包括解析。这些融合模型表是通过各个业务部门梳理而得到的常用的宽表，比如读者终端使用行为表、读者阅读偏好表等，目前这类已经梳理出来的模型已有20余个。最后是挖掘模型。常规理解，挖掘模型这类偏向机器学习得到的数据模型往往其是偏应用的，不应该存在于基础模型层。但是在本馆的馆情分析中，例如读者阅读偏好模型这类常用的挖掘模型（基于朴素贝叶斯模型），由于使用频次和需求的关系，笔者也把它规整到中台模型。在数据模型这个层次上的性能优化主要是梳理常用的数据查询方法，对于需要筛选和排序的**字**段加上适当的索引，如唯一性索引、非唯一性索引和联合索引。但是需要注意，虽然索引大大提高了查询速度，但同时却会降低更新表的速度，如insert、update和delete。因为更新表时，不仅要更新数据，还要更新索引文件。另外，建立索引会占用一定的磁盘空间，但是在一般情况下，这个问题不太严重。笔者在实践中发现，如果在一个大表上创建了多种组合索引，索引文件的体积才会增长很快。最后，虽然索引能够解决大部分问题，但它也不是万能的。对于一些加上索引仍然有性能问题的表，笔者通过开启 MYSQL 慢查询日志检查有问题的SQL日志，然后并对 SQL 执行计划进行分析，这个要具体案例具体分析，这里不再展开。

在数据服务方面，可常规的业务中台的概念一样，笔者将数据模型按照应用要求做服务封装，就构成了数据服务。只是相比于业务中台的功能封装，数据封装要更难一点，毕竟OLTP功能的变化有限，而数据分析受业务因素的影响很大，变化更快，导致服务封装的难度变大。在这个层次上，笔者采用通过微服务架构，依据服务类型的不同，分摊给不同的项目组进行开发形成若干个微服务。微服务之间通过HTTP协议进行集成。之所以不采用业内主流的RPC协议主要是考虑到数据服务之间很少会发生频繁调用，主要还是和底层数据模型或者服务网关的交互,引入RPC并不会带来多大性能上的提升。但是为了确保服务的可靠性，笔者还是使用微服务治理方式对微服务进行管理。主要是采用开源中间件 Consult进行服务注册、服务发现、健康检查与负载均衡，具体的注册和发现实现细节限于篇幅，在此不多加赘述。另一方面为了确保服务性能，笔者在这个层次上引入了数据缓存的设计，目的是减少数据库资源的争夺，提升系统的响应时间。该数据中台使用 Redis集群实现热点数据的缓存，并采用主从复制技术实现缓存层的高可用与读写分离。当数据请求到达时，首先判断数据是否在缓存中，若命中直接返回。若涉及到数据库写入操作，则先将数据写入数据库，并将缓存标记失效。另一方面是MYSQL数据库读写分离的设计。对于MYSQL数据库实现一主多从的设计，所有读操作全部在副本上完成，主库与从库通过传递 binlog 实现数据同步。读写分离通常来说需要引入MYCAT之类的中间件进行读写的自动代理，或者在业务逻辑层上分离读写的对象实现读写分离。通过仔细分析，笔者从代码维护的角度考虑，选择引入中间件MYCAT来实现读写分离。通过读写分离技术，实现了数据库层面的负载均衡，避免了单机系统的故障与IO资源的争夺，极大提升了数仓操作的效率，确保了开放接口业务的稳定性。

但有数据模型和数据服务还是远远不够的，因为再好的现成数据和服务也往往无法满足前端个性化的要求。这时候就得授人以鱼。数据中台的最后一层就是数据开发，其按照开发难度也分为三个层次，最简单的是提供标签库（DMP），用户可以基于标签的组装快速形成分析报表，一般面向普通馆员。这里的标签库实际使用非常简单，馆员以拖拽的形式选择感兴趣的标签字段，比如性别、常驻区域、阅读偏好（分类号），通过添加逻辑运算符组合，并点击确定就可以快速生成目标读者群体。然后可以通过电话、短信、APP等推送方式对读者进行针对性的书籍、活动等推送服务。其次是提供数据开发平台，用户可以基于该平台访问到所有的数据并进行可视化开发，一般面向技术部SQL开发人员。最后就是提供应用环境和组件，让技术人员可以自主打造个性化数据产品，以上层层递进，满足不同层次人员的要求。由于引入了前端技术，因此这部分主要侧重于前端的优化。前端优化的主要目的是减少HTTP请求和缓存数据来提高页面的响应速度。一般在前端工具链中可以使用 Webpack 合并和压缩JS和CSS，还有使用专门的雪碧图制作工具来合并图片，减少图片请求。在数据方面，习惯将缓存分为强缓存和协商缓存两种。两者的主要区别是使用本地缓存的时候，是否需要向服务器验证本地缓存是否依旧有效。顾名思义，协商缓存，就是需要和服务器进行协商，最终确定是否使用本地缓存。般，我们会设置Cache-Control的值为“public, max-age=xxx”，表示在xxx秒内再次访问该资源，均使用本地的缓存，不再向服务器发起请求。显而易见，如果在xxx秒内，服务器上面的资源更新了，客户端在没有强制刷新的情况下，看到的内容还是旧的。因此笔者采用协商缓存。协商缓存意味着，当浏览器对某个资源的请求没有命中强缓存，就会发一个请求到服务器，验证协商缓存是否命中，如果协商缓存命中，请求响应返回的http状态为304并且会显示一个Not Modified的字符串。主要使用ETAG 和 Last-Modified 标志位在后端进行判断请求的资源是否过期。

通过指定合理的架构风格，实现了系统的水平分层，各个层次得到较为细致的设计。项目自正式上线以来，已正常运行了近半年时间，且部署实施后投入运行一直比较稳定。通过引入数据中台的思想来进行数据的治理和服务，提升了馆情数仓设计的质量，实现了数据的复用和规范，满足了前端馆员多变的业务需求。从取数角度看，根据统计，目前馆内80%以上的取数工作全部由前台自助完成，极大缩短了取数周期。但是从实践角度来看，随着接入数据的不断增长，传统的基于单机的关系型数据库会逐渐显现出瓶颈。因此项目二期的主要工作会放在基础数据库的改造，暂定使用 Hive 来实现分布式数据存储，使用 SPARK 进行基于分布式数据的内存计算，以及在此之上强化机器学习和数据挖掘的作用，进一步挖掘基础数据的内在规律，为更广泛的业务需求提供数据支持。