软件系统分析与设计

**IShare-学术成果分享平台**

**新浪微博架构调研报告**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **项目组成员信息** | | | | |
| **小组编号** | **3** | | | |
| **小组名称** | **就要整整齐齐队** | | | |
| **小组联系人** | **唐熙程** | | | |
| **学号** | **姓名** | **联系电话** | **本次实践中主要承担的工作内容** | **本次作业占比**  **(95%-105%之间)** |
| 20373809 | 周骁睿 | 18322377517 | 完成拓展性分析部分 | 100% |
| 20373677 | 陈楚凡 | 18159211139 |  | 100% |
| 20373481 | 刘谕笑眉 | 18811596392 |  | 100% |
| 20373114 | 唐熙程 | 18873578949 | 完成文档架构分析部分 | 100% |
| 20373571 | 吕新月 | 13305226276 | 完成引言、结论并合并校对文档 | 100% |
| 20373565 | 黄瑞 | 15828398946 |  | 100% |
| 20373661 | 徐亦佳 | 13857760431 | 完成性能分析部分 | 100% |
| 20373721 | 洪秋訸 | 17398568289 | 完成性能分析部分 | 100% |
| 20373420 | 张铭轩 | 18601000101 | 完成可靠性分析部分 | 100% |
| 20373460 | 张博皓 | 17704295626 | 完成审核 | 100% |

2022

版本变更历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 提交日期 | 主要编制人 | 审核人 | 版本说明 |
| 1.0 | 2022/10/17 | 吕新月 | 吕新月 | 完成文档大纲 |
| 1.1 | 2022/10/19 | 张铭轩 | 吕新月 | 完成新浪微博可靠性分析 |
| 1.2 | 2022/10/19 | 周骁睿 | 吕新月 | 完成新浪微博拓展性分析 |
| 1.3 | 2022/10/22 | 洪秋訸 | 吕新月 | 完成了性能分析 |
| 1.4 |  | 唐熙程 | 吕新月 | 完成文档架构分析部分 |
| 1.5 | 2022/10/25 | 吕新月 | 吕新月 | 完成了文档总结 |
| 2.0 | 2022/10/26 | 吕新月 | 张博皓 | 完成了文档合并 |
| 2.1 |  | 张博皓 | 张博皓 | 完成了文档修订 |

目 录

[1. 引言 1](#_Toc15563)

[1.1 文档概述 1](#_Toc7818)

[1.2 调研对象 1](#_Toc12360)

[1.3 参考文献 1](#_Toc4558)

[2. 架构分析 2](#_Toc13807)

[2.1 应用层 2](#_Toc2207)

[2.1.1 jersey框架 2](#_Toc18374)

[2.2 服务层 3](#_Toc18263)

[2.2.1 消息队列 MCQ 3](#_Toc10628)

[2.2.2 Motan RPC框架 4](#_Toc22669)

[2.3 资源层 5](#_Toc8755)

[2.3.1 DAL中间件 5](#_Toc31402)

[2.3.2 Redis Proxy 6](#_Toc25599)

[3. 可靠性分析 8](#_Toc22280)

[3.1 可靠性实现方案 8](#_Toc6982)

[3.1.1 数据备份 8](#_Toc14130)

[3.1.2 软件容错 8](#_Toc10089)

[3.2 新浪微博可靠性设计 10](#_Toc20737)

[3.2.1 数据库选择 10](#_Toc21235)

[3.2.2 备份与恢复策略 10](#_Toc18591)

[4. 性能分析 12](#_Toc19973)

[4.1 高性能实现方案 12](#_Toc16322)

[4.1.1 应用、数据、文件分离 12](#_Toc1331)

[4.1.2 利用缓存改善网站性能 12](#_Toc17303)

[4.1.3 数据库优化 13](#_Toc51)

[4.1.4 使用分布式系统 13](#_Toc14137)

[4.2 新浪微博性能优化 13](#_Toc3847)

[4.2.1 多级双机房缓存架构 13](#_Toc1704)

[4.2.2 分库分表 14](#_Toc404)

[4.2.3 Weibo Mesh 14](#_Toc3732)

[5. 扩展性分析 15](#_Toc17594)

[5.1 扩展性实现方案 15](#_Toc18306)

[5.1.1 事件驱动架构（EDA） 15](#_Toc22895)

[5.1.2 分布式服务 17](#_Toc30479)

[5.1.3 可扩展数据结构 17](#_Toc32103)

[5.2 微博扩展性设计 18](#_Toc99)

[5.2.1 架构细分 18](#_Toc23123)

[5.2.2 应用服务拆分 18](#_Toc29962)

[5.2.3 MCQ消息队列 19](#_Toc30652)

[6. 调研总结 19](#_Toc20946)

# 引言

## 文档概述

本文档的目标在于通过调研流行的技术栈，为项目自身的架构设计提供参考。

本文档将具体阐述新浪微博的架构设计，介绍新浪微博从应用层到数据层的技术栈。本文还将从安全性、可靠性、性能、扩展性四个方面对新浪微博进行剖析，并总结这四个方面常见的解决方案。

## 调研对象

微博，即微博客（MicroBlog），是基于用户关系的信息分享、传播以及获取平台。由新浪公司推出的新浪微博作为社会化媒体的典型代表，源自美国的Twitter。自2009年8月以来，就一直保持着爆发式增长。

新浪微博支持用户通过手机、PC等多种移动终端接入，以文字、图片、视频等多媒体形式实现信息的即时分享、传播互动。其主要功能有发布、转发、关注、评论、搜索和私信等，具有信息发布门槛低、随时随地传播信息、传播方式呈裂变、信息交互简便快捷、有用户排行等产品特点。

此外，微博还在2010年11月推出微群功能，即微博群的简称。微群兼备了通讯和媒体传播的双重功能，被是为网页版的“QQ群”。这项产品能够聚合有相同兴趣爱好或者相同标签的用户，将所有与之相应的话题全都聚拢在微群里。同时也有向未加入微群的用户随机推荐热门微群或其他机制的推荐功能。

近4年来，新浪微博并未曝出有较大影响的安全事故，且网站全天候不间断正常运行。

微博具有广泛的全国性影响以及足够的用户分布，有使用人群数量基数大，状态信息更新频繁、信息传播迅速等特点。根据2021年微博数据显示，微博月活跃用户达到5.3亿，移动端占比94%，日活跃用户达到2.3亿。在2021年微博发布的《微博牛年新春用户行为报告》中显示，在新年的第一分钟共有来自69个国家和地区的网友共发布1374668条微博。面对如此庞大的用户规模和业务量，需要高可用、高并发访问、低延时的强大后台系统支撑。

新浪微博能保证长时间无障碍运行，在高压的情况下不宕机，宕机后也能较快恢复。

新浪微博兼具媒体和社交属性，业务涵盖热搜、推荐、社交、直播、广告、公益等方面，需要高拓展性来满足这些需求。经过长时间的重构、线上运行、思索与沉淀，如今平台已迭代形成了第三代架构体系。

综上所述，可以认为新浪微博作为超大型网站拥有高安全性、高可靠性、高性能、强扩展性的特点并予以研究。

## 参考文献

1. [亿级用户下的新浪微博平台架构](https://cloud.tencent.com/developer/article/1963636)
2. [新浪微博千万级规模高性能、高并发的网络架构究竟如何实现](https://zhuanlan.zhihu.com/p/114532303)
3. [微博平台架构官微](https://weibo.com/tplatform?sudaref=www.baidu.com&is_all=1" \l "_rnd1666099967159)
4. [Spring技术栈官网](https://spring.io/)

# 架构分析



## 应用层

### jersey框架

微博的接口框架简化和规范了业务接口开发工作，将通用的接口层功能打包到框架中，采用了Spring的面向切面（AOP）设计理念。接口框架基于Jersey 进行二次开发，

Jersey是一个RESTful服务JAVA框架，与常规的JAVA编程使用的Struts框架类似，它主要用于处理业务逻辑层。

通过简单的配置，即可打造极简主义的REST接口。其实就是可以把我们的服务层开放API供前端调用。

**Jersey的优势有：**

* 强大而清晰的标准化注解

Jersey提供了非常容易理解和强大的注解机制，仅需要很少的代码就能完成复杂的事情，用户还可以基于Jersey提供的注解定义自己的注解。

* 高并发支持

Jersey为每一个用户访问开启新的进程，支持高并发访问，根据搭建的容器，并发数会有所不同，如Jetty容器可以轻松满足1000用户的并发访问。

* 可与Spring生态组件完美整合

Jersey支持集成Spring核心组件，既可以通过注解配置，也可以通过配置文件，比在Struts中集成Spring还要简单。

* 搭载Jetty容器，部署简单

Jetty容器的最大特点就是方便易携，体积小，作为内置化的容器，部署起来十分方便。

* RESTful Client支持，单元测试轻松完成

Jersey支持Server和Client端两种形式的应用开发，在编写单元测试时，Client机制会十分有用。

* 技术文档全面

Jersey提供丰富、条理清晰的使用用户指南文档。

**与SpringMVC的区别：**

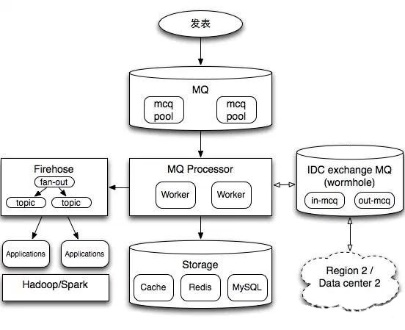
1. Jersey同样提供DI，是由glassfish hk2实现，也就是说，如果想单独使用Jersey一套，需要另外学习Bean容器；
2. MVC出发点是WEB，但Jersey出发点是RESTful，体现点在与接口的设计方面，如MVC返回复杂结构需要使用ModelAndView，而Jersey仅仅需要返回一个流或者文件句柄；
3. Jersey提供一种子资源的概念，这也是RESTful中提倡所有url都是资源；
4. Jersey直接提供application.wadl资源url说明；
5. MVC提供Session等状态管理，jersey没有，这个源自RESTFull设计无状态化；
6. Response方法支持更好返回结果，方便的返回Status，包括200，303，401，403；
7. 提供非常方便的方式访问RESTful;

## 服务层

### 消息队列 MCQ

消息队列提供一种先入先出的通讯机制，在平台内部，最常见的场景是将数据的落地操作异步写入队列，队列处理程序批量读取并写入DB，消息队列提供的异步机制加快了前端机的响应时间，其次，批量的DB操作也间接提高了DB操作性能，另外一个应用场景，平台通过消息队列，向搜索、大数据、商业运营部门提供实时数据。消息队列的核心作用就是三点；解耦、异步和并行。

微博平台内部大量使用的MCQ(SimpleQueue Service Over Memcache)消息队列服务，基于MemCache协议，消息数据持久化写入BerkeleyDB，只有get/set两个命令，同时也非常容易做监控（stats queue），有丰富的client library，线上运行多年，性能比通用的MQ高很多倍。



**图2-1消息队列架构图**

当前的主要消息队列分成如图三部分：

1、feed信息流主流程处理，图中中间的流程，通过相关MQ worker将数据写入cache、Redis及MySQL，以便用户浏览信息流。传统的队列使用主要是为了将操作异步处理，起到削峰填谷的作用，并解除多个序列操作之间的耦合关系。

2、流式计算，图中左边的流程，主要进行大数据相关实时处理。

3、多机房处理，将数据分发到多个机房，由于微博使用了一种多机房数据对等架构，通过消息队列将数据同步到多个机房，因此在每个机房都会有一个类似虫洞这样的消息收发模块IDC exchange MQ。

**消息系统的主要单元：**

mq(mcq)：消息队列，新浪的memcacheq，是一种单机的类似redis queue的一个一对一队列系统。

firehose：微博数据统一队列，使用http及memcache协议，由于在业务上每份数据会被多个项目使用，firehose主要用在一对多场景，且支持at-least-once投递保证，类似Apache Kafka。

worker：队列处理程序，自己开发的程序，完成特定的任务如将数据从队列取出入库，类似Apache Storm的bolt。

**架构主要特点**

实时性：所有的数据在100ms之内处理完成，包括存储型任务。

可扩展性：系统主要单元mq, firehose, worker都是无状态设计，同一个pool每个节点可以处理相同的工作，因此可以线性扩展。

可用性：由于上述的无状态设计，可用性可达 99.999%，自动failover，无单点。

**数据流设计特点**

统一的数据推送通道 firehose；

统一的标准化的格式，所有数据采用内部的protocol buffers格式输出。

### Motan RPC框架

RPC是指远程过程调用，也就是说两台服务器A，B，一个应用部署在A服务器上，想要调用B服务器上应用提供的函数/方法，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语义和传达调用的数据。

Motan是新浪微博开源的一套轻量级、方便使用的RPC框架，从功能角度来讲，RPC框架可以分为服务治理型和多语言型；motan属于前者，因此对motan框架可以简单的理解为：分离方法的调用和实现，并具双端服务治理功能；

Motan 提供的主要功能包括：

服务发现：服务发布、订阅、通知

高可用策略：失败重试（Failover）、快速失败（Failfast）、异常隔离（Server 连续失败超过指定次数置为不可用，然后定期进行心跳探测）

负载均衡：支持低并发优先、一致性 Hash、随机请求、轮询等

扩展性：支持 SPI 扩展（service provider interface）

其他：调用统计、访问日志等

**Motan 的交互流程如下**

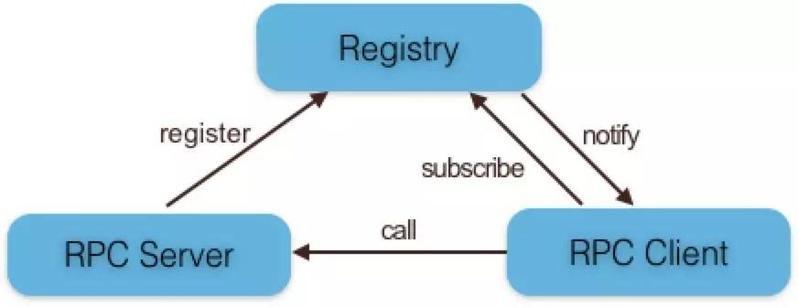
Motan 中有服务提供方 RPC Server，服务调用方 RPC Client 和服务注册中心 Registry 三个角色。

Server 向 Registry 注册服务，并向注册中心发送心跳汇报状态。

Client 需要向注册中心订阅 RPC 服务，Client 根据 Registry 返回的服务列表，对具体的 Sever 进行 RPC 调用。

当 Server 发生变更时，Registry 会同步变更，Client 感知后会对本地的服务列表作相应调整。

交互关系如下图：



**图2-2 RPC交互关系图**

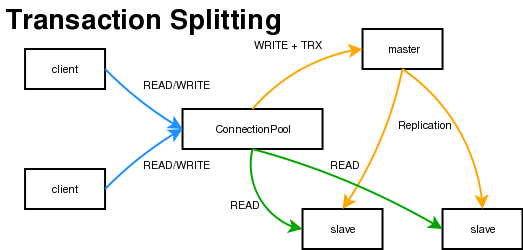
## 资源层

### DAL中间件

DAL是数据访问层的英文缩写，即为数据访问层(Data Access Layer)。

Mysql作为大量互联网公司使用的数据库，如果在使用过程中出现性能问题，会采用mysql的横向扩展，使用主从复制来提高读性能，要是解决写入问题，需要进行分库分表。DAL是链接数据前端和数据库的中间件，对于使用者来说是以数据库呈现，发出的请求由DAL管理并交由数据库处理。下面介绍一种可能的微博DAL，MySQL Router。

MySQL  Router是mysql官方发布的数据库中间件，是处于应用client和dbserver之间的轻量级代理程序，它能检测，分析和转发查询到后端数据库实例，并把结果返回给client。是mysql-proxy的一个替代品。其架构图和功能如下。



**图2-3事务分离**

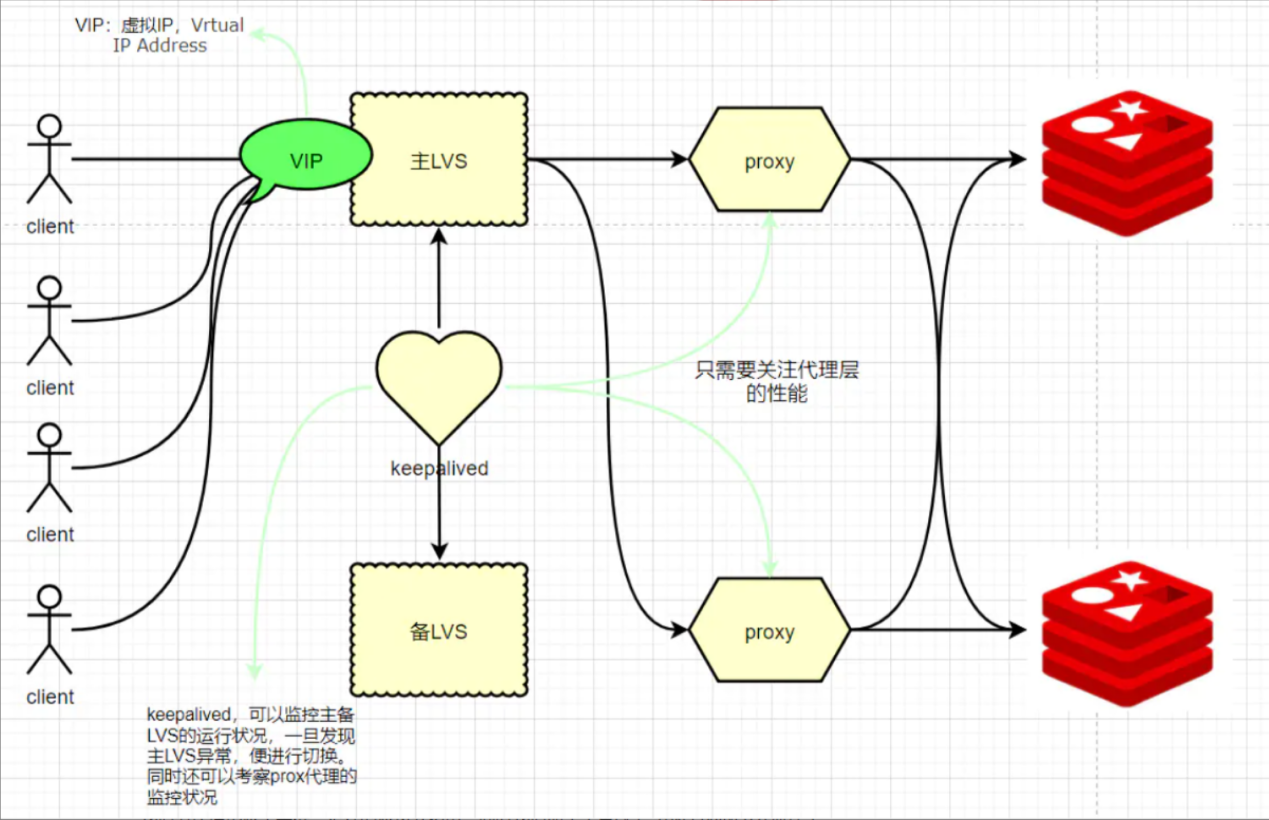
其读写分离的原理为：MySQL  Router接受前端应用程序请求后，根据不同的端口来区分读写，把连接写端口的所有更新和事务（TRX）发往主库，把连接只读端口的select查询以轮询方式发往多个从库，从而实现读写分离的目的。读写返回的结果会交给MySQL  Router,由MySQL Router返回给客户端的应用程序。

### Redis Proxy

随着业务规模的逐渐扩大，单Redis节点模式的负载增加。在客户端和Redis服务器之间加一层代理成了一种理想的解决方案，代理屏蔽后端Redis实现细节并向客户端提供Redis服务，可以完美解决Redis的高可用和扩展性的问题。

Redis集群首先要解决数据分片的问题。假设Redis集群有16384个hash槽。常见的分片方法即将客户端发送来的请求对key通过CRC16校验之后对16384取模来决定使用哪个槽，集群的每一个节点负责一部分hash槽。Redis Proxy可根据路由规则，即分片规则，转发给后台的各个Redis服务器，再原路返回，实现了管理Redis集群的能力。

对于Redis Proxy，一种可能的架构如下：



**图2-4 Redis Proxy**

其中VIP表示虚拟IP，LVS是负载平衡节点。主LVS节点拥有一个备份LVS节点，节点之间使用心跳包确保存活，当节点失效时，VIP自动修改接入备份节点从而实现高可用。LVS将请求分发给不同的代理节点，代理节点再根据路由规则查询Redis节点。网络的数据链路变长之后，数据在链路上的传输可能造成效率瓶颈。

另外，使用Redis Proxy还可以实现以下功能：

* 集群版使用模式转换

Proxy能够实现架构转换，帮助使用者如同在使用标准版一样地使用集群版。当标准版无法支撑业务发展时，无需修改代码即可将标准版的数据迁移至带有Proxy的集群版，大幅度降低业务改造成本。

* 管理只读节点流量

Proxy会实时探测只读节点的状态，当出现异常情况时，Proxy会执行流量管控动作

* 缓存热点Key信息

开启代理查询缓存功能（Proxy Query Cache）后，Proxy会缓存热点Key对应的请求和返回信息，当在有效时间内收到同样的请求时直接返回结果至客户端，无需和后端的数据分片交互，可更好地改善对热点Key的发起大量读请求导致的访问倾斜。

# 可靠性分析



## 可靠性实现方案

### 数据备份

数据备份是软件容灾的基础，可以防止系统出现操作失误或系统故障导致数据丢失，是将全部或部分数据集合从应用主机的硬盘或阵列复制到其它的存储介质，进行离线存储的过程。当出现人为误操作或者不可逆故障的时候，能够使用副本进行数据的恢复，它是软件可靠性的数据层支撑设计。

具体备份类型主要有三种：

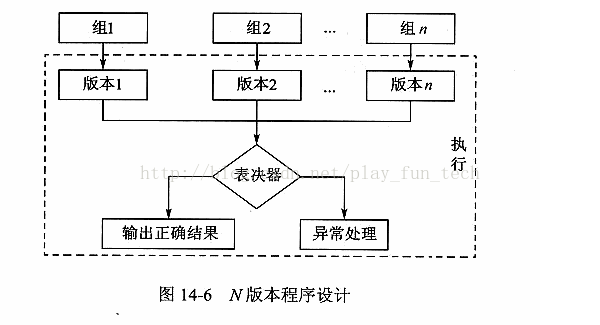
1. 完全备份：每次均对系统产生的所有数据进行备份。优点是恢复快，仅需上一次备份后的数据即可恢复；缺点是备份数据量大，每次完全备份所需时间很长，不太适用于业务繁忙的软件平台。
2. 增量备份：每次备份只更新的自上次备份（包括完全备份和增量备份）后，产生的新数据或者被修改过的数据。其优点是每次备份的数据量少，备份时间缩短；缺点是恢复时需要上一个全备份及多份增量备份，各分备份数据之间还存在互相依赖的关系，可靠性相对较差。
3. 差分备份：每次备份更新的自上次完全备份后，产生的新数据或者被修改过的数据。其优点综合上述两者，且一定程度规避两者的缺点。

### 软件容错

软件容错的主要目的是提供足够的冗余信息和算法程序，使系统在实际运行时能够及时发现程序设计错误，采取补救措施，以提高软件可靠性。具有容错设计的软件架构，在发生故障的情况下，系统可以保证不失效，仍然能够正常工作。

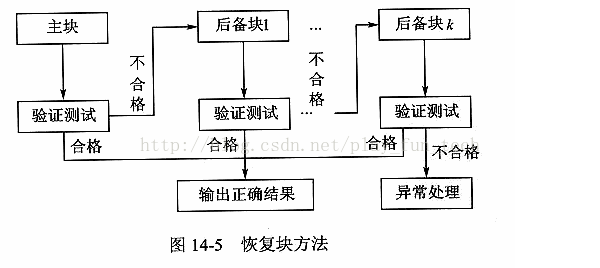
具体手段有如下三种：

1. N-版本程序设计：对于一个给定的功能，由多个不同的设计组独立编制出多个不同的程序，给定相同初始条件和相同输入的操作结果，然后在多台机器上运行并比较结果。如果多个版本运行的结果一致，则认为结果正确。若不一致，则按照多数表决的原则，判定结果的正确性。



**图3-1 N版本程序设计**

1. 恢复块方法：提供具有相同功能的运行程序主块和几个备份程序块文本，一个恢复块包含若干个功能相同、设计差异的程序块文本。一个运行程序块文本和多个备份程序块文本构成“动态冗余”，运行程序主块首先投入运行，结束后进行验收测试。如果没有通过验收测试，则判定出现故障，一旦运行程序块文本出现故障，则备份程序块文本进行替换。这一过程可以重复到耗尽所有的后备块，或者某个程序故障行为超出了预料，从而导致不可恢复的后果。设计时应保证实现主块和后备块之间的独立性，避免相关错误的产生，使主块和后备块之间的共性错误降到最低限度。



**图3-2恢复块方法**

1. 防卫式程序设计：对于程序中存在的错误和不一致性，防卫式程序设计是一种不采用任何传统的容错技术就能实现软件容错的方法，它的基本思想是通过在程序中包含错误检查代码和错误恢复代码，使得错误发生时程序能恢复到正确状态中去。实现策略包括以下三项：错误检测、破坏估计和错误恢复。具体的例子有try\_catch等。

## 新浪微博可靠性设计

### 数据库选择

新浪微博主要是在资源层框架中搭建了数据库，资源层的框架非常多，有封装MySQL与HBase的Key-List DAL中间件、有定制化的计数组件，有支持分布式MC与Redis的Proxy。新浪微博的存储除了大量使用MySQL和Memcached以外，还有一种存储也被广泛使用，那就是Redis。并且基于微博自身的业务特点，我们对原生的Redis进行了改造，因此诞生了两类主要的Redis存储组件：CounterService和Phantom。

对象库支持便捷的序列化与反序列化微博中的对象数据，序列化时，将JVM内存中的对象序列化写入在HBase中并生成唯一的ObjectID，当需要访问该对象时，通过ObjectID读取，对象库支持任意类型的对象，支持PB、JSON、二进制序列化协议，微博中最大的应用场景将微博中引用的视频、图片、文章统一定义为对象，一共定义了几十种对象类型，并抽象出标准的对象元数据Schema，对象的内容上传到对象存储系统（Sina S3）中，对象元数据中保存Sina S3的下载地址。

随着SSD硬盘的普及，其优越的IO性能被越来越多的替换传统的SATA和SAS磁盘，常见的应用场景有三种：1）替换MySQL数据库的硬盘，目前社区还没有针对SSD优化的MySQL版本，即使这样，直接升级SSD硬盘也能带来8倍左右的IOPS提升；2）替换Redis的硬盘，提升其性能；3）用在CDN中，加快静态资源加载速度。

微博平台将SSD应用在分布式缓存场景中，将传统的Redis/MC + Mysql方式，扩展为 Redis/MC + SSD Cache + Mysql方式，SSD Cache作为L2缓存使用，第一降低了MC/Redis成本过高，容量小的问题，也解决了穿透DB带来的数据库访问压力。

### 备份与恢复策略

新浪微博面对复杂的业务场景、超大规模的数据量，微博数据恢复中心需要同时权衡可用性、经济成本、安全性、效率。在整个设计过程中，所有的设计策略都围绕恢复时效性展开，把数据与恢复链路中涉及的所有环节都进行标准化与自动化。

1. 数据分级标准化

按照8：2原则，数据的重要性是有差异的。如果不加区分的恢复新浪微博高达100PB 级别的数据，无论是从成本上还是效率上都是无法承受的。微博从垂直与水平两个方向对数据的优先级进行拆分：

垂直层面：按业务的重要程度划分核心与非核心。核心业务的确认相对复杂，通常需要公司层面的领导拍板，但核心业务变更的频率非常低。微博上千个对外提供的API，核心API只有十几个。

水平层面：数据的访问是有时效性的，在微博场景下表现更加明显，7天内的数据访问量超过 98%。部分超过1年的数据被访问的吞吐基本维持在个位数甚至是零，简单的使用吞吐量作为数据的访问热力值，通过热力值对数据进行二次分级。

通过垂直与水平数据分级之后，核心热门数据的数据规模下降两个数量级到PB 级别，这使得整个数据在1小时内重建成为了可能

1. 数据资源服务拓朴构建自动化

有了数据分级标准后，需要按照分级标准找出备份哪些资源。数据是由服务生产的，所有的数据都会归约到一个特定的服务，因此数据的备份转化成了服务的备份，这样就可以通过追踪流量路径依赖的方式，发现流量路径中的服务节点，从而完成个服务网络拓朴图的构建。拓扑中服务依赖关系的一个核心准则是：核心业务不依赖非核心业务。避免核心业务拓扑扇出不可控，影响数据备份的范围与可用性。

微博服务的注册通过 weibo-mesh 启动完成后自动注册构建，形成服务依赖关系拓扑，开源或者第三方资源类的服务包括 MySQL、Redis 类的数据，微博通过 resource mesh agent 发现并自动注册到所属的服务池。

1. 数据备份标准化与自动化

不同的数据类型，备份方式各有不同。包括流量最开始入口四七层的配置、RPC 服务需要备份的二进制版本（镜像 URL）和 MySQL、Redis 等业务数据。所有接入数据恢复中心的服务，都需要提供 snapshot+streaming 两个 API。Snapshot 用于生成数据的快照，streaming 提供自上一次快照以来产生的所有的 operation。服务提供 API 后，数据恢复中心就能自动备份数据，实现备份与业务的解耦合。目前数据恢复中心提供了常用数据类型的 snapshot 与 streaming 的 API，相关业务只要上线，即可纳入数据恢复中心进行备份。

1. 数据备份服务化：微博的321备份机制

数据备份两个最基本的要求是数据的一致性与数据完整性。单个文件的数据一致性通过数据摘要进行动态存储验证，使用纠错码有效处理 bit 反转静默存储错误。

对于数据的完整性，微博使用大块存储结合 Merkle Hash Tree 来解决。所有的数据文件，都拆分或者合并成1GB的一个数据块(1GB大小的块是一个最佳实践值，过小网络传输效率低，过大单个块传输耗时长，不利于提升并发效率）。一个完整的数据备份元数据由＜版本号，所属业务，数据类型，数据块列表＞四元组构成，数据备份服务提供 APl 可以进行全网备份或者指定业务与数据类型的备份。备份 APl 与业务无关，与数据类型无关。

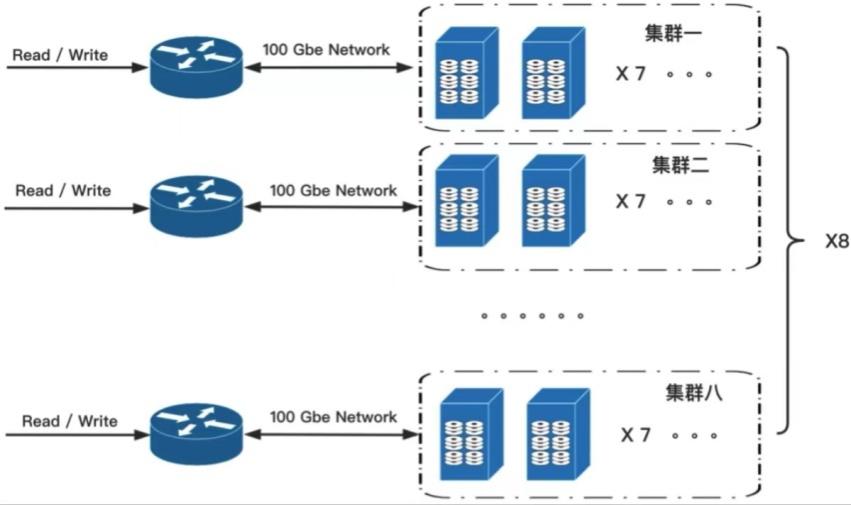
数据备份利用 snapshot 进行全量备份，使用 streaming 支持动态增量备份。我们采用了 watch 变化的机制，数据容量累计到一定程度或者超时则会把增量数据做一次 checkpoint 。 snapshot 通常是天级别， checkpoint 一般是小时级。

在微博使用的资源中，一类是类似 Redis ,读写量非常高，增量数据产生的 operation streaming 量非常大，但这类资源的单实例容量一般控制在10GB级别，可以提升 snapshot 的频率，降低两个 checkpoint 之间的数据量以提升恢复效率。另一类是 MySQL ，单实例容量非常大，通常在 TB 级别，写入量较低，可以降低 snapshot 的频率，提升 checkpoint 的频率，以在存储成本与数据恢复效率上达到平稳。

数据服务服务通过 checkpoint 机制，将数据备份转换成了数据块的存储与备份。

微博在数据备份上遵循了321策略，1)所有的数据备份都至少包含2个热备，2个冷备；2）在线热备数据存储在 SSD 设备，冷备数据存储在独立的 OSS 集群；3）数据会在异地离线存储，通过专线进行数据同步与传输。

数据备份服务存储中心选择的是在云原生场景下应用广泛的对象存储 OSS 。在逻辑上，恢复中心由管理端与存储端组成，且二者逻辑上是独立的。存储端支持多物理存储，具体来说，支持在物理上各个机房内的自己 OSS 存储集群，同时还支持接入云厂商的 OSS 服务；由管理端来统一调度。管理端本身亦支持异地多活。自建存储共8个集群，每个小集群7台存储主机，单机容量20T,单集群容量140T,8个集群1120 TB 。集群对外网络带宽为100Gbps。自建存储集群示意图如下，能够支撑 PB 级别的数据备份与快速读取。



**图3-3自建存储集群示意图**

# 性能分析



## 高性能实现方案

### 应用、数据、文件分离

将应用程序、数据库、文件各自部署在独立的服务器上，并且根据服务器的用途配置不同的硬件，达到最佳的性能效果。

### 利用缓存改善网站性能

本地缓存，顾名思义是将数据缓存在应用服务器本地，可以存在内存中，也可以存在文件，组件。本地缓存的特点是速度快，但因为本地空间有限所以缓存数据量也有限。OSCache就是常用的本地缓存。

分布式缓存的特点是，可以缓存海量的数据，并且扩展非常容易，在门户类网站中常常被使用，速度按理没有本地缓存快，常用的分布式缓存是Memcached、Redis。

反向代理指的是部署在网站的机房，当用户请求达到时首先访问反向代理服务器，反向代理服务器将缓存的数据返回给用户，如果没有缓存数据才会继续访问应用服务器获取，这样做减少了获取数据的成本。反向代理有Squid，Nginx。

假设我们的服务器都部署在杭州的机房，对于浙江的用户来说访问是较快的，而对于北京的用户访问是较慢的，这是由于浙江和北京分别属于电信和联通的不同发达地区，北京用户访问需要通过互联路由器经过较长的路径才能访问到杭州的服务器，返回路径也一样，所以数据传输时间比较长。对于这种情况，常常使用CDN解决，CDN将数据内容缓存到运营商的机房，用户访问时先从最近的运营商获取数据，这样大大减少了网络访问的路径。比较专业的CDN运营商有蓝汛、网宿。

### 数据库优化

首先是读写分离和分库分表。随着用户量的增加，数据库成为最大的瓶颈，改善数据库性能常用的手段是进行读写分离以及分库分表，读写分离顾名思义就是将数据库分为读库和写库，通过主备功能实现数据同步。分库分表则分为水平切分和垂直切分，水平切分则是对一个数据库特大的表进行拆分，例如用户表。垂直切分则是根据业务的不同来切分，如用户业务、商品业务相关的表放在不同的数据库中。

还可以使用NoSql数据库和搜索引擎。对于海量数据的查询和分析，我们使用nosql数据库加上搜索引擎可以达到更好的性能。并不是所有的数据都要放在关系型数据中。常用的NOSQL有mongodb、hbase、redis，搜索引擎有lucene、solr、elasticsearch。

### 使用分布式系统

#### 分布式文件系统

用户一天天增加，业务量越来越大，产生的文件越来越多，单台的文件服务器已经不能满足需求，这时就需要分布式文件系统的支撑。常用的分布式文件系统有GFS、HDFS、TFS。

GFS（Google File System）可以给大量的用户提供总体性能较高的服务

• 适合部署在廉价的普通硬件上

• 提供容错功能

HDFS（Hadoop Distributed File System）能提供高吞吐量的数据访问，非常适合大规模数据集上的应用

• 运行在通用硬件(commodityhardware)

• 高度容错

• 适合部署在廉价的机器上

TFS（Taobao Flies System）主要针对海量的非结构化数据，提供高可靠和高并发的存储访问

• 高可扩展、高可用、高性能

• 面向互联网服务

• 适合海量小文件存储

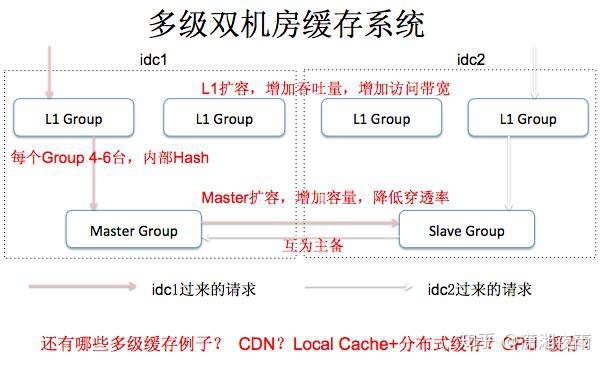
#### 分布式服务

各个业务应用都会使用到一些基本的业务服务，例如用户服务、订单服务、支付服务、安全服务，这些服务是支撑各业务应用的基本要素。我们将这些服务抽取出来利用分部式服务框架搭建分布式服务。

## 新浪微博性能优化

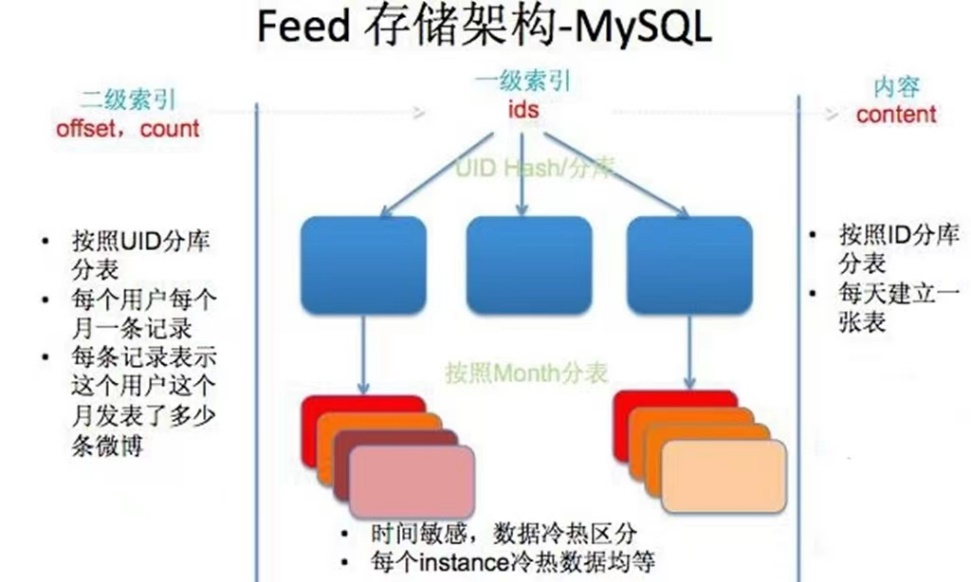
### 多级双机房缓存架构

微博使用了双层缓存，每个L1上都是一组 (包含4-6台机器)，左边的框相当于一个机房，右边又是一个机房。L1缓存增加整个系统的QPS，其次以低成本灵活扩容的方式增加系统的带宽。两个机房是互为主备，或者互为热备。如果两个用户在不同地域，他们访问两个不同机房的时候，假设用户从IDC1过来，因为就近原理，他会访问L1，没有的话才会跑到 Master，当在IDC1没找到的时候才会跑到IDC2来找。同时有用户从IDC2访问，也会有请求从L1和Master返回或者到IDC1去查找。IDC1和IDC2，两个机房都有全量的用户数据，同时在线提供服务，但是缓存查询又遵循最近访问原理。



**图4-1 多级双机房缓存架构图**

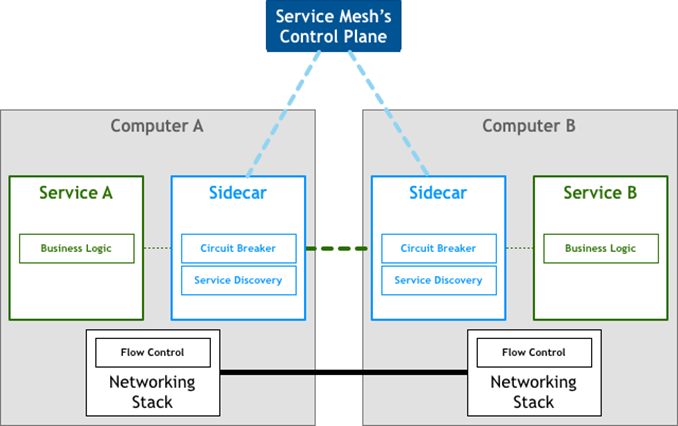
### 分库分表

微博采用Feed存储架构，微博的博文主要存在 MySQL中。分为内容表和索引表，内容表即每条内容一个索引，每天建一张表，而对于索引表，共建了两级索引。在做分库分表时，同时考虑了按照UID和按照时间维度哈希。根据用户使用微博的特性，一级索引的时候会先根据关注的用户，取他们的前条微博ID，然后聚合排序。而对于热点新闻，由于热度与时间相关性极高，这种场景就需要按照时间维度做分表，首先冷热数据做了分离(可以对冷热数据采用不同的存储方案来降低成本)，其次，可以控制数据库表的爆炸。而快要速找到某用户发布的某一时段的微博时，可以通过二级索引快速定位。

**图4-2 Feed存储架构图**

### Weibo Mesh

Weibo Mesh是新浪微博对于Service Mesh的一个实践。



**图4-3 Service Mesh示意图**

微博将Motan RPC胖客户端实现的治理功能下沉到Agent上，服务注册和发现依赖微博自研Vintage命名和配置服务，对服务的订阅和发现来建立服务间依赖的逻辑网络。业务与的通信协议保持一致，Agent支持HTTP和RPC的调用，业务只需把原有的调用指向Agent即可，不需要改造业务代码。微博实现了全新语言无关的通信协议 Motan2和跨语言友好的数据序列化协议Simple来应对跨语。

除了代理Service的能力外，Mesh体系提供了缓存、队列等服务化代理，业务方可以与依赖缓存、队列资源治理解耦的能力。可以大幅提高那些治理能力比较薄弱的业务和语言的架构水平。随着云原生技术的日趋完善，会有越来越多的基础设施从原有的 SDK 中抽象出来。未来数据库访问会以 Database Mesh形式提供访问，封装数据分片、读写分离、从库负载均衡、熔断、链路路采集能力。

# 扩展性分析

扩展性指对现有系统影响最小的情况下，系统功能可持续扩展或提升的能力。表现在系统基础设施稳定不需要经常变更，应用之间较少依赖和耦合，对需求变更可以敏捷响应。它是系统架构设计层面的开闭原则（对扩展开放，对修改关闭），架构设计考虑未来功能扩展，当系统增加新功能时，不需要对现有系统的结构和代码进行修改。

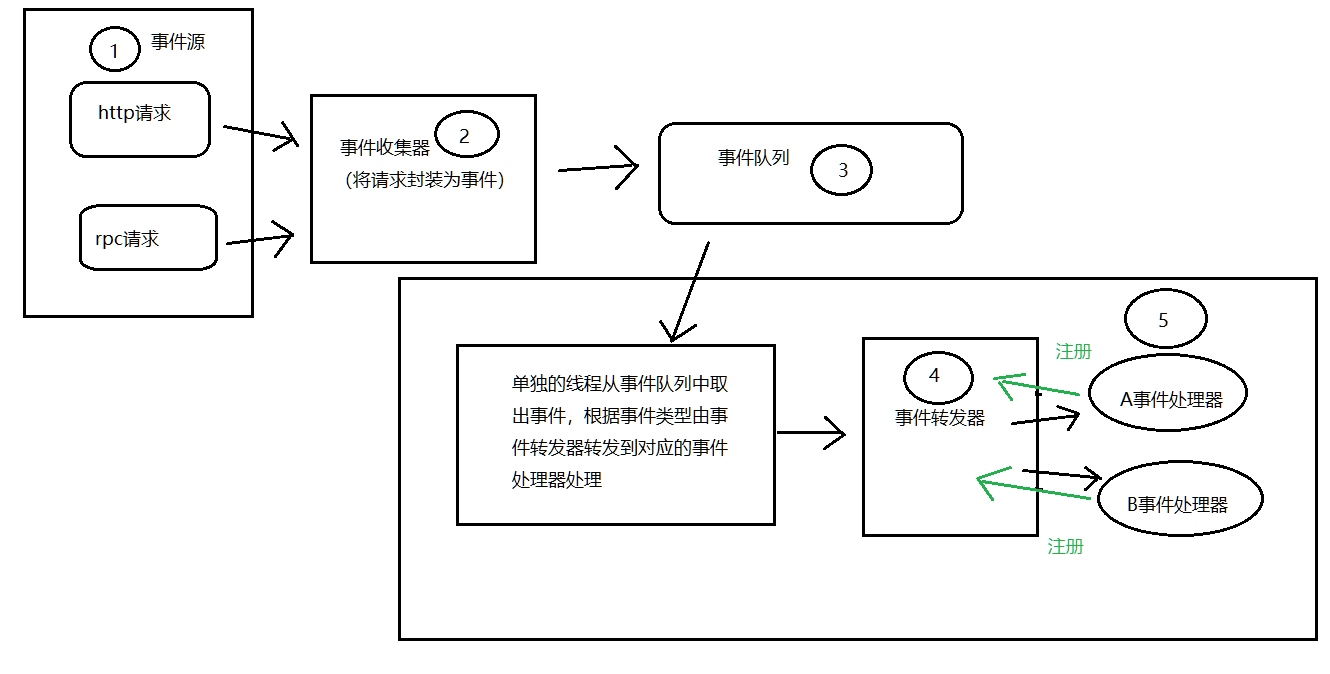


## 扩展性实现方案

### 事件驱动架构（EDA）

如果模块之间不存在直接调用，那么新增模块或者修改模块就对其他模块影响最小，这样系统的可扩展性无疑更好一些。

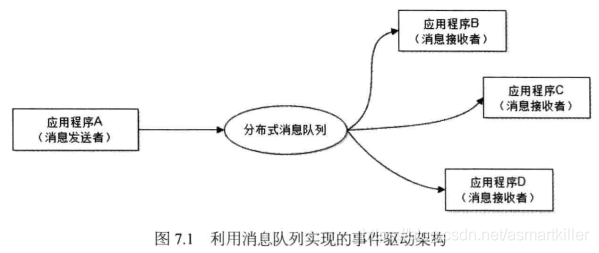
事件驱动架构通过在低耦合的模块之间传输事件消息，以保持模块的松散耦合，并借助事件消息的通信完成模块间合作，典型的EDA架构就是操作系统中常见的生存者消费者模式。在大型网站架构中，具体实现手段有很多，最常用的是分布式消息队列。



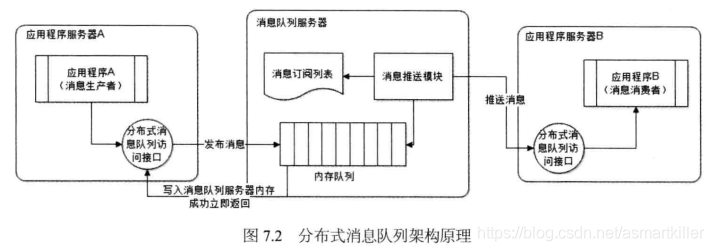
**图5-1 事件驱动架构示意图**

队列是一种先进先出的数据结构，分布式消息队列可看作将这种数据结构部署到独立的服务器上，应用程序可以通过远程访问接口使用分布式消息队列，进行消息存取操作，进而实现分布式的异步调用。

消息队列基于发布—订阅模式工作，消息发送者发布消息，一个或多个消息接收者订阅消息。消息发送者把消息发送至分布式消息队列后就处理完毕，然后由消息订阅者从消息队列中获取消息进行处理。对于新增的业务，只要对某个消息感兴趣，就可以订阅该消息，而这对原有的系统和业务没有任何影响，从而实现系统的可扩展性设计。



**图5-2 利用消息队列实现的事件驱动架构**

****

**图5-3 消息队列架构原理**

分布式消息队列可以很复杂，比如可以支持ESB（企业服务总线）、支持SOA（面向服务的架构）等；也可以很简单，比如用MySQL也可以当作分布式消息队列：消息生产者程序将消息当作数据记录写入数据库，消息消费者程序查询数据库并按记录写入时间戳排序，就实现了一个事实上的分布式消息队列，而且这个消息队列使用成熟的MySQL运维手段，也可以达到较高的可用性和性能指标。

### 分布式服务

随着我们的业务量越来越大和越重要，单体的架构模式已经无法对应大规模的应用场景，而且系统中决不能存在单点故障导致整体不可用，所以只有垂直或是水平拆分业务系统，使其形成一个分布式的架构，利用分布式架构来冗余系统消除单点的故障，从而提高整个系统的可用性。同时分布式系统的模块重用度更高，速度更快，扩展性更高是大型的项目必不可少的环节。

分布式服务可以通过接口降低系统的耦合性，不同的子系统之间通过相同的接口描述调用服务。分布式服务的拆分主要分为两类，一类是纵向拆分，把一个大应用拆分为多个小应用。如果新增的业务较为独立，就直接将其设计并部署为一个独立的 Web 应用。一类是横向拆分，把复用的业务拆分出来，独立部署为分布式服务，新增的业务只需要调用这些分布式的服务，就可以快速搭建出一个应用系统。即使模块内的业务逻辑发生变化，只要保持接口一致，就不会影响其他模块。

纵向拆分较简单，通过梳理业务，把关联较少的业务剥离，使其成为独立的 Web 应用。而横向拆分不仅需要识别出可复用的业务、设计服务接口以及规范服务之间的依赖关系，而且还需要一个完善的分布式服务管理框架，使公用模块服务化，提供其他系统使用，提高可重用性，扩展性。

### 可扩展数据结构

使用 NoSQL 数据库的 ColumnFamily技术可以做到可扩展的数据结构设计。它是一种面向列族的稀疏矩阵的存储格式。

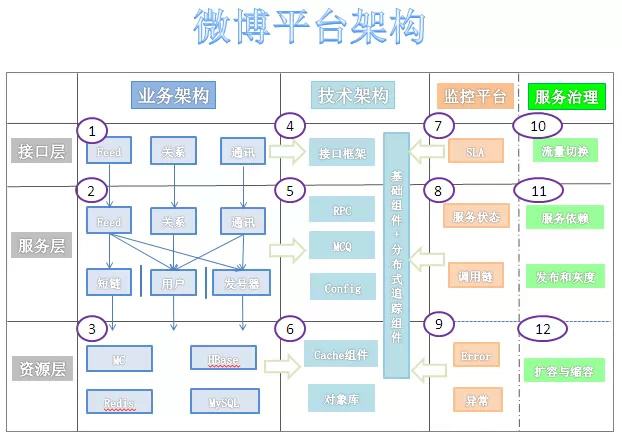
只需要指定 ColumnFamily 的名字，即可创建表。字段可以在写入数据时再指定，通过这种方式，一张表可以包含数百万个字段。这就使得应用的数据结构可以随意扩展。只需要指定任意字段名称和值即可查询。

## 微博扩展性设计

微博作为用户上亿级的软件，可扩展性较好，前端机、业务服务、资源层面支持按需快速扩容，可以通过快速增加私有云、公有云的服务器数量来提升集群的整体性能。

### 架构细分

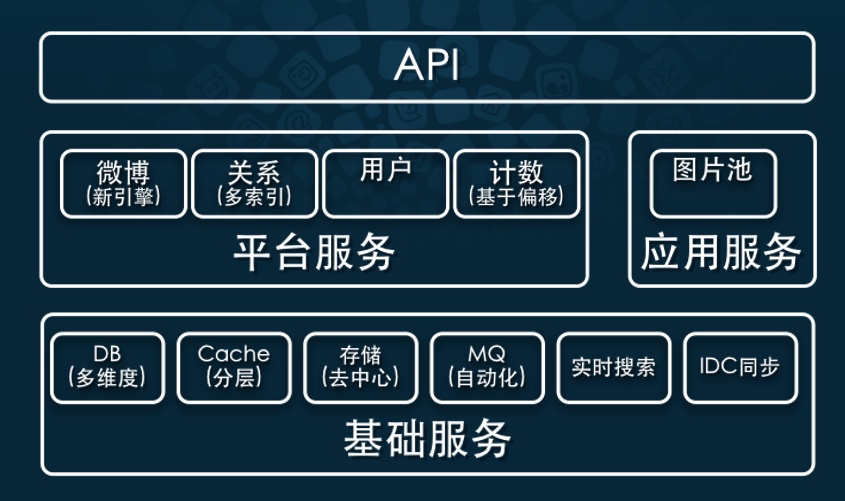
微博将架构更加细分，或增加数据架构师，cache架构师，分布式架构师；数据库sharding碰到瓶颈，开始使用分布式数据服务；数据访问需要根据业务特点细分；开发、运维、测量、调优具备有自己的专有工具；所有服务需要地理多机房分布，具备IDC容灾设计。



**图5-4 微博平台架构细分提高可拓展性**

### 应用服务拆分

当前微博对外主要以移动客户端、web主站、开放平台三种方式提供服务，并通过平台接入层访问微博Feed平台体系。其中平台服务层按照业务板块来划分应用代码，使单个应用的职责更清晰，相互之间可以做到独立升级迭代。各种业务进行模块化拆分，诸如feed计算、微博内容、关系、用户、评论、短链、私信等分解为独立的服务模块，对每个模块实现服务化架构，通过标准化协议进行统一访问。中间层通过各种服务组件来构建统一的标准化服务体系，如motan提供统一的rpc远程访问，configService提供统一的服务发布、订阅，cacheService提供通用的缓存访问，SLA体系、Trace体系、TouchStore体系提供系统通用的健康监测、跟踪、测试及分析等。存储层主要通过Mysql、HBase、Redis、分布式文件等对业务数据提供落地存储服务。整体服务的可拓展性大幅提升。



**图5-5 微博服务拆分**

### MCQ消息队列

微博模块化的系统，通过消息队列进行交互，使模块之间的依赖解耦。

微博平台内部大量使用的MCQ(SimpleQueue Service Over Memcache)消息队列服务，基于MemCache协议，消息数据持久化写入BerkeleyDB，只有get/set两个命令，同时也非常容易做监控（stats queue），有丰富的client library，线上运行多年，性能比通用的MQ高很多倍。

# 调研总结

本次调研，我们先调研了新浪微博的基础架构设计，理解了新浪微博水平方向分层式架构、垂直方向进一步细分的正交式架构模型。即新浪微博水平方向分为应用层、服务层、资源层三层，垂直方向进一步细分为业务架构、技术架构、监控平台和服务治理平台。而正交分解法把整个架构又分为12个区域，每个区域定义了一个核心功能点。

随后，我们调研了三个主要评价指标的解决方案，并对新浪微博的这三个方面进行了对应的评估，新浪微博作为大型网站，在这三个方面均有一套完善独特的的解决方案。

总的来说，新浪微博作为国内最大的社交媒体网络平台之一，其资本、运营、技术等方面实力都非常强悍，可以为这个大型网站专门构建出一整套架构设计与各个方面优化的解决方案。这样的大型解决方案虽然不适用于小公司以及类似于我们十人左右的小团队，但是其架构思路及常见问题的解决方案仍然值得借鉴。