百度万人协同规模下的代码管理架构演进

廖超超 CSDN 2017-03-15

互联网研发，唯快不破。为了提升公司整体研发效率，百度引入了业界的优秀工程实践，设计开发了一整套研发工具链。主要包括项目管理平台、代码开发协作平台和持续交付平台，分别针对需求、开发和交付场景，提供工具、流程和数据支持，如图1所示。

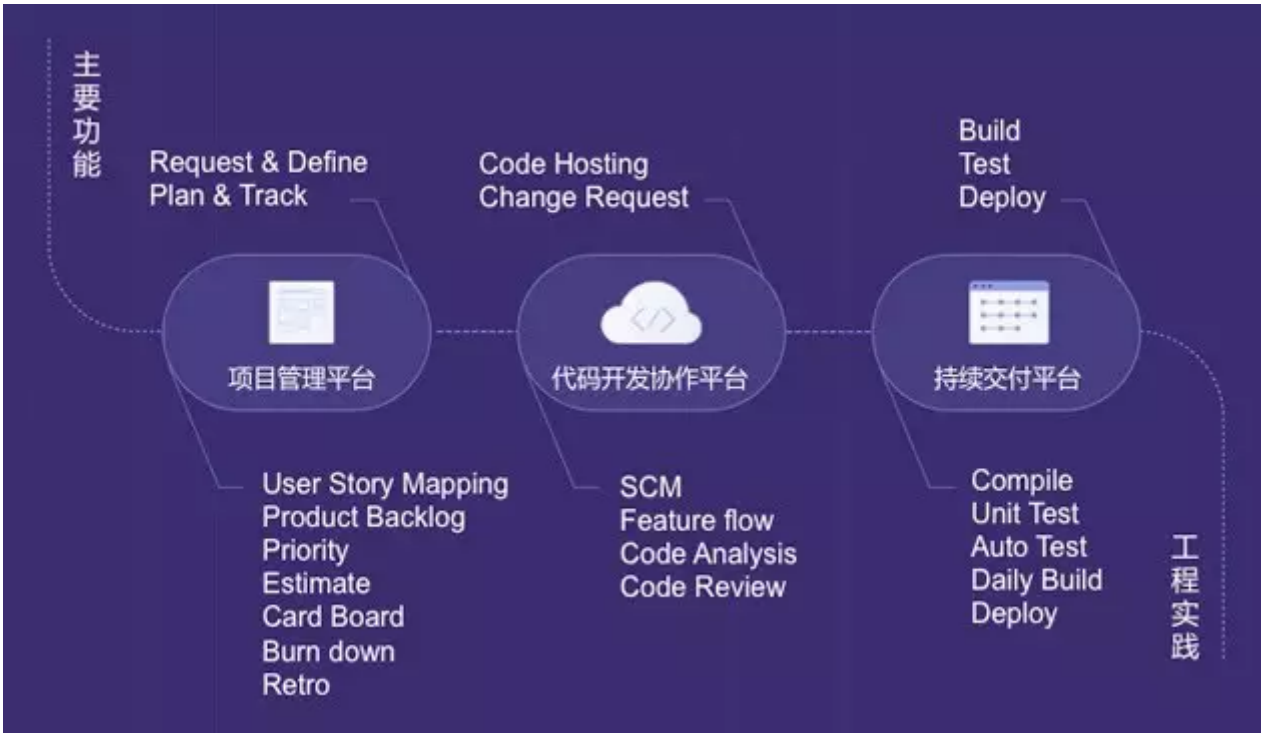

图1 百度研发工具链

代码管理的目标场景就是开发场景，是研发活动的核心环节，承载着打通需求、交付上下游的作用。百度代码管理建设分别从文化传播、工程实践和产品建设三个方面入手推进公司代码管理水平的不断提升。为此，我们推出了代码管理建设的五级金字塔模型，如图2所示，分别代表了代码管理建设的不同能力水平。

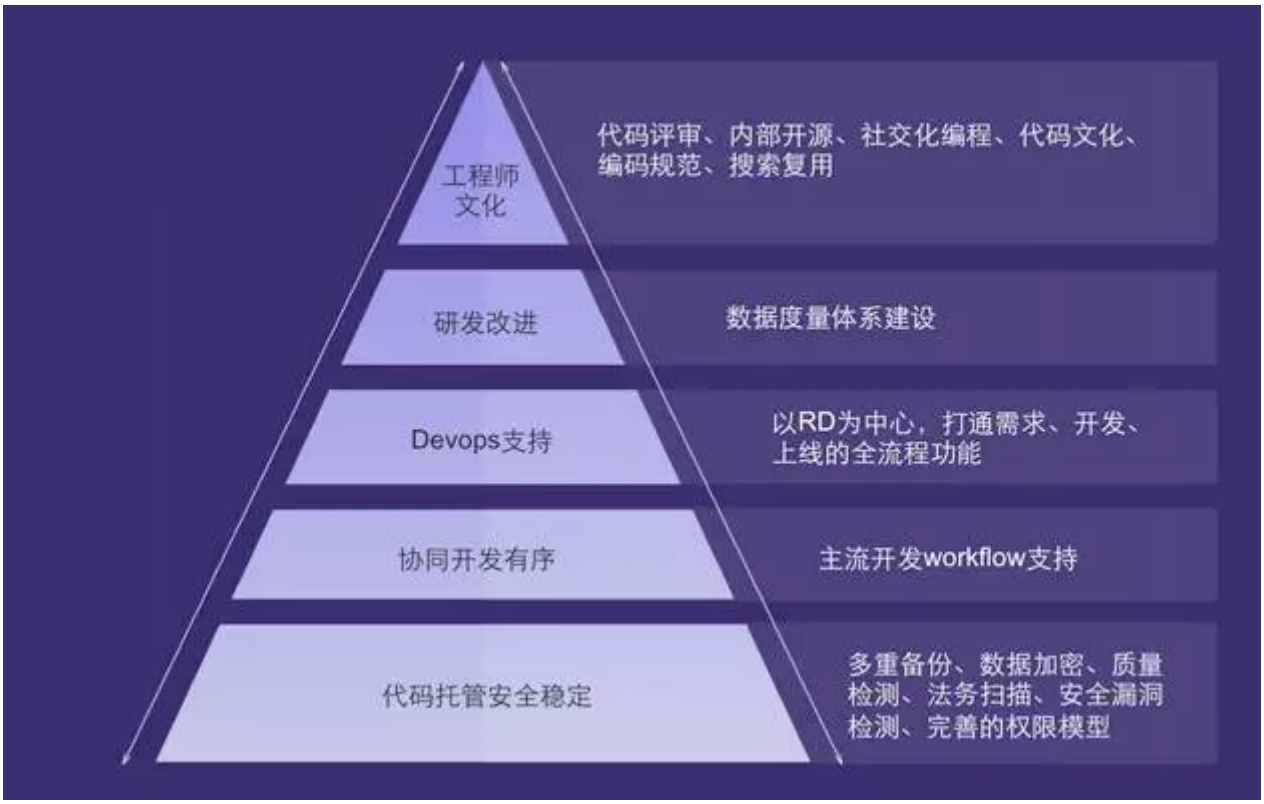

图2 百度代码管理概况

最底层是代码托管，这是代码管理最基础的能力。

第二层是协同开发，就是支持各个业务线在不同的研发模式下进行快而有序地协作开发。百度的产品、业务线众多，不同的团队规模、不同的开发语言、不同的研发模式都给开发协同提出了不同的需求。

第三层是DevOps支持，实现产品全生命周期的工具全链路打通与自动化。

第四层通过研发数据度量体系的建设，给公司提供研发数据参考，促进研发流程的改进。

第五层工程师文化建设，在公司内部推行代码评审、内部开源、社交化编程等工程师文化。

百度代码管理的挑战

百度拥有万人开发团队，近十万项目，每周代码自动检出的问题超二十万，每天发起评审超1万次。为了保证代码质量，我们要求代码提交前和提交后都进行自动化检查。为了加速编译和集成，我们有大规模的分布式编译系统和持续集成系统。百度C/C++语言是源码依赖，编译系统需要检出所有的依赖代码，这样代码库的访问压力呈指数级增长。这些都是百度代码管理面临的挑战，总结起来就是这三点：代码质量、规模协同和安全稳定的服务，如图3所示。

图3 百度代码管理遇到的挑战

面对这三大挑战，代码开发协作平台重点解决代码管理五个方面的问题：代码托管、协同开发、代码质量、代码安全与开放、研发改进。

1. 代码托管

代码托管是研发的基础设施。代码托管需要保证服务的安全、稳定和可靠，同时保证在大规模协同场景下的高性能。

2. 代码质量

基于代码入库流程，提供简单易用的代码评审，并且在评审环节支持代码扫描、编码规范、安全扫描等自动化检查，同时支持打通持续集成进行自动化测试，从而保证代码入库前就得到充分的质量检验。

3. 代码安全与开放

代码安全要求对访问控制权限做严格的限制，需要支持安全扫描和安全审计等；代码开放鼓励代码共享、开源，从而实现代码复用。

4. 协同开发

支持主流的Workflow以满足各业务线不同的研发模式的需求，如：传统的分支开发、主干开发、特性分支、git flow等工作流。

5. 研发改进

研发管理需要有数据支撑，用数据度量一切，不断地优化研发流程，促进高效协同，提升研发效率。

百度代码管理架构演进的历程

代码开发协作平台经历了这四个阶段的演进。面对不同阶段的不同问题，我们所采用的方案也不同，如图4所示。

图4 百度代码管理架构演进的历程

产品初创时期

为了快速验证产品，我们采用了精益的思路快速实现了MVP。在代码库服务设计上，暂时不考虑容量和性能的问题，采用了Master-Slave这种单实例结构，如图5所示。

图5 产品初创时期——架构

在这种架构下，我们主要从两个方面保证服务的安全可靠。

1. 存储上做了RAID，同时使用DRBD实时备份数据，保证数据可靠性。我们采用的是DRBD的同步复制协议，也就是说master的数据都会实时同步到slave上。
2. 为了保证服务的高可用，使用KeepAlived，确保在master故障时能够快速切换到slave，实现自动failover。

产品发展时期

随着平台的快速发展，代码库的并发和容量都在急剧增长。我们首先采用大内存、SSD硬盘来提升硬件性能。然后进行I/O、网络、缓存等优化。经过反复的性能测试得到了单机的最优配置。

在扩容方案上我们主要考虑了两种方案：分布式存储和数据分片。

1. 分布式存储
   1-1. 优点是架构简单，数据有备份，容量可以横向伸缩。
   1-2. 缺点是I/O性能下降。
2. 数据分片
   2-1. 优点是性能可靠，控制灵活，便于扩展（根据业务需求实现不同的分片策略和负载均衡方案）。
   2-2. 缺点是对现有的架构改变较大；跨分片的操作实现成本较高。

我们经过性能测试和MVP验证后，最终选择了数据分片的方案。主要的原因就是代码服务是高I/O的服务，分布式存储的I/O性能较本地存储差距比较明显，尤其写的性能更是下降了一个数量级。

图6 产品发展时期——架构

图6所示的这版架构的主要改变是git服务分实例部署。根据Repositories进行分片，将不同的Repositories分配到不同的实例。数据库服务采用主备的方式独立部署，同时支持了数据库访问的读写分离。

用户请求首先经过proxy，调用统一的路由服务将请求转发到对应实例。认证服务独立部署，proxy集成认证模块，加强了用户身份认证。

路由服务是核心服务。为了降低业务系统的改造成本，设计了统一的路由服务和路由模块，通过切面的方式拦截所有对代码库的访问请求，从而实现对业务代码的较低侵入和对调用方的透明。在路由设计上，因为首先使用了去中心化的微服务架构，所以采用客户端路由的方式。同时，增加了本地缓存，即使路由服务宕机，路由依然可以正常运行，如图7所示。

图7 产品发展时期——路由设计

产品成熟时期

由于编译、自动化测试、持续集成等需求出现了爆发式的增长，代码库每日读的请求超过30万次，每日写的请求超2万次。高峰时段，TPS将近1000，千兆网卡全部被打满。经过对吞吐量的需求的评估，我们预计TPS将突破10000。为了保证性能，高峰时段下载代码的速率，自动化系统应该在30MB/s以上，开发人员必须在5MB/s以上。因此，吞吐量不足的问题已经成为最核心的问题。我们的改进方案如下：

1. 增加带宽，千兆网卡换成万兆网卡。

2. 增加机器。通过拆分更小的实例来分摊带宽压力。增加每组实例的只读节点，因为我们的场景是读远大于写的，吞吐量的压力大多来自读请求。同时将闲置的冷备节点升级成只读节点。

图8 产品成熟期——架构

图8是读写分离的架构图，通过proxy判断读写请求，将写请求发给master节点，读请求通过负载均衡模块分别发给实例的所有节点。在这版架构升级的过程中，我们还是采用DRBD+KeepAlived实现容灾备份方案。读写分离大大提高了系统吞吐量，但是DRBD冷备机器闲置，严重浪费资源。所以，我们做了进一步的改进。

图9 产品成熟期——架构优化

图9是改进后的架构图，我们废弃了DRBD备份，并且实现自己的高可用方案。我们的方案主要分为两个阶段：

1. master节点的失效判定。某个proxy节点的心跳检测捕获master节点异常后，发起投票，如果过半数的proxy节点都判断master节点异常，就判定master失效。
2. slave节点提升。再进行一轮投票，将这组git实例中一个slave节点提升为master节点。投票完成之后，将新的实例信息写入路由服务，路由服务通知所有调用方路由变更，及时更新本地路由缓存。

代码开发协作平台的整体架构

图10 百度代码开发协作平台架构

百度代码开发协作平台整体上采用微服务架构，基于自主研发的微服务框架构建各个业务服务单元，独立开发、发布、部署和运行。整体的架构如图10所示。

1. 接入服务

Httpd Proxy主要用于Web访问，Sshd Proxy用于Git命令行操作，API Gateway用于统一提供开放API，便于API的安全授权、管理。

在接入服务之上采用百度统一前端接入服务构建高可用负载均衡器，一方面提高系统的并发访问能力，另一方面提高系统的防攻击能力，保证平台的安全性。

2. 访问控制

平台构建统一的安全策略和用户认证体系，确保系统安全。

3. 服务中心

服务中心是服务治理的核心，提供服务注册/发现、服务路由、服务配置、服务降级、服务熔断、流量控制等功能，保证平台整体的服务稳定性。通过服务路由、配置管理中心、服务注册/发现等机制来统一管理服务，另外提供统一的管理控制台管理应用服务集群、Git集群和基础服务集群等。

4. 开放服务

平台同时支持Webhook和Plugin两种开放能力的方案，支持第三方系统方便集成。Webhook主要的应用场

景是当开发人员提交代码变更后，自动触发持续集成构建。Plugin主要应用在代码评审环节的自动代码检查。

5. 业务服务

业务服务通过微服务架构组织服务单元，每一个业务服务都会注册到服务中心，在调用其他业务服务时也是通过服务中心的服务发现机制去获取某一特定服务的具体提供实例列表，通过客户端路由方式来决定具体调用哪个服务提供者，从而既保证服务可靠性，又能提高系统吞吐量。平台提供代码管理、代码浏览、代码评审、代码搜索、代码扫描等业务组件。

6. Git集群

Git集群是平台的最核心、最基础的部分。为了保证Git集群的安全、高可用和高性能，平台提供了如下能力：

6-1. 同时支持数据的软实时和硬实时备份能力。

6-2. 提供三重备份，每一份代码至少有三份拷贝。

6-3. 提供根据不同的分片策略对数据分片的能力，支持Git集群动态扩容。

6-4. 提供读写分离的能力，支持一主多备。

6-5. 提供HA方案，支持自动failover。

7. 基础服务

平台依赖数据库、索引、缓存、用户管理、通知、存储等多个基础服务。这些基础服务在保障自身可用性的同时，提高了平台整体可靠性。

企业级SaaS服务下代码管理架构实践

经过一系列的架构改进，代码开发协作平台的容量、性能、可靠性都已经得到提升和验证。随着百度效率云产品对外服务，代码管理在其他方面遇到了更大的挑战。

1. 安全性，代码是企业核心资产，只有确保企业代码不泄漏、不丢失才能赢得企业的信任。
2. 容量和性能的要求更高，对外服务以后会有更多的用户，更多的用户就意味着更多的Repositories和更高的并发需要平台支撑。
3. 弹性伸缩的需求，因为企业不同发展阶段，对代码库容量和性能的需求是不同的，那就需要实现弹性伸缩以满足企业对资源的变化需求。
4. 自动化运维主要考虑两个方面，能够支持企业快速接入，可以方便大规模集群管理。

结合以上所述企业级SaaS对代码管理的要求，我们提出了企业专属云方案，如图11所示。

图11 专属云

我们主要从三个方面实现企业专属云：

1. 接入层，通过统一的代理服务，集成安全认证模块，支持企业账号接入实现企业请求隔离。
2. 应用层，采用共享服务的方式，通过统一的访问控制层实现企业隔离。
3. 对于企业的核心资产（如：代码库、产品库等）我们在资源层做隔离，不同的企业服务运行在不同的资源上，实现真正地物理隔离。

围绕着企业专属云方案，我们在架构设计上加强了多租户的管理、资源的管理，如图12所示。

图12 企业级SaaS服务下代码管理架构

在多租户管理方面，在统一的访问控制层增加了租户认证，所有请求都需要带上租户信息才可以通过认证。

在资源管理方面，同时支持Docker资源和虚拟机资源。企业接入时，Admin系统将自动从统一的资源池申请资源，通过Docker的方式完成自动化部署。我们同时支持混部和独立部署的方式，混部就是在同一个资源上部署多个企业的代码库实例。对于对代码安全有更高隔离要求的客户，我们将他们的服务独立部署在一台虚拟机上。

总结

百度代码开发协作平台使用微服务架构构建业务服务，一方面整合了现有的业务系统，另一方面提高了系统的稳定性和性能。使用数据分片和读写分离相结合的方式解决了代码库服务容量和性能的问题。使用专属云方案处理多租户的问题，帮助企业客户快速接入，实现资源隔离。但是，我们还有很多不足的地方有待提高和完善。比如，目前我们考虑到性能和开发成本的问题，选择了数据分片来扩容。但是，随着代码库容量的不断提升，数据分片带来的架构复杂、运维成本、性能瓶颈等问题也开始显现出来。读写分离和主备切换的方案，在高并发读的场景下工作尚可，但是面对高并发写的场景性能和可靠性就难以满足。

架构设计是和业务需求紧密相关的，只有合适的架构才是好的架构，因此，产品发展的不同阶段需要选择不同的技术架构方案。同时，一种可演进的架构是应对业务需求发展和变化的较优选择。

> **作者：**廖超超，百度代码开发协作平台架构师，加入百度之前曾就职于阿里巴巴。在高可用架构和分布式系统设计方面有一定经验，关注互联网大规模协同开发、持续集成和持续交付等业界优秀实践。
> **责编：**钱曙光，关注架构和算法领域，寻求报道或者投稿请发邮件qianshg@csdn.net，另有「CSDN 高级架构师群」，内有诸多知名互联网公司的大牛架构师，欢迎架构师加微信qianshuguangarch申请入群，备注姓名+公司+职位。
>

阅读原文