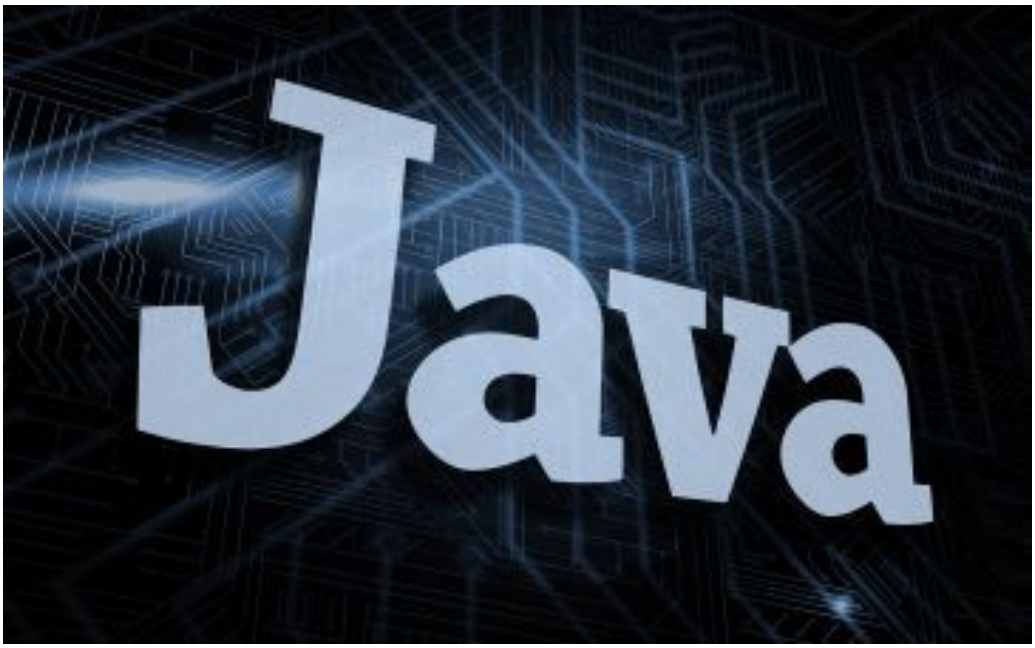


架构设计之高可用架构设计

欣然
java架构交流群：473984645

高可用架构设计总结：

前言:海恩法则和墨菲定律

海恩法则

> · 事故的发生是量的积累的结果。
> · 再好的技术、再完美的规章， 在实际操作层面也无法取代人自身的素质和责任心 。

墨菲定律

> · 任何事情都没有表面看起来那么简单 。
> · 所有事情的发展都会比你预计的时间长 。
> · 会出错的事总会出错。
> · 如果你担心某种情况发生，那么它更有可能发生 。

警示我们，在互联网公司里，对生产环境发生的任何怪异现象和问题 都不要轻易忽视，对于其背后的原因一定要彻查。同样，海恩法则也强调任何严重事故的背后 都是多次小问题的积累，积累到一定的量级后会导致质变，严重的问题就会浮出水面 。 那么，我们需要对线上服务产生的任何征兆，哪怕是一个小问题，也要刨根问底: 这就需要我们有技术攻关的能力，对任何现象都要秉着以下原则： 为什么发生？ 发生了怎么应对？ 怎么恢复？ 怎么避免？ 对问题要彻查，不能因为问题的现象不明显而忽略 。

1、可用性度量和考核

业务可用性：

所谓业务可用性(availability)也即系统正常运行时间的百分比，架构组最主要的 KPI (Key Performance Indicators，关键业绩指标)。对于我们提供的服务（web，api）来说，现在业界更倾向用 N 个9 来量化可用性， 最常说的就是类似“4个9(也就是99.99%)”的可用性。

描述	通俗叫法	可用性级别	年度停机时间
基本可用性	2个9	99%	87.6小时
较高可用性	3个9	99.9%	8.8小时
具有故障自动恢复能力的可用性	4个9	99.99%	53分钟
极高可用性	5个9	99.999%	5分钟

故障时间=故障修复时间点-故障发现（报告）时间点
服务年度可用时间%=（1-故障时间/年度时间）× 100%。

故障的度量与考核

对管理者而言：可用性是产品的整体考核指标。 每个工程师而言：使用故障分来考核：

类别	描述	权重
高危S级事故故障	一旦出现故障，可能会导致服务整体不可用	100
严重A级故障	客户明显感知服务异常：错误的回答	20
中级B级故障	客户能够感知服务异常：响应比较慢	5
一般C级故障	服务出现短时间内抖动	1

考核指标：故障分=故障时间分钟* 故障级别权重。

服务级别可用性：

如果是一个分布式架构设计，系统由很多微服务组成，所有的服务可用性不可能都是统一的标准。

为了提高我们服务可用性，我们需要对服务进行分类管理并明确每个服务级别的可用性要求。

类别	服务	可用性要求	描述
一级核心服务	核心产品或者服务	99.99%（全年53分钟不可用）	系统引擎部分：一旦出现故障，整个系统瘫痪
二级重要服务	重要的产品功能	99.95%（全年260分钟不可用）	类比汽车轮子：该服务出现问题，该重要功能不可用。
三级一般服务	一般功能	99.9%（全年8.8小时不可用）	类比汽车倒车影像：该部分出现问题，稍微影响用户体验
四级工具服务	工具类是服务	99%	非业务功能：比如爬虫、管理后台、运维工具

2、典型架构分层设计

典型架构分层设计如下：按照功能处理顺序划分应用，这是面向业务深度的划分。

每个公司的架构分层可能不一样，但是目的都是为了统一架构术语，方便团队内部沟通。

接入层：主要流量入口，经过简单

应用层：直接对外提供产品功能，例如网站、API接口等。应用层不包含复杂的业务逻辑，只做呈现和转换。

服务层：根据业务领域每个子域单独一个服务，分而治之。数据层：数据库和NoSQL，文件存储等。

我们先列出目前我们系统有哪些环节,每个环节是否薄弱. 客户端访问服务器端，经过很多环节，任何环节出问题，都不能访问：

接入层：

1、dns被劫持：域名是否使用https。

2、黑客攻击：是否有弱密，服务器权限，数据库权限

3、ddos攻击：是否有必要使用高防IP接入流量。

4、CC攻击：免费和收费版域名分开，网关是否有限流和防刷措施。

应用层：

1、应用服务器宕机。

2、应用服务bug。

3、第三方服务不可用。

服务层：

1、服务不可用或者出现bug

2、第三方服务不可用。

数据层：

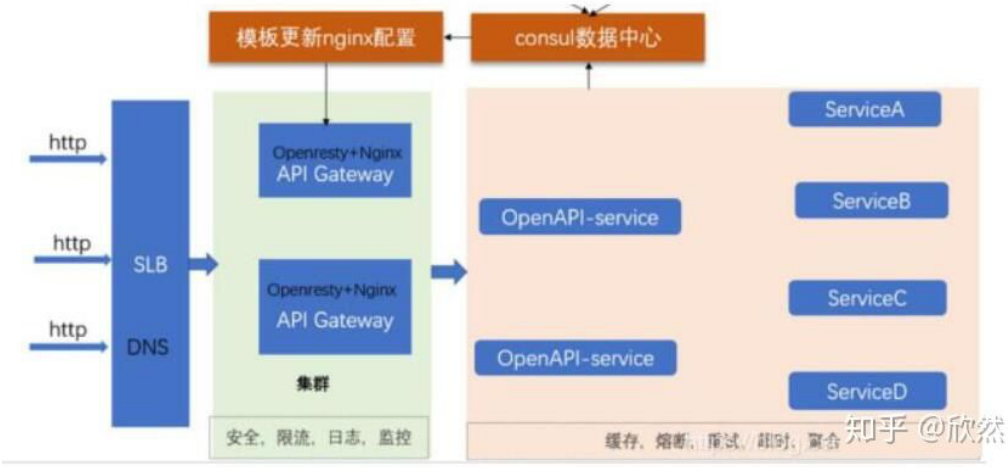
1、数据库服务器磁盘损坏导致数据库不可用等

3、接入层高可用设计

在接入层，这里主要是架构运维的高可用要求的事项：

1、 域名规范解析和规范化管理，应该制定《域名规范管理说明》，例如根据产品重要等级，制定使用高防ip的策略。
2、 规范API管理。
3、 明确各个API限流和防刷策略。

因此我们设计接入层架构：



目前我们对外的接口繁多，同时不同的项目不同的接口，没有一个统一管理的系统，也不方便监控和

跟踪 api 的健康状况。因此搭建我们 api 网关，方便 api 日常管理，包括控版本管理，升级，回滚。同时提供调试工具，方便开发人员， qa 调试和测试。 更重要的是 api 网关起到限流防刷（CC攻击）作用，保护后端服务。

4、应用层高可用设计

应用层设计主要原则：

1、可以水平扩展：通过接入层的负载均衡，实现故障自动转移。

2、无状态设计：无状态的系统更利于水平扩展，更利于做负载均衡。

状态是系统的吞吐量、易用性、可用性、性能和可扩展性的大敌，要尽最大可能避免。

3、回滚设计 ： 确保系统可以向后兼容，如果应用服务上线后出现bug，可以紧急回滚。

4、灰度发布：结合接入层设计A/B 功能，实现灰度发布，比如按ip，请求参数等分发流量。

5、服务层高可用设计

服务层设计最主要原则：服务分级管理

线上有很多服务，每个服务的可用性要求不一样，我们需要先这些服务做分级。

1、各级服务的部署原则：核心服务：独立服务器且N+1部署。三级和四级服务可以共享服务器部署。

2、各级服务上线发布原则：核心和重要服务：晚上12点上线。，三级和四级随时可上线

3、各级服务监控原则

一级核心服务：

定义：

可用性:99.99%，极高可用性，全年53分钟不可用。条件：

1、服务自身可用性：99.99%。

2、依赖数据资源服务可用性要求：（应用服务研发方自定义）。

3、依赖第三方服务可用性要求：（应用服务研发方自定义）。

4、需要部署的服务器数：N台。

服务设计满足以下原则：

1、冗余N+1部署：故障自动转移到多部署一个节点，避免单点问题。

2、可监控：服务流量预警、端口存活、进程占用的资源、服务接口功能逻辑是否正常，应用FGC等情况。

3、可回滚、灰度：灰度部署服务，部署的服务出现问题可快速回滚。

4、可独立部署：可以直接在运维平台打包部署，而不需要依赖其他服务部署完成后才能部署运行。

5、可独立测试：可以单独测试。

6、水平扩展：流量激增可快速扩容。

7、异步设计：服务需要通知第三方服务，必须通过消息队列进行异步方式完成。

8、幂等设计：服务可以重复调用，不影响结果。

7、可容错：自身有容错和修复能力：

1)、隔离手段：服务使用的资源（CPU、线程、IO等）隔离，使用舱壁模式；

2)、自我保护手段：快速失败(failfast)、流控、超时、熔断；

3)、失效转移或恢复手段：失效检测、重试、转移(failover)、回退恢复（failback)；

4)、降级手段：依据依赖服务的重要性或依赖程度（强、弱）,同步变异步，降级开关、拒绝部分服务等。

二级重要服务：

定义：

可用性99.95%（故障具备自动恢复的能力，全年260分钟不可用）。条件：

1、服务自身可用性：99.95%。

2、依赖数据资源服务可用性要求：（应用服务研发方自定义）。

3、依赖第三方服务可用性要求：（应用服务研发方自定义）。

4、需要部署的服务器数：N台。

服务设计满足以下原则：

1、冗余N+1部署：故障自动转移到多部署一个节点，避免单点问题。

2、可监控：监控进程、端口存活、进程占用的资源，应用FGC等。

3、可回滚、灰度：灰度部署服务，部署的服务出现问题可快速回滚。

4、故障隔离：服务器只部署唯一该应用服务，该应用服务出现问题，只影响自身服务问题。

5、可独立部署：可以直接在运维平台打包部署，而不需要依赖其他服务部署完成后才能部署运行。

6、可独立测试：可以单独测试。

7、水平扩展：流量激增可快速扩容。

8、可容错：自身有容错和修复能力。

三级一般服务：

定义：

可用性99.9%（较高可用性，全年260分钟不可用）。条件：

1、服务自身可用性：99.95%。

2、依赖数据资源服务可用性要求：（应用服务研发方自定义）。

3、依赖第三方服务可用性要求：（应用服务研发方自定义）。

4、需要部署的服务器数：N台。服务设计满足以下原则：

1、冗余N+1部署：可以单点部署。

2、可监控：可监控服务进程、端口存活是否正常。

3、可回滚、灰度：灰度部署服务，部署的服务出现问题可快速回滚。

4、故障隔离：一个服务器上可以部署多个应用，但保证服务器资源充足。

5、可独立部署：需要独立部署。

6、可独立测试：可以单独测试。

7、水平扩展：流量激增可快速扩容。

8、可容错：需要具备一般的容错能力。

四级工具服务：

定义：

可用性99.9%（全年8.8小时分钟不可用）。条件：

1、服务自身可用性：99.9%。

2、依赖数据资源服务可用性要求：（应用服务研发方自定义）。

3、依赖第三方服务可用性要求：（应用服务研发方自定义）。

4、需要部署的服务器数：N台。

服务设计满足以下原则：

1、冗余N+1部署：可以单点部署，只要有个进程存活就可以。

2、可监控：不需要监控。

3、可回滚、灰度：只要部署成功就可以。

4、故障隔离：哪个服务器有资源就可以部署。

5、可独立部署：不用考虑。

6、可独立测试：不用考虑。

7、水平扩展：不用考虑。

8、可容错：不用考虑。

6、高可用的数据库架构

数据架构设计原则

7、高质量的服务管理

1、规范服务管理：CMDB对项目、服务、服务器进行统一管理。

2、自动化发布：发布不影响用户，完善发布流程，自动化发布，可以及时回滚 。

3、自动化测试： 上线完成后进行全面自动化测试。

4、性能压测：通过对服务压测，了解服务可以承载并发能力，以致可以让运维通过预警进行服务器扩容 。

5、代码控制：测试环境使用测试分支，beta环境发布tag，线上使用该tag发布。

6、发布流程：规范上线发布流程。

7、灰度发布：灰度发布服务 。

8、应急处理机制 。

8、完善的监控告警机制

在高可用服务设计章节提到，核心服务可以监控：服务流量预警、端口存活、进程占用的资源、服务接口功能逻辑是否正常，应用FGC等情况，需要一个完善监控告警机制，并在告警后，通过一定的策略进行处理，以致服务可以快速恢复。例如，监控FGC，如果在一分钟内存出现10次FGC，自动重启服务。

1、网络流量监控 。

2、系统监控：服务器资源和网络相关监控（CPU、内存等）

3、日志监控：统一日志收集（各个服务）监控，跟踪(log2) 。

4、应用监控：端口存活、进程占用的资源，应用FGC等情况

5、业务监控 ： 服务接口功能逻辑是否正常

6、立体监控 监控数据采集后，除了用作系统性能评估、集群规模伸缩性预测等， 最终目标是还可以根据实时监控数据进行风险预警，并对服务器进行失效转移，自动负载调整，最大化利用集群所有机器的资源。

9、容灾

- 1、备份：数据备份（热备，冷备（冗余），异地）
- 2、过载保护：
- 3、同城多活-》异地多活
- 4、流量切换
- 5、重试，防雪崩（概率很小，成本很高）

10、职责

海恩法则提到：再好的技术、再完美的规章， 在实际操作层面也无法取代人自身的素质和责任心 。

因此要做到高可用的架构设计，职责也要清晰明确，要不然出现问题，相互推诿，问题解决进度很慢，会直接影响业务服务可用性。

1、架构师职责：

1、高可用架构设计：包括业务流程，模块划分组合，框架设计，流程纰漏，最后架构设计，技术实现步骤。 系统性的思考，权衡利弊，综合各种因素，设计出具有前瞻性的架构。

2、和运维协调沟通，提出高效的服务治理解决方案，把控服务质量管理。

3、协调沟通：开发之间沟通，产品之间沟通，市场沟通，运维沟通、沟通后产出图形化文档及设计。

4、规范和统筹：保证系统秩序，统一，规范，稳定，高效运行。

2、运维职责：

1)、熟悉系统技术架构，和架构师制定各种规范化要求。

2)、和架构师共同协调沟通，对系统架构提出可靠性，伸缩，扩展，数据库切分，缓存应用等解决方案。

3)、提供监控系统，自动化发布系统，代码管理，文档平台，自动运维平台等基础设施

4)、制定运维规范。

5)、建立运维安全体系。

6)、建立容灾备份体系。

3、研发职责：

- 1)、参与架构师的架构师设计，并根据设计实现具体细节。
- 2)、针对开发功能进行自测，压测。

- 3)、开发代码，使用工具或组件符合架构师制定规范。 包括代码规范、文档规范。
- 4)、代码部署符合运维部署规范要求。

我这儿整理了比较全面的JAVA相关的面试资料，

需要领取面试资料的同学，请加群：473984645

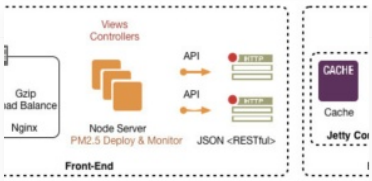
发布于 01-11

软件架构　高可用架构（书籍）　Java

文章被以下专栏收录

JAVA高级进阶　进入专栏

推荐阅读



架构设计原则整理

冯庆 发表于极简码农

什么才是真正的架构设计?

一. 什么是架构和架构本质在软件行业，对于什么是架构，都有很多的争论，每个人都有自己的理解。此君说的架构和彼君理解的架构未必是一回事。因此我们在讨论架构之前，我们先讨论架构的概念...

白衣少年

面试官：小伙子，说说你对分布式系统原理的看法吧

前程有光 发表于程序员的加...

阿
解

爪

还没有评论

写下你的评论...