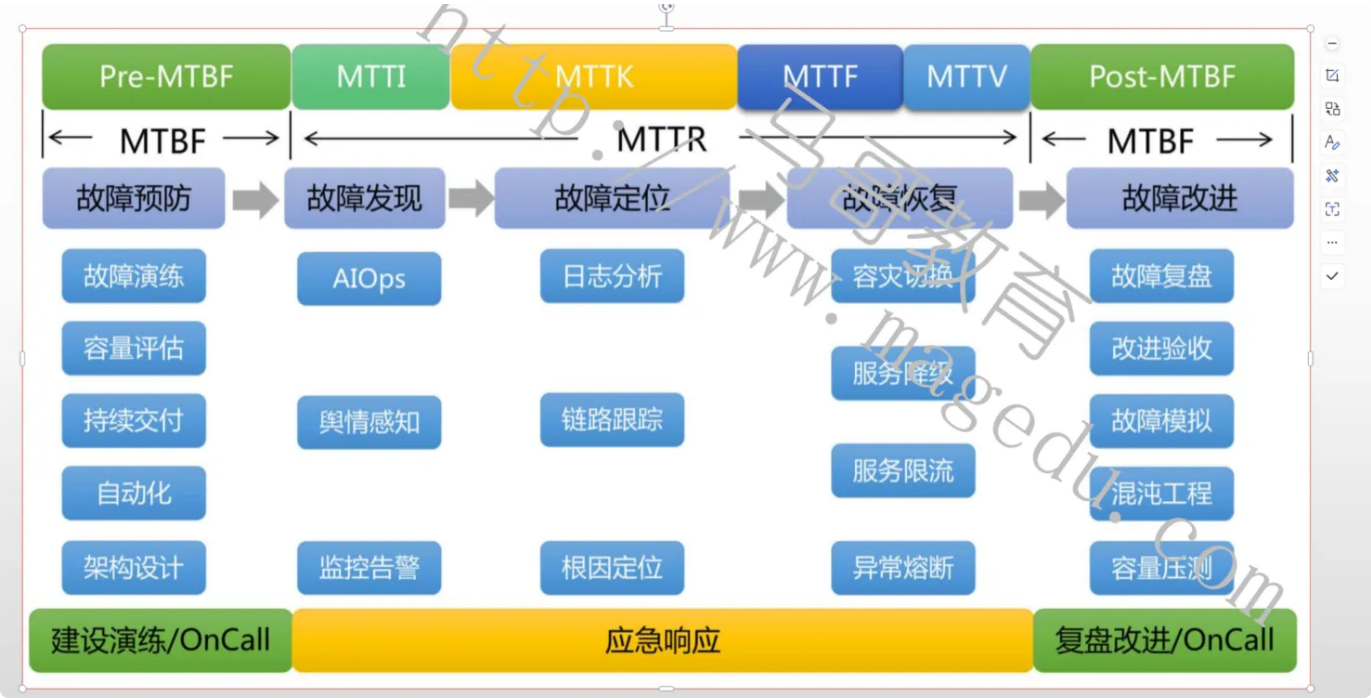
故障恢复的各阶段及应对措施 与MTTR

MTBF：Mean Time Between Failure，平均故障时间间隔，即平均无故障时间

MTTR：Mean Time To Repair，平均故障修复时长



MTTR的细分指标

MTTI：Mean Time To Identify，即平均故障发现时间

■从故障发生到SRE团队开始响应之间的时长

■该过程可能由用户或客服反馈、舆情监控或者监控告警所触发

MTTK：Mean Time to Know，平均故障获知时间

■更通俗的说法是，平均故障定位时间

■找出故障根因，完成故障定位

MTTF：Mean Time To Fix，平均故障修复时间

■从定位故障根因开始至业务恢复之间的时长

■对应的是解决问题的措施（例如限流、降级、熔断等）及效率

MTTV：Mean Time To Verify，平均故障验证时间

故障解决后，借助于用户反馈、监控指标等手段，来确认业务是否真 正恢复所消耗的时长

提升系统可用性

提升可用性的方式

■缩短MTTR指标的时长

■拉长MTBF指标的时长

措施方向

■Pre-MTBF阶段

◆在架构设计层面提升系统可用性，以冗余机制通过故障隔离降低MTTR：主 从模式、集群模式、多AZ（AvailabilityZone）的高可用、单元化、跨地域（Region）容灾、异地多活容灾等

◆在架构设计层面提升系统韧性：异步、重试、限流、降级、熔断、反压（降低下游效率进而迫使上游承压）

◆面向失败设计（Design-for-Failure），实现故障修复自动化

●常见的失败：导致崩溃的Bug、OMM、系统负载过高导致的夯死、系统级软件的问题，以及混部场景中的作业干扰等

●自动化修复是指在故障发性时自动触发事先准备的修复机制

◆借助于“混沌工程”模拟故障场景，并设计合理的应对举措

■Post-MTBF阶段

◆故障复盘，找出不足，落地改进措施

◆将更好地应对问题的预防机制常态化、例行化，并固化至日常操作流程中

■MTTR阶段

◆完善监控系统，提升故障告警的及时性和精准度，必要时可引入AIOps

服务的风险容忍度

为了辨别风险容忍度，SRE必须与产品负责人一起努力，将一组商业目标转化为明确可行的工程目标

消费者服务的风险容忍度

每个消费者服务通常有一个对应的产品团队及产品经理，这些产品经理负责了解用户和业务，以及在市场上塑造产品的定位，SRE应该同产品团队一同确立起相关产品的可用性标准

评估消费者服务的风险容忍度时需要考虑如下因素

- 需要的可用性水平
- 不同类型的失败（例如持续的低故障率或偶尔发生的全网故障）对服务是　否存在不同的影响
- 基于服务的成本来帮助在风险曲线上定位该服务

- 其它要考虑在内的重要指标

确立风险容忍度

■目标应该因服务而异，通常取决于服务提供的功能，以及服务在市场上的定位

- 用户期望的服务水平是什么?
- 该服务是否直接关系到收入（自身的收入或者客户的收入）？
- 是有偿服务，还是免费服务?
- 市场上竞争对手提供的服务水平如何?
- 该服务是ToB的业务，还是ToC的业务?

■成本是决定一项服务的可用性目标的一个关键因素

- 构建和运维可用性再多一个“9”的系统，收益会增加多少?
- 额外的收入是否能够抵消为了达到这一可靠水平所付出的成本

基础设施服务的风险容忍度

与消费者服务一个典型的不同是，基础设施服务可能会有多个存在不同需求的客户

■可用性目标水平

◆ 以某数据库服务为例，前端的消费者服务需要很低的延迟和较高的可靠性，而前端的离线分析服务则更关注吞吐量指标，这两类情形下的风险容忍度差别巨大

◆ 高可靠性的代价巨大，因而，分别使用两套各自独立的数据库系统（低延迟集群和高吞吐量集群）来应对这两种需求场景更为实际可行

■故障类型

◆低延迟的用户希望请求队列（几乎总是）为空，从而可以立刻处理每个到达的请求

◆高吞吐量的用户更感兴趣的是系统的吞吐量，因此希望请求队列永远不空

■成本

◆符合成本效益的条件下同时满足低延迟和高吞吐量两个竞争性约束的方式，就是分而治之

◆满足低延迟和高可靠的集群需要更高的冗余度，而高吞吐量的集群冗余度较低，利用率较高

基于错误预算管理风险

设立“错误预算”

■错误预算提供了一个明确的、客观的指标来决定服务在一个时间　　窗口内能接受多少不可靠性

■这能够排除SRE与产品研发部门谈判时的政治因素

■设立错误预算的方法：“错误预算= 1–SLO

◆由产品管理层设定一个SLO，从而确定该服务在指定时间窗口内的预计正常运行时间

◆由监控系统来判定服务的实际在线时间

◆存在未消耗的错误预算空间时，就可以发布新版本

需要建立的共识，以帮助确立错误预算

■软件对故障的容忍度

◆投入资源较少，故障率太高，程序过于脆弱；

◆投入资源太多，运行非常稳定，也未必意味着产品用户会很多

■测试

◆测试不充分，故障可能会频发，甚至出现严重问题，如泄露隐私数据

◆过于强调测试，可能会失去市场先机

发布频率

- 每一次的发布都是有风险的；团队应该在减少风险上投入多少时间?

- 金丝雀测试的持续时间和大小
  - 测试范围多大?
  - 要等多久?