什么是系统的鲁棒性?

嗨，你好啊，我是**猿java**

现实中，系统面临的异常情况和不确定性因素是不可避免的。例如，网络系统可能会遭受网络攻击、服务器宕机等问题；金融系统可能会受到市场波动、黑天鹅事件等因素的影响；自动驾驶系统可能会遇到天气恶劣、道路状况复杂等情况。

在这些情况下，系统的鲁棒性就显得尤为重要，它能够确保系统能够正确地处理各种异常情况，保持正常运行。因此，这篇文章我们将分析什么是系统的鲁棒性？如何保证系统的鲁棒性?

什么是系统的鲁棒性?

鲁棒性，英文为 `Robustness`，它是一个多学科的概念，涉及控制理论、计算机科学、工程学等领域。

在计算机领域，系统的鲁棒性是指系统在面对各种异常情况和不确定性因素时，仍能保持稳定运行和正常功能的能力。

鲁棒性是系统稳定性和可靠性的重要指标，一个具有良好鲁棒性的系统能够在遇到各种异常情况时做出正确的响应，不会因为某些异常情况而导致系统崩溃或失效。

鲁棒性要求系统在在遇到各种异常情况都能正常工作，各种异常很难具像化，这看起来是一种比较理想的情况，那么系统的鲁棒性该如何评估呢?

系统鲁棒性的评估

系统的鲁棒性可以从多个方面来考虑和评估，这里主要从三个方面进行评估：

首先，系统的设计和实现应该考虑到各种可能的异常情况，并采取相应的措施来应对

例如，在网络系统中，可以采用防火墙、入侵检测系统等技术来保护系统免受网络攻击；在金融系统中，可以采用风险管理技术来降低市场波动对系统的影响；在自动驾驶系统中，可以采用传感器融合、路径规划等技术来应对复杂的道路状况。

其次，系统在面临异常情况时应该具有自我修复和自我调整的能力

例如，当网络系统遭受攻击时，系统应该能够及时发现并隔离攻击源，同时自动恢复受影响的服务；当金融系统受到市场波动影响时，系统应该能够自动调整投资组合，降低风险；当自动驾驶系统面临复杂道路状况时，系统应该能够根据实时的道路情况调整行驶策略。

此外，系统的鲁棒性还包括对数据异常和不确定性的处理能力

在现实生活中，数据往往会存在各种异常情况，例如数据缺失、噪声数据等。系统应该能够对这些异常数据进行有效处理，保证系统的正常运行。同时，系统也应该能够对数据的不确定性进行有效处理，例如通过概率模型、蒙特卡洛方法等技术来处理数据不确定性，提高系统的鲁棒性。

鲁棒性的架构策略
========

对于系统的鲁棒性，有没有一些可以落地的策略?

如下图，展示了一些鲁棒性的常用策略，核心思想是：**事前-事中-事后**!

预防故障(事前)

对于技术人员来说，要有防范未然的意识，因此，对于系统故障要有预防措施，主要的策略包括：

* **代码质量**：绝大部分软件系统是脱离不了代码，因此代码质量是预防故障很核心的一个前提。
* **脱离服务**：脱离服务（Removal from service）这种策略指的是将系统元素临时置于脱机状态，以减轻潜在的系统故障。
* **替代**：替代（Substitution）这种策略使用更安全的保护机制–通常是基于硬件的–用于被视为关键的软件设计特性。
* **事务**：事务（Transactions）针对高可用性服务的系统利用事务语义来确保分布式元素之间交换的异步消息是原子的、一致的、隔离的和持久的。这四个属性被称为“ACID属性”。
* **预测模型**：预测模型（Predictive model.）结合监控使用，用于监视系统进程的健康状态，以确保系统在其标称操作参数内运行，并在检测到预测未来故障的条件时采取纠正措施。
* **异常预防**：异常预防（Exception prevention）这种策略指的是用于防止系统异常发生的技术。
* **中止**：如果确定某个操作是不安全的，它将在造成损害之前被中止（Abort）。这种策略是确保系统安全失败的常见策略。
* **屏蔽**：系统可以通过比较几个冗余的上游组件的结果，并在这些上游组件输出的一个或多个值不同时采用投票程序，来屏蔽（Masking）故障。
* **复盘**：复盘是对事故的整体分析，发现问题的根本原因，查缺补漏，找到完善的方案。

检测故障(事中)

当故障发生时，在采取任何关于故障的行动之前，必须检测或预测故障的存在，故障检测策略主要包括：

* **监控**：监控（Monitor）是用于监视系统的各个其他部分的健康状态的组件：处理器、进程、输入/输出、内存等等。
* **Ping/echo：**Ping/echo是指在节点之间交换的异步请求/响应消息对，用于确定通过相关网络路径的可达性和往返延迟。
* **心跳**：心跳（Heartbeat）是一种故障检测机制，它在系统监视器和被监视进程之间进行周期性的消息交换。
* **时间戳**：时间戳（Timestamp）这种策略用于检测事件序列的不正确性，主要用于分布式消息传递系统。
* **条件监测**：条件检测（Condition monitoring.）这种策略涉及检查进程或设备中的条件或验证设计过程中所做的假设。
* **合理性检查**：合理性检查（Sanity checking）这种策略检查特定操作或计算结果的有效性或合理性。
* **投票**：投票（Voting）这种策略的最常见实现被称为三模块冗余（或TMR），它使用三个执行相同操作的组件，每个组件接收相同的输入并将其输

出转发给投票逻辑，用于检测三个输出状态之间的任何不一致。
* **异常检测**：异常检测（Exception detection）这种策略用于检测改变执行正常流程的系统状态。
* **自检测**：自检测（Self-test）要求元素（通常是整个子系统）可以运行程序来测试自身的正确运行。自检测程序可以由元素自身启动，或者由系统监视器不时调用。

故障恢复(事后)

故障恢复是指系统出现故障之后如何恢复工作。这是对团队应急能力的一个极大考验，如何在系统故障之后，将故障时间缩小到最短，将事故损失缩减到最小？这往往决定了一个平台，一个公司的声誉，决定了很多技术人员的去留。故障恢复的策略主要包括：

* **冗余备用**：冗余备用（Redundant spare）有三种主要表现形式：主动冗余（热备用）、被动冗余（温备用）和备用（冷备用）。
* **回滚**：回滚（Rollback）允许系统在检测到故障时回到先前已知良好状态，称为“回滚线”——回滚时间。
* **异常处理**：异常处理（Exception handling）要求在检测到异常之后，系统必须以某种方式处理它。
* **软件升级**：软件升级（Software upgrade）的目标是在不影响服务的情况下实现可执行代码映像的在线升级。
* **重试**：重试（Retry）策略假定导致故障的故障是暂时的，重试操作可能会取得成功。
* **忽略故障行为**：当系统确定那些消息是虚假的时，忽略故障行为（Ignore faulty behavior）要求忽略来自特定来源的消息。
* **优雅降级**：优雅降级（Graceful degradation）这种策略在元素故障的情况下保持最关键的系统功能，放弃较不重要的功能。
* **重新配置**：使用重新配置（Reconfiguration），系统尝试通过将责任重新分配给仍在运行的资源来从系统元素的故障中恢复，同时尽可能保持关键功能。

上述这些策略看起来很高大上，好像离你很远，但是其实很多公司都有对应的措施，比如：系统监控，系统告警，数据备份，分布式，服务器集群，多活，降级策略，熔断机制，复盘等等，这些术语应该就和我们的日常开发息息相关了。

总结
==

系统的鲁棒性是指系统在面对各种异常情况和不确定性因素时，仍能保持稳定

运行和正常功能的能力。系统鲁棒性看似一个理想的状态，却是业界一直追求的终极目标，比如，系统稳定性如何做到 5个9（99.999%），甚至是 6个9（99.9999%），这就要求技术人员时刻保持工匠精神、在自己的本职工作上多走一步，只有在各个相关岗位的共同协作下，才能确保系统的鲁棒性。

学习交流
=====

如果你觉得文章有帮助，请帮忙点个赞呗，公众号：`猿java`，持续输出硬核文章。

原文链接: https://juejin.cn/post/7393312386571370536