# 分布式系统核心功能

* 配置管理
* 服务治理
* 负载均衡
* 日志中心
* 动态扩容
* 数据分片
* 调用链路追踪

## 配置管理

* 本地配置，使用properties、yml等
* 配置中心，nacos、GitConfig、Apollo等

配置中心需要考虑的问题：

1. 读取性能
2. 变更实时性
3. 分布式环境数据一致性
4. 本地缓存
5. 高可用

## 服务治理

服务治理包括：

* 服务发现：一般使用注册中心实现，比如zookeeper、Eureka、consul
* 负载均衡
* 限流
* 熔断
* 超时
* 重试
* 服务追踪

## 负载均衡

## 日志中心

## 动态扩容

## 数据分片

## 调用链路追踪

# 分布式一致性

# 服务降级与熔断

## 基本概念

### 服务降级

服务的访问压力剧增的时候，根据当前的业务情况以及流量对一些服务和页面有策略的降级，帮助缓解服务的压力，保证服务能够正常运行不受影响（这里的服务包括应用和数据等）。同时也得保证部分甚至是大部分客户能得到正确的响应，至于请求处理不了或出错的，给一个默认的响应即可。

举个比较简单的例子，当服务器的TCP请求数量大于500的时候，大于500的请求会直接返回提示信息--服务繁忙请稍后再试，500以内的请求正常跑即可。

### 服务熔断

在股票市场，熔断这个词大家都不陌生，是指当股指波幅达到某个点后，交易所为控制风险采取的暂停交易措施。相应的，服务熔断一般是指软件系统中，由于某些原因使得服务出现了过载现象，为防止造成整个系统故障，从而采用的一种保护措施，所以很多地方把熔断亦称为过载保护。

## 降级分类

按照是否自动化：自动开关降级、人工开关降级

按照功能分类：读服务降级、写服务降级

按照系统层级降级分类：多级降级

### 自动降级分类

（1）、超时降级：主要配置好超时时间和超时重试次数和机制，并使用异步机制探测回复情况

（2）、失败次数降级：主要是一些不稳定的api，当失败调用次数达到一定阀值自动降级，同样要使用异步机制探测回复情况

（3）、故障降级：比如要调用的远程服务挂掉了（网络故障、DNS故障、http服务返回错误的状态码、rpc服务抛出异常），则可以直接降级。降级后的处理方案有：默认值（比如库存服务挂了，返回默认现货）、兜底数据（比如广告挂了，返回提前准备好的一些静态页面）、缓存（之前暂存的一些缓存数据）

（4）、限流降级

当我们去秒杀或者抢购一些限购商品时，此时可能会因为访问量太大而导致系统崩溃，此时开发者会使用限流来进行限制访问量，当达到限流阀值，后续请求会被降级；降级后的处理方案可以是：排队页面（将用户导流到排队页面等一会重试）、无货（直接告知用户没货了）、错误页（如活动太火爆了，稍后重试）。

## 服务熔断和服务降级比较

两者其实从有些角度看是有一定的类似性的：

目的很一致，都是从可用性可靠性着想，为防止系统的整体缓慢甚至崩溃，采用的技术手段；

最终表现类似，对于两者来说，最终让用户体验到的是某些功能暂时不可达或不可用；

粒度一般都是服务级别，当然，业界也有不少更细粒度的做法，比如做到数据持久层（允许查询，不允许增删改）；

自治性要求很高，熔断模式一般都是服务基于策略的自动触发，降级虽说可人工干预，但在微服务架构下，完全靠人显然不可能，开关预置、配置中心都是必要手段；

而两者的区别也是明显的：

触发原因不太一样，服务熔断一般是某个服务（下游服务）故障引起，而服务降级一般是从整体负荷考虑；

管理目标的层次不太一样，熔断其实是一个框架级的处理，每个微服务都需要（无层级之分），而降级一般需要对业务有层级之分（比如降级一般是从最外围服务开始）

实现方式不太一样

## 服务降级需要考虑的问题

1.降级的服务是否是核心服务

2.是否支持降级，降级的策略是什么

3.业务场景

## Hystrix

该库旨在通过控制那些访问远程系统、服务和第三方库的节点，从而对延迟和故障提供更强大的容错能力。Hystrix具备拥有回退机制和断路器功能的线程和信号隔离，请求缓存和请求打包（request collapsing，即自动批处理，译者注），以及监控和配置等功能。

