背景

JD.COM 京东

- 随着业务复杂度的增加，依赖的服务也逐步增加，服务调用出现异常问题几率也会增加：
  - 系统依赖的某个服务发生延迟或者故障，数秒内导致所有应用资源（线程，队列等）被耗尽，造成所谓的雪崩效应 (Cascading Failure)，导致整个系统拒绝对外提供服务。
  - 对下游依赖服务没做好限速处理，最终导致下游服务崩溃。
- 容错是一个很大的话题，受篇幅所限，将介绍仅限定在服务调用间常用的一些容错模式。

原则- Design for Failure

JD.COM 京东

服务容错的设计有个基本原则，就是“Design for Failure”。为了避免出现“千里之堤溃于蚁穴”这种情况，在设计上需要考虑到各种边界场景和对于服务间调用出现的异常或延迟情况，同时在设计和编程时也要考虑周到。这一切都是为了达到以下目标：

- 一个依赖服务的故障不会严重破坏用户的体验。
- 系统能自动或半自动处理故障，具备自我恢复能力。

基于这个原则和目标，衍生出下文将要介绍的一些模式，能够解决分布式服务调用中的一些问题，提高系统在故障发生时的存活能力。

场景

JD.COM 京东

- 系统整体负荷较高
- 需要优先保证重要的服务
- 系统资源耗尽过快，出现拒绝对外提供服务。

服务降级(gracefully degrade)

JD.COM 京东

- 需要针对业务进行服务分级

  比如分为一级服务，二级服务，三级服务

  紧急情况下，务必保证一级服务的稳定性，然后可以牺牲二级三级服务

场景

JD.COM 京东

- 依赖服务出现建立网络连接或响应延迟
- 调用方无限等待，不能及时释放关键资源，如Web容器的连接数，数据库连接数等
- 整个系统资源耗尽出现拒绝对外提供服务这种情况。

超时（Timeout）模式

JD.COM 京东

- 超时模式，是一种最常见的容错模式。常见的有设置网络连接超时时间，一次RPC的响应超时时间等。在分布式服务调用的场景中，它主要解决了当依赖服务出现建立网络连接或响应延迟，不用无限等待的问题，调用方可以根据事先设计的超时时间中断调用，及时释放关键资源，如Web容器的连接数，数据库连接数等，避免整个系统资源耗尽出现拒绝对外提供服务这种情况。

场景

JD.COM 京东

- 对于下游服务的数据强依赖的场景
- 网络抖动等导致服务调用出现超时

重试（Retry）

JD.COM 京东

- 重试模式，一般和超时模式结合使用，适用于对于下游服务的数据强依赖的场景（不强依赖的场景不建议使用！），通过重试来保证数据的可靠性或一致性，常用于因网络抖动等导致服务调用出现超时的场景。与超时时间设置结合使用后，需要考虑接口的响应时间分布情况，超时时间可以设置为依赖服务接口99.5%(tp99)响应时间的值，重试次数一般1-2次为宜，否则会导致请求响应时间延长，拖累到整个系统。

场景

JD.COM 京东

- 下游服务容量有限
- 出现突发流量猛增（节假日大促等）而导致下游服务因压力过大而拒绝服务。

限流(Rate Limiting)

JD.COM 京东

限流模式，常用于下游服务容量有限，但又怕出现突发流量猛增而导致下游服务因压力过大而拒绝服务的场景。常见的限流模式如下：

- 控制并发，限制并发的数量
- 控制速率，限制并发访问的速率。

场景

JD.COM 京东

- 服务由于网络连接超时
- 系统有异常或load过高出现暂时不可用
- 可能需要一段时间才能修复
- 这种对请求的阻塞可能会占用宝贵的系统资源，如：内存，线程，数据库连接等等
- 最坏的情况下会导致这些资源被消耗殆尽，使得系统里不相关的部分所使用的资源也耗尽从而拖累整个系统。

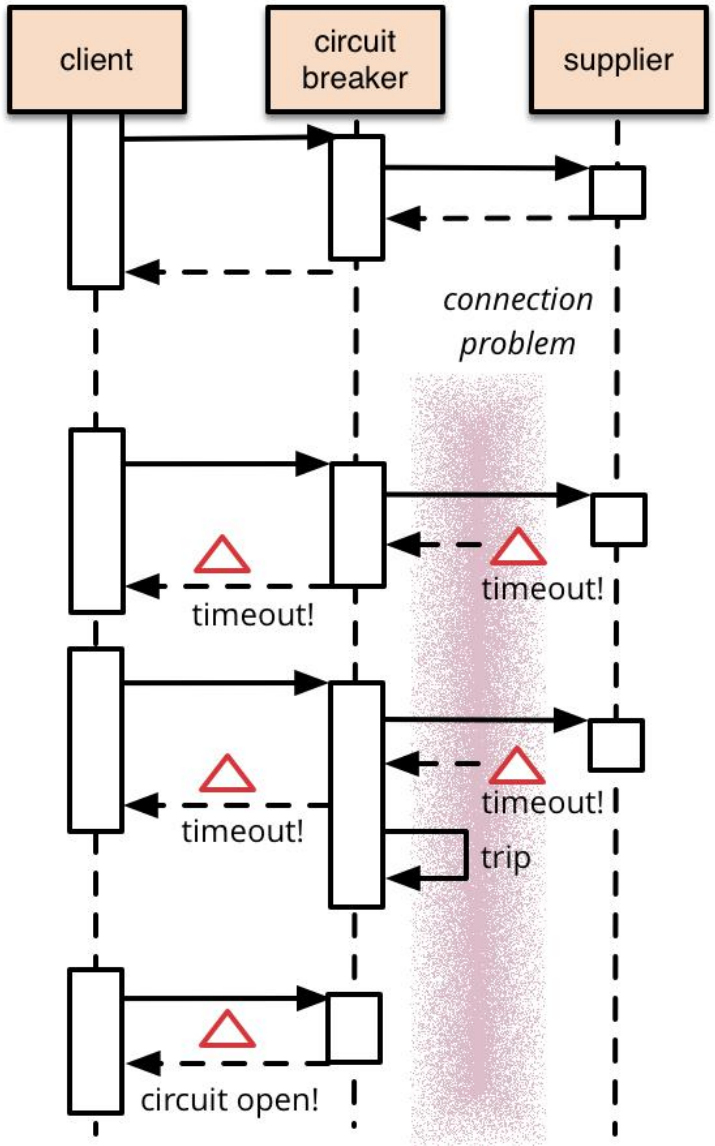
电路熔断器(Circuit Breaker)

JD.COM 京东

- 一些服务由于网络连接超时，系统有异常或load过高出现暂时不可用等情况，导致对这些服务的调用失败，可能需要一段时间才能修复，这种对请求的阻塞可能会占用宝贵的系统资源，如：内存，线程，数据库连接等等，最坏的情况下会导致这些资源被消耗殆尽，使得系统里不相关的部分所使用的资源也耗尽从而拖累整个系统。在这种情况下，调用操作能够立即返回错误而不是等待超时的发生或者重试可能是一种更好的选择，只有当被调用的服务有可能成功时我们再去尝试。防止我们的系统不断地尝试执行可能会失败的调用，使得我们的系统继续执行而不用等待修正错误，或者浪费CPU时间去等到长时间的超时产生。熔断器模式也可以使我们系统能够检测错误是否已经修正，如果已经修正，系统会再次尝试调用操作。

电路熔断器(Circuit Breaker)

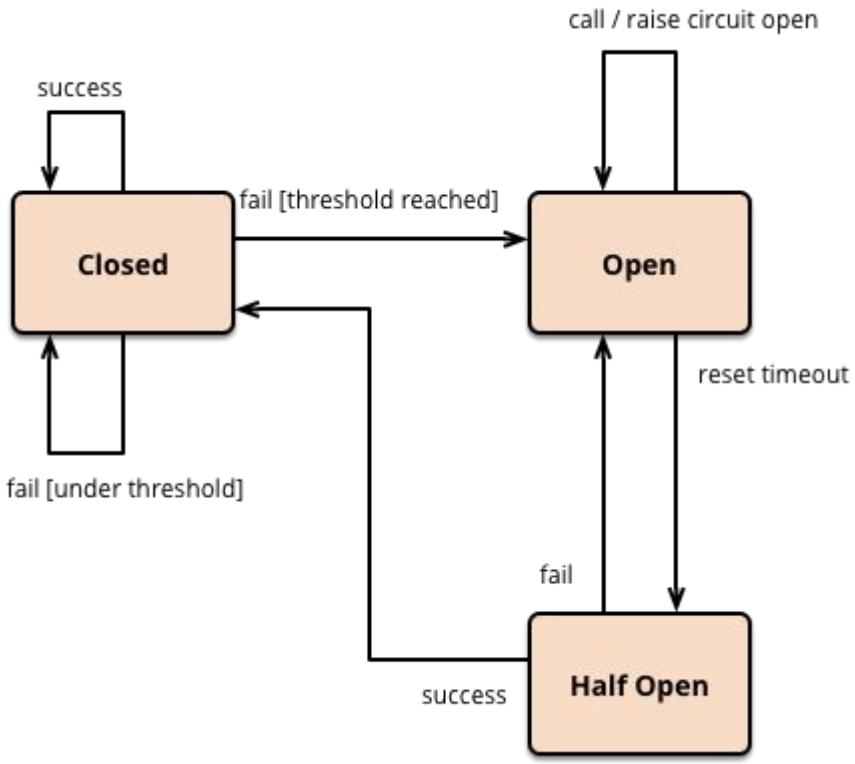
JD.COM 京东



电路熔断器(Circuit Breaker)

JD.COM 京东

- 在熔断器内部，往往有以下几种状态：



电路熔断器(Circuit Breaker)

JD.COM 京东

- 闭合（closed）状态：该状态下能够对目标服务或方法进行正常的调用。熔断器类维护了一个时间窗口内调用失败的次数，如果某次调用失败，则失败次数加1。如果最近失败次数超过了在给定的时间窗口内允许失败的阈值(可以是数量也可以是比例)，则熔断器类切换到断开(Open)状态。此时熔断器设置了一个计时器，当时钟超过了该时间，则切换到半断开（Half-Open）状态，该睡眠时间的设定是给了系统一次机会来修正导致调用失败的错误。
- 断开(Open)状态：在该状态下，对目标服务或方法的请求会立即返回错误响应，如果设置了fallback方法，则会进入fallback的流程。
- 半断开（Half-Open）状态：允许对目标服务或方法的一定数量的请求可以去调用服务。如果这些请求对服务的调用成功，那么可以认为之前导致调用失败的错误已经修正，此时熔断器切换到闭合状态（并且将错误计数器重置）；如果这一定数量的请求有调用失败的情况，则认为导致之前调用失败的问题仍然存在，熔断器切回到断开方式，然后开始重置计时器来给系统一定的时间来修正错误。半断开状态能够有效防止正在恢复中的服务被突然而来的大量请求再次拖垮。

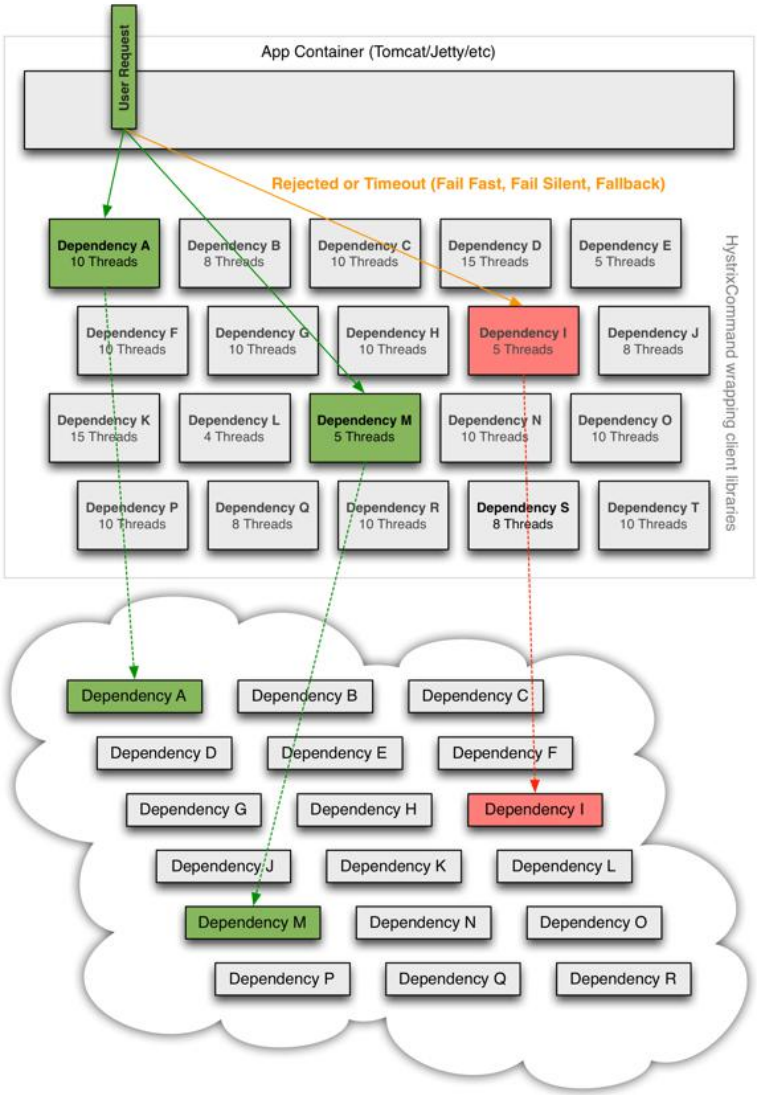
场景

JD.COM 京东

- 如某接口由于数据库慢查询，外部RPC调用超时导致整个系统的线程数过高，连接数耗尽等
- 资源很快被慢的服务调用吃光，整个系统会全部慢下来，甚至出现系统停止响应的情况

舱壁隔离(Bulkhead Isolation)

JD.COM 京东



回退(Fallback)

JD.COM 京东

- 超时，重试失败，熔断或者限流发生的时候，为了及时恢复服务或者不影响到用户体验，需要提供回退的机制，常见的回退策略有：
- 自定义处理：在这种场景下，可以使用默认数据，本地数据，缓存数据来临时支撑，也可以将请求放入队列，或者使用备用服务获取数据等，适用于业务的关键流程与严重影响用户体验的场景，如商家/产品信息等核心服务。
- 故障沉默（fail-silent）：直接返回空值或缺省值，适用于可降级功能的场景，如产品推荐之类的功能，数据为空也不太影响用户体验。
- 快速失败（fail-fast）：直接抛出异常，适用于数据非强依赖的场景，如非核心服务超时的处理。

回退(Fallback)

JD.COM 京东

- 超时，重试失败，熔断或者限流发生的时候，为了及时恢复服务或者不影响到用户体验，需要提供回退的机制，常见的回退策略有：
- 自定义处理：在这种场景下，可以使用默认数据，本地数据，缓存数据来临时支撑，也可以将请求放入队列，或者使用备用服务获取数据等，适用于业务的关键流程与严重影响用户体验的场景，如商家/产品信息等核心服务。
- 故障沉默（fail-silent）：直接返回空值或缺省值，适用于可降级功能的场景，如产品推荐之类的功能，数据为空也不太影响用户体验。
- 快速失败（fail-fast）：直接抛出异常，适用于数据非强依赖的场景，如非核心服务超时的处理。

Netflix

JD.COM 京东