分布式系统原理

Lease机制

Lease 机制是最重要的分布式协议，广泛应用于各种实际的分布式系统中。即使在某些系统中相似的设计不被称为 lease，但我们可以分析发现其本质就是一种 lease 的实现。

本节从一个分布式cache 系统出发介绍最初的 lease 机制，接着加以引申，探讨 lease 机制的本质。最后介绍了 lease 机制最重要的应用：**判定节点状态**。

1. 基于 lease 的分布式 cache 系统

本节先通过讨论一种分布式 cache 系统的实例来介绍 lease 机制。Lease 机制最初也是被运用于这种系统。

基本的问题背景如下：在一个分布式系统中，有一个中心服务器节点，中心服务器存储、维护着一些数据，这些数据是系统的元数据。系统中其他的节点通过访问中心服务器节点读取、修改其上的元数据。由于系统中各种操作都依赖于元数据，如果每次读取元数据的操作都访问中心服务器节点，那么中心服务器节点的性能成为系统的瓶颈。为此，设计一种元数据 cache，在各个节点上cache 元数据信息，从而减少对中心服务器节点的访问，提高性能。另一方面，系统的正确运行严格依赖于元数据的正确，这就要求各个节点上 cache 的数据始终与中心服务器上的数据一致，cache中的数据不能是旧的脏数据。最后，设计的 cache 系统要能最大可能的处理节点宕机、网络中断等异常，最大程度的提高系统的可用性。

为此，利用 lease 机制设计一套 cache 系统，其基本原理为如下。中心服务器在向各节点发送数据时同时向节点颁发一个 lease。每个 lease 具有一个有效期，和信用卡上的有效期类似，lease 上的有效期通常是一个明确的时间点，例如 12:00:10，一旦真实时间超过这个时间点，则 lease 过期失效。这样 lease 的有效期与节点收到 lease 的时间无关，节点可能收到 lease 时该 lease 就已经过期失效。这里首先假设中心服务器与各节点的时钟是同步的，下节中讨论时钟不同步对 lease 的影响。中心服务器发出的 lease 的含义为：在 lease 的有效期内，中心服务器保证不会修改对应数据的值。因此，节点收到数据和 lease 后，将数据加入本地 Cache，一旦对应的 lease 超时，节点将对应的本地 cache数据删除。中心服务器在修改数据时，首先阻塞所有新的读请求，并等待之前为该数据发出的所有lease 超时过期，然后修改数据的值。

具体的服务器与客户端节点一个基本流程如下：

```
流程 2.3.1：基于 lease 的 cache，客户端节点读取元数据：
1.  判断元数据是否已经处于本地 cache 且 lease 处于有效期内
1.1 是：直接返回 cache 中的元数据
1.2 否：向中心服务器节点请求读取元数据信息
    1.2.1  服务器收到读取请求后，返回元数据及一个对应的 lease
    1.2.2  客户端是否成功收到服务器返回的数据
```

```
        1.2.2.1 失败或超时：退出流程，读取失败，可重试
        1.2.2.2 成功：将元数据与该元数据的 lease 记录到内存中，返回元数据
```

```
流程 2.3.2：基于 lease 的 cache，客户端节点修改元数据流程
1.  节点向服务器发起修改元数据请求。
2.  服务器收到修改请求后，阻塞所有新的读数据请求，即接收读请求，但不返回数据。
3.  服务器等待所有与该元数据相关的 lease 超时。
4.  服务器修改元数据并向客户端节点返回修改成功
```

- 上述机制可以保证各个节点上的 cache 与中心服务器上的中心始终一致。这是因为中心服务器节点在发送数据的同时授予了节点对应的 lease，在 lease 有效期内，服务器不会修改数据，从而客户端节点可以放心的在 lease 有效期内 cache 数据。

- 述基础流程有一些性能和可用性上的问题，但可以很容易就优化改性。优化点一：服务器在修改元数据时首先要阻塞所有新的读请求，造成没有读服务。进一步的优化是，当进入修改流程，服务器颁发的lease 有效期限选择为已发出的 lease 的最大有效期限。实际使用中，第一层优化就足够了。优化点二：服务器在修改元数据时需要等待所有的 lease 过期超时，从而造成修改元数据的操作时延大大增大。优化的方法是，在等待所有的 lease 过期的过程中，服务器主动通知各个持有lease的节点放弃lease并清除cache中的数据，如果服务器收到客户端返回的确认放弃lease的消息，则服务器不需要在等待该 lease 超时。

2. lease 机制的分析

首先给出本文对 lease 的定义：**Lease 是由颁发者授予的在某一有效期内的承诺**。

- 颁发者一旦发出 lease，则无论接受方是否收到，也无论后续接收方处于何种状态，只要 lease 不过期，颁发者一定严守承诺；另一方面，接收方在 lease的有效期内可以使用颁发者的承诺，但一旦 lease 过期，接收方一定不能继续使用颁发者的承诺。

- Lease 机制具有很高的容错能力。首先，通过引入有效期，Lease 机制能否非常好的容错网络异常。再者，Lease 机制能较好的容错节点宕机。最后，lease 机制不依赖于存储。

- Lease 机制依赖于有效期，这就要求颁发者和接收者的时钟是同步的。对于这种时钟不同步，实践中的通常做法是将颁发者的有效期设置得比接收者的略大，只需大过时钟误差就可以避免对 lease 的有效性的影响。

3. 基于 lease 机制确定节点状态

在分布式系统中确定一个节点是否处于正常工作状态是一个困难的问题。由于可能存在网络分化，节点的状态是无法通过网络通信来确定的。下面举一个较为具体的例子来讨论这个问题。

例 2.3.1：在一个 primary-secondary 架构的系统中，有三个节点 A、B、C 互为副本，其中有一个节点为 primary，且同一时刻只能有一个 primary 节点。另有一个节点 Q 负责判断节点 A、B、C 的状态，一旦 Q 发现 primary 异常，节点 Q 将选择另一个节点作为 primary。假设最开始时节点 A 为 primary，B、C 为 secondary。节点 Q 需要判断节点 A、B、C 的状态是否正常。

首先需要说明的是基于“心跳”(Heartbeat)的方法无法很好的解决这个问题。节点 A、B、C 可以周期性的向 Q 发送心跳信息，如果节点 Q 超过一段时间收不到某个节点的心跳则认为这个节点异常。

上述问题的出现的原因在于虽然节点 Q 认为节点 A 异常，但节点 A 自己不认为自己异常，依旧作为 primary 工作。其问题的本质是由于网络分化造成的系统对于“节点状态”认知的不一致。

上面的例子中的分布式协议依赖于对节点状态认知的全局一致性，即一旦节点 Q 认为某个节点A 异常，则节点 A 也必须认为自己异常，从而节点 A 停止作为 primary，避免“双主”问题的出现。解决这种问题有两种思路，第一、设计的分布式协议可以容忍“双主”错误，即不依赖于对节点状态的全局一致性认识，或者全局一致性状态是全体协商后的结果；第二、利用 lease 机制。

由中心节点向其他节点发送 lease，若某个节点持有有效的 lease，则认为该节点正常可以提供服务。用于例 2.3.1 中，节点 A、B、C 依然周期性的发送 heart beat 报告自身状态，节点 Q 收到 heart beat后发送一个 lease，表示节点 Q 确认了节点 A、B、C 的状态，并允许节点在 lease 有效期内正常工作。节点 Q 可以给 primary 节点一个特殊的 lease，表示节点可以作为 primary 工作。一旦节点 Q 希望切换新的 primary，则只需等前一个 primary 的 lease 过期，则就可以安全的颁发新的 lease 给新的primary 节点，而不会出现“双主”问题。

4. lease 的有效期时间选择

Lease 的有效期虽然是一个确定的时间点，当颁发者在发布 lease 时通常都是 *将当前时间加上一个固定的时长从而计算出 lease 的有效期*。

如何选择 Lease 的时长在工程实践中是一个值得讨论的问题。如果 lease 的时长太短，例如 1s，一旦出现网络抖动 lease 很容易丢失，从而造成节点失去 lease，使得依赖 lease 的服务停止；如果 lease 的时长太大，例如 1 分钟，则一旦接受者异常，颁发者需要过长的时间收回 lease 承诺。例如，使用 lease 确定节点状态时，若 lease 时间过短，有可能造成网络瞬断时节点收不到 lease 从而引起服务不稳定，若 lease 时间过长，则一旦某节点宕机异常，需要较大的时间等待 lease 过期才能发现节点异常。

工程中，常选择的 lease 时长是 **10 秒级别**，这是一个经过验证的经验值，实践中可以作为参考并综合选择合适的时长。

5. 工程投影

- GFS 中的 Lease。GFS 中使用 Lease 确定 Chuck 的 Primary 副本。Lease 由 Master 节点颁发给 primary 副本，持有Lease 的副本成为 primary 副本。

- Niobe 中的 Lease。Niobe 中虽然没有明确说明使用了 Lease 机制，但是通过分析可以发现，这是一个 Lease 机制。Niobe 协议中，也是通过 Lease 机制维持 Primary 副本的选择，不同的是 Niobe 中的 Lease 是由 Secondary 节点向 Primary 节点发送。在 Niobe 协议中，每个 Secondary 副本都会给 Primary 副本发送 Lease，这个 Lease 的含义是： 在 Lease 时间内，本副本承认你是 Primary 节点。从这里我们可以看到，niobe 中的 lease 含义也可以理解为：“我承诺在接下来 lease 时间内，我不会 kill 你”。

- Chubby 与 Zookeeper 中的 Lease。我们知道 chubby 通过 paxos 协议实现去中心化的选择 primary 节点（见 2.8.6 ）。一旦某个节点获得了超过半数的节点认可，该节点成为 primary 节点，其余节点成为 secondary 节点。Secondary节点向 primary节点发送 lease，该 lease 的含义是： “承诺在

lease 时间内，不选举其他节点成为 primary节点”。除了 secondary 与 primary 之间的 lease，在 chubby 中，primary 节点也会向每个 client 节点颁发lease。该 lease 的含义是用来判断 client 的死活状态，一个 client 节点只有只有合法的 lease，才能与 chubby 中的 primary 进行读写操作。与 Chubby 不同，Zookeeper 中的 secondary 节点（在 zookeeper 中称之为 follower）并不向 primary 节点（在 zookeeper 中称之为 leader）发送 lease，zookeeper 中的 secondary 节点如果发现没有 primary节点则发起新的 paxos 选举，只要 primary 与 secondary 工作正常，新发起的选举由于缺乏多数secondary的参与而不会成功。

- 间接使用 Lease。笔者很难想象，如何在工程上既不使用 Lease 而又实现一个一致性较高的系统。直接实现 lease机制的确会对增加系统设计的复杂度。然而，由于有类似 Zookeeper 这样的开源的高可用系统，在工程中完全可以间接使用 Lease。借助 zookeeper，我们可以简单的实现高效的、无单点选主、状态监控、分布式锁、分布式消息队列等功能，而实际上，这些功能的实现都是依赖于背后 zookeeper与 client 之间的 Lease 的。

参考：《分布式系统原理介绍》 - 刘杰