链滴

乐观锁

作者：ixiaozhi
原文链接：https://ld246.com/article/1501949089119
来源网站：链滴

假设有一个库存表为 table, 结构如下：

```
<table border=1>
<tr>
<td>列名</td><td>描述</td>
</tr>
<tr>
<td>ID</td><td>主键</td>
</tr>
<tr>
<td>number</td><td>数量</td>
</tr>
</table>
```

这里暂时不考虑服务器数量、页面 CDN 缓存、带宽等问题。我们常需要的操作是在 A 上同时增加 B C，要的结果是 ABC 或者 ACB，而不希望是 AB 或 AC。再其次之，如果结果是 AB，则需要给 C 返一个错误。

数据库乐观锁

数据库表中增加一个 version 字段，默认值为 0。在需要修改前，先查询 version，在修改时验证 vers on 且令 version = version + 1。在大多数的场合中，用来控制并发时的更新丢失是行之有效的手段。

所以我们可以用类似 update table set number = number - 1 where version = ? and number > 0 来操作数据库，基于判断 update 的条件来确定是否成功并保证 number 一直大于零。

memcached cas 锁 或者其他锁

Memecached 支持的多线程同步的方法有 Add命令、incr/decr命令、gets/cas操作。我们使用 gets cas 操作来进行同步锁。

Memcached 1.2.5 以及更高版本，提供了 gets 和 cas 命令。如果您使用 gets 命令查询某个 key 的 ache 会给您返回该 item 当前值的唯一标识。如果您覆写了这个 item 并想把它写回到 Memcached 中，您可以通过 cas 命令把那唯一标识送给 Memcached。如果该 item 存放在 Memcached 中的 一标识与您提供的一致，您的写操作将会成功。如果另一个进程在这期间也修改了这个 item，那么该 tem 存放在 Memcached 中的唯一标识将会改变，您的写操作就会失败。

示例代码如下：

```
    MemcachedClient cache = null;
    AuthDescriptor ad = new AuthDescriptor(new String[]{"PLAIN"}, new PlainCallbackHandl
r(username, password));

    try {
        cache = new MemcachedClient(

                new ConnectionFactoryBuilder().setProtocol(ConnectionFactoryBuilder.Protocol.B
NARY)
                        .setAuthDescriptor(ad)
                        .build(),
                AddrUtil.getAddresses(host + ":" + port));
    } catch (IOException e) {
```

```
        e.printStackTrace();
    }

    int y = 0;
    CASValue<Object> uniqueValue = cache.gets("testobject-1");
    if (uniqueValue == null || uniqueValue.getValue() == null) {
        cache.add("testobject-1", 300, 0);
        uniqueValue = cache.gets("testobject-1");
    } else {
        y = (Integer) uniqueValue.getValue();
    }

    CASResponse response = cache.cas("testobject-1", uniqueValue.getCas(), (Integer) uniqu
Value.getValue() + 1);

    if (response.toString().equals("OK")) {
        System.out.println("ok," + y);
        return "ok," + y;
    } else {
        System.out.println("fail," + y);
        return "fail," + y;
    }
```

消息队列

业界已经有很多 MQ 产品，我们选适合自己的使用便可。如 RabbitMQ、ActiveMQ、ZeroMQ、Jafa/Kafka 等等。

对于短时间的突然大量并发来说，利用队列来缓存请求并依次进行处理也是个很好的方案。

题外：消息队列一般用于其他场景，如应用间的异步通讯、消息通知、日志记录等等。

其他

有很多时候，配合具体的业务逻辑来操作才是真正的好系统。 当预期人数并没有太多时，直接用数据乐观锁来实现既节约开发成本也节约维护成本，并不是什么东西重要就是好。

业务层面也可以帮助系统更好的运作。比如用户量过大时且有很多为了刷单而注册的用户时，可以使会员等级、用户积分等方式预先筛选出优质用户来参与。投机取巧类的，比如可以在前台只取 1/10 用户请求进入后台进行处理，其他直接返回繁忙。如果只是秒杀少量商品，也可以使用预先生成订单再填入用户来实现商品不会超卖。

过早或者没必要地让软件复杂化，而往软件系统中添加组件就是严重增加复杂性是一种很差的做法，认同网络上一作者的想法，链接见参考链接。

参考链接

- http://cloudate.net/?p=306
- https://zh.wikipedia.org/zh/消息队列
- http://www.oschina.net/translate/top-10-uses-for-message-queue
- 你可能并不需要消息队列