蘑菇街交易平台 数据库架构演进历程

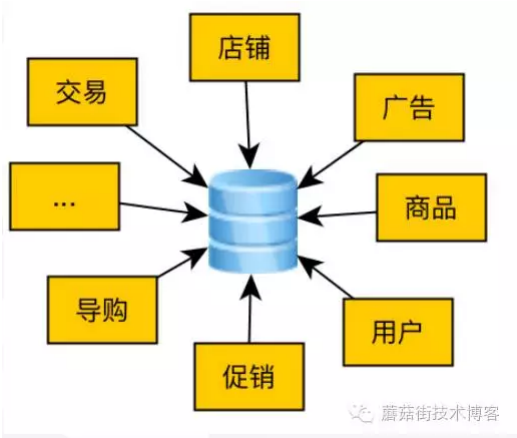
原创：张龙－蘑菇街九如 蘑菇街技术博客 2016-07-25

提纲

- 蘑菇街交易平台初期数据存储面临的问题
- 数据库架构两板斧
- 交易平台实践之路
- 总结及下一步展望

> 假如大家家里有一个小鱼缸，每周买5条不同品种的鱼放入鱼缸，那么随着时间的推移，会出现什么问题？
>
> 1. 早期放入的鱼越来越大，后续不停的有新鱼加入，鱼缸太小，已经无法容纳这么多鱼
> 2. 如果一条价格很低的鱼生病之后，会迅速传染到价格很高的鱼，整个鱼缸的鱼全部被感染
> 3. 想找到其中一条鱼耗费时间越来越长
> 4. ...

交易平台初期数据存储面临的问题



蘑菇街一开始做导购，后期转型电商，中间穿插各种业务，如广告、网红、直播、良品...，从前面的小故事可以套过来，老业务数据还在不停的膨胀，新业务又在不停的拓展，那么在数据库这层面会遇到哪些问题呢？

1. DB磁盘空间
2. DB读写压力
3. DB连接数过多
4. 事故影响大
5. ...

当数据库遇到的这些问题的时候，怎么样去解决，按照什么样的方法去解决呢？

数据库架构两板斧

- scale up （垂直扩展）

  在同一个逻辑单元内增加资源来提高处理能力。

  - 更大的内存
  - 更好的磁盘
  - 核数更多的CPU

鱼缸故事来说，换更大的鱼缸。

从数据库层面来讲，垂直扩展主要是考虑磁盘的问题，从容量、性能、成本三方面来考量。关于磁盘的种类（SATA、SAS、SSD、PCIE）以及他们之间的区别不在此讲述了，大家搜一下相关资料看一看。

磁盘类型	成本	性能
sas	单盘2T，1000	写：300 读：1W
ssd	1. 单盘1.6T，10000 2. 单盘3.2T，22000	写：1300 读：2.5W
pcie	单盘3.2T，44000	写：3000 读：4W

- scale out（横向扩展）

  增加更多逻辑单元的资源，并令它们像是一个单元一样工作。

  > 鱼缸故事来说，将鱼放入各个小鱼缸。

  从数据库层面来讲，横向扩展主要有读写分离、垂直拆分、水平拆分三种手段。

	怎样做	优点	缺点
读写分离	Master挂从库，将读流量切到从库	1. 实现简单，应用程序改动小 2. 读的压力得到分摊	1. 扩展能力有限，性能有上限 2. 磁盘空间问题不能解决

	怎样做	优点	缺点
垂直拆分	根据业务模型将相关联的表迁移，放入同一个DB	1. 实现简单，应用程序改动小 2. 业务数据物理隔离 3. DB的容量压力得到缓解 4. DB的读写压力得到缓解	1. 扩展能力有限，性能有上限 2. 磁盘空间问题不能彻底解决
水平拆分	1. 分库分表: 将一张表数据按照一定的规则分拆不同的DB 2. 冷热分离：将冷数据迁移到另外的存储设备	1. 性能无上限，扩展性强 2. DB容量、读写压力都能得到彻底解决	1. 节点多，运维成本高 2. 实现复杂，应用程序改动大，依赖强大的中间件

- 上面介绍了scale up（垂直扩展） & scale out（横向扩展）是什么，以及怎样去做，那么在遇到问题的时候该怎样去选择方案，我们先对两种方案进行对比，列出各自的优缺点

	优点	缺点
scale up（垂直扩展）	1. 节点少，维护相对简单 2. 数据集中式，应用架构简单，开发容易	1. 性能上限 2. 数据集中式，业务隔离性差，出现故障时影响大
scale out（横向扩展）	1. 可以通过扩容廉价机器搭建一个处理能力强大的集群，降低成本 2. 数据库处理能力没有上限，添加节点增加集群处理能力 3. 单个节点故障对系统整体影响小	1. 节点多，维护成本大 2. 数据分散式，应用架构复杂，开发难度高

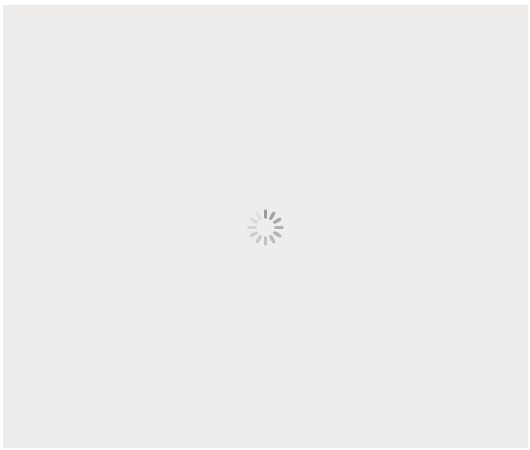
  - 短期来看，scale up更有优势，简化系统架构和应用系统的开发，集中精力发展业务。
  - 长期来看，scale out是必然的趋势，硬件提升总有一个上限，当业务发展到一定的规模的时候，scale up已经无法满足业务的需求。

交易平台实践之路

> 互联网项目一般都是读多写少，在业务发展初期的时候，第一步就是读写分离，蘑菇街也一样，读写分离很早就已经做了，后来随着业务的快速发展，读写压力越来越大，数据量也越来越多，DB的IO性能严重下降，我们先升级了硬件，换成FIO，此时再从硬件上去提升性能已经无法做到。

> 蘑菇街转型电商初期，所有的表都在同一个DB当中，随着业务的发展，单台Master已经无法满足业务的需求。由于电商业务的复杂，研发人员也快速扩充，为了保障交易资金的数据安全，第一步进行的就是垂直拆分。

- 第一阶段 -- 停机垂直拆分
  - 2014 - 2015上半年 交易资金从主站拆分，交易资金拆分。
    业务停机迁移



经过停机垂直拆分之后，此时交易业务全部在单独的一个DB，数据层面和主站、资金等其他业务完全隔离开来，运行一段时间过后，我们此时又遇到了如下几个问题：

- 交易Master出现磁盘空间危机，使用空间已达84%

> 交易链路各业务边界明确
>
> 不同业务间耦合度低，表关联少
>
> 垂直拆分实现简单，应用程序改动也比较少，那么根据交易业务进行垂直拆分（快照、物流、购物车、退款），就能够解决Master磁盘空间的危机。

- 交易Master写瓶颈，在15年双11期间交易链路无法满足业务的需求，技术成为瓶颈，制约着业务的发展

> 经过压测FIO单实例3000TPS是安全水位，如果大促期间要支撑3K-5K的订单量，就算把订单表垂直拆分出去也无法满足要求，只有进行分库分表

- 业务超高速发展，**不允许停机**

> 业务发展太快，如果还像去年那样停机操作对用户体验影响太大，只有进行在线迁移

- 当时距离双11还有四个月，我们觉得最为紧迫的事情是Master磁盘危机，尤其是要实施在线迁移，之前没有相关的经验。经过分析交易相关的业务表，快照数据最大（快照迁移出去就能降低25%的存储空间）、业务简单（只有Insert，没有Update）、表结构简单，基于这三个原因我们选择了先进行在线迁移快照业务。

- 第二阶段 -- 不停机垂直拆分

  要进行在线迁移，有三个问题需要解决

- 全量

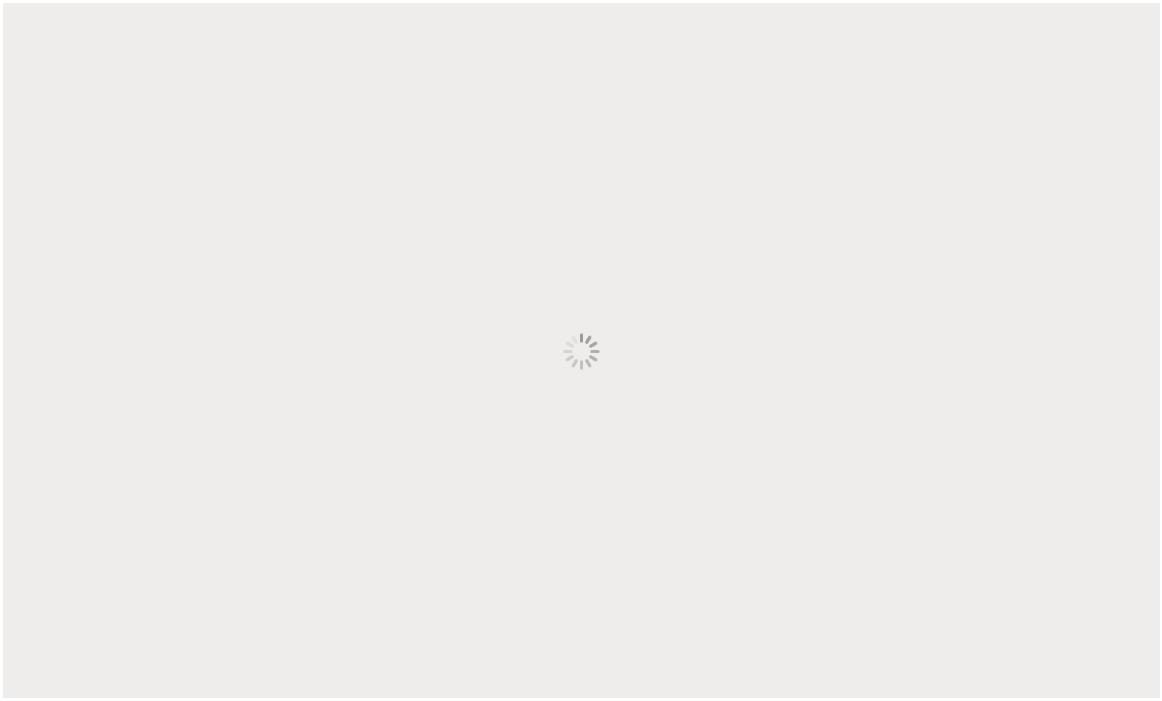
> 表结构不变：挂主从
>
> 表结构变化：同步脚本，扫描老表数据写入到新表

- 增量

- canal：基于数据库增量日志解析，提供增量数据订阅&消费。开源地址
- pigeon：基于canal开发的一个多数据源支持、低延时、高可靠的数据变更事件捕获、分发平台。
- kafka：是一种高吞吐量的分布式发布订阅消息系统。官方文档
- corgi：自研的一种高吞吐量的分布式发布订阅消息系统，和kafka类似。
- 增量我们用到了几个组件，先了解几个概念

- 要进行在线迁移，有三个问题需要解决

- > 实时性：中间件团队当时基于canal研发了数据变更分发平台pigeon，将数据变更记录放入kafka（后来自研了Corgi），业务团队消费kafka/Corgi的消息进行业务逻辑处理，全程多线程处理
  >
  > 顺序性：为了保障数据的实时性，是多线程处理，每一条数据根据主键ID取模，保障每条被同一个线程处理
  >
  > 扩展性：数据迁移时可能会有表字段变化、多表合一表、一表拆多表等业务场景，扩展性要求高
  >
  > 高可用：Pigeon本身是无状态，所有状态都在ZK中，一个pigeon节点如果挂掉了，直接重新起一个节点；Corgi的Keeper有主从的，可以自动切换。两个Keeper都挂了，Client还可以使用自己的缓存元数据。一个Topic至少分布在3个Broker上，Broker挂掉不影响正常的发送，只是挂掉的Broker上的消息不能消费。这种情况，我们Broker还有主从结构，半同步复制，还可以启动Slave代替挂掉的Broker，保证数据不丢；消费端分布式多台机器部署，避免单点；

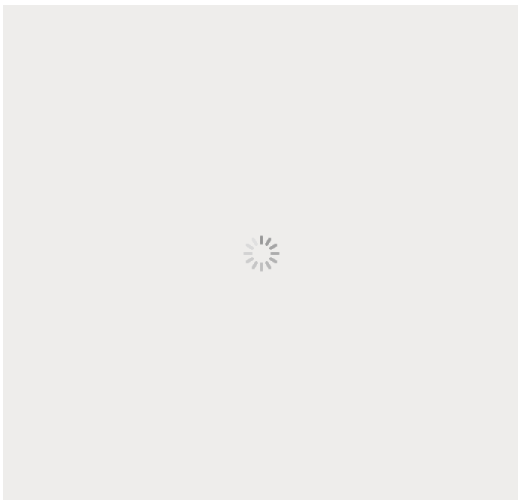


- 灰度开关

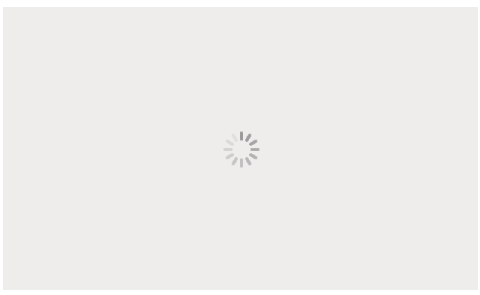
> 以往迁移都是停机一个晚上操作，有充足的时间进行业务验证，那么现在在线迁移，就必须有灰度开关来保障出现问题时尽可能快的恢复，灰度开关设计文档。

- 快照垂直拆分

为了解决以上问题，我们开发出了***数据同步工具***、***高可用灰度开关***2套工具，去支持快照的在线迁移，快速验证在线迁移可行性。



- 快照在线迁移整个过程比较顺利，经过快照业务验证方案之后，交易各链路按照业务域划分，也相继实现了垂直拆分。下图是垂直拆分之后的交易数据库架构



- 第三阶段 -- 不停机分库分表

> 交易Master库磁盘空间危机得到解决后，开始为双11大促做准备，虽然各个业务都进行了垂直拆分，但是老DB还是有比较多的业务比如退款、订单操作、下单等，下单业务无法满足大促期间的峰值要求，垂直拆分验证了在线迁移的可行性之后，我们决定开始进行订单的分库分表

  - 分库分表我们先要考虑如下四个问题
  - 拆分的路由key

    > 交易的订单业务只有三种维度，买家ID、卖家ID、订单ID，交易的订单ID很早之前就为水平拆分做了准备，订单ID后四位为买家ID后四位。所以交易的业务可以划分为两类，买家ID、卖家ID

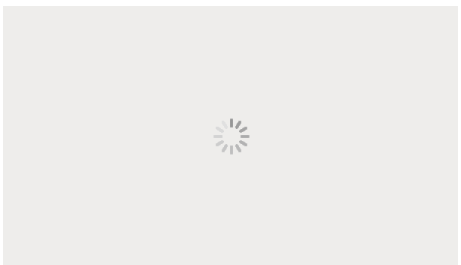
  - 水平拆分策略

    > 分库分表一般采用hash切分
    >
    > 冷热分离一般采用id区间或时间区间来进行的范围切分

  - 容量预估

    > 2^n

  - 为什么是2^N，二叉树，扩容的时候更方便，只需要基于表级别分，不需要数据重新分配。比如分了8张表，分别放在2个库B、C，一段时间后，需要再次扩容，只需要将H、I迁移放到D库，J、K迁移放到E库，数据不需要重新分配。



  - > 根据单表规划存储数据来计算需要多少表

2^9=512，交易团队根据自身的数据量、DB的机器性能评估出需要512张表。计算公式如下图（模拟数据），2017年为了支撑百亿级订单，最少2^10=1024张表

  -

- 单实例3KTPS 计算需要多少库 8

  - 库 2^3=8 单实例一个库3KTPS，8个实例8个库可以支撑2.4WTPS。足以支撑未来3年的量，不需要再次扩容。根据当前的订单量，从成本考虑，我们准备了2个Master实例，一个实例4个库。

> hash规则

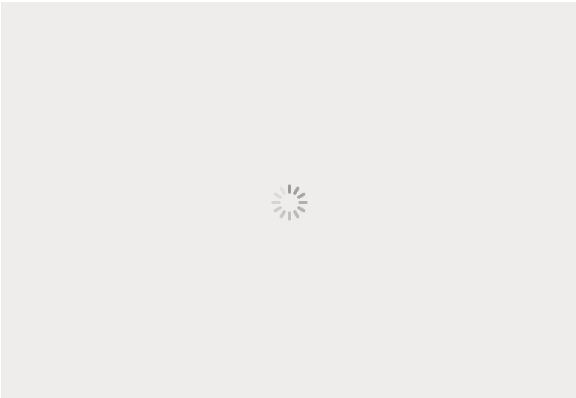
  - 库：((shardingPara % 10000) % 512) / 64
  - 表：(shardingPara % 10000) % 512

- 业务ID -- 全局唯一

> 很多地方说主键ID要唯一，其实不对，主键ID可以使用表自增。业务ID可以是一个字段，也可以是多个字段。 交易订单业务相关表使用的OrderId字段。

- 上面四个问题考虑清楚之后，我们准备了五种手段来进行订单分库分表

- > 数据同步 trade-data-sync
  >
  > 数据对账：基于Pigeon拿到数据变更记录，延时队列去新DB对比数据。
  >
  > 灰度开关：灰度开关设计文档。
  >
  > 分库分表中间件：自研T-Sharding中间件。
  >
  > 分布式主键生成器：中间件团队提供。



- 订单分库分表完成之后：

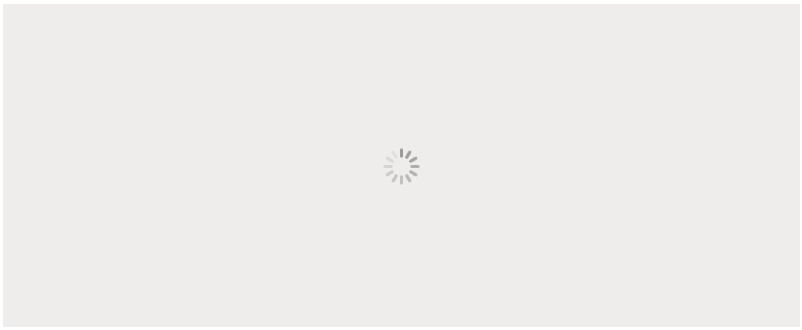
- 第四阶段 -- 冷热分离

> 随着业务的持续增长，预计在16年双11，订单表需要从2个Master扩为8个Master，购物车需要分库分表，快照数据又快要将DB撑满，物流数据也持续增长。
> 如果全部进行分库分表，DB的机器成本越来越高，每次都是成本翻倍。

> - 背景
>   - 冷数据访问频率低，占用昂贵的磁盘资源
>   - 冷数据和热数据放一起，数据量过大，IO性能下降，影响热数据访问性能
> - 冷数据定义
>   - 数据的生命周期已结束
>
>> 数据写入之后再也不会被修改，我们成为生命周期已经结束，比如订单，订单从下单一直到订单完成之后，不能再进行任何操作了，这样的订单就算订单生命周期结

> 束的订单
>
> - 访问频率低
>
> - 对交易业务适合进行冷热的表分析

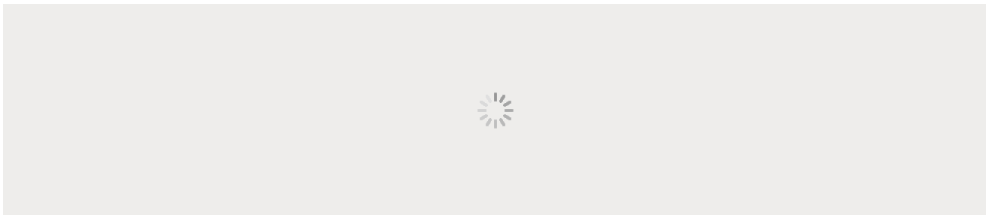
- 应用程序修改



我们再次用快照业务来进行试水冷热分离，快照数据冷数据大小

- 六个月前冷数据 1.9T
- 单条数据平均大小 4K

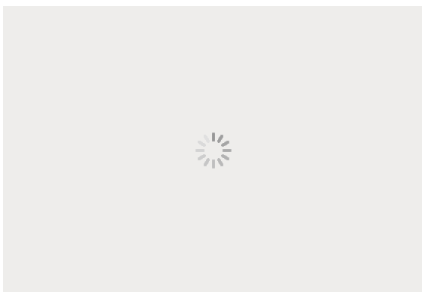
冷数据存储方案



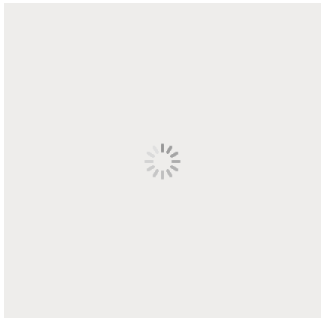
存储成本的解释：假如1T数据，tair需要2T的存储；Infobright需要0.2T的存储；hbase需要1.5T的存储

技术方案

- 应用程序修改

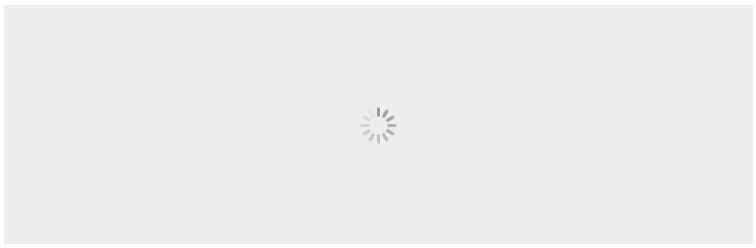


- metabase灰度开关工具，之前灰度开关升级版本，由公司中间件团队统一维护

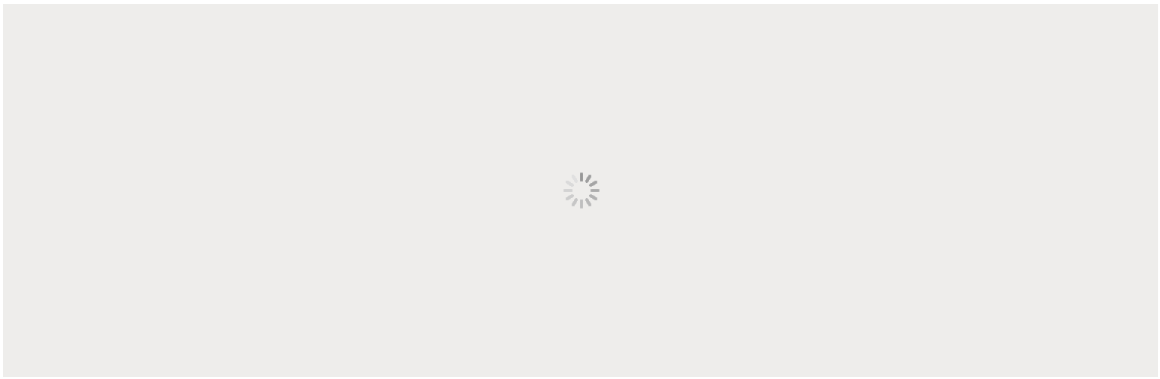


> HBase有公司团队进行搭建、维护，我们业务团队进行冷数据迁移，线上业务切换还是比较快，快速的验证了冷热分离的方案可行性

- 运行效果



- 成本对比



快照目前的方案使用了38W，但是数据存储只用了20%

总结及下一步展望

- 总结

1. 合适的阶段做合适的事情，选择最适合当前阶段的方案

> 抓主要矛盾，集中精力做最重要的事情。比如上面第二阶段的时候，既可以做垂直拆分，又可以做分库分表，我们是基于各种因素选择了优先做快照垂直拆分，快照拆分快速验证了在线迁移的可行性，为后续垂直拆分、水平拆分打下了良好的基础。

2. 要前瞻性的规划，支撑业务的快速发展

> 在15年垂直拆分的时候，基于当时业务情况的评估，为了节省机器成本将快照、购物车数据迁移在同一个DB实例中；16年美丽说、淘世界融合，快照购物车本身蘑菇街数据的增长，美丽说、淘世界购物车数据的迁移当时就快把DB磁盘空间撑满了，只有临时停止数据融合的工作，紧急将购物车、快照的数据进行拆分到不同的物理机器，才让数据融合的工作继续进行下去。

- 展望

1. 冷热分离大规模上线（卖家数据、买家数据、物流数据），降低机器成本
2. 运维自动化（扩容、上下线、MHA）
3. 监控、报警体系化

小编后记：

作者九如提交给小编这边文章的时候，是他在蘑菇街的最后一个工作日，他说他要好好完成这篇文章，因为在蘑菇街学会很多，希望能整理出来，让更多人看到。

他说从蘑菇街出去，不会选择跳槽到其他公司，而选择再去创业，和一群兄弟们再去做点事情。

祝福九如，有机会常回来看看！