分布式事务解决方案「手写代码」

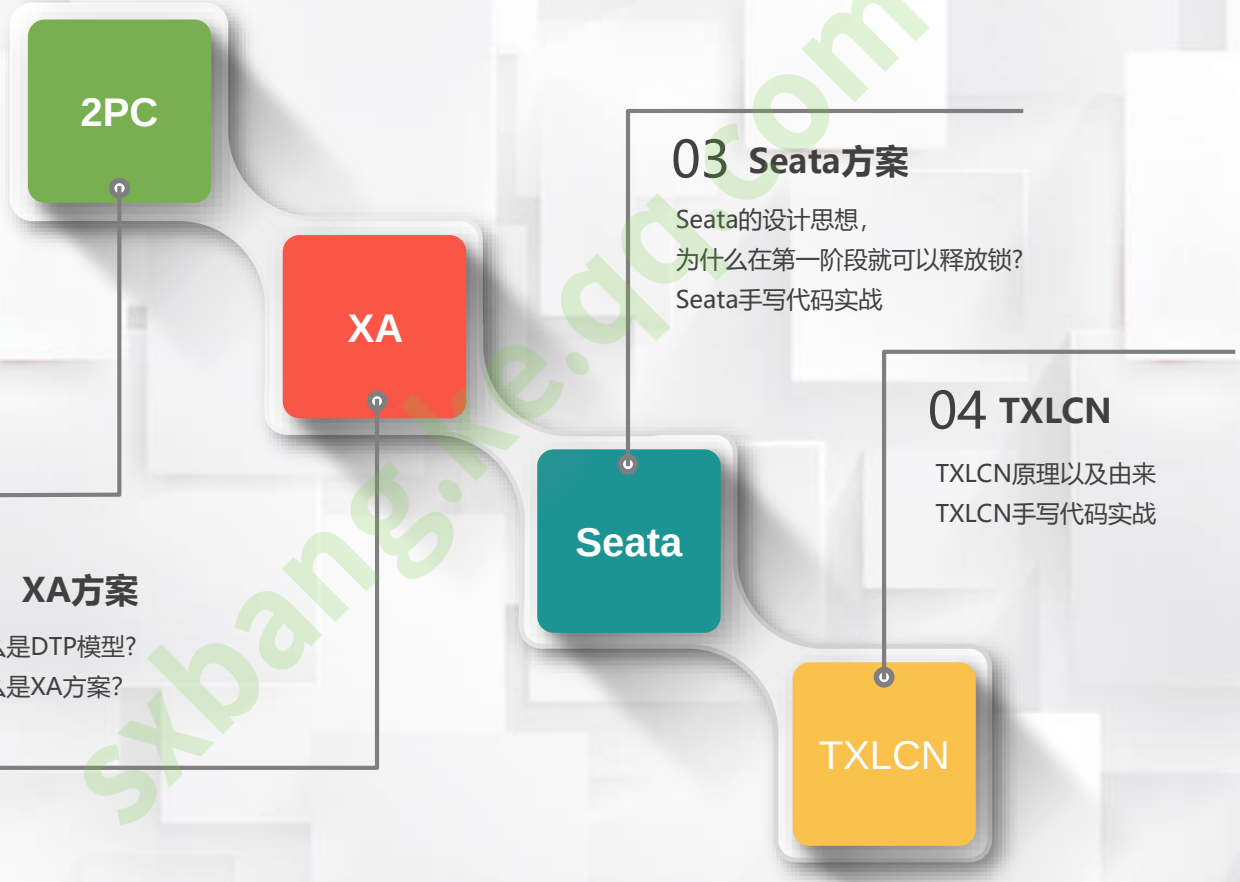
大道至简 悟在天成

Andy | 悟纤

目录

基本概念

2PC/3PC



01 2PC/3PC概念

什么是2PC/3PC?
之间有什么区别?

02 XA方案

什么是DTP模型?
什么是XA方案?

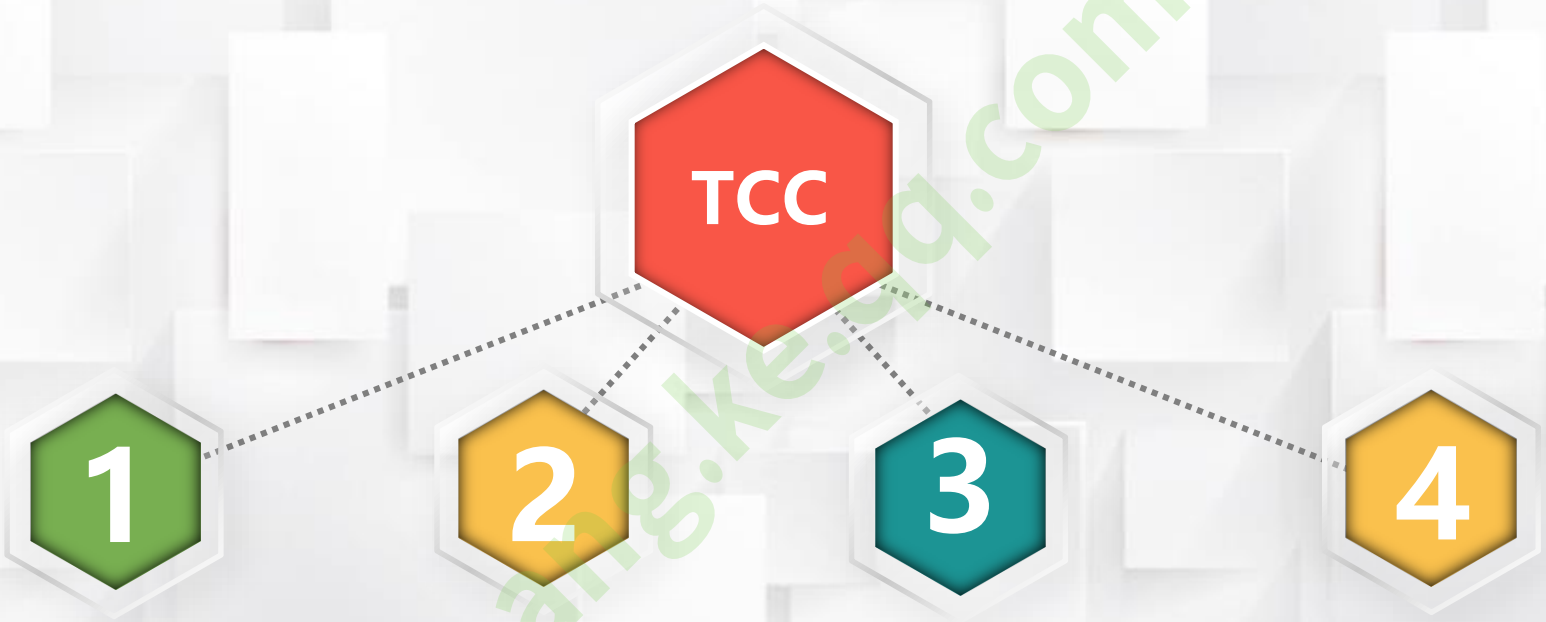
03 Seata方案

Seata的设计思想,
为什么在第一阶段就可以释放锁?
Seata手写代码实战

04 TXLCN

TXLCN原理以及由来
TXLCN手写代码实战

TCC



TCC概念

什么是TCC

TCC异常处理

空回滚
幂等性
悬挂

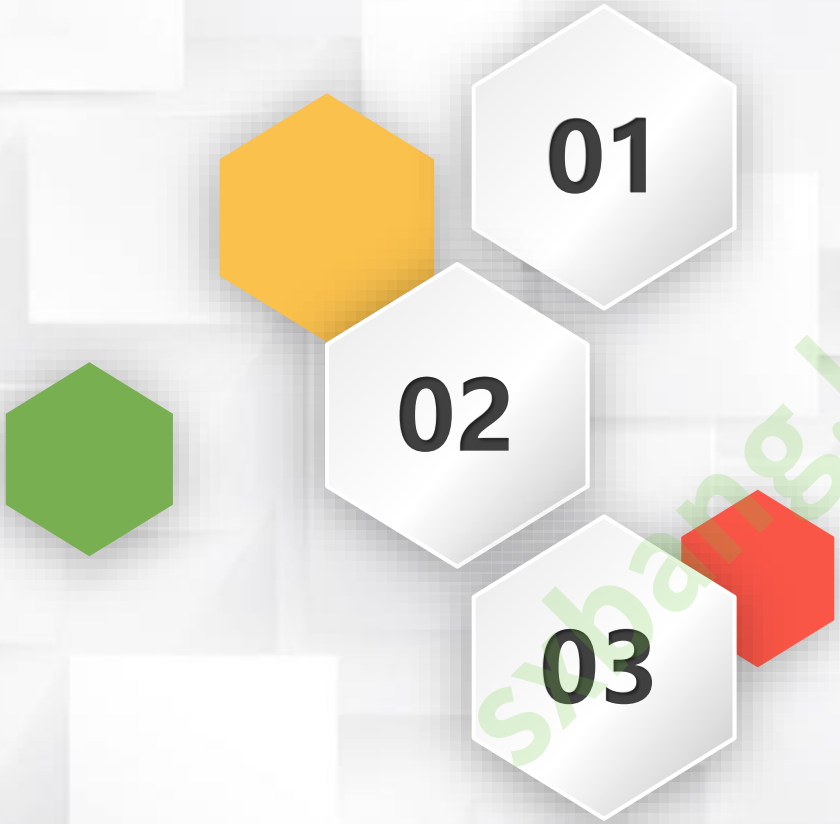
TCC案例分析

通过方案讲解，了解透TCC

TCC框架实战

常用的TCC框架
TCC使用框架实战

可靠消息最终一致性



概念篇

什么是可靠消息最终一致性?

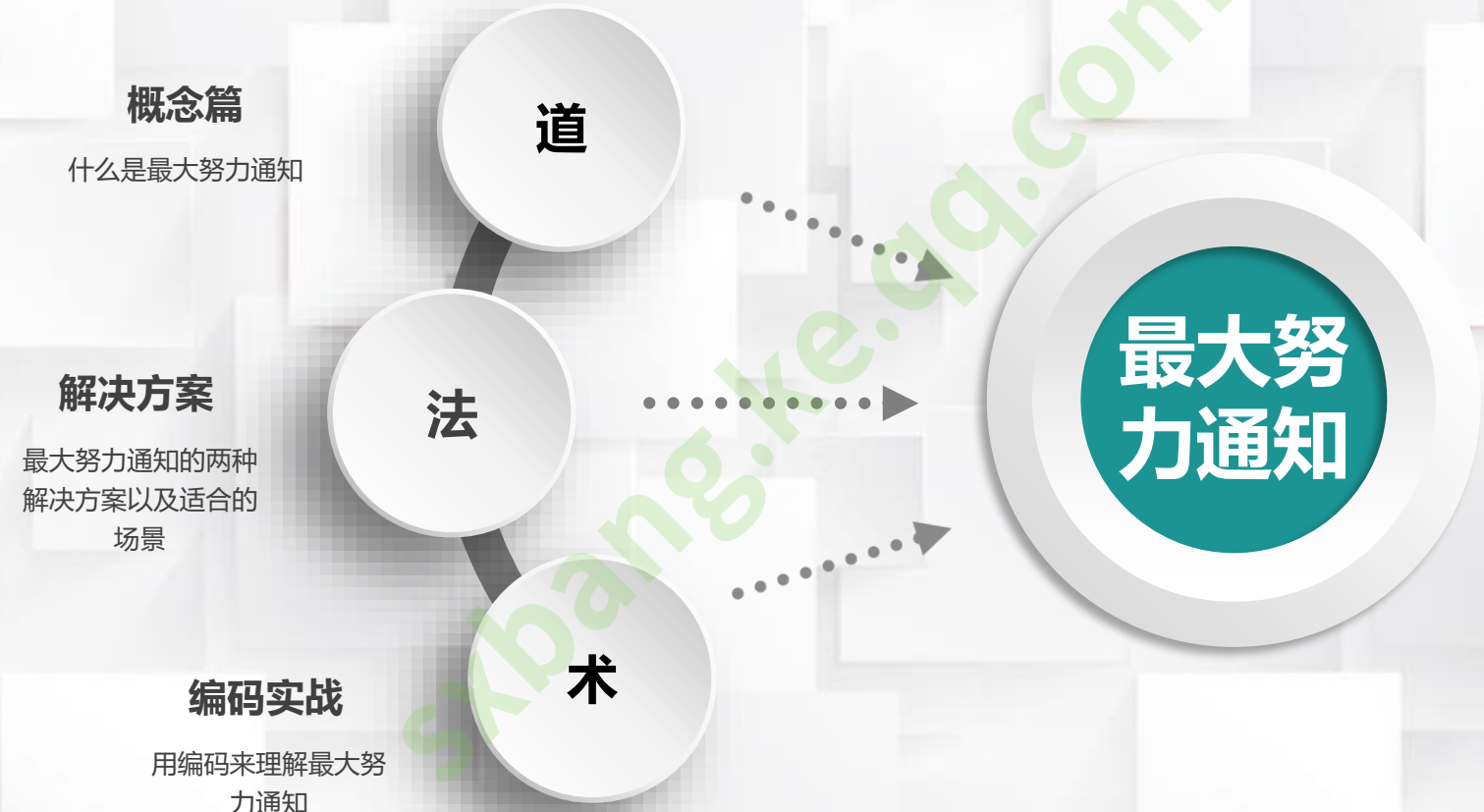
本地消息表

可靠消息最终一致性解决方案：本地消息表

RocketMQ

可靠消息最终一致性解决方案：RocketMQ

最大努力通知



适合人群

JAVA

有一定的Java基础

Spring Boot

有一定的Spring Boot基础

JPA

对Spring Data JPA有一定的了解

MySQL

有一定的SQL编写能力

使用技术

1. Spring Boot 2.1.5
2. Spring Data JPA
3. MySQL 8.0
4. JDK 1.8
5. Seata 0.9
6. LCN 5.0
7. Spring Cloud Greenwich.SR3
8. RocketMQ 4.4

……

「提供每个视频的源码下载」

事务的基本概念

大道至简 悟在天成

Andy | 悟纤

一、什么是事务

1.1 举个生活中的例子

去超市买东西，一手交钱，一手交货就是一个事务的例子。

事务的体现：交钱和交货必须全部成功，事务才能算成功，任何一个活动失败，整个事务就失败了了。

1.2 举个SQL例子

张三要给李四转账100元，那么我们会有这样的一段SQL：

```
begin transaction;
        update account set money = money-100 where name = '张三';
        update account set money = money+100 where name = '李四';
commit transaction;
```

事务的体现：这两个SQL要么全部成功，要么全部失败。

1.3 事务定义

事务是一系列操作组成的工作单元，该工作单元内的操作是不可分割的，即要么所有操作都做，要么所有操作都不做，这就是事务。

理解一：事务可以看做是一次大的活动，它由不同的小活动组成，这些活动要么全部成功，要么全部失败。

理解二：事务可以看做是一个大的操作，它由一系列操作组成，这些操作要么全部成功，要么全部失败。

二、本地事务/数据库事务

数据库事务：在计算机系统中，更多的是通过关系型数据库来控制事务，这是利用数据库本身的事务特性来实现的，因此叫数据库事务。

本地事务：由于应用主要靠关系数据库来控制事务，而数据库通常和应用在同一个服务器，所以基于关系型数据库的事务又被称为本地事务。

BTW：本地事务也称为数据库事务或传统事务（相对于分布式事务而言）

三、分布式事务

分布式系统会把一个应用系统拆分为可独立部署的多个服务，因此需要服务与服务之间远程之间远程协作才能完成事务操作， 这种分布式系统环境下由不同的服务之间通过网络协作完成事务称为分布式事务。

例如：用户注册送积分事务、创建订单减库存事务，银行转账事务。

3.1 转账例子说明分布式事务

张三转账100元给李四：

本地事务的实现：

```
begin transaction;
  //1. 本地数据库操作：张三减少100元。
  //2. 本地数据库操作：李四增加100元。
commit transaction;
```

3.1 转账例子说明分布式事务

张三转账100元给李四：

分布式事务的实现：

```
begin transaction;
  //1. 本地数据库操作：张三减少100元。
  //2. 远程调用：让李四增加100元。
commit transaction;
```

事务的四大特性ACID

大道至简 悟在天成

Andy | 悟纤

一、事务的四大特性

事务的四大特性主要是：

原子性（Atomicity）

一致性（Consistency）

隔离性（Isolation）

持久性（Durability）

一、事务的四大特性

1.1 原子性（Atomicity）

原子性是指事务是一个不可分割的工作单位，事务中的操作要么全部成功，要么全部失败。比如在同一个事务中的SQL语句，要么全部执行成功，要么全部执行失败。

```
begin transaction;
        update account set money = money-100 where name = '张三';
        update account set money = money+100 where name = '李四';
commit transaction;
```

1.2 一致性（Consistency）

官网上事务一致性的概念是：事务必须使数据库从一个一致性状态变换到另外一个一致性状态。

换一种方式理解就是：事务按照预期生效，数据的状态是预期的状态。

举例说明：张三向李四转100元，转账前和转账后的数据是正确的状态，这就叫一致性，如果出现张三转出100元，李四账号没有增加100元这就出现了数据错误，就没有达到一致性。

1.3 隔离性（Isolation）

事务的隔离性是多个用户并发访问数据库时，数据库为每一个用户开启的事务，不能被其他事务的操作数据所干扰，多个并发事务之间要相互隔离。

1.4 持久性（Durability）

持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

二、数据库ACID的体现

2.1 原子性

原子性说的是数据要么一起成功，要么一起失败，那么就有两种情况：

事务提交（commit）和事务回滚（rollback）。

二、数据库ACID的体现

2.1 原子性

成功提交：

```
mysql>
mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |  1000 |
|  2 | 李四   |     0 |
+----+--------+-------+
2 rows in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money-100 where name='张三';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set money = money+100 where name='李四';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |   900 |
|  2 | 李四   |   100 |
+----+--------+-------+
2 rows in set (0.00 sec)
```

1.开启事务
2.张三扣减金额
3.李四增加金额
4.提交事务。

这里的条件实际开发中使用id，这里只是为了方便理解。

二、数据库ACID的体现

2.1 原子性

失败回滚：

```
mysql>
mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |   900 |
|  2 | 李四   |   100 |
+----+--------+-------+
2 rows in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money-100 where name='张三';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set money = money+100 where name='李四';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> rollback;
Query OK, 0 rows affected (0.03 sec)

mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |   900 |
|  2 | 李四   |   100 |
+----+--------+-------+
2 rows in set (0.00 sec)
```

1.开启事务
2.张三扣减金额
3.李四增加金额
4.事务回滚

二、数据库ACID的体现

2.2 一致性

一致性主要说明的是事务的前后，数据库中的数据的状态要确保一致。

事务提交成功，那么张三账户上的余额是900元，李四账户上的余额是100元。

事务提交失败，那么张三和李四的账户的金额不变。

这说明现在在数据库的事务的控制下，确保了数据的一致性。

2.3 隔离性

隔离性的体现，多个并发事务之间是隔离的。

张三给李四转账，如果事务没有提交的话，那么在另外一个session中并不能查看另外一个session未提交的数据。

二、数据库ACID的体现

2.3 隔离性

```
Database changed
mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |   900 |
|  2 | 李四   |   100 |
+----+--------+-------+
2 rows in set (0.00 sec)

mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money = money-100 where name='张三';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set money = money+100 where name='李四';
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input state

mysql> use tx_demo;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select *from account;
+----+--------+-------+
| id | name   | money |
+----+--------+-------+
|  1 | 张三   |   900 |
|  2 | 李四   |   100 |
+----+--------+-------+
2 rows in set (0.00 sec)

mysql>
```

事务没有提交，在另外一个session中查看的还是原先的数据。

2.4 持久性

持久性的体现就是数据一旦commit之后，那么对于数据的改变就是永久的。我们commit之后，张三的账户就永久减少了100元，李四的账户就永久增加了100元。

三、现JDBC ACID的体现

我们使用JDBC连接MYSQL数据库的代码：

```
* 1/ 注册JDBC驱动；
* 2/ 获取连接；
* 3/ 开启事务
* 4/ 创建SQL语句;
* 5/ 执行SQL语句;
* 6/ 提交事务
* 7/ 关闭连接.
```

三、现JDBC ACID的体现

```
// 1/ 注册JDBC驱动;
Class.forName("com.mysql.cj.jdbc.Driver");

String url = "jdbc:mysql://127.0.0.1:3306/tx_demo";
String user = "root";
String password = "root";

// 2/ 获取连接;
Connection conn = DriverManager.getConnection(url, user, password);

// 3/ 开启事务
conn.setAutoCommit(false);
```

三、现JDBC ACID的体现

```
// 4/ 创建SQL语句; 注意：实际使用account的主键，这里主要是为了方便理解。
String sql = "update account set money = money-100 where name = ?";
PreparedStatement ps = conn.prepareStatement(sql);
ps.setString(1,"张三");

// 5/ 执行SQL语句;
int rs = ps.executeUpdate();
if(rs>0) {
    System.out.println("张三-扣减成功");
}

// 给李四增加金额
sql = "update account set money = money+100 where name = ?";
ps = conn.prepareStatement(sql);
ps.setString(1,"李四");
rs = ps.executeUpdate();
if(rs>0) {
    System.out.println("李四-增加成功");
}
```

三、现JDBC ACID的体现

```
// 6/ 提交事务,更标准的写法应该拦截异常，有异常的情况下rollback();
conn.commit();

// 7/ 关闭连接.
ps.close();
conn.close();
```

三、现JDBC ACID的体现

如果代码正常运行的话，那么张三会扣减金额，李四会增加金额，这就确保的原子性；

一旦数据保存到数据库之后，数据就永久被改变了，这就是持久性；

事务前后，数据的状态也是我们所期望的状态，这就保证了数据的一致性；

如果在事务未commit的话，那么在另外一个线程发起查询请求的话，那么并不能查看到最近的数据（这里未进行编码），这就是隔离性。

分布式事务产生的场景

大道至简 悟在天成

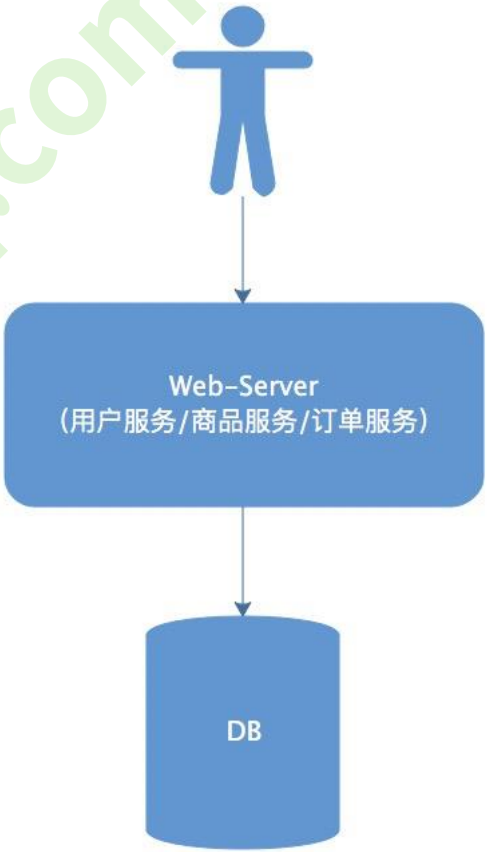
Andy | 悟纤

一、分布式事务产生的场景

分布式系统是从单体系统演变过来的，

我们要理解分布式事务产生的场景，

需要从单体应用进行理解。



一、分布式事务产生的场景

（1）Web-Server中的服务都是在一个项目中，此时只有一个数据库。

（2）Web-Server集群的时候，每一个Web-Server都有每个服务，属于同进程调用。

（3）数据库只有一个，Web-Server获取的数据库的连接也就是由同一个数据库处理的，满足本地事务的基本要求，能够满足事务的四大特性。

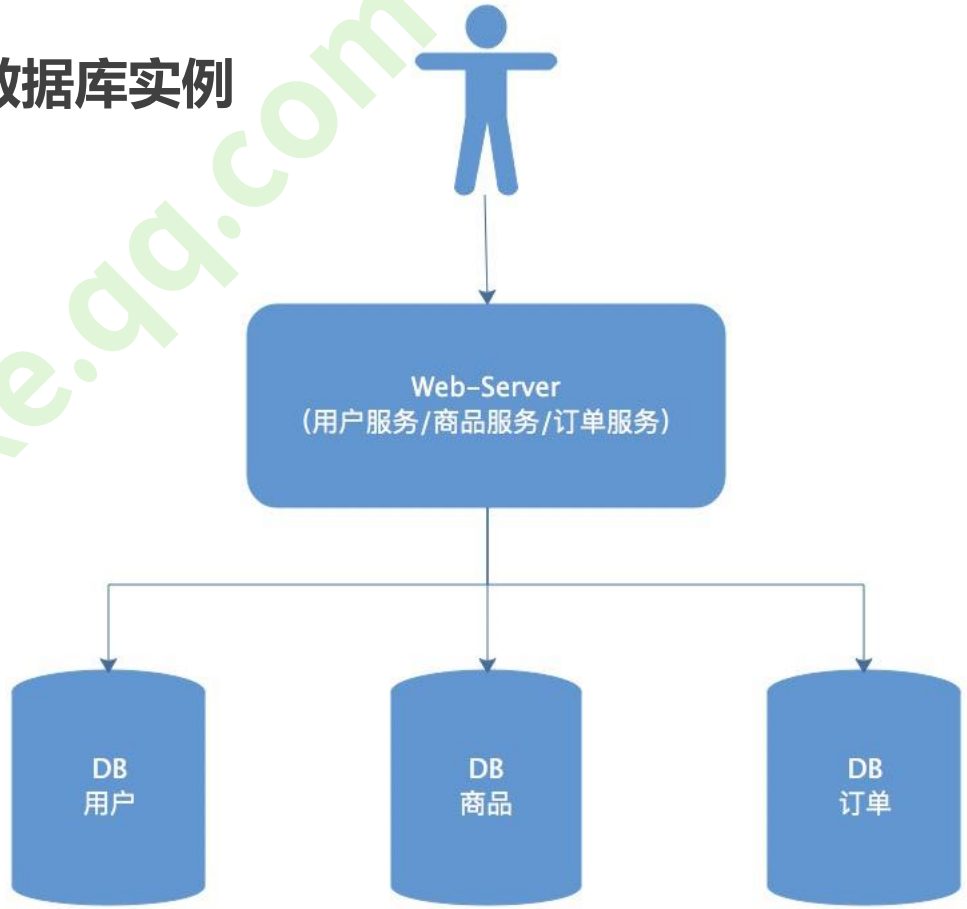
一、分布式事务产生的场景

1.1 数据库拆分：单体系统访问多个数据库实例

随着用户增多，数据库压力越来越大，此时我们会进行数据库的拆分，怎么拆分呐？比如：拆分成用户数据库、商品数据库、订单数据库。

一、分布式事务产生的场景

1.1 数据库拆分：单体系统访问多个数据库实例



一、分布式事务产生的场景

1.1 数据库拆分：单体系统访问多个数据库实例

此时：单体系统需要访问多个数据库时就会产生分布式事务。

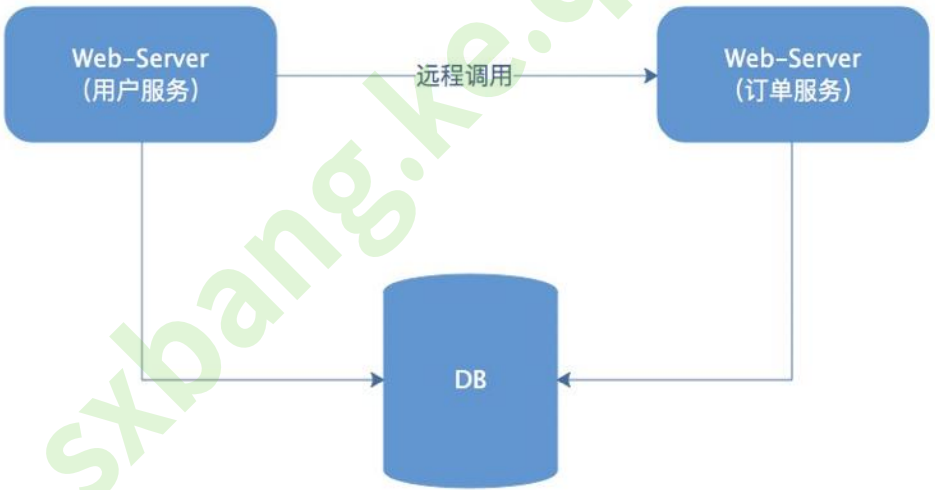
产生原因：由于数据分布在不同的数据库实例，需要通过不同的数据库连接去操作数据，此时就会产生分布式事务。

简单理解：跨数据库实例产生分布式事务。

1.2 服务拆分：多服务访问同一个数据库实例

随着模块越来越多，项目越来越庞大，我们会将模块拆分成不同的项目，那么两个服务需要跨网络远程调用，两个服务持有了不同的数据库连接进行数据库操作，此时就会产生分布式事务。

1.2 服务拆分：多服务访问同一个数据库实例

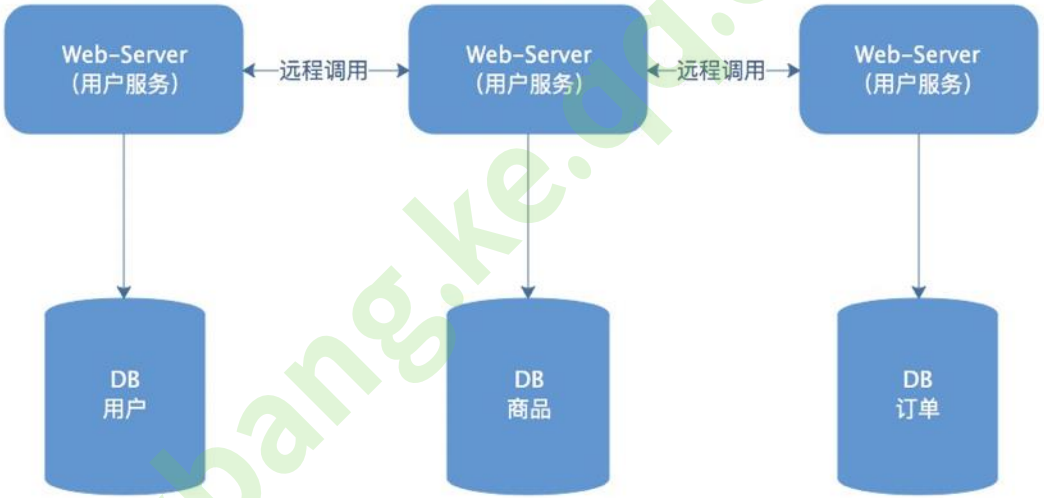


简单理解：跨网络远程调用产生分布式事务。

1.3 服务+数据库拆分：微服务架构

随着项目的发展，到了后期，我们就会一个服务一个数据库，比如：用户服务对应一个用户数据库，商品服务对应一个商品数据库，订单服务对应一个订单数据库，这就是微服务的产生，微服务之间通过远程调用完成事务操作，这就产生了分布式事务。

1.3 服务+数据库拆分：微服务架构



简单理解：跨网络远程调用产生分布式事务。

二、小结

通过上面分析，有两种情况会产生分布式事务：

（1）跨网络远程调用完成事务协作，就会产生分布式事务。

（2）跨数据库实例完成事务协作，就会产生分布式事务。

我们可以在进一步分析，上面这两种情况有一个共性就是在一个事务中，数据库的连接是不一样的，所以简单理解就是：一个事务中，当操作使用了不同的数据库连接就会产生分布式事务。

分布式事务的CAP理论

大道至简 悟在天成

Andy | 悟纤

前言

通过前面的学习，我们了解到了分布式的基础概念，与本地事务不同的是，分布式系统之所以叫分布式系统，是因为提供服务的若干个节点分布在不同机器上，相互之间通过网络交互，不能因为有一点网络问题就导致整个系统无法提供服务，网络因素成为了分布式事务的考量标准之一。因此，分布式事务需要进一步的理论基础，接下来，我们先来学习一下分布式事务的CAP理论。

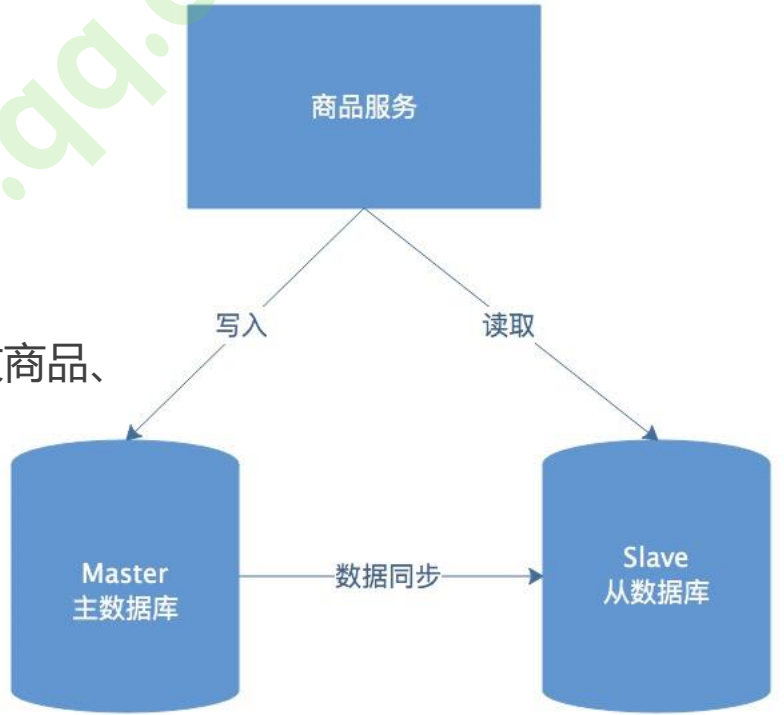
在讲解分布式事务事务控制解决方案之前需要先学习一些基础理论，通过理论知识指导我们确定分布式事务控制的目标，从而帮助我们理解每个解决方案。

一、CAP理论

CAP是Consistency、Avaliability、Partition tolerance三个词语的缩写，分别表示一致性、可用性、分区容忍性。

一、CAP理论

为了方便对CAP理论的理解，我们结合电商系统中的一些业务场景来理解CAP，如下图，是商品信息管理的执行流程：



执行流程如下：

（1）商品请求主数据库写入商品信息（添加商品、修改商品、删除商品）；

（2）主数据库向商品服务响应写入成功；

（3）商品服务请求从数据库读取商品信息；

C - 一致性

一致性是写操作后的读操作可以读取到最新的数据状态，当数据分布在多个节点时，从任意节点读取到的数据都是最新的状态。

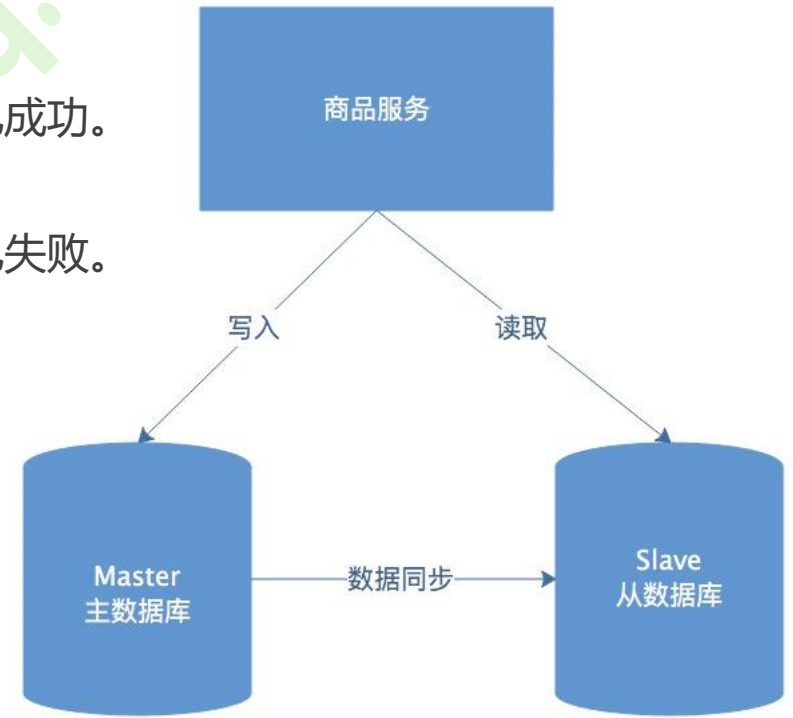
一、CAP理论

C - 一致性

商品信息的读写要满足一致性就是要实现如下目标：

（1）商品服务写入主数据库成功，则向从数据库查询新数据也成功。

（2）商品服务写入主数据库失败，则向从数据库查询新数据也失败。



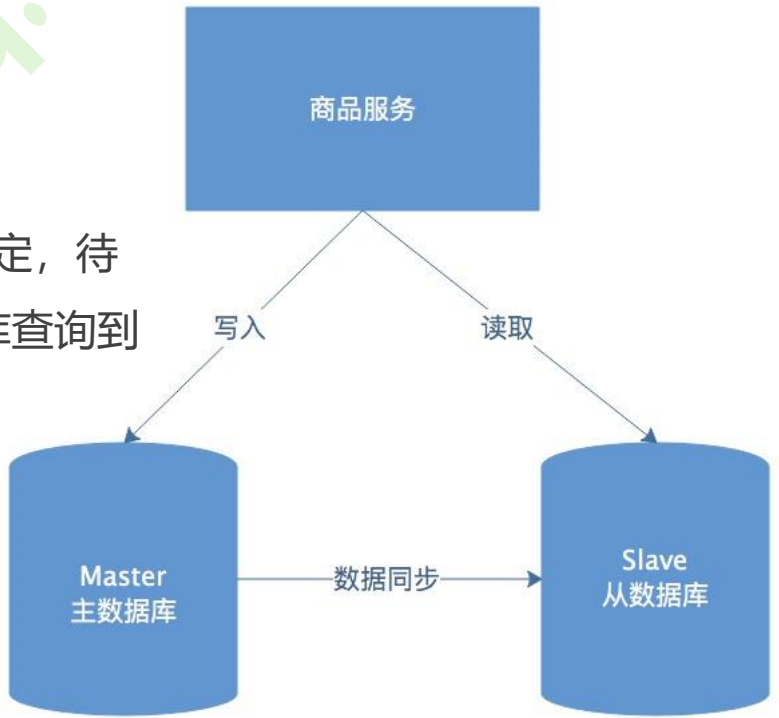
一、CAP理论

C - 一致性

如何实现一致性?

(1) 写入主数据库后要将数据同步到从数据库。

(2) 写入主数据库后，在向从数据库同步期间要将从数据库锁定，待同步完成后再释放锁，以免在新数据库写入成功后，向从数据库查询到旧的数据。



一、CAP理论

C - 一致性

分布式一致性的特点：

(1) 由于存在数据同步的过程，写操作的相应会有一定延迟。

(2) 为了保证数据一致性会对资源暂时锁定，待数据同步完成释放锁定资源。

(3) 如果请求数据同步失败的节点则会返回错误信息，一定不会返回旧信息。

一、CAP理论

A – 可用性

可用性是指任何事务操作都可以得到相应结果，且不会出现响应超时或响应错误。

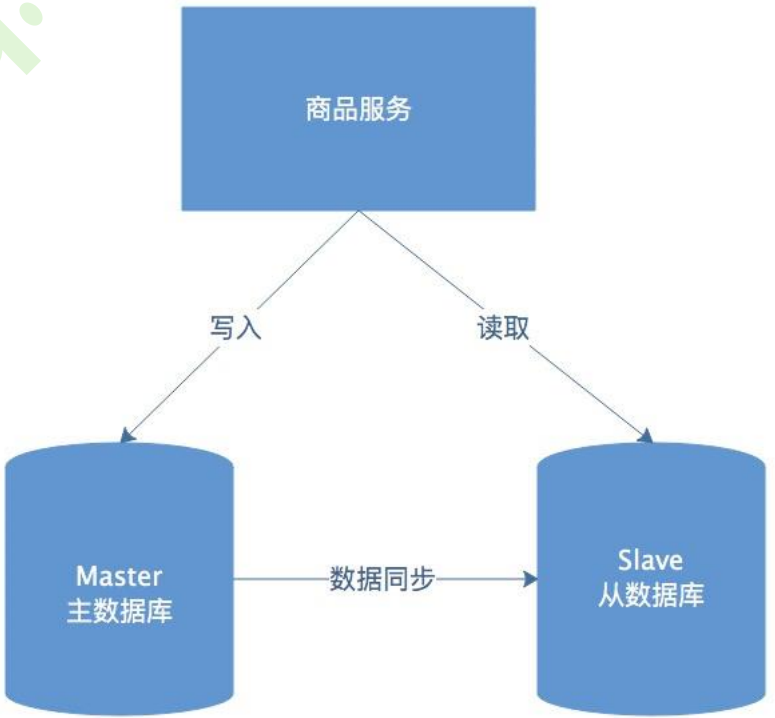
一、CAP理论

A – 可用性

商品信息的读取要满足可用性就是要实现如下目标：

(1) 从数据库接收到查询的请求则立即能够响应数据查询结果。

(2) 从数据库查询不允许出现响应超时或者响应错误。



一、CAP理论

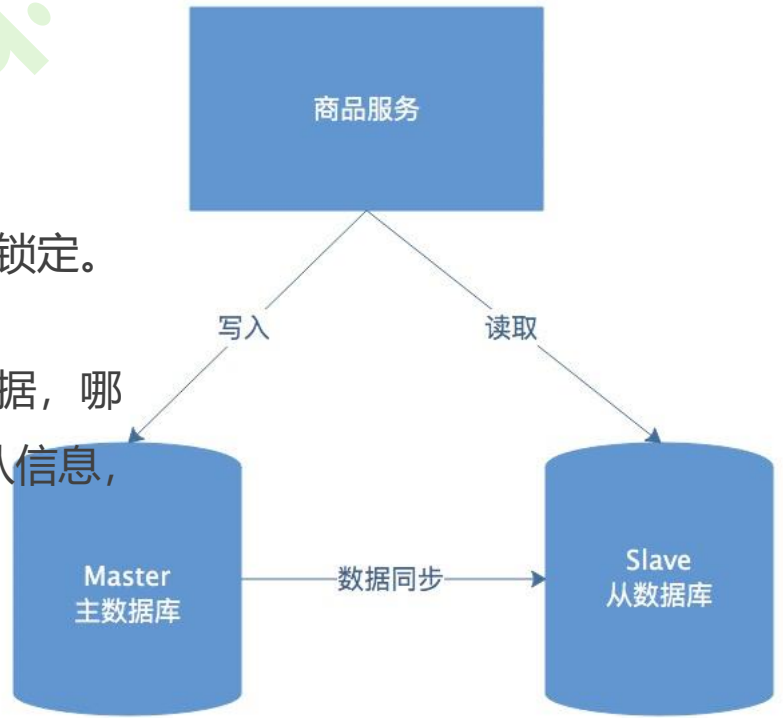
A – 可用性

如何实现可用性?

(1) 写入主数据库要将数据同步到从数据库。

(2) 由于要保证从数据库的可用性，不可将从数据库中的资源锁定。

(3) 即时数据还没有同步过来，从数据库也要返回要查询的数据，哪怕是旧数据，如果连旧数据也没有则可以按照约定返回一个默认信息，但不能返回错误或相应超时。



一、CAP理论

A – 可用性

分布式系统可用性的特点：

(1) 所有请求都有响应，且不会出现响应超时或者响应错误。

一、CAP理论

P – 分区容忍性

通常分布式系统的各个节点部署在不同的子网，这就是网络分区，不可避免的会出现由于网络问题而导致节点之间通信失败，此时仍可对外提供服务，这叫分区容忍性。

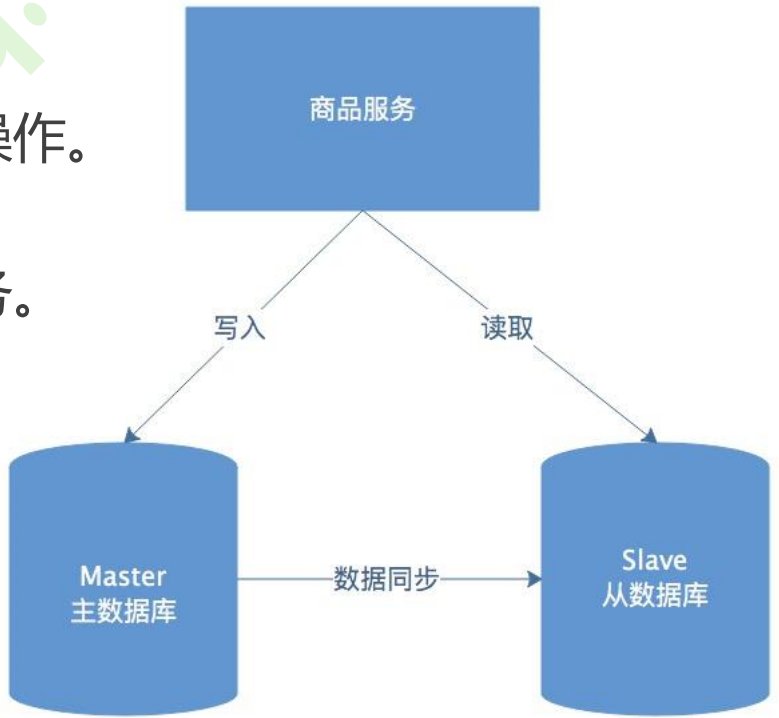
一、CAP理论

P – 分区容忍性

商品信息读写要满足分区容忍性就是要实现如下目标:

(1) 主数据向从数据库同步数据失败不影响读写操作。

(2) 一个节点挂掉不影响另一个节点对外提供服务。



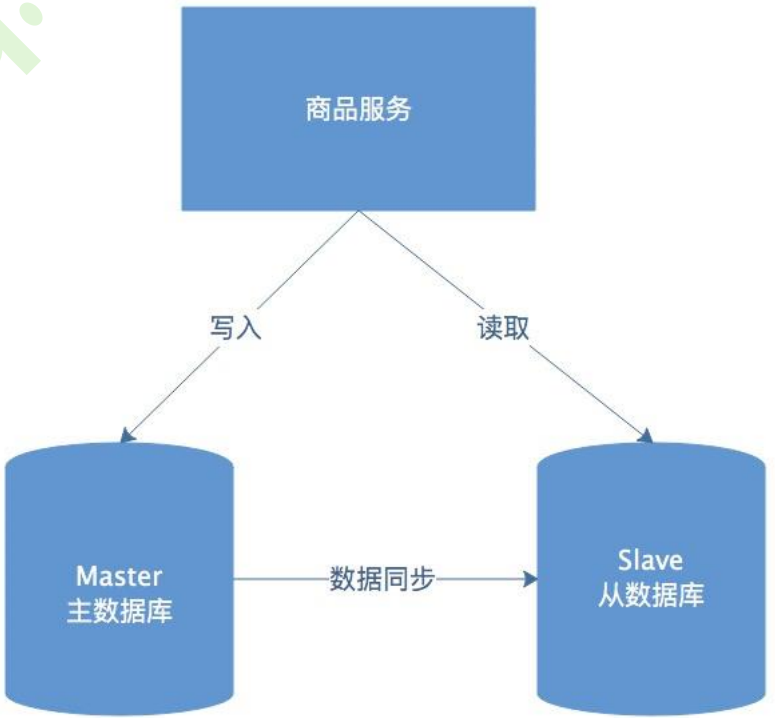
一、CAP理论

P – 分区容忍性

如何实现分区容忍性?

（1）尽量使用异步取代同步操作，例如使用异步方式将数据从主数据库同步到从数据库，这样节点之间有效的实现松耦合。

（2）添加从数据库节点，其中一个节点挂掉其它节点提供服务。



一、CAP理论

P – 分区容忍性

分布式分区容忍性的特点：

（1）分区容忍性是分布式系统具备的基本能力。

二、CAP组合方式

在所有的分布式事务场景中不会同时具备CAP三个特性，因为在具备了P的前提下C和A是不能共存的。

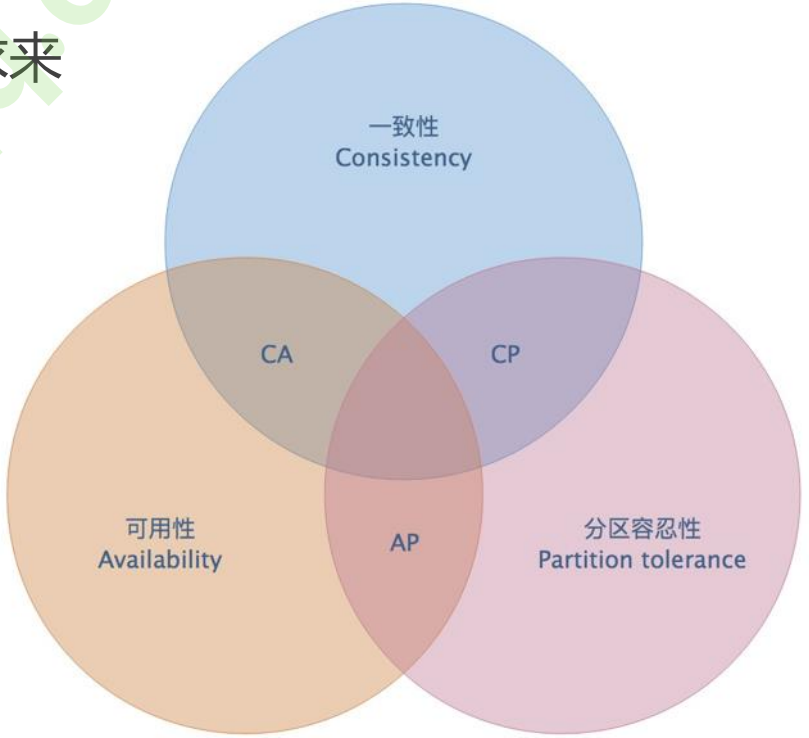
如果要实现C则必须保证数据一致性，在数据同步的时候为防止向从数据库查询的不一致则需要从数据库锁定，待完成同步之后解锁，如果同步失败从数据库要返回错误信息或超时信息。

如果要实现A则必须保证数据可用性，不管任何时候都可以向从数据库进行查询数据，并且不能够返回错误信息或者超时信息

二、CAP组合方式

通过分析在满足P的前提下，C和A存在矛盾。

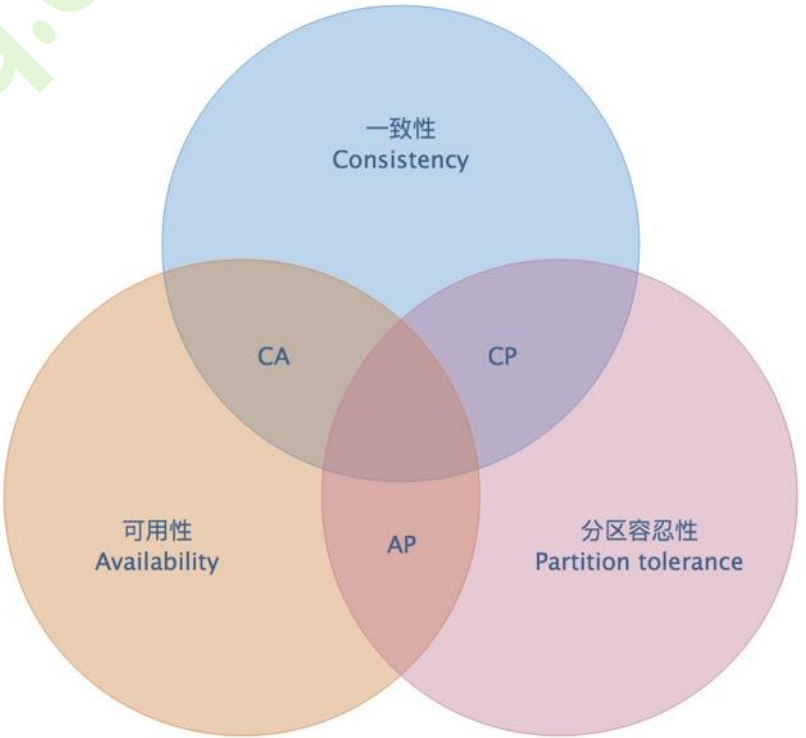
所以在生产中对分布式事务处理时需要根据需求来确定满足CAP的哪两个方面。



二、CAP组合方式

1.1 CA组合

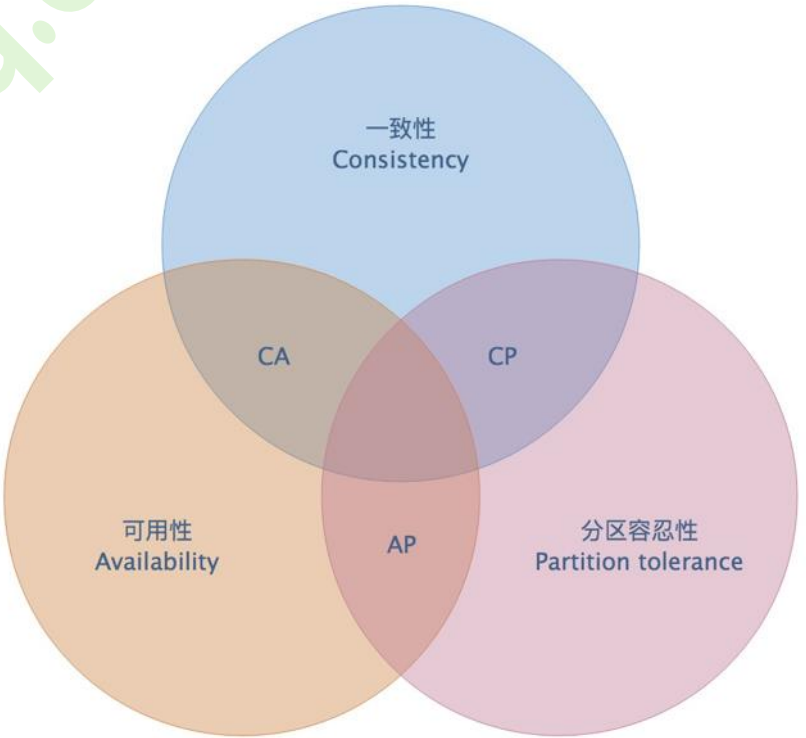
CA组合就是保证一致性和可用性，放弃分区容忍性，即不进行分区，不考虑由于网络不通或节点挂掉的问题。那么系统将不是一个标准的分布式系统，我们最常用的关系型数据库就满足了CA。



二、CAP组合方式

1.2 CP组合

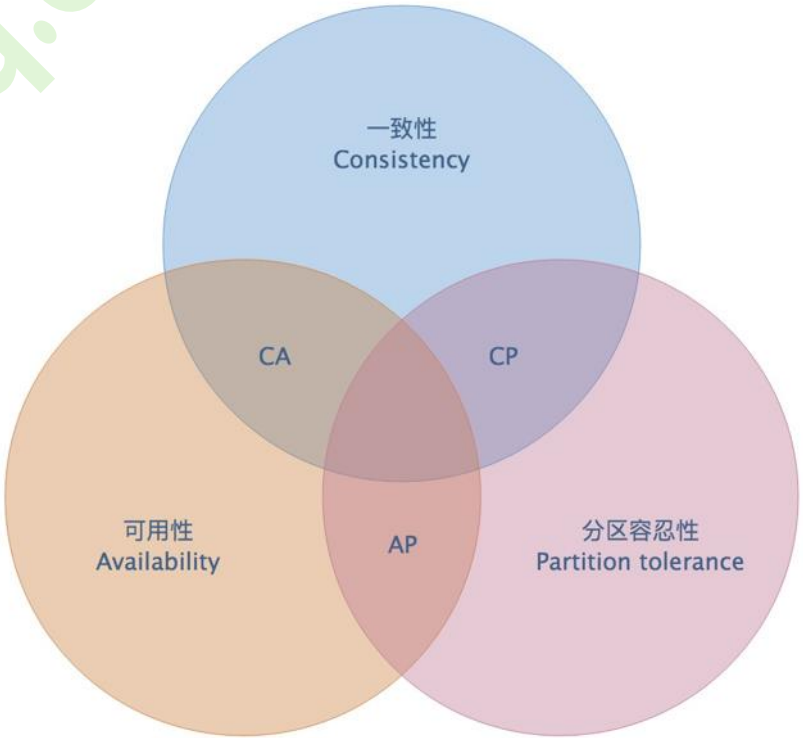
CP组合就是保证一致性和分区容忍性，放弃可用性。Zookerper就是追求强一致性，放弃了可用性， 还有跨行转账，一次转账请求要等待双方银行系统都完成整个事务才能完成。



二、CAP组合方式

1.3 AP组合

AP组合就是保证可用性和分区容忍性，放弃一致性。这是分布式系统设计时的选择。



三、小结

通过上面我们学习了CAP的基础理论知识，CAP是一个已经证实的理论：一个分布式系统做多只能满足CAP中的两项，为达到良好的响应性能来提高用户体验，因此一般会做出如下选择：*保证A和P，舍弃C强一致性，保证最终一致性*。

分布式事务的BASE理论

大道至简 悟在天成

Andy | 悟纤

前言

CAP理论告诉我们分布式系统最多只能满足CAP中的两项，其中AP在实际应用中较多。AP舍弃一致性，保证可用性和分区容忍性，但在实际生产中很多场景都要实现一致性，比如前面举的例子主数据库向从数据库同步数据，即使不要一致性，但是最终也要将数据同步成功来确保数据一致，这种一致性和CAP的一致性不同。

CAP中的一致要求在任何时间查询每个节点数据都必须一致，它强调的是强一致性，但是最终一致性是允许在一段时间内每个节点的数据不一致，但是经过一段时间每个节点的数据必须一致，它强调的是最终数据的一致性。

一、BASE理论

BASE是Basically Avaliable（基本可用）、Soft state（软状态）和Eventually consistent（最终一致性）三个短语的缩写。

BASE理论是对CAP中AP的一个扩展，通过牺牲强一致性来获得可用性，当出现故障部分不可用时需要确保核心功能可用，允许数据在一段时间内是不一致的，但最终达到一致状态。满足BASE理论的事务，我们称之为“柔性事务”。

一、BASE理论

1.1 BA – 基本可用

分布式系统在出现故障时，允许损失部分可用功能，保证核心功能可用。举个例子：电商网站交易付款出问题了，商品依然可以正常浏览。

1.2 S-软状态

由于不要求强一致性，所以BASE允许系统中存在中间状态（也叫软状态），这个状态不影响系统可用性，如订单的“支付中”、“数据同步中”等状态，待数据最终一致后状态改为“成功”状态。

1.3 E - 最终一致性

经过一段时间，所有节点数据都会达到一致。如订单中的“支付中”状态，最终会变成“支付成功”或者“支付失败”，使订单状态与实际交易结果达成一致，但需要一定时间的延迟和等待。