# 分布式锁初探

## 为什么需要分布式锁？

需要保证在分布式场景下只有一个线程同时在执行一个代码片段或者一个资源同时只有一个线程在使用。

常见的场景: 分布式任务，秒杀。

## 需要什么样的分布式锁?

高可用

有失效时间（避免死锁）

可重入（结合具体业务）

阻塞（结合具体业务）

性能高

公平锁（需要？）

## 实现

### 基于数据库实现

1. 基于唯一索引

我们知道，数据库可以通过唯一索引来保证数据的唯一性，我们可以利用数据库的这个特点来设计我们的锁。

通过向数据库中插入数据，插入成功即获得锁，如果已经存在记录，则插入失败，无法获得锁。

解锁操作删除对应的记录即可。

通过以上的思路，我们可以实现一个最简单的分布式锁。

表结构。

CREATE TABLE `simple\_lock` (

`id` bigint(20) NOT NULL AUTO\_INCREMENT,

`lock\_key` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `uk\_key` (`lock\_key`) USING BTREE

) ENGINE=InnoDB AUTO\_INCREMENT=930 DEFAULT CHARSET=utf8

通过lock\_key来标识锁。

核心代码:

加锁操作:

public boolean tryLock() {  
 Connection connection = null;  
 PreparedStatement preparedStatement = null;  
 try {  
 connection = getConnection();  
 preparedStatement = connection.prepareStatement(*LOCK\_SQL*);  
 preparedStatement.setString(1, key);  
  
 int i = preparedStatement.executeUpdate();  
 return i != 0;  
 } catch (Exception e) {  
 …  
 } finally {  
 …

}  
 return false;  
}

解锁操作:

@Override  
public void unlock() {  
 Connection connection = null;  
 PreparedStatement preparedStatement = null;  
 try {  
 connection = getConnection();  
 preparedStatement = connection.prepareStatement(*UNLOCK\_SQL*);  
 preparedStatement.setString(1, key);  
 preparedStatement.executeUpdate();  
 } catch (Exception e) {  
 …

} finally {  
 …

}  
}

虽然简单实现了，但是存在很多问题

没有做到高可用，可以通过主备数据库来解决。

没有失效时间，可以通过添加定时任务定时扫描数据库来使数据失效。

不可重入，可以通过增加锁标识字段来解决。

阻塞锁，可以通过不断尝试加锁来解决。

公平锁？