```
    // 检查缓存
    Object cachedValue = cacheService.get(key);
    if (cachedValue != null) {
        return cachedValue;
    }

    // 从数据库中加载数据
    Object data = databaseService.loadData(key);

    // 更新缓存
    cacheService.put(key, data);

    return data;
} finally {
    // 释放锁
    lock.unlock();
}
```

这些只是一些 Redisson 分布式锁在 Java 中的使用场景示例。Redisson 分布式锁可以用于各种场景来保护共享资源并防止并发冲突，从而提高应用程序的性能和可靠性。

以下是一些有关 Redisson 分布式锁的额外提示：

* 使用 Redisson 分布式锁时，请务必设置合理的过期时间，以避免死锁。
* Redisson 分布式锁是可重入的，这意味着同一个线程可以多次获得同一把锁。但是，请注意不要在不必要的情况下持有锁太长时间，以免降低应用程序的性能。
* Redisson 分布式锁是公平的，这意味着每个线程都有机会获得锁。但是，在某些情况下，可能需要使用非公平锁来优先考虑某些线程。

设置一定的超时时间
---

1. 使用 lock.lock(long timeout, TimeUnit unit) 方法

```
RedissonClient redissonClient = RedissonClient.create();
RLock lock = redissonClient.getLock("myLock");

try {
    // 设置超时时间 10 秒，并获取锁
    boolean acquired = lock.tryLock(10, TimeUnit.SECONDS);
    if (acquired) {
        // 执行业务逻辑
        // ...
    } else {
        // 未能获得锁，处理超时情况
        System.out.println("未能获得锁，请稍后再试");
    }
} finally {
    // 释放锁
    if (lock.isLocked() && lock.isHeldByCurrentThread()) {
        lock.unlock();
    }
}

```

2. 使用 lock.lock(AtomicReference leaseTime, TimeUnit unit) 方法

```
RedissonClient redissonClient = RedissonClient.create();
RLock lock = redissonClient.getLock("myLock");

AtomicReference<Long> leaseTime = new AtomicReference<>(10L lient
= RedissonClient.create();
RLock lock = redissonClient.getLock("myLock");

try {
    // 设置动态最大超时时间和过期时间，并获取锁
    boolean acquired = lock.tryLock(r -> {
        // 根据实际情况计算最大超时时间
        long timeout = calculateTimeout();
        return timeout;
    }, TimeUnit.SECONDS, r -> {
        // 根据实际情况计算过期时间
        long expireTime = calculateExpireTime();
        return expireTime;
    }, TimeUnit.SECONDS);
    if (acquired) {
```

```
        // 执行业务逻辑
        // ...

        // 续期锁
        while (true) {
            long timeRemaining =
lock.remainingLeaseTime(TimeUnit.SECONDS);
            if (timeRemaining <= 5) {
                long newLeaseTime = calculateTimeout();
                long newExpireTime = calculateExpireTime();
                if (lock.tryLock(newLeaseTime, TimeUnit.SECONDS,
newExpireTime, TimeUnit.SECONDS)) {
                    break;
                }
            }
        }
    } else {
        // 未能获得锁，处理超时情况
        System.out.println("未能获得锁，请稍后再试");
    }
} finally {
    // 释放锁
    if (lock.isLocked() && lock.isHeldByCurrentThread()) {
        lock.unlock();
    }
}

```

请注意，在实际使用中，您需要根据具体的业务需求来选择合适的设置最大超时时间和过期时间的方法。

以下是一些有关 Redisson 分布式锁设置最大超时时间和过期时间的额外提示：

* 最大超时时间和过期时间是两个不同的概念。最大超时时间是客户端尝试获取锁的最长时间，而过期时间是锁的有效时间。
* 设置最大超时时间可以防止客户端长时间等待锁，而设置过期时间可以防止由于意外情况导致锁无法释放而造成死锁。
* 在使用动态最大超时时间或过期时间函数时，请确保该函数能够根据实际情况返回合理的数值。