专业技能

熟悉Redis及其线程模型、持久化机制，熟悉缓存雪崩、穿透、击穿解决方案

Redis基础

Redis的数据结构有哪些，怎么实现的

字哈列集序位H地

字符串（String）： Simple Dynamic Strings实现，允许动态地扩展缩小长度

哈希表（Hash）：通过哈希值将数据分布到不同的槽位上，以提高查找效率。

列表（List）：双向链表结构实现。在两端执行插入和删除操作的效率高。

集合（Set）：哈希表结构实现，支持高效的添加、删除和判断元素是否存在的操作。

有序集合（Sorted Set）：跳表和哈希表结合。跳表提供有序性，哈希表提供快速查找。

位图（Bitmap）

HyperLogLog：用于估计基数（集合中不重复元素的数量）

地理空间索引（Geospatial Index）： ZSET的分数表示地理位置的坐标。

Redis为什么快？

内单非数持协

内存存储： Redis主要将数据存储在内存中，内存的读写速度远高于磁盘

单线程模型： Redis采用单线程模型，通过事件循环来处理多个客户端请求。避免了多线程之间的锁竞争，减少了上下文切换的开销。

非阻塞式 I/O： Redis使用非阻塞式的I/O操作，使得在进行网络通信和文件操作时，能够更好地利用系统资源，提高吞吐量。

高效的数据结构

持久化选项： Redis支持多种持久化选项，包括快照和追加式文件（AOF）。

优化的网络协议： Redis使用自定义的协议进行客户端和服务器之间的通信。这种协议是文本协议，易于理解和调试。同时，它也是二进制安全的，支持存储和传输任意类型的数据。

**详细讲一讲IO多路复用**

注册监听通知操作

Select poll epoll kqueue

1.定义：

IO多路复用它允许单一的线程同时监听多个IO通道上的事件。通过这种方式，一个线程能够有效地管理多个IO通道，从而减少线程的数量，提高系统的性能和资源利用率。如果一个IO通道上的IO操作是阻塞的，可能会影响整个IO多路复用的性能，因此通常建议使用非阻塞IO。

2.常见的IO多路复用机制：

select：通过select系统调用，允许程序监视一组文件描述符，一旦其中一个文件描述符准备好进行IO操作，就通知程序进行相应的读写操作。

poll：与select类似，但采用链表，避免了select中的文件描述符数量限制。

epoll：是Linux下效率最高的IO多路复用机制，使用红黑树实现高效的事件查找。epoll的性能随着文件描述符的增加而近乎线性增长，适用于高并发的场景。

kqueue：是BSD和macOS中的IO多路复用机制，与epoll类似，通过红黑树实现，具有高性能和可扩展性。

3. 工作流程：

程序通过IO多路复用调用（如select、epoll）将一组文件描述符注册到内核中。

内核监视这些文件描述符上的事件，当有IO事件发生时，通知应用程序。

程序通过遍历通知，找到准备好的文件描述符，进行相应的IO操作。

这样，一个线程就能够同时管理多个IO通道，而不需要为每个IO通道创建一个独立的线程。

Redis 消息队列

消息队列的选型，Kafka和RabbitMQ的区别，各自的应用场景

Kafka更稳定大规模

RabbitMQ更灵活易用

Kafka:

设计理念：

Kafka设计为分布式、高可用、持久化的消息系统。它的主要目标是提供高吞吐量、低延迟和容错性。

数据模型：

Kafka采用发布-订阅的模型，消息以主题（Topic）为单位组织。消息存储在分布式日志中，允许多个消费者并行消费。

可靠性：

Kafka强调消息的持久性，一旦消息被写入，就会被持久存储。它支持多副本复制，确保数据可靠性。

性能：

Kafka在高吞吐量和低延迟方面表现出色，特别适合大规模数据流处理。

适用场景：

日志收集与分析、事件溯源、大数据处理、实时数据流处理等。

RabbitMQ:

设计理念：

RabbitMQ设计为一个灵活、易用的消息队列系统。它提供多种消息传递模式，包括点对点和发布-订阅。

数据模型：

RabbitMQ支持多种消息传递模式，包括点对点、发布-订阅和路由。消息存储在队列中，每个队列有一个或多个消费者。

可靠性：

RabbitMQ提供可靠性传递，支持持久化消息。它可以通过不同的交换机类型和队列设置来实现不同的消息路由策略。

性能：

RabbitMQ在一般情况下表现良好，但相对于Kafka，它在处理大量数据和高吞吐量方面可能稍显逊色。

适用场景：

任务队列、点对点通信、发布-订阅、应用解耦等。

如何选择：

数据处理特点：

如果需要处理大规模的实时数据流、日志等，Kafka可能是更合适的选择。

如果强调消息传递的可靠性和灵活性，以及更广泛的消息模型，RabbitMQ可能更适用。

复杂性和易用性：

RabbitMQ通常被认为更易于入门和使用，适用于相对简单的消息场景。

可靠性需求：

如果对消息的持久性和数据不丢失有较高要求，Kafka可能更符合需求。

Redis线程模型

Redis持久化机制

Redis 缓存

Redis 缓存雪崩

Redis缓存穿透

Redis缓存击穿

加了过期时间的LRU缓存，缓存里的内容x秒后会过期，如何实现

实现带有过期时间的LRU（Least Recently Used）缓存通常需要结合两个主要概念：LRU算法和定时过期。以下是一个简单的Python示例，使用collections.OrderedDict来实现LRU缓存，并通过定时任务来处理过期时间：

这个示例中，LRUCacheWithExpiration类继承自OrderedDict，通过使用OrderedDict的有序字典来实现LRU缓存，同时通过定时任务检查过期时间并清理过期的缓存项。注意，这里使用了一个简单的线程来定期清理过期项，你可能需要根据实际需求进行调整，例如使用定时器、协程等机制。