明确目的：

Redis在Java中怎么使用？有哪些应用？涉及什么？【缓存，分布式锁】

锁可以设置过期时间吗？红锁？

缓存的三个问题，以及解决方案（缓存击穿，缓存穿透，缓存雪崩）

为什么他会高性能？（基于内存，单线程，线程模型，高效的数据结构）

作为缓存，他的回收策略是什么样的？

有持久化策略吗？

**多级缓存**的概念？

**应用部分**：

**依赖添加**：

<dependency>

<groupId> org.springframework.boot </groupId>

<artifactId> spring-boot-starter-data-redis </artifactId>

</dependency>

Redis在Spring中使用的依赖

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-pool2</artifactId>

</dependency>

连接池（复用连接对象能提高性能）依赖🡪加上这个，Redis连接池才有效，默认客户端是Lettuce

**配置文件**：（这里展示的是单体环境的配置）

spring:

redis:

host: localhost

port: xxxx

password: xxxx

database: 0

lettuce:

pool:

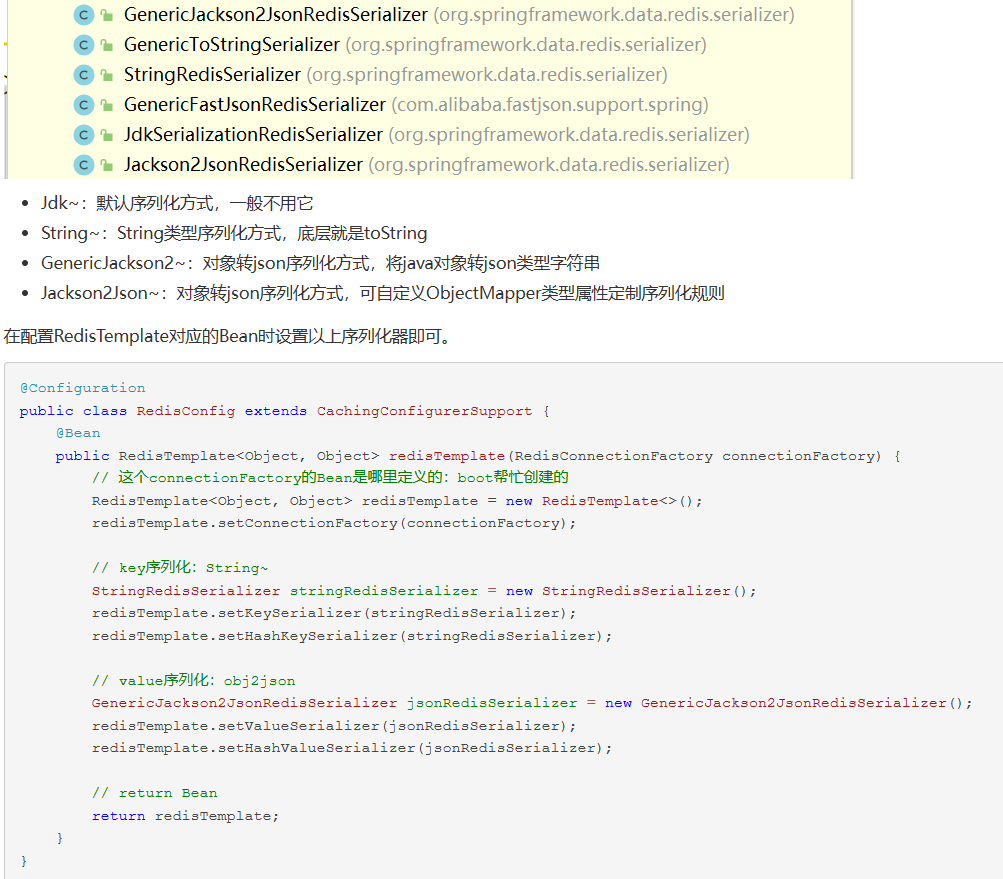
max-idle: 16

max-active: 32

min-idle: 8

**设置序列化方式**：

因为Redis并不“认识”Java对象，所以他们之间的通信需要将Java对象转换成Redis认识的形式，如“String字符串”，“二进制格式”，“json字符串”



一般的json序列化用GenericJackson2JsonRedisSerializer即可🡪但对于像LocalDateTime，格式比较特殊的类，默认的json序列化方式可能会出错

因此**Jackson2JsonRedisSerializer能自定义序列化规则**就有必要了；

应用于缓存时，建议使用上述的RedisTemplate进行操作，但对于分布式锁，建议用上一下的Redisson操作

**Redisson对应依赖**：

<dependency>

            <groupId>org.redisson</groupId>

            <artifactId>redisson</artifactId>

            <version>3.6.5</version>

        </dependency>

**配置Redisson客户端**【也可以通过yml文件进行配置】：

Config config = new Config();

config.useSingleServer()【**单体模式的配置**】

.setAddress("redis://127.0.0.1:6379")

.setPassword("123456")

.setDatabase(0);

**操作（使用Redison也能实现缓存，所以最好用Redisson客户端操作Redis🡪且提供的锁操作功能更加完善🡪但缺点是只能用jdk自带的序列化器进行序列化-->有看门狗，红锁等机制，待完善说明）**：

[Redis实战之Redisson使用技巧详解，干活！-redis原理详解 (51cto.com)](https://www.51cto.com/article/743175.html)

详细看这个文章🡪提供了5种基本数据类型的操作过程，以及布隆过滤器使用，各种环境锁的使用（集群，主从，单体环境），**用的时候复制即可（注意，操作ZSet结构时需要实现Comparable接口；需要序列化时需要实现serializable接口）**；

**缓存的问题，解决方法**：

缓存穿透：访问不存在/无效key，导致大量请求打向数据库🡪在缓存中给无效key设置null值，设置布隆过滤器(用位图实现，占用内存小，但是会有误差)，再或者**对传进来的参数进行限制（比如说有效数据是 1-10，那么只要请求的参数小于1或者大于10都直接拒绝请求）**

缓存击穿：短时间高频访问缓存中不存在/已经失效的key，导致大量请求打向数据库🡪通过分布式锁确保只有一个线程能获取资源

缓存雪崩：Redis崩溃了🡪设置多级缓存，避免大量请求同时访问数据库（**存在缓存一致性问题需要解决**）

**高效的数据结构：**

**5种基本数据结构**：动态字符串(记录字符串长度)，List(双向链表/压缩列表【类似于数组，用连续内存空间进行存储，只适合少量数据使用，内存的**空间局部性**会更好】)，Hash（数组+链表组成的哈希表/压缩列表），Set(整数集合【只能存放整数类型数据，连续内存空间】/哈希表)，ZSet(哈希表+跳表/压缩列表)

**4种特殊类型**：bitmap位图(连续内存空间,每个“位”用于表示一个元素的状态)，hyperloglog（可以大致估计集合中不重复的元素个数，具体实现不了解），GEO(用于存储“地理位置”，利用ZSet实现，基于“经纬度”计算出用于排序的score)，Stream（貌似可用来实现“消息队列”？）

不用Stream，只能用List实现消息队列有什么问题？🡪没有ACK确认机制，不知道消息传递后是否消费成功

但一般不建议拿Redis实现消息队列，**“持久性”**无法得到保证【Redis的单线程模型，是**不允许添加消息时，同步持久化消息队列的🡪但说不定会有业务能接收**】

**线程模型**：

连接socket->事件分发器->事件队列->事件处理器

事件处理器是单线程，没有“锁”等额外开销；且基于内存，不会出现“因IO阻塞线程”的情况【或者说需要用上io的情况，交给其他线程去做，比如写AOF文件，RDB文件之类的🡪**对于Redis来说，应该不用保证一个Redis指令完成之前写入AOF文件？**】

**内存回收策略**：

**Redis使用的是惰性删除+定时删除**

定时删除：创建一个定时任务，再key过期时删除🡪过期key某段时间很多时，cpu开销比较大

惰性删除：过期key在再次访问时才会删除🡪不访问仍然会占用内存

定时删除：Redis会定期从会过期的key中随机取一定数量的key，并删除已经过期的key🡪和惰性删除一起使用，解决惰性删除的缺点

**拓展**：

**看门狗**：

持有锁时间为1s，但是业务执行时间需要2s，此时为了保证线程安全，需要对持有锁的时间进行“续费”，这就是看门狗机制（该机制只对“没有设置过期时间的锁”生效）

**红锁机制**：

线程a在节点a上分布式锁成功，但是节点a马上宕机，会导致该分布式锁🡪因此线程a在Redis集群模式上分布式锁时，需要在 (n/2)+1个集群节点都上锁成功，才认为线程a上锁成功

**处理“哈希冲突”的方法**：链式哈希，还有呢？

**Redis怎么处理hash扩容的情况的**？

默认情况，元素个数/桶>0.75（默认负载因子）的情况，会进行rehash；在旧hash1存在的情况，会创建一个新hash2，之后会将hash1中的元素逐步迁移到hash2中🡪在rehash途中进行读操作时，会先从hash1中读取，再去hash2中读取，确保不丢失数据

**Redis的有序集合ZSet的数据结构**（哈希表和跳表）

哈希表负责“时间复杂度1”的get/put操作

跳表则是负责“按score获取范围内的一段数据”🡪这种**功能为什么不用b+树或者红黑树去实现呢？**

因为跳表一般需要遍历链表才能获取一定范围内的数据🡪但是红黑树只能通过树的遍历，效率差，范围查询效率差

此外，b+树因为叶节点中存在指针指向相邻的叶节点，是可以通过遍历链表进行“范围查询”的，但是其在进行“插入/删除/修改”操作时，可能会涉及子节点的分裂，开销会更大，因此没用上b+树（但是，在**MySQL中，因为不会将数据以Map结构存储从而进行快速查找，所以B+树的搜索效率是十分重要的，所以MySQL才不能用跳表**）

**持久化策略**：

AOF日志：记录执行的Redis指令（持久化效率较快，重启时要执行Redis指令，启动较慢）

RDB快照：记录内存快照（持久化效率较慢，启动快）

AOF+RDB混合使用：保留了双方的优点，即“持久化较快，启动不至于太慢”【每隔一段时间才进行RDB持久化，这段期间用AOF的方式进行持久化】

**刷盘策略**：

对于AOF日志文件来说，有三种刷盘策略，分别是：

方案1：每执行一个Redis语句就刷一次盘

方案2：将Redis语句写在缓冲区中，每隔1s进行一次刷盘

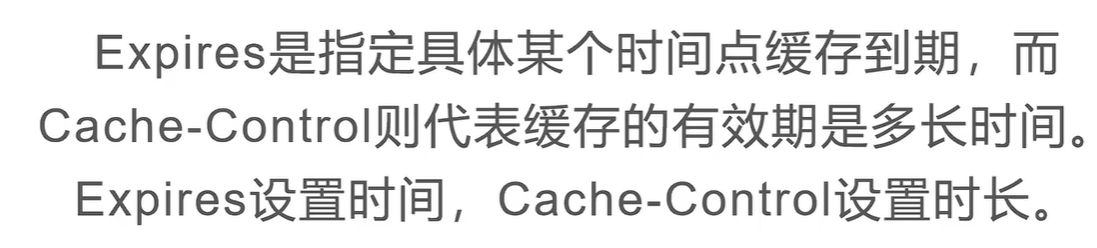
方案3：将Redis语句写在缓冲区中，让os决定什么时候刷盘

但是，考虑到Redis的事件处理器是单线程的，因此方案1每执行一次Redis语句，就要进行一次io刷盘，是**完全不能接受**的

可能会有疑问，“**那Redis的持久性不就不能保证了吗？**”🡪是的，但是Redis的应用，比如“缓存”、“分布式锁”等等，对其**持久性要求并没有MySQL那些关系型数据库那么高**，因此是可以接受的

**多级缓存架构**：

**客户端层面**：利用HTTP头中的cache-control或expires字段添加客户端缓存（一般用来缓存常用的jpg文件等静态资源）



**CDN加速/缓存**：在上海请求广东网站资源时，会先从上海的CDN服务器中寻找有没有“请求资源的缓存”；没有的情况才会直接访问广东网站资源，再将其缓存在CDN服务器中🡪以便下次访问/其他人访问时能更高效地获取资源

**Nginx缓存**：Nginx是Web服务器的一种，可以做Tomcat服务器集群的代理，提供了“负载均衡”和“静态资源缓存”的功能【有些情况只用Nginx缓存，不用CDN加速也能满足需求】

以下是**业务数据层面的缓存，也分层次**：

**进程外缓存**：Redis常用于做分布式缓存

**进程内缓存：**如Caffeine等本地缓存

那么，**多级缓存有什么问题呢(一般来说是先访问本地缓存，再访问Redis缓存，节省了与Redis间的网络通信开销)**？🡪缓存的一致性，当缓存中的内容被修改时，不同机器的Caffeine本地缓存需要进行同步，Redis缓存也要进行同步修改