JAVA Summary

（大佬勿进）

面试真的很靠运气+实力，很多时候你准备的在充分也会崩。推荐书单：jvm 虚拟机（周志明）thinking in Java

项目相关：

项目最好准备 2-3 个这样，面试官一定会让你选一个项目讲！这是套路，一定要让面试官跟着你走！这样他会觉得你啥都会。以我为例，选了牛客网的中级项目课，进行改进。然后第二个项目做了一个 GitHub 上的商城 project。主要以牛客网项目为例，面试官一定会对你的 redis，异步框架感兴趣，你一定要结合海量知识点去分析为什么要用这些。***Eg***. 异步框架怎么会设计出来的，我一定会说，因为我看了 i/o 模型的设计模式，发现我的项目里如果使用同步 block 会有很大可能崩盘，所以我将他的思路用到了我的项目。然后面试官一定会问你 I/O 复用等，这时候就是你的 style 了。

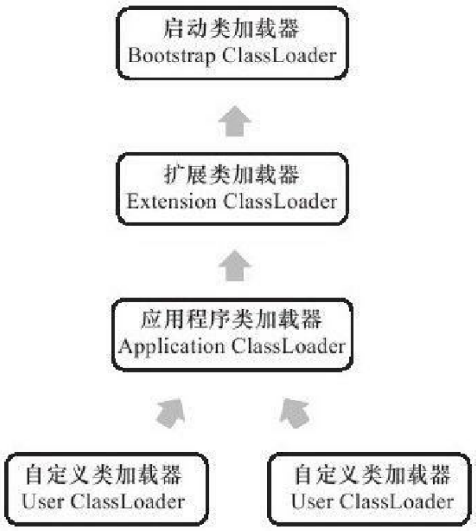
我感觉面试官要的是你的解决方案的能力，你的思考过程，而并不是 solution。

Java 基础

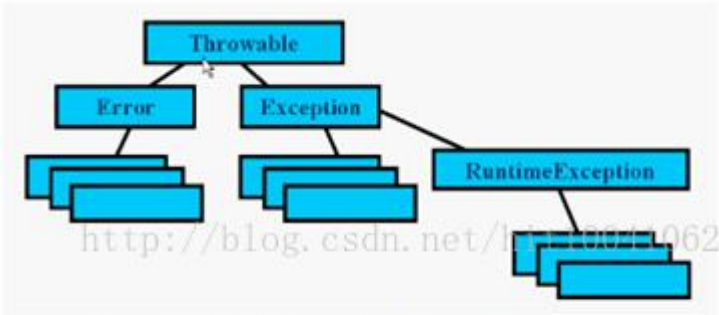
1. 重载和重写：
   重载：参数列表不一样和返回值类型无关（**次序不一样也可以**）
   重写：方法重写是在子类存在方法与父类的方法的名字相同,而且参数的个数与类型一样,返回值也一样的方法，**什么都的一样。子类方法权限要大于等于父类**
2. 成员变量：存在堆内存中，连人带马存在于堆
   局部变量：存在于栈中（stack frame 中）reference！！
   静态变量：存在于方法区中（常量池），是随着类加载而加载的，只执行一次，相当于“共享变量”。
3. 创建了一个对象在内存中发生了一些什么：
   a) 先将硬盘上指定位置的 Person.class 文件加载进内存。
   b) 执行 main 方法时，在栈内存中开辟了 main 方法的空间(压栈-进栈)，然后在 main 方法的栈区分配了一个变量 p。
   c) 在堆内存中开辟一个实体空间，分配了一个内存首地址值。
   假设 Java 堆中内存是绝对规整的，所有用过的内 存都放在一边，空闲的内存放在另一边，中间放着一个指针作为分界点的指示器，那所分配内存就仅仅是把那个指针向空闲空间那边挪动一段与对象大小相等的距离，这种分配方式称 为“指针碰撞”（Bump the Pointer）。如果 Java 堆中的内存并不是规整的，已使用的内存和空 闲的内存相互交错，那就没有办法简单地进行指针碰撞了，虚拟机就必须维护一个列表，记 录上哪些内存块是可用的，在分配的时候从列表中找到一块足够大的空间划分给对象实例，并更新列表上的记录，这种分配方式称为“空闲列表”（Free List）。
   d) 在该实体空间中进行属性的空间分配，并进行了默认初始化

e) 对空间中的属性进行显示初始化。
f) 进行实体的构造代码块初始化。
g) 调用该实体对应的构造函数，进行构造函数初始化。
h) 将首地址赋值给 p ，p 变量就引用了该实体。(指向了该对象)

4. 继承
   a) Super:代表子类所属父类的空间引用。
   b) 子类先调用父类的无参构造函数（Super（））
   c) 如果子类中重写了该方法，那么父类类型的引用将会调用子类中的这个方法，这就是动态连接
5. 接口和抽象类：
   a) 接口对外的功能，声明。
   b) 接口和抽象类中有抽象方法就必须重写。
   c) 接口中修饰符是固定的全是 public 的，而抽象类就没有
   d) 接口没有 main 方法，因此我们不能运行它。
   e) 抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是 public static final 类型的；
6. Object 类：
   a) 所有类的直接或间接父类
   b) 重写 equals 必须重写 hashcode 因为相同的对象必须返回相同的哈希值
7. String 类：（Stringbuffer，Stringbuilder 区别），不要求考虑线程安全的话 builder 快
8. 类加载（触发类的初始化）：
   a) 常量（编译期间放入常量池的静态字段）不会造成类的初始化。
   b) 因此通过其子类来引用父类中定义的静 态字段，只会触发父类的初始化而不会触发子类的初始化
9. 类加载双亲委派模型
   a) 启动类加载器（Bootstrap Classloader）来自虚拟机内部（）<HAVA_HOME>\lib：
   b) 其他的类加载器：扩展类加载器<HAVA_HOME>\lib\ext，应用程序加载器
   c) 过程：先检查是否已经被加载过，若没有加载则调用父加载器的 loadClass（）方 法，若父加载器为空则默认使用启动类加载器作为父加载器。如果父类加载失败，抛出 ClassNotFoundException 异常后，再调用自己的 findClass（）方法进行加载。



基础类调用用户代码，这时候还没有被加载。用 thread context classloader。



Throwable是所有异常的根，java.lang.Throwable

Error是错误，java.lang.Error

Exception是异常，java.lang.Exception

10. Java 异常：

exception 又分为 unchecked 和 checked，这个自己看哈

高并发：

写在前面：高并发呢，Java 这块主要是一些是这些，但是会涉及数据库优化，nginx 负载均衡，redis，cdn 等等技术，万变不离其宗我觉得就是减少对于不要同时对一个服务器访问，能用缓存尽量用缓存，能用线程池千万别去显式创建线程，能用不加锁的尽量不用。因为设计的技术栈很全所以要重点看。

1. Synchronize：monitor 监控的独占锁。（monitorenter 指令时，首先要尝试获取对象的锁。如果这 个对象没被锁定，或者当前线程已经拥有了那个对象的锁，把锁的计数器加 1，相应的，在 执行 monitorexit 指令时会将锁计数器减 1，当计数器为 0 时，锁就被释放）。另外，java 对其优化，加入了**自适应的自旋锁**：自适应意味着自旋的时间不再固定了，而是由前 一次在同一个锁上的自旋时间及锁的拥有者的状态来决定。如果在同一个锁对象上，自旋等 待刚刚成功获得过锁，并且持有锁的线程正在运行中，那么虚拟机就会认为这次自旋也很有 可能再次成功，进而它将允许自旋等待持续相对更长的时间，比如 100 个循环。另外，如果 对于某个锁，自旋很少成功获得过，那在以后要获取这个锁时将可能省略掉自旋过程，以避 免浪费处理器资源。还有一些锁粒度的粗化等等

2. ReentrantLock：比 syn 多了等待可中断、可实现公平锁，以及锁可以绑定多个条件，主要实现是 ***AQS，*** 源码自己看，通篇全是 CAS，底层维护一个类似于队列的东西。推荐并发编程网上面写的很好。
3. CAS：重点关注一个 ABA 问题。可以和 mysql 的 mvcc 一起看
4. 线程池：（源码自己看，有时间就看看）
   若当前线程池中线程数<corepoolsize，则每来一个任务就创建一个线程去执行。若当前线程池中线程数>=corepoolsize，会尝试将任务添加到任务缓存队列中去，若添 加成功，则任务会等待空闲线程将其取出执行，若添加失败，则尝试创建线程去执行这个任 务。若当前线程池中线程数>= Maximumpoolsize，则采取拒绝策略（有 4 种，1）abortpolicy 丢 弃任务，抛出 RejectedExecutionException 2）discardpolicy 拒绝执行，不抛异常 3）discardoldestpolicy 丢弃任务缓存队列中最老的任务，并且尝试重新提交新的任务 4） callerrunspolicy 有反馈机制，使任务提交的速度变慢）。
5. Volatile：http://ifeve.com/java-memory-model-4/，这个网站很详细，我的总结就是利用内存屏障的方法来禁止重排序以达到 volatile 的内存语义。
6. CountDownLatch： CountDownLatch 是通过“共享锁”实现的。在创建 CountDownLatch 时， 会传递一个 int 类型参数，该参数是“锁计数器”的初始状态，表示该“共享锁” 最多能被 count 个线程同时获取，这个值只能被设置一次， 而且 CountDownLatch 没有提供任何机制去重新设置这个计数值。主线程必须在启 动其他线程后立即调用 await()方法。这样主线程的操作就会在这个方法上阻 塞，直到其他线程完成各自的任务。当某线程调用该 CountDownLatch 对象的 await()方法时，该线程会等待“共享锁”可用时，才能获取“共享锁”进而继 续运行。而“共享锁”可用的条件，就是“锁计数器”的值为 0！而“锁计数器” 的初始值为 count，每当一个线程调用该 CountDownLatch 对象的 countDown() 方法时，才将“锁计数器”-1；通过这种方式，必须有 count 个线程调用 countDown()之后，“锁计数器”才为 0，而前面提到的等待线程才能继续运行！
7. Thread 类：
   Sleep，让线程休眠阻塞一段时间，时间满了不会立刻执行。唤醒后进入就绪状态
   Yield：让同等优先级的线程运行,也有可能不停下

Join：交出对象持有的锁，阻塞状态。
Interrupt：中断状态设置为 true
Isinterrupt：检测当时的中断状态
Interrupted：清楚当时的中断状态

8. 阅读源码区：不是死看，要学会他的设计模式，这样和面试官才有谈的资本，（当然自己看懂了更好，自己写项目也要用）

Collection
├List
│ ├LinkedList：双端队列。
│ ├ArrayList
│ └Vector
│ └Stack
└Set
Map
├Hashtable
├**HashMap**（!!!!!!!! 看懂了哈，还有 concurrenthashmap，和前面的一起看）
└WeakHashMap

GC 相关：

1. Finalize() 方法：可达性发现之后标记，（需要调用 finalize（）方法）然后进入 F-queue 之后在进行标记。属于自救。只能这么搞一次。
2. 方法区的回收：无用的类，（classloader 已经被回收）
3. 算法：
   标记清除：1. 内存碎片 2.效率低迷
   标记整理（老年代）：向一端移动。
   复制算法（新生代）eden 和两块 siriovor。用 eden 和 一块 survivor，然后复制。8：1
   分代收集算法；前面几个的大整合。
   安全区和安全点开始 GC
4. 垃圾收集器：
   1. Serial 收集器：stop the world。
   2. parNew： 多线程版本，不好
   3. parallel： stop the world。 吞吐量
   4. serial old：标记整理
   5. CMS：初始标记（CMS initial mark） 并发标记（CMS concurrent mark） 重新标记（CMS remark） 并发清除（CMS concurrent sweep）
   6. G1 收集器：分代收集不需要配合，空间整合，可预测停顿通过一个预先列表。初始标记（Initial Marking） 并发标记（Concurrent Marking） 最终标记（Final Marking）通过 remember set1 筛选回收（Live Data Counting and Evacuation）
5. 内存分配：
   Eden 满了触发 minor GC 老年代满了 full gc。 每熬过一次+1 岁。

动态担保：只使用其中一个 Survivor 空间来作为轮换备份，因此当出现大量对象在 Minor GC 后仍然存活的情况（最极端的情况就是内存回收后新生代中所有对

象都存活），就需要 老年代进行分配担保，把 Survivor 无法容纳的对象直接进入老年代。与生活中的贷款担保类 似，老年代要进行这样的担保，前提是老年代本身还有容纳这些对象的剩余空间，一共有多 少对象会活下来在实际完成内存回收之前是无法明确知道的，所以只好取之前每一次回收晋 升到老年代对象容量的平均大小值作为经验值，与老年代的剩余空间进行比较，决定是否进 行 Full GC 来让老年代腾出更多空间。

Database：

索引相关：

1. 数据结构 B 树，b+ 树，链接：https://blog.csdn.net/aqzwss/article/details/53074186
2. 为什么不用红黑树做索引，磁盘 IO 预读，可以了解一下。
3. 索引失效：1.查询条件包含 or 2. 组合索引，不是使用第一列索引，索引失效 3. like 以%开头 4. 如何列类型是字符串，where 时一定用引号括起来，否则索引失效 5.当全表扫描速度比索引速度快时，mysql 会使用全表扫描，此时索引失效
4. 最左前缀匹配原则，非常重要的原则，mysql 会一直向右匹配直到遇到范围查询(>、<、between、like)就停止匹配
5. 聚集索引和非聚集索引：（**自己总结；聚集索引表示表中存储的数据按照索引的顺序存储，检索效率比非聚集索引高，但对数据更新影响较大。非聚集索引表示数据存储在一个地方，索引存储在另一个地方，索引带有指针指向数据的存储位置，非聚集索引检索效率比聚集索引低，但对数据更新影响较小。**）聚集索引表记录的排列顺序和索引的排列顺序一致，所以查询效率快，只要找到第一个索引值记录，其余就连续性的记录在物理也一样连续存放。聚集索引对应的缺点就是修改慢，因为为了保证表中记录的物理和索引顺序一致，在记录插入的时候，会对数据页重新排序。非聚集索引：非聚集索引制定了表中记录的逻辑顺序，但是记录的物理和索引不一定一致，两种索引都采用 B+树结构，非聚集索引的叶子层并不和实际数据页相重叠，而采用叶子层包含一个指向表中的记录在数据页中的指针方式。非聚集索引层次多，不会造成数据重排。
6. 怎么判断一个 mysql 中 select 语句是否使用了索引，可以在 select 语句前加上 explain， 比如 explain select * from tablename;返回的一列中，若列名为 key 的那列为 null，则没有使 用索引，若不为 null，则返回实际使用的索引名。让 select 强制使用索引的语法：select * from tablename from index(index_name)；

7. 第一范式（1NF）：属性不可分
   第二范式（2NF）：符合 1NF，并且非主属性完全依赖于码。
   第三范式（3NF）：符合 2NF，并且，消除传递依赖。
8. JDBC
   连接数据库的基本过程
   加载驱动程序
   创建连接对象
   创建语句对象

编写 SQL 语句
使用语句对象执行 SQL 语句
如果有结果集对结果集进行处理
关闭结果集对象（如果有），关闭语句对象，关闭连接对象

9. 主从复制：
   master 将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events，可以通过 show binlog events 进行查看）。
   slave 将 master 的 binary log events 拷贝到它的中继日志(relay log)。
   slave 重做中继日志中的事件，将改变反映它自己的数据。
10. RR 和 RC
    Innodb 解决幻读的问题是用的 gap 锁+行级锁！不是 MVCC!!! 记住了，建议翻墙去看官方文档，lz 打不动字了。。

HTTP：

1. Https：
   总的来说，https 比 http 多了一个 SSL。先用非对称加密再用对称加密。过程如下：
   1、客户端请求建立 SSL 链接，并向服务端发送一个随机数 – Client random 和客户端支持的加密方法，比如 RSA 公钥加密，此时是明文传输。
   2、服务端回复一种客户端支持的加密方法、一个随机数 – Server random、授信的服务器证书和非对称加密的公钥。
   3、客户端收到服务端的回复后利用服务端的公钥，加上新的随机数 – Premaster secret 通过服务端下发的公钥及加密方法进行加密，发送给服务器。
   4、服务端收到客户端的回复，利用已知的加解密方式进行解密，同时利用 Client random、Server random 和 Premaster secret 通过一定的算法生成 HTTP 链接数据传输的对称加密 key
   5、Java 配置 https：
   keytool -genkey -alias tomcat -storetype PKCS12 -keyalg RSA -keysize 2048 -keystore keystore.p12 -validity 3650 在 spring application.properties 中配置。
   6、**https 抓包**
2. HTTP 详解：
   1. 1.0 和 1.1 区别：长连接！重点看长连接怎么区分 Transfer-Encoding: chunked 和 Content-Length
   2. 1.1 和 2.0 的区别：***i/o 复用模型*** 再次提醒一下各位朋友，这个东西很重要，redis 里面也用到了，所以一定要摸清楚，不要以为不重要。
   3. Post 和 get 区别
   4. 码：4xx 3xx 5xx 重点看！
   5. TCP/IP： 三次握手和四次挥手：为什么要三次握手？（朋友们自己找~~~~）

Servlet：

Servlet 工作原理：客户端发起一个请求，servlet 调用 service()方法时请求进行 响应，service 对请求的方式进行了匹配，选择调用 dopost 或者 doget 等这些方法，然后进 入对应方法中调用逻辑层上的方法，实现对客户的响应。
2）响应客户请求：对于用户到达 servlet 的请求，servlet 容器会创建特定于

该请求的 servletRequest 和 servletresponse 对象， 然后调用 servlet 的 service 方法，service 方法从 servletrequest 对象中获得客户请求的信息，处理该请求，并通过 servletresponse 对象向客户返回响应消息。3）终止：当 web 应用终止 或者 servlet 容器终止或 servlet 容器重新装载 servlet 新实例时，servlet 容器会调用 servlet 对象的第 destory 方法，在 destory 方法中可释放 servlet 占用的资源。

字节是一种计量单位，表示数据量多少，他是计算机信息技术用于计量存储容量的一种 单位。字符：计算机中使用的文字和符号。不用编码里，字符和字节对应关系不同。

1）ASCII 中，一个英文字母（不论大小写）占一个字节，一个汉字占 2 个字节。2）UTF-8，一个英文 1 个字节，一个中文，3 个字节。3）unicode ，中文、英文都是两个字节。

Spring：

1. AOP:

Aspect 切面：一个关注点的模块化

连接点（joinpoint）：执行过程中的某个点

通知：连接点和切面的动作

契入点：切面中匹配连接点

运用的是动态代理模式，JDK 的动态代理机制只能代理实现了接口的类，而不能实现接口的类就不能实现 JDK 的动态代理，cglib 是针对类来实现代理的，他的原理是对指定的目标类生成一个子类，并覆盖其中方法实现增强，但因为采用的是继承，所以不能对 final 修饰的类进行代理。

Cglib

implements MethodInterceptor

2. IOC：松耦合咯，

   单利一个 bean 容器中只有一个。

   http 请求只会有一个实例

   IOC 容器初始化：

   Resource 定位，我们一般使用外部资源来描述 Bean 对象，所以 IOC 容器第一步就是需要定位 Resource 外部资源 xml 文件。载入：第二个过程就是 BeanDefinition 的载入。BeanDefinitionReader 读取、解析 Resource 定位的资源，也就是将用户定义好的 Bean 表示成 IOC 容器的内部数据结构也就是 BeanDefinition

3. Bean 加载：容器初始化的时候会预先对单例和非延迟加载的对象进行预先初始化。其他的都是延迟加载是在第一次调用 getBean 的时候被创建。从 DefaultListableBeanFactory 的 preInstantiateSingletons 里可以看到这个规则的实现。

4. 依赖注入：

依赖注入发生在 getBean 方法中，getBean 又调用 dogetBean 方法。 getBean 是依赖注入的起点，之后调用 createBean 方法，创建过程又委托给了 docreateBean 方法。

5. **拦截器相关：**

   HandlerInterceptoer 拦截的是请求地址，所以针对请求地址做一些验证、预处

理等操作比较合适。当你需要统计请求的响应时间时 MethodInterceptor 将不太容易做到，因为它可能跨越很多方法或者只涉及到已经定义好的方法中一部分代码。MethodInterceptor 利用的是 AOP 的实现机制，在本文中只说明了使用方式，关于原理和机制方面介绍的比较少，因为要说清楚这些需要讲出 AOP 的相当一部分内容。在对一些普通的方法上的拦截 HandlerInterceptoer 就无能为力了，这时候只能利用 AOP 的 MethodInterceptor。

6. Spring 事务：spring 事务传播性质：PROPAGATION_REQUIRED，两个事务一起提交一起回滚 PROPAGATION_REQUIRES_NEW 将原有的事务挂起（还有其他的不重要）还有声明式事务等等
7. 源码。。有空看，主要是学习设计！

Mybatis：

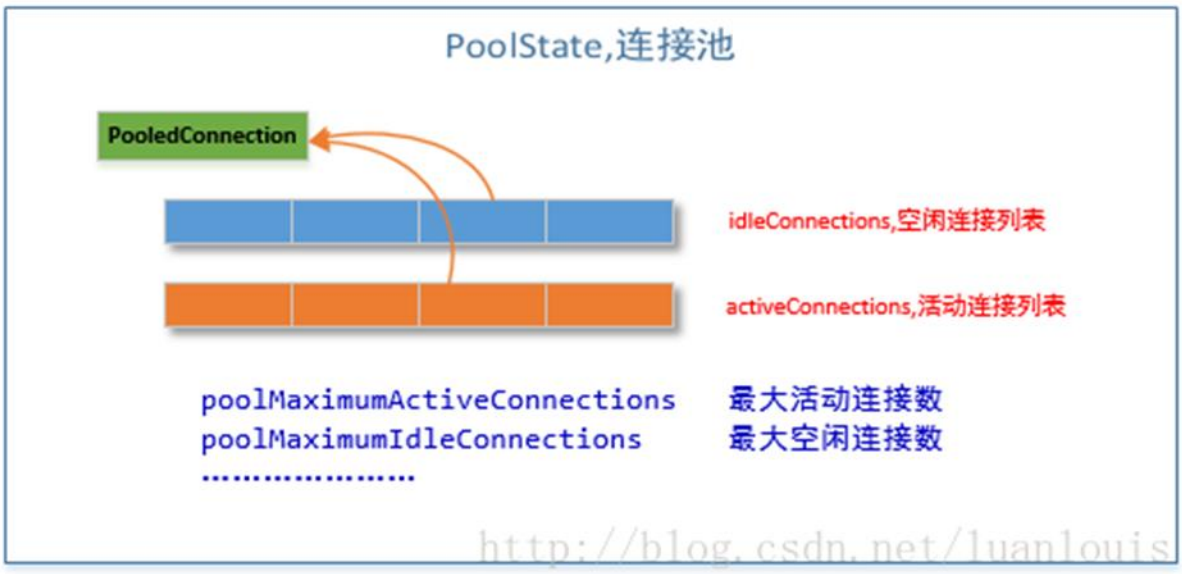
1. 连接池：
    - Jndi，unpooled，pooled。
    - Pooledatasource 持有 unpooleddatasuorce 的一个引用。
    - Sqlsession.select..这时候才会去创建一个 connection 对象。
    - Unpooled：

    初始化驱动：　　判断 driver 驱动是否已经加载到内存中，如果还没有加载，则会动态地加载 driver 类，并实例化一个 Driver 对象，使用 DriverManager.registerDriver()方法将其注册到内存中，以供后续使用。

    创建 Connection 对象：　　使用 DriverManager.getConnection()方法创建连接。

    配置 Connection 对象：　　设置是否自动提交 autoCommit 和隔离级别 isolationLevel。

    返回 Connection 对象



2. mybatis 中的#和$的区别：

    #{}：在预编译过程中，会把#{}部分用一个占位符?代替，执行时，将入参替换编译好的 sql 中的占位符“?”，能够很大程度防止 sql 注入

    ${}：在预编译过程中，${}会直接参与 sql 编译，直接显示数据，无法防止 Sql 注入，一般用于传入表名或 order by 动态参数

Sql 注入：select * from ${tableName} where name = #{name}
如果表名为 user; delete user; --
上述 sql 就会变为： select * from user; delete user; -- where name = ?;就会达到非法删除 user 表的功效！！！！

3. Orm 模型要看看！

Redis：

1. 持久化：
   RDB：虽然 Redis 允许你设置不同的保存点（save point）来控制保存 RDB 文件的频率， 但是， 因为 RDB 文件需要保存整个数据集的状态， 因此你可能会至少 5 分钟才保存一次 RDB 文件。你就可能会丢失好几分钟的数据。每次保存 RDB 的时候，Redis 都要 fork() 出一个子进程，并由子进程来进行实际的持久化工作。 在数据集比较庞大时， fork() 可能会非常耗时，造成服务器在某某毫秒内停止处理客户端；

AOF：redis-check-aof –fix 修复工具，aof 复制的是指令。一般现在 aof 和 rdb 一起用效果会好很多

2. Redis 为什么快呢？基于内存，又得益于 io 复用模型（典型的 reactor 网络 io 多路复用）
3. Redis 集群：（官方文档，本人写不动了）
4. 一致性 hash 和 redis 用的 hash 区别
5. 主从同步：主要是一个全量同步和增量同步。甩一个好用链接：https://jiachuhuang.github.io/2017/07/17/Redis%E4%B8%BB%E4%BB%8E%E5%A4%8D%E5%88%B6%E2%80%94%E2%80%94Slave%E8%A7%86%E8%A7%92/
6. 缓存雪崩，缓存标记，标记一个过期时间，通常比缓存时间短一半。缓存穿透，设置一个 key
7. 缓存穿透：缓存穿透是指查询一个一定不存在的数据，由于缓存是不命中时被动写的，并且出于容错考虑，如果从存储层查不到数据则不写入缓存，这将导致这个不存在的数据每次请求都要到存储层去查询，失去了缓存的意义。在流量大时，可能 DB 就挂掉了，要是有人利用不存在的 key 频繁攻击我们的应用，这就是漏洞。
8. 缓存击穿： 对于一些设置了过期时间的 key，如果这些 key 可能会在某些时间点被超高并发地访问，是一种非常“热点”的数据。这个时候，需要考虑一个问题：缓存被“击穿”的问题，这个和缓存雪崩的区别在于这里针对某一 key 缓存，前者则是很多 key。
9. 内存不足，开启虚拟内存，两个点：1. 页面置换页面很小 2. 只换 value

RabbitMq：

虚拟主机：一个虚拟主机持有一组交换机、队列和绑定。为什么需要多个虚拟主机呢？很简单，RabbitMQ 当中，用户只能在虚拟主机的粒度进行权限控制。 因此，如果需要禁止 A 组访问 B 组的交换机/队列/绑定，必须为 A 和 B 分别创建一个虚拟主机。每一个 RabbitMQ 服务器都有一个默认的虚拟主机“/”。

交换机：Exchange 用于转发消息，但是它不会做存储 ，如果没有 Queue bind 到 Exchange 的话，它会直接丢弃掉 Producer 发送过来的消息。 这里有一个比较重要的概念：路由键 。消息到交换机的时候，交互机会转发到对应的队列中，

那么究竟转发到哪个队列，就要根据该路由键。**绑定：**也就是交换机需要和队列相绑定，这其中如上图所示，是多对多的关系。
选择改变的原因：客户端接收消息的模式默认是自动应答，但是通过设置 autoAck 为 false 可以让客户端主动应答消息。当客户端拒绝此消息或者未应答便断开连接时，就会使得此消息重新入队。同时，想用 redis 专做缓存。

算法：未完待续。。。。。