Java 基础

1. hashMap 的原理以及和 hashtable 的区别？

答：HashMap 基于 hashing 原理，我们通过 put()和 get()方法储存和获取对象。当我们将键值对传递给 put()方法时，它调用键对象的 hashCode()方法来计算 hashcode，让后找到 bucket 位置来储存值对象。当获取对象时，通过键对象的 equals()方法找到正确的键值对，然后返回值对象。HashMap 使用链表来解决碰撞问题，当发生碰撞了，对象将会储存在链表的下一个节点中。 HashMap 在每个链表节点中储存键值对对象。当两个不同的键对象的 hashcode 相同时会发生什么？ 它们会储存在同一个 bucket 位置的链表中。键对象的 equals()方法用来找到键值对。HashMap 和 Hashtable 都实现了 Map 接口，但决定用哪一个之前先要弄清楚它们之间的分别。主要的区别有：线程安全性，同步(synchronization)，以及速度。HashMap 几乎可以等价于 Hashtable，除了 HashMap 是非 synchronized 的，并可以接受 null(HashMap 可以接受为 null 的键值(key)和值(value)，而 Hashtable 则不行)。HashMap 是非 synchronized，而 Hashtable 是 synchronized，这意味着 Hashtable 是线程安全的，多个线程可以共享一个 Hashtable；而如果没有正确的同步的话，多个线程是不能共享 HashMap 的。Java 5 提供了 ConcurrentHashMap，它是 HashTable 的替代，比 HashTable 的扩展性更好。另一个区别是 HashMap 的迭代器(Iterator)是 fail-fast 迭代器，而 Hashtable 的 enumerator 迭代器不是 fail-fast 的。所以当有其它线程改变了 HashMap 的结构（增加或者移除元素），将会抛出 ConcurrentModificationException，但迭代器本身的 remove()方法移除元素则不会抛出 ConcurrentModificationException 异常。但这并不是一个一定发生的行为，要看 JVM。这条同样也是 Enumeration 和 Iterator 的区别。由于 Hashtable 是线程安全的也是 synchronized，所以在单线程环境下它比 HashMap 要慢。如果你不需要同步，只需要单一线程，那么使用 HashMap 性能要好过 Hashtable。HashMap 不能保证随着时间的推移 Map 中的元素次序是不变的。

2. 数组和 ArrayList 的区别以及 ArrayList 和 LinkedList 的区别？

答：数组为容量固定，在内存当中是连续分配的空间，效率极高，但在不确定有多少元素的情况下，数组无法解决动态适应的问题，ArrayList 基于数组实现，在容量不够时可以动态扩容，每次扩容都是之前数组大小的 old Capacity + old Capacity >> 1(大致为 1.5 倍)，ArrayList 在动态适应元素添加，但是在扩容时会带来性能损耗，因为存在整个数组的拷贝;
ArrayList 是实现了基于动态数组的数据结构，而 LinkedList 是基于链表的数据结构;
对于随机访问 get 和 set，ArrayList 要优于 LinkedList，因为 LinkedList 要移动指针;
对于随机插入和删除操作 add 和 remove，一般大家都会说 LinkedList 要比 ArrayList 快，因为 ArrayList 要移动数据。而 LinkedList 只需要移动指针的指向就行，LinkedList 是没有大小限制的

3. java 多态的内存实现方式，内存中是如何调用子类的方法？

JVM 在执行 class 文件的时候，也就是在内存中执行 Java 的程序的时候，会将父类的 Method，放到 Method Table 里面，子类复写了父类中的方法后，初始化子类的时候，就会将 Method Table 中相应的 Method 进行覆盖。
另一方面，所有派生类中继承于基类的方法在方法表中的偏移量跟该方法在基类方法表中的偏移量保持一致这也就是为什么可以准确的调用 Method。

4. 父类声明对象和子类初始化，两者在内存中的位置？

5. String 的哈希实现,String StringBuffer 和 StringBuilder 的区别等

答：

```
int h = hash;
if (h == 0 && value.length > 0) {
    char val[] = value;

    for (int i = 0; i < value.length; i++) {
        h = 31 * h + val[i];
    }
    hash = h;
}
return h;
```

String 的 hash 实现为 s[0] + s[1]*31+s[2]*31*31....
String 为字符串常量，而 StringBuilder 和 StringBuffer 均为字符串变量，即 String 对象一旦创建之后该对象是不可更改的，但后两者的对象是变量，是可以更改的。在线程安全上，StringBuilder 是线程不安全的，而 StringBuffer 是线程安全的，
String：适用于少量的字符串操作的情况
StringBuilder：适用于单线程下在字符缓冲区进行大量操作的情况
StringBuffer：适用多线程下在字符缓冲区进行大量操作的情况

6. Java 程序的特征，Java 面向对象的特点（详细说明），Java 可以继承多个类吗？

答： 特征：跨平台：指可以不受计算机的硬件和操作系统的限制，而独立于平台。因为 java 中有自带的 JVM，它执行经过 javac 命令编译完成的 java 源代码所生成的 class 文件转换成机器码语言在平台上操作。

面向对象：指的是以对象为基本粒度。对象中包含属性和方法，对象的说明是用属性来表式，方法是用来操作对象。这样可以对应用程序解耦，提高代码的扩展和重用性。

安全性：就是指以语言级安全：表示以对象为基表粒度、编译时安全性：表示在 java 源代码编译时进行了语义和语法的检查、运行时安全性：表示在运行 java 类时进行字节码校检器校检限制、可执行代码安全性：就是对 java 类的访

问范围进行了限制。

多线程：就是指 java 内置了多线程技术和实现多线程的内置方法。

简单易用：就是指 java 源代码可以不用在特定环境下编写。

可维护性高

面向对象的特点：

封装：就是指对对象中的属性进行了私有化，只提供了两个公共的 set 和 get 方法，设置属性和获取属性值。提供数据的安全性

继承：就是指一个类不仅拥有已有类的属性和方法，还可以拥有自己的属性和方法。注意：子类并不能真正继承父类的构造，但可以调用。

多态：就是指重载和重写。重载：指一个类中有两个同名不同参的方法。重写：指子类中拥有与父类同名同参的方法。

Java 不可以继承多个类，只支持单继承，接口支持多继承

7. 说一下类里的 Static，静态代码块和构造代码块的不同？

在 Java 语言中，static 表示“静态”的意思，使用场景可以用来修饰成员变量和成员方法，当然也可以是静态代码块。static 的主要作用在于创建独立于具体对象的域变量或者方法，即使没有创建对象，也能使用属性和调用方法，static 目的就是在于解决这个问题。static 静态方法，被 static 修饰的方法也叫做静态方法，因为对于静态方法来说是不属于任何实例对象的，那么就是说在静态方法内部是不能使用 this 的，因为既然不属于任何对象，那么就更谈不上 this 了，实例变量：每次创建对象，都会为每个对象分配成员变量内存空间，实例变量是属于实例对象的，在内存中，创建几次对象，就有几份成员变量。静态变量：静态变量由于不属于任何实例对象，是属于类的，所以在内存中只会有一份，在类的加载过程中，JVM 为静态变量分配一次内存空间。静态代码块是在类加载的时候执行，属于类范围，而构造代码块只有在 new 对象时才会执行，属于对象范围，普通代码块存在于方法体内，只有在方法调用时才执行

8. 接口和抽象类的区别，创建对象有哪几种方式，比较对象有哪些方法？

答：1 接口里只能包含抽象方法，静态方法和默认方法，不能为普通方法提供方法实现，抽象类则完全可以包含普通方法。

2 接口里只能定义静态常量，不能定义普通成员变量，抽象类里则既可以定义普通成员变量，也可以定义静态常量。

3 接口不能包含构造器，抽象类可以包含构造器，抽象类里的构造器并不是用于创建对象，而是让其子类调用这些构造器来完成属于抽象类的初始化操作。

4 接口里不能包含初始化块，但抽象类里完全可以包含初始化块。

5 一个类最多只能有一个直接父类，包括抽象类，但一个类可以直接实现多个接口，通过实现多个接口可以弥补 Java 单继承不足。

创建对象的方式：

使用 new 关键字	} → 调用了构造函数
使用 Class 类的 newInstance 方法	} → 调用了构造函数

使用 Constructor 类的 newInstance 方法	} → 调用了构造函数
使用 clone 方法	} → 没有调用构造函数
使用反序列化	} → 没有调用构造函数

java 中对象比较大小有两种方法
1：实现 Comparable 接口的 public int compareTo(T o) 方法；
2：实现 Comparator 接口的 int compare(T o1, T o2)方法；

9. Error/Exception 的区别

首先，Error 类和 Exception 类都是继承 Throwable 类
Error（错误）是系统中的错误，程序员是不能改变的和处理的，是在程序编译时出现的错误，只能通过修改程序才能修正。一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程序终止。
Exception（异常）表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。

JVM

1. 介绍一下 JVM 的内存分区和垃圾回收？jvm 内存模型中老年代满了，继续往里添加会怎样？

答：JVM 内存主要分为 5 部分：堆、虚拟机栈、本地方法栈、方法区、程序计数器，其中堆和方法区是线程共享的区域，其余为线程私有；
垃圾回收：Serial 收集器，ParNew（Serial 的多线程版本），Parallel（并行）收集器，Serial-Old 收集器，Parallel-Old 收集器，CMS 收集器，G1 收集器
如何判断对象已死：
　　引用计数法（无法解决相互循环引用的问题）
　　可达性分析算法（GC Root）
以下几种对象可以作为 GC Root：
　　虚拟机栈中的引用对象
　　方法区中类静态属性引用的对象
　　方法区中常量引用对象
　　本地方法栈中 JNI 引用对象
垃圾收集算法：
　　标记-清除法
　　复制算法
　　标记整理法
　　分代收集算法

2. Minor GC 和 Full GC 的触发时机，堆和栈的区别，内存管理机制，问了内存机制？

答：Minor GC 触发条件：当 Eden 区满时，触发 Minor GC。
Full GC 触发条件：
（1）调用 System.gc 时，系统建议执行 Full GC，但是不必然执行
（2）老年代空间不足
（3）方法区空间不足
（4）通过 Minor GC 后进入老年代的平均大小大于老年代的可用内存
（5）由 Eden 区、From Space 区向 To Space 区复制时，对象大小大于 To Space 可用内存，则把该对象转存到老年代，且老年代的可用内存小于该对象大小

3. jvm 类加载过程，JVM 为何 xmx 和 xms 设成一样？

答：其中类加载过程包括加载、验证、准备、解析和初始化五个阶段（此外还有使用和卸载）。-Xss 规定了每个线程堆栈的大小。一般情况下 256K 是足够了。影响了此进程中并发线程数大小，-Xms 初始的 Heap 的大小，-Xmx 最大 Heap 的大小。在很多情况下，-Xms 和-Xmx 设置成一样的。这么设置，是因为当 Heap 不够用时，会发生内存抖动，影响程序运行稳定性。
(https://www.cnblogs.com/KingIceMou/p/6967129.html)

线程

4. 手写一个多线程代码：两个线程共同打印一个数组

5. 创建线程的方式，你一般用哪种，为什么，如何实现多线程 ？

答：继承 Thread、实现 Runnable 或者实现 Callable 接口包装为 FutureTask 来执行、Executor 框架，如何需要返回值一般选用 Callable，不需要返回值则选择 Runnable，java 不允许多继承，因此实现了 Runnable 接口的类可以再继承其他类，灵活性更大

6. 什么是死锁，必要条件，怎么解决死锁？

死锁：所谓死锁，是指多个进程循环等待它方占有的资源而无限期地僵持下去的局面。很显然，如果没有外力的作用，那麼死锁涉及到的各个进程都将永远处于封锁状态
产生原因：
a 竞争资源
b 进程间推进顺序非法
产生条件（4 个）：
互斥条件：进程要求对所分配的资源（如打印机）进行排他性控制，即在一段时间内某 资源仅为一个进程所占有。此时若有其他进程请求该资源，则请求进程只能等待。
不剥夺条件：进程所获得的资源在未使用完毕之前，不能被其他进程强行夺走，即只能 由获得该资源的进程自己来释放（只能是主动释放）。
请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而

该资源 已被其他进程占有，此时请求进程被阻塞，但对自己已获得的资源保持不放。
循环等待条件：存在一种进程资源的循环等待链，链中每一个进程已获得的资源同时被 链中下一个进程所请求。即存在一个处于等待状态的进程集合{P1, P2, ..., pn}，其中 Pi 等 待的资源被 P(i+1)占有（i=0, 1, ..., n-1)，Pn 等待的资源被 P0 占有
解决办法：

预防死锁

破坏死锁的四个必要条件中的一个或多个来预防死锁。

避免死锁

和预防死锁的区别就是，在资源动态分配过程中，用某种方式防止系统进入不安全的状态。

检测死锁

运行时出现死锁，能及时发现死锁，把程序解脱出来

解除死锁

发生死锁后，解脱进程，通常撤销进程，回收资源，再分配给正处于阻塞状态的进程。

如何避免

加锁顺序（线程按照一定的顺序加锁）

加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）

7. 线程状态，sleep/wait 方法的区别

答：对于 sleep()方法，我们首先要知道该方法是属于 Thread 类中的。而 wait()方法，则是属于 Object 类中的。sleep()方法导致了程序暂停执行指定的时间，让出 cpu 该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。在调用 sleep()方法的过程中，线程不会释放对象锁。而当调用 wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用 notify()方法后本线程才进入对象锁定池准备，获取对象锁进入运行状态。

8. 讲一讲进程和线程

答：**进程**:是程序的一次动态执行，它对应着从代码加载，执行至执行完毕的一个完整的过程，是一个动态的实体，它有自己的生命 周期。它因创建而产生，因调度而运行，因等待资源或事件而被处于等待状态，因完成任务而被撤消。反映了一个程序在一定的数据 集上运行的全部动态过程。通过进程控制块(PCB)唯一的标识某个进程。同时进程占据着相应的资源（例如包括 cpu 的使用 ，轮转时间以及一些其它设备的权限）。是系统进行资源分配和调度的一个独立单位。

线程: 可以理解为进程的多条执行线索，每条线索又对应着各自独立的生命周期。线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位。一个线程可以创建和撤销另一个线程，同一个进程中的多个线程之间可以并发执行。

9. 线程安全实现存取款类（手写）

数据库

10. Mysql 的索引,从 B 树，B+树开始，数据库的索引的优缺点,索引分类

答：在 MySQL 中，主要有四种类型的索引，分别为： B-Tree 索引， Hash 索引，Fulltext 索引和 R-Tree 索引，Innodb、MyISAM 默认是 B+树作为索引结构
索引的作用：数据库索引，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。索引的实现通常用 B-TREE。
优点：大大加快数据的查询速度
缺点：
1、创建索引和维护索引要耗费时间，并且随着数据量的增加所耗费的时间也会增加
2、索引也需要占空间，我们知道数据表中的数据也会有最大上线设置的，如果我们有大量的索引，索引文件可能会比数据文件更快达到上线值
3、当对表中的数据进行增加、删除、修改时，索引也需要动态的维护，降低了数据的维护速度。
分类：索引我们分为四类来讲 单列索引(普通索引，唯一索引，主键索引)、组合索引、全文索引、空间索引

11. 数据库的事务 ACID,数据库的范式，数据库连接池原理，了解和使用过哪些开源的连接池 Druid、C3P0

答：
(1) 原子性（Atomicity）
原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，这和前面两篇博客介绍事务的功能是一样的概念，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。
(2) 一致性（Consistency）
一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。
拿转账来说，假设用户 A 和用户 B 两者的钱加起来一共是 5000，那么不管 A 和 B 之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是 5000，这就是事务的一致性。
(3) 隔离性（Isolation）
隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。
即要达到这么一种效果：对于任意两个并发的事务 T1 和 T2，在事务 T1 看来，T2 要么在 T1 开始之前就已经结束，要么在 T1 结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。
关于事务的隔离性数据库提供了多种隔离级别，稍后会介绍到。
(4) 持久性（Durability）
持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永

久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。
隔离级别：

① Serializable (串行化)：可避免脏读、不可重复读、幻读的发生。

② Repeatable read (可重复读)：可避免脏读、不可重复读的发生。

③ Read committed (读已提交)：可避免脏读的发生。

④ Read uncommitted (读未提交)：最低级别，任何情况都无法保证。

12. 数据库优化的方法，四种，什么时候索引失效，四种，视图和表的区别，数据库查询有问题，怎么找出问题，数据库存储过程的特点？

答：
优化方法：
1. 选取最适用的字段属性（长度以及类型）
2. 使用连接（JOIN）来代替子查询(Sub-Queries)
3. 添加缓存（如 Redis）
4. 使用联合(UNION)来代替手动创建的临时表
5. 使用索引
索引失效：
1. 如果条件中有 or,即使其中有条件带索引也不会使用(这也是为什么尽量少用 or 的原因)
2. 对于多列索引，不是使用的第一部分，则不会使用索引
3. like 查询是以%开头
4. 如果列类型是字符串，那一定要在条件中将数据使用引号引用起来,否则不使用索引
5. 如果 mysql 估计使用全表扫描要比使用索引快,则不使用索引
视图和表的区别：
数据库中的数据都是存储在表中的，而视图只是一个或多个表依照某个条件组合而成的结果集，一般来说你可以用 update，insert，delete 等 sql 语句修改表中的数据，而对视图只能进行 select 操作。但是也存在可更新的视图，对于这类视图的 update，insert 和 delete 等操作最终会作用于与其相关的表中数据。因此，表是数据库中数据存储的基础，而视图只是为了满足某种查询要求而建立的一个对象。
表是物理存在的，你可以理解成计算机中的文件！
视图是虚拟的内存表，你可以理解成 Windows 的快捷方式！
存储过程的特点：
优点
1.运行速度：对于很简单的 sql，存储过程没有什么优势。对于复杂的业务逻辑，因为在存储过程创建的时候，数据库已经对其进行了一次解析和优化。存储过程一旦执行，在内存中就会保留一份这个存储过程，这样下次再执行同样的存储过程时，可以从内存中直接调用，所以执行速度会比普通 sql 快。
2.减少网络传输：存储过程直接就在数据库服务器上跑，所有的数据访问都在数据库服务器内部进行，不需要传输数据到其它服务器，所以会减少一定的网络传输。但是在存储过程中没有多次数据交互，那么实际上网络传输量和直接 sql 是一样的。而且我们的应用服务器通常与数据库是在同一内网，大数据的访问的

瓶颈会是硬盘的速度，而不是网速。

3.可维护性：存储过程有些时候比程序更容易维护，这是因为可以实时更新 DB 端的存储过程，有些 bug，直接改存储过程里的业务逻辑，就搞定了。

4.增强安全性：提高代码安全，防止 SQL 注入。这一点 sql 语句也可以做到。

5.可扩展性：应用程序和数据库操作分开，独立进行，而不是相互在一起。方便以后的扩展和 DBA 维护优化。

缺点

1.SQL 本身是一种结构化查询语言，但不是面向对象的的，本质上还是过程化的语言，面对复杂的业务逻辑，过程化的处理会很吃力。同时 SQL 擅长的是数据查询而非业务逻辑的处理，如果如果把业务逻辑全放在存储过程里面，违背了这一原则。

2.如果需要对输入存储过程的参数进行更改，或者要更改由其返回的数据，则您仍需要更新程序集中的代码以添加参数、更新调用，等等，这时候估计会比较繁琐了。

3.开发调试复杂，由于 IDE 的问题，存储过程的开发调试要比一般程序困难。

4.没办法应用缓存。虽然有全局临时表之类的方法可以做缓存，但同样加重了数据库的负担。如果缓存并发严重，经常要加锁，那效率实在堪忧。

5.不支持群集，数据库服务器无法水平扩展，或者数据库的切割（水平或垂直切割）。数据库切割之后，存储过程并不清楚数据存储在哪个数据库中。

总结

1. 适当的使用存储过程，能够提高我们 SQL 查询的性能，
2. 存储过程不应该大规模使用，滥用。
3. 随着众多 ORM 的出现，存储过程很多优势已经不明显。
4. SQL 最大的缺点还是 SQL 语言本身的局限性——SQL 本身是一种结构化查询语言，我们不应该用存储过程处理复杂的业务逻辑——让 SQL 回归它“结构化查询语言”的功用。复杂的业务逻辑，还是交给代码去处理吧。

数据库出错：检查错误信息，观察是否 SQL 语句有误，如果不是，则根据错误信息排查是否为数据库服务出现故障，检查对应数据库服务

Spring

13. Spring 的 ioc aop，spring 如何初始化类注解 bean？

什么是 DI 机制？

答：依赖注入（Dependecy Injection）和控制反转（Inversion of Control）是同一个概念，具体的讲：当某个角色需要另外一个角色协助的时候，在传统的程序设计过程中，通常由调用者来创建被调用者的实例。但在 spring 中创建被调用者的工作不再由调用者来完成，因此称为控制反转。创建被调用者的工作由 spring 来完成，然后注入调用者

面向切面编程（AOP）完善 spring 的依赖注入（DI），面向切面编程在 spring 中主要表现为两个方面

1.面向切面编程提供声明式事务管理

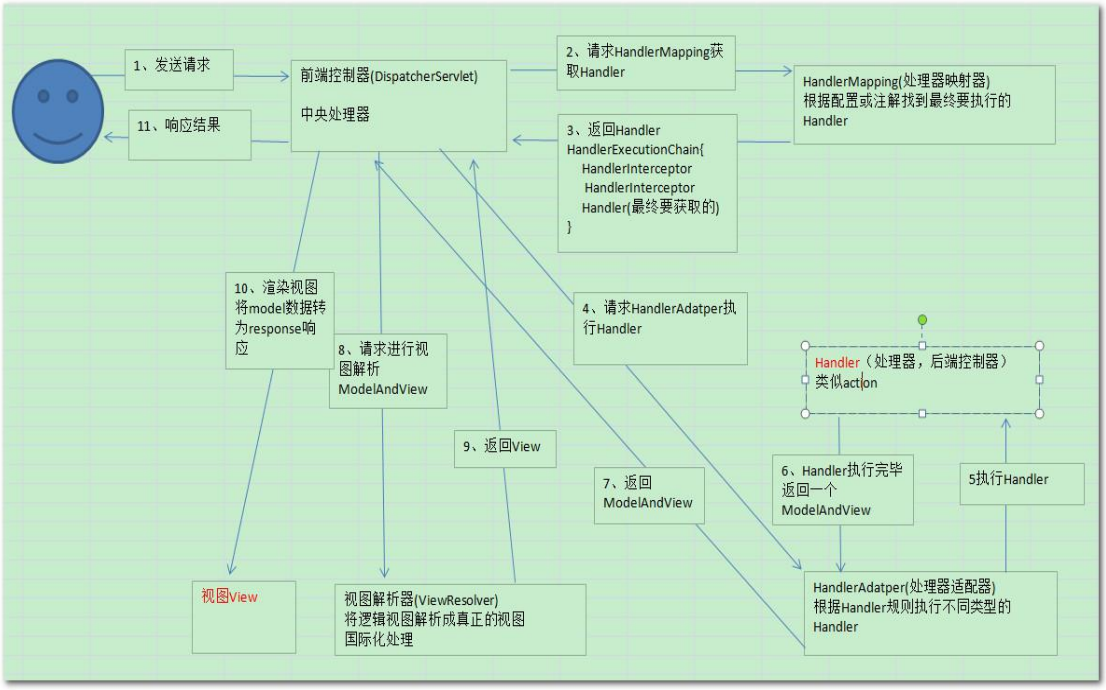
2.spring 支持用户自定义的切面

面向切面编程（aop）是对面向对象编程（oop）的补充，（主要用于日志记录，权限控制、事物管理），面向对象编程将程序分解成各个层次的对象，面向切面编程将程序运行过程分解成各个切面。AOP 从程序运行角度考虑程序的结构，提

取业务处理过程的切面，oop 是静态的抽象，aop 是动态的抽象，是对应用执行过程中的步骤进行抽象，从而获得步骤之间的逻辑划分。
aop 框架具有的两个特征：
1. 各个步骤之间的良好隔离性
2. 源代码无关性
Spring 通过扫描包，查看对应需要初始化 bean 的注解，将依赖注入，然后通过反射生成对应的 bean

14. SpringMVC 流程和原理？

答 ：



设计模式

15. 说一下设计模式的原则？用过哪些？单例模式，特别问了观察者模式？

答：

单一职责原则(Single Responsibility Principle, SRP)：
一个类只负责一个功能领域中的相应职责，或者可以定义为：就一个类而言，应该只有一个引起它变化的原因。

开闭原则(Open-Closed Principle, OCP)：
一个软件实体应当对扩展开放，对修改关闭。即软件实体应尽量在不修改原有代码的情况下进行扩展。

里氏代换原则(Liskov Substitution Principle, LSP)：
子类必须能够替换其父类，否则不应当设计为其子类。换句话说，父类出现的地方，都应该能由其子类代替。所以，子类只能去扩展基类，而不是隐藏或者覆盖基类。

依赖倒转原则(Dependency Inversion Principle, DIP)：
抽象不应该依赖于细节，细节应当依赖于抽象。换言之，要针对接口编程，

而不是针对实现编程。

接口隔离原则(Interface Segregation Principle, ISP)：

使用多个专门的接口，而不使用单一的总接口，即客户端不应该依赖那些它不需要的接口。

迪米特法则(Law of Demeter, LoD)：

软件实体应当尽可能少地与其他实体发生相互作用，也叫最少知道原则。

算法

16. 你知道几种排序算法？

答：堆排序，归并排序，快速排序。

17. 手写 AVL 树

18. 自己写 socket

扩展

19. 传统分布式系统与微服务的区别？微服务的技术点有哪些？

20. 用户同时用到几个微服务，那么如何保证用户状态？

21. 除了单独 redis 记录外，还有什么方法？

22. 二维码扫描登录的处理过程？后台登录的安全机制有哪些？

23. mq 有什么用途？

答：应用解耦（异步），通知，数据发布订阅，削峰（限流）

总结

一定要对自己简历描述的项目很熟悉，会切合简历进行相关提问：遇到的问题、如何解决等等

一道系统设计，有些难度，不过面试官会启发并根据你的回答一直讨论下去并提出新问题

你对哪方面比较熟悉，为什么？

一道算法简单题

TCP/IP 中长短连接是用来干什么的，TCP 三次握手和四次分手过程，TCP 和 UDP 的区别？

短连接在请求完资源后就可以释放，用来防止长期占用服务器资源。

长连接用于保持连接，防止多次建立连接浪费资源，有一种 DDOS 攻击就利用了长连接。

https://blog.csdn.net/ZWE7616175/article/details/80432486

redis?