# JAVA

## 线程

### 1.多线程有几种实现方案，分别是哪几种？

答：两种。

(1)继承Thread类

(2)实现Runnable接口

扩展一种：实现Callable接口，得和线程池结合

### 2.同步有几种方式，分别是什么？

答：两种

(1)同步代码块

(2)同步方法

### 3.启动一个线程是run()还是start()?他们的区别是什么？

答：start();

区别：

run()：封装了被线程执行的代码，直接调用仅仅是普通方法的调用；

start()：启动线程，并由JVM自动调用run()方法

### 4.sleep()和wait()的区别是什么？

答：sleep()：必须指定时间；不释放锁

wait()：可以指定时间，也可以不指定时间；会释放锁

### 5.为什么wait(),notify(),notifyAll()等方法都定义在Object类中？

答：因为这些方法的调用是依赖于锁对象的，而同步代码块的锁对象是任意锁。而Object是所有对象的父类，所以定义在这里面。

### 6. lock和synchronized的区别

(1) lock是一个接口，而synchronized是java的一个关键字。

(2) synchronized在发生异常时会自动释放占有的锁，因此不会出现死锁；而lock发生异常时，不会主动释放占有的锁，必须手动来释放锁(建议在finally中释放锁)，不然可能引起死锁的发生。

(3)使用synchronized时无法判断锁的状态，使用lock可以通过tryLock()来判断有没有获取锁。

(4)使用synchronized时，如果有一个线程获得锁，则另一个线程会一直等待；而使用lock时，如果有一个线程获得锁，则另一个线程会不断尝试获得锁，不用一直等待。

(5)synchronized和lock在竞争资源不激烈的情况下，性能相差不大。synchronized引入了偏向锁、轻量级锁、重量级锁以及锁优化机制来实现锁的优化；lock则是使用了自旋锁实现锁的优化。

### 7.线程的生命周期图

答：新建—就绪—运行—死亡

新建—就绪—运行—阻塞—就绪—运行—死亡



### 8.线程的状态转换图



### 9.线程池的理解

线程池是一种池化技术，用于提高资源的复用性，常见的还有内存池、连接池和对象池。线程池复用的是线程资源，它减少了线程的频繁创建和销毁带来的性能开销，其本身也会有参数来控制线程创建的数量，避免了无休止创建线程带来的资源利用率过高的问题。

### 10.Thread和Runnable的区别

(1)Thread是一个类，Runnable是一个接口，类只能被单继承，而接口可以被多实现。

(2)Runnable表示一个线程的顶层接口，Thread则是实现了Runnable。无论是使用Thead类还是实现Runnable接口，都需要去实现run()。

(3)站在面向对象的思想来说，Runnable是一个任务，而Thread才是真正处理线程，一般是实现Runnable用于定义具体的任务，然后交由Thread来处理，达到松耦合的目的。

### 11.ThreadLocal的底层原理

(1)ThreadLocal是Java中所提供的线程本地存储机制，可以利用该机制将数据缓存在某个线程内部，该线程可以在任意时刻、任意方法中获取缓存的数据。

(2)ThreadLocal底层是通过ThreadLocalMap来实现的，每个Thread对象(注意不是ThreadLocal对象)中都存在一个ThreadLocalMap，Map的key为ThreadLocal对象，Map的value为需要缓存的值。

(3)如果在线程池中使用ThreadLocal会造成内存泄漏，因为当ThreadLocal对象使用完之后，应该要把设置的key、value，也就是Entry对象进行回收，但线程池中的线程不会回收，而线程对象是通过强引用指向ThreadLocalMap，ThreadLocalMap也是通过强引用指向Entry对象，线程不被回收，Entry对象也就不会被回收，从而出现内存泄漏。解决办法是，在使用了ThreadLocal对象之后，手动调用ThreadLocal的remove方法，手动清除Entry对象。

(4)ThreadLocal经典的应用场景就是连接管理(一个线程持有一个连接，该连接对象可以在不同的方法之间进行传递，线程之间不共享同一个连接)。

### 12.线程池的底层和工作原理

(1)如果此时线程池中的线程数量小于corePoolSize，即使线程池中的线程都处于空闲状态，也要创建新的线程来处理被添加的任务。

(2)如果此时线程池中的线程数量等于corePoolSize，但是缓冲队列workQueue未满，那么任务被放入缓冲队列。

(3)如果此时线程池中的线程数量大于等于corePoolSize，缓冲队列workQueue满，并且线程池中的数量小于maximumPoolSize，会创建新的线程来处理被添加的任务。

(4)如果此时线程池中的线程数量大于corePoolSize，缓冲队列workQueue满，并且线程池中的数量等于maximumPoolSize，那么通过 handler所指定的策略来处理此任务。

(5)当线程池中的线程数量大于corePoolSize时，如果某线程空闲时间超过keepAliveTime，线程将被终止。这样，线程池可以动态的调整池中的线程数。

## 设计模式

### 1.单例模式的思想是什么？请写一个代码体现

答：思想：保证类在内存中只有一个对象。

代码在项目learn的pattern.pattern3包下。

补充：

开发时：饿汉式(不会出问题)

面试时：懒汉式(多线程环境下可能会出问题，需要使用同步锁)

jdk中的Runtime类是单例模式的饿汉式

Runtime：每个Java应用程序都有一个Runtime类实例，使应用程序能够与其运行的环境

相连接。

exec(String command)：可以用来执行cmd命令

## 基础知识

### 1.byte型变量和常量相加的问题

byte b1 = 3,b2 = 4,b;

b=b1+b2;

b=3+4;

哪句会编译失败？为什么？

答：第二句会失败。因为byte类型的变量做加法，首先b1和b2都转化成int，再做加法，而结果自然也是int型，但b是byte型，不是int，所以报错，int转byte可能会损失精度；

但是常量相加，会先将结果计算出来，然后看是否在byte范围内，如果在就不报错，所以第三句不报错。

### 2. byte型数据溢出的计算

byte b = 130;有没有问题？如果想正常赋值，怎么做？结果是多少呢？

答：有问题，因为byte的范围是-128~127，130超出了byte的范围。要想正确赋值，可以强制转换，即byte b = (byte)130;结果是-126。

### 3.short型变量相加的问题

short s=1; s = s+1;

short s=1; s+=1;

以上两个代码有没有问题，如果有，那里有问题？

答：第一个会报可能损失精度的错误(参考byte型变量的相加)，而第二个不会。因为s+=1;等价于s = (s的数据类型)(s + 1);相当于强制转换了。

### 4.请自己实现两个整数变量值的交换

int a = 10;

int b = 20;

答：(1) 借助第三个变量

int c = a;

a = b;

b = c;

(2) 使用异或运算符^

a = a ^ b;

b = a ^ b;

a = a ^ b;

(3) 用变量相加的做法

a = a + b;

b = a – b;

a = a – b;

(4) 一句话搞定

b = (a+b) – (a=b);

### 5.用最有效率的方式写出2乘以8的结果

答：2 << 3，即2 \* 2的3次幂，等于16。

<<即为左移，即按二进制形式把所有的数字向左移动对应的位数，高位移出(舍弃)，低位的空位补零。

### 6.创建类的对象时，静态代码块、构造代码块、构造方法的执行顺序

答：(1)若有父类，顺序为：

静态代码块Fu—>

静态代码块Zi—>

构造代码块Fu—>

构造方法Fu—>

构造代码块Zi—>

构造方法Zi。

(2)若没有父类，顺序为：

静态代码块—>

构造代码块—>

构造方法。

补充：static{语句} 为静态代码块，{语句}为构造代码块。

### 7.Override和Overload的区别，Overload能改变返回值类型吗？

答：Override：方法重写，指在子类中，出现和父类中一模一样的方法声明的现象。

Overload：方法重载，指在用一个类中，出现的方法名相同，但参数列表不同的现象。

方法重载能改变返回值类型，因为它和返回值类型无关。

### 8.this和super分别代表什么，以及他们各自的使用场景

答：this：代表当前类的对象引用。

super：代表父类存储空间的标识。(可以理解为父类的引用，通过这个东西可以访问父类的成员)

场景：

成员变量：

this.成员变量

super.成员变量

构造方法：

this(…)

super(…)

成员方法：

this.成员方法

super.成员方法

### 9.String s = new String(“hello”)和String s = “hello”的区别？

答：前者会创建1个或2个对象，后者创建1个或0个对象。

String s = new String("hello")首先会在堆中创建一块内存，内存地址返回给栈中s，然后因为"hello"是常量，JVM会去方法区中的字符串常量池查看是否有"hello"字符串的对象，没有的话就分配一个空间来存放"hello"，并将其空间地址存入堆中new出来的对象中；

对于String s="hello"这样创建的对象，JVM会直接检查字符串常量池是否已有"hello"字符串对象，如没有，就分配一个内存存放"hello"，如有了，则直接将字符串常量池中的地址返回给栈中的s而不经过堆中new的对象。

### 10.String、StringBuffer、StringBuilder的区别？

答：String内容是不可变的，其内部的value值被final修饰，变更String值时会产生新的String对象，线程安全。而StringBuffer、StringBuilder内容可变。StringBuffer是同步的，线程安全，底层使用了synchronized，但是效率低；StringBuilder是不同步的，线程不安全，但效率高。

String的值最终都会存储在运行时常量池中，运行时常量池属于方法区；StringBuffer、StringBuilder的值都是存储在堆内存中。

### 11.StringBuffer和数组的区别？

答：二者都可以看作是一个容器，用来装其他的数据。但是StringBuffer的数据最终是一个字符串数据；而数组可以放置多种数据，但必须是同一种数据类型的。

### 12.Integer赋值问题

Integer i1 = 128;

Integer i2 = 128;

System.out.println(i1 == i2);

Integer i3 = 127;

Integer i4 = 127;

System.out.println(i3 == i4);

二者的输出结果是什么？为什么？

答：==比较的是两边的地址值，结果是false和true。

因为java有一个内置的数据缓冲池，范围-128~127，i1和i2的值是128，不在范围内，所以会创建两个不同的对象；而i3和i4的值是127，直接去缓冲池里拿数据，因为他们的地址一致，都是127的地址，所以是true。

### 13.final、finally和finalize的区别

答：final修饰类，类不能被继承；修饰变量，变量是常量；修饰方法，方法不能被重写。

finally：是异常处理的一部分，用来释放资源。

finalize：是Object的一个方法，用于垃圾回收。

### 14.如果catch里面有return语句，请问finally里面的代码还会执行吗？如果会，是在return前还是后呢？

答：会执行，会在return前执行，准确的说是在中间执行。

### 15.java常见的异常

(1) java.lang.NullPointerException  
这个异常大家肯定都经常遇到，异常的解释是"程序遇上了空指针"，简单地说就是调用了未经初始化的对象或者是不存在的对象，这个错误经常出现在创建图片，调用数组这些操作中，比如图片未经初始化，或者图片创建时的路径错误等等。  
对数组操作中出现空指针，很多情况下是一些刚开始学习编程的朋友常犯的错误，即把数组的初始化和数组元素的初始化混淆起来了。数组的初始化是对数组分配需要的空间，而初始化后的数组，其中的元素并没有实例化，依然是空的，所以还需要对每个元素都进行初始化。  
(2) java.lang.ClassNotFoundException  
因找不到指定的类而导致的异常，一般是使用Class.forName()时类名写错了，或者没有导包。  
(3) java.lang.ArithmeticException  
这个异常的解释是"数学运算异常"，比如程序中出现了除以零这样的运算就会出这样的异常，对这种异常，大家就要好好检查一下自己程序中涉及到数学运算的地方，公式是不是有不妥了。  
(4) java.lang.ArrayIndexOutOfBoundsException  
这个异常相信很多朋友也经常遇到过，异常的解释是"数组下标越界"，现在程序中大多都有对数组的操作，因此在调用数组的时候一定要认真检查，看自己调用的下标是不是超出了数组的范围。  
一般来说，显示调用不太容易出这样的错，但隐式调用就经常出错了，还有一种情况，是程序中定义的数组的长度是通过某些特定方法决定的，不是事先声明的，这个时候，最好先查看一下数组的length，以免出现这个异常。  
(5) java.lang.IllegalaccessException  
这个异常的解释是"没有访问权限"，当应用程序要调用一个类，但当前的方法即没有对该类的访问权限便会出现这个异常。对程序中用了package的情况下要注意这个异常。

(6) ClassCastException

类转换异常，将一个不是该类的实例转换成这个类就会抛出这个异常。

(7) NoSuchMethodException

没有这个方法异常，一般发生在反射调用方法的时候

(8) FileNotFoundException

找不到文件异常，如果文件不存在就会抛出这种异常。

(9) IOException

IO，即：Input、Output，我们在读写磁盘文件、网络内容的时候经常会生的一种异常，这种异常是受检查异常，需要进行手工捕获。

### 16.所有整型包装类对象之间值的比较

比如：Integer与Integer之间值的比较

在-128~127之间时，Integer会从常量池中取数据，而==比较对象类型时比较的是地址值。

所以Integer a = 1;Integer b = 1;System.out.println(a==b);结果为true。

但Integer a = 200;Integer b = 200;System.out.println(a==b);结果为false。因为200不在-128~127之间，a和b都是new出来的对象，二者的地址值根本不相等。

所以如果要进行整型包装类对象之间值的比较，最好用equals()。

### 17. java面向对象有哪些特征

(1) 封装

封装隐藏了类的内部实现机制。对属性进行了封装：外界只能通过特定的方法进行访问。对方法进行了封装：外界只能通过定制好的方式调用，不用了解方法内部逻辑，方便使用。保护了数据。便于修改，增强了代码的可维护性和复用性。

(2) 继承

继承是从已有的类中派生出新的类，即子类继承自父类。当子类通过extends关键字继承了父类后，便继承了父类的属性和方法（注意：子类继承了父类的所有属性和方法。但父类的私有属性和方法，子类是不能直接访问的，只是拥有但无法使用）。同时子类还可以具备父类所不具备的属性或方法。

通过继承避免了对各个类中重复的属性和方法进行反复描述。增强了代码的复用性。代码更加简洁。

(3) 多态

多态是类和类的关系。多态必须具备三个条件：1.要有继承关系 2.要有方法的重写 3.父类的引用指向子类的对象。

正是因为两个类有继承关系，并存在方法的重写，才能在调用时有父类的引用指向子类的对象。

通过多态可以是一个对象在不同载体中呈现不同的状态。即同一个父类引用在调用同一个方法时会得到不同的结果。多态减少了类的代码量，提高了代码的可维护性、可扩展性和复用性。

### 18.jdk1.8中的新特性

(1) 接口的默认方法

(2) Lambda表达式

(3) 函数式接口

仅有一个抽象方法的接口被称为函数式接口。

(4) StreamAPI

(5) Optional类

(6) 新的日期时间类

### 19.抽象类和接口的区别

(1) 抽象类

可以定义构造方法；

可以有抽象方法和具体方法；

可以定义成员变量；

可以有静态方法；

(2) 接口

没有构造方法；

jdk1.8之前只有抽象方法，没有具体方法；

没有静态方法；

没有成员变量，只能是成员常量；

其成员只能被public修饰；

### 20.序列化和反序列化的理解

序列化的核心目的是为了解决网络通信之间对象传输的问题，即如何把JVM进程中的一个对象跨网络传输到另一个JVM进程中进行恢复。序列化就是把内存中的对象转化为字节流，以便实现存储和传输。反序列化就是根据文件或者网络上获取到对象的字节流，根据流中的对象描述信息和状态重新构建一个新的对象。序列化的前提是为了保证通信双方对于对象的一个可识别性，所以很多时候会把对象先转化为通用的解析格式，比如json、xml，然后再把它们转化为数据流进行网络传输。市面上开源的序列化技术有Json、Xml、Protobuf、Kyro等，实际选择时需要考虑序列化之后的数据大小、序列化的性能、是否支持跨平台和跨语言、技术的成熟度。

### 21.IO和NIO的区别

(1)IO流可以实现数据从磁盘、内存、网络中读取以及写入，java中提供了字符流和字节流两种方式实现数据流的操作。

(2)当程序面向网络进行数据IO操作的时候，java提供了Socket的方式实现网络IO的通信，这种方式可以实现数据的网络传输。基于Socket的IO通信属于阻塞IO，在连接以及IO事件未就绪的情况下，当前的连接会处于阻塞等待的状态，一般有连接处于阻塞状态，后续的连接都得去等待。NIO是jdk1.4新增的一种new IO，它提供了一种非阻塞的特性，通过NIO进行网络数据传输的时候，如果连接或IO未就绪，服务端不会阻塞当前连接，而是继续轮询后续的连接来处理，能处理更多的并发连接。

### 22.为什么比较两个Integer不能使用==来判断

先说应该使用equals()来进行比较！

在Integer中引入了享元模式，对-128到127之间的数据做了一层缓存，如果Integer类型的值在-128到127之间，会直接从缓存中获取Integer对象的一个实例，否则会创建一个新的Integer对象，优点是减少频繁创建Integer对象所带来的内存消耗，从而提升性能。因此如果定义了两个Integer对象且它们的值都在-128到127之间，用==来判断则必然是true，因为它俩指向的内存地址是同一个。



## 集合

### 1.Map集合的key可以为null吗？

答：HashMap可以，键[和值](https://wenwen.sogou.com/s/?w=%E5%92%8C%E5%80%BC&ch=ww.xqy.chain)都可以为空或null，但是不能同时存在两个相同的键，意思就是不能把两个不同的键都设为空或null，因为键是唯一的。



### 2. ArrayList和LinkedeList的区别

(1) 数据结构不同

ArrayList是Array(动态数组)的数据结构，LinkedList是Link(链表)的数据结构。

(2) 执行不同操作时效率不同

当取值和赋值时，ArrayList可以直接通过下标来操作，因此速度比LinkedList快。而对数据进行增加和删除时，LinkedList更快，因为ArrayList在增加和删除后，会对操作点之后所有数据的下标索引产生影响，需要进行操作点之后所有数据的移动。

(3) 自由性不同

ArrayList使用时需要指定容量，但它有下标，使用方便；LinkedeList使用时不需要指定容量，能够动态的随数据量的变化而变化，但是它没有下标，使用不方便。

(4) 主要控件开销不同

ArrayList主要控件开销在于需要在List列表预留一定空间；而LinkList主要控件开销在于需要存储节点信息以及节点指针。

### 3. 高并发中的集合有哪些问题

(1) 第一代线程安全集合类：Vector、Hashtable

保证线程安全的方法：使用synchronized修饰方法 效率低下。

(2) 第二代非线程安全集合类：ArrayList、HashMap

线程不安全，但是性能好，用于代替Vector、Hashtable。

如果需要线程安全，可以调用一层方法(装饰器模式)来实现：

Collections.synchronizedList(list);

Collections.synchronizedMap(map);

底层使用的也是synchronized。

(3) 第三代线程安全集合类

java.util.concurrent.ConcurrentHashMap(jdk1.8的该类底层更新，放弃使用Lock锁)

java.util.concurrent.CopyOnWriteArrayList：

a.首先CopyOnWriteArrayList内部也是通过数组来实现的，在添加元素时，会复制一个

新的数组，写操作在新数组上进行，读操作在原数组上进行。

b.写操作会加锁，防止出现并发写入数据会丢失的情况。

c.写操作结束后，会把原数组指向新数组。

d. CopyOnWriteArrayList允许在写操作时读取数据，大大提高了读的性能，因此适合读

多写少的场景。但是CopyOnWriteArrayList也因此比较占用内存，同时读取的数据

可能不是最新的数据，所以不适合实时性很高的场景。

java.util.concurrent.CopyOnWriteArraySet

### 4.Arraylist的扩容机制

Arraylist是一个数组结构的存储容器，在创建对象时默认数组长度是10，也可以在创建对象时手动指定初始长度。当Arraylist没有足够的容量存储数据时，会触发自动扩容。首先创建一个新的数组，其长度为原来数组长度的1.5倍，然后用Arrays.copyOf()将老数组中的数据拷贝到新数组里，扩容完成以后，再将新数据添加到新的数组里。

### 5.HashMap的扩容机制

(1)jdk1.7版本：

a.首先生成一个长度两倍于原数组的新数组。

b.遍历老数组中每个位置上的链表上的每个元素。

c.取每个元素的key，并基于新数组长度，计算出每个元素在新数组中的下标。

d.将元素添加到新数组中。

e.所有元素转移完后，将新数组赋值给HashMap对象的table属性。

(2)jdk1.8版本：

a.首先生成一个长度两倍于原数组的新数组。

b.遍历老数组中每个位置上的链表上或红黑树上的每个元素。

c.如果是链表，则直接将链表中的每个元素重新计算下标，并添加到新数组中。

d.如果是红黑树，则先遍历红黑树，先计算出红黑树中每个元素对应在新数组中的下标

位置。

e.统计每个下标位置的元素个数，如果该位置下的元素个数超过了8，则生成一个新的

红黑树，并将根节点的添加到新数组的对应位置。如果该位置下的元素个数没有超

过8，那么则生成一个链表，并将链表的头节点添加到新数组的对应位置。

f.所有元素转移完后，将新数组赋值给HashMap对象的table属性。

## 并发

### 1.如何理解volatile关键字

在并发领域中，存在三大特性：原子性、有序性、可见性。volatile关键字用来修饰对象的属性，在并发环境下可以保证这个属性的可见性。对于加了volatile关键字的属性，在对这个属性进行修改时，会直接将CPU高级缓存中的数据写回到主内存，对这个变量的读取也会直接从主内存中读取，从而保证了可见性，底层是通过操作系统的内存屏障来实现的，由于使用了内存屏障，所以会禁止指令重排，同时也就保证了有序性，在很多并发场景下，如果用好volatile关键字可以很好的提高执行效率。

### 2. ReentrantLock中的公平锁和非公平锁的底层实现

首先不管是公平锁和非公平锁，它们的底层实现都会使用AQS来进行排队，它们的区别在于：线程在使用lock()方法加锁时，如果是公平锁，会先检查AQS队列中是否存在线程在排队，如果有线程在排队，则当前线程也进行排队，如果是非公平锁，则不会去检查是否有线程在排队，而是直接竞争锁。

不管是公平锁还是非公平锁，一旦没竞争到锁，都会进行排队，当锁释放时，都是唤醒排在最前面的线程，所以非公平锁只是体现在了线程加锁阶段，而没有体现在线程被唤醒阶段。

另外，ReentrantLock是可重入锁，不管是公平锁还是非公平锁都是可重入的。

### 3. ReentrantLock中的tryLock()和lock()方法的区别

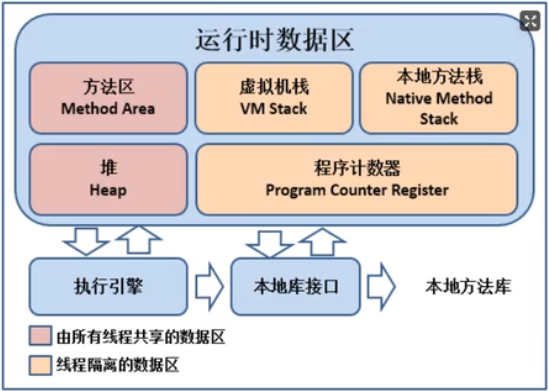
(1)lock()没有返回值，且这个方法在没有获取到锁的时候会阻塞，这样后续代码就不会执行。

(2)tryLock()会返回一个boolean类型的返回值，当返回值为true时表示获取到了锁，当返回值为false时表示没有获取到锁。且该方法不会阻塞。

## JVM

### 1.JVM中哪些是线程共享区

堆区和方法区是所有线程共享的。栈、本地方法栈、程序计数器是每个线程独有的。



### 2.JVM中哪些可以作为GC Root

什么是GC Root。JVM在进行垃圾回收时，需要找到“垃圾”对象，也就是没有被引用的对象，但是直接找“垃圾”对象是比较耗时的，所以反过来，先找“非垃圾”对象，也就是正常对象。那么就需要从某些“根”开始去找，根据这些“根”的引用路径找到正常对象，而这些“根”有一个特征，就是它只会引用其他对象，而不会被其他对象引用，例如：栈中的本地变量、方法区中的静态变量、本地方法栈中的变量、正在运行的线程等可以作为GC Root。

# 其他

## Web

### 1.Tomcat如何优化

对于Tomcat调优，可以从两个方面来进行调整：内存和线程。

首先启动Tomcat，实际上就是启动了一个JVM，所以可以按JVM调优的方式来进行调整，从而达到Tomcat优化的目的。另外Tomcat中设计了一些缓存区，比如appReadBufSize、bufferPoolSize等缓存区来提高吞吐量。

还可以调整Tomcat的线程，比如调整minSpareThreads参数来改变Tomcat空闲时的线程数，调整maxThreads参数来设置Tomcat处理连接的最大线程数。并且还可以调整IO模型，比如使用NIO、APR这种相比于BIO更加高效的IO模型。

# 框架

## Mybatis

### 1. #{}和${}的区别

(1) #是一个占位符，$是拼接符

使用#{}方式引用参数的时候，Mybatis会把传入的参数当成是一个字符串，自动添加单引号；使用${}引用参数时，不做任何处理，会直接将值拼接在sql语句中，一般用作表名或者字段名的拼接。

(2) 使用#{}能够防止sql注入，${}不能避免注入攻击

#{}的方式引用参数，mybatis会先对sql语句进行预编译，然后再引用值，能够有效防止sql注入，提高安全性。${}的方式引用参数，sql语句不进行预编译。因此建议使用#{}的方式。

## Spring

### 1.spring中有两个相同id的bean，会报错吗

(1)在同一个xml配置文件中，不能存在id相同的两个bean，因为id表示bean的唯一标志符号。当id重复时，spring容器在启动时会报错，错误发生在spring对xml文件进行解析转化为BeanDefinition的阶段。

(2)在两个不同的spring配置文件中，可以存在id相同的两个bean，此时ioc容器在加载bean时，默认会把多个相同id的bean进行覆盖。

(3)spring3.x版本后，出现了@Configuration来声明一个配置类，@Bean用来注册bean。如果有多个相同名称的Bean，则ioc只会注册第一个声明Bean的实例。



### 2.@Resource和@Autowired的区别

(1)@Resource是jdk提供的一个注解，它默认先根据名称实现bean的依赖注入，如果找不到该名称的bean，会继续根据类型来注入。它有name和type两种属性，分别手动指定根据名称、类型注入。



(2)@Autowired是spring提供的一个注解，它默认根据类型实现bean的依赖注入。它有一个required属性，默认值为true，指强制要求bean实例的注入。当容器中有多个相同类型的bean时，启动时会报错，解决方法有两个。一是用@Primary修饰@Bean修饰的方法，@Primary表示主要的bean，这样@Autowired会优先使用被@Primary修饰的bean；二是和@Qualifier一起修饰变量，@Qualifier起到筛选的作用，可以根据bean的名称找到唯一的bean。



# 数据库

## 通用

### 1.数据库连接池的作用

(1)数据库连接池是一种池化技术，旨在实现资源的复用，避免资源的重复创建和销毁带来的开销。实际应用程序每一次向数据库发起CRUD操作时都需要数据库的连接。

(2)其核心思想是应用程序在启动时，去提前初始化一部分的连接，保存在连接池里，当应用程序需要用到连接时，直接从连接池里取，用完后再将连接放到连接池中，从而避免了数据库连接的多次创建与销毁。

(3)数据库连接池的关键参数

初始化连接数(最小连接数)：表示项目启动时，初始化多少个连接保存在连接池中。该数量

是连接池一直保持的数据连接数。

最大连接数：表示连接池能申请的最大连接数，如果连接数不够，后续要获取连接的线程就

会阻塞。

最大空闲连接数：表示没有请求的时候，连接池中要保留的最大空闲连接数。

最小空闲连接数：表示当空闲连接数小于这个值的时候，连接池需要再创建连接来补充至这

个值。

最大等待时间：表示当连接池中的连接用完了以后，新的请求要等待的时间，超过这个时间

就会提示超时异常。

## Mysql

### 1.索引的优点与缺点

索引是一种能够帮数据库高效的从磁盘去检索数据的一种数据结构。在mysql的InnoDB引擎里采用的是B+树的结构来去实现索引和数据的存储。

优点：通过B+树的结构存储数据，可以大大减少数据检索时的磁盘IO次数，从而提升数据查询的性能；B+树索引在进行范围查找时，只需要找到起始节点，然后基于叶子节点的链表结构往下读取即可，查询效率较高；通过唯一索引约束，可以保证数据表中每一行数据的唯一性。

缺点：数据的增加、修改、删除，需要涉及到索引的维护，当数据量较大的情况下，索引的维护会带来较大的性能开销；一个表中允许存在一个聚簇索引和多个非聚簇索引，但是索引数量过多会造成索引维护的成本过高；创建索引时，需要考虑索引字段值的分散性，如果该字段的重复数据过多，创建索引反而会带来性能的下降。