

史上最全的Java面试题独家整理

公众号：程序IT圈，独家整理



长按识别**关注我们**

Java基础面试题169提以及答案

1、一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制？**

可以有多个类，但只能有一个public的类，并且public的类名必须与文件名相一致。

2、Java有没有goto？

java中的保留字，现在没有在java中使用。

3、说说&和&&的区别。

&和&&都可以用作逻辑与的运算符，表示逻辑与（and），当运算符两边的表达式的结果都为true时，整个运算结果才为true，否则，只要有一方为false，则结果为false。

&&还具有短路的功能，即如果第一个表达式为false，则不再计算第二个表达式，例如，对于if(str!=null&&!str.equals(s))表达式，当str为null时，后面的表达式不会执行，所以不会出现NullPointerException如果将&&改为&，则会抛出NullPointerException异常。If(x==33 & ++y>0) y会增长，If(x==33 && ++y>0)不会增长

&还可以用作位运算符，当&操作符两边的表达式不是boolean类型时，&表示按位与操作，我们通常使用0x0f来与一个整数进行&运算，来获取该整数的最低4个bit位，例如，0x31 & 0x0f的结果为0x01。

4、在JAVA中如何跳出当前的多重嵌套循环？

在Java中，要想跳出多重循环，可以在外面的循环语句前定义一个标号，然后在里层循环体的代码中使用带有标号的break语句，即可跳出外层循环。

例如：

```
for(int i=0;i<10;i++){
    for(int j=0;j<10;j++){
        System.out.println("i=" + i + ",j=" + j);
        if(j == 5) break ok;
    }
}
```

另外，我个人通常并不使用标号这种方式，而是让外层的循环条件表达式的结果可以受到里层循环体代码的控制，例如，要在二维数组中查找到某个数字。

```
int arr[][] ={{1,2,3},{4,5,6,7},{9}};
boolean found = false;
for(int i=0;i<arr.length&&!found;i++) {
    for(int j=0;j<arr[i].length;j++){
        System.out.println("i=" + i + ",j=" + j);
        if(arr[i][j] ==5) {
            found =true;
            break;
        }
    }
}
```

5、switch语句能否作用在byte上，能否作用在long上，能否作用在String上？

在switch (e) 中，e只能是一个整数表达式或者枚举常量（更大字体），整数表达式可以是int基本类型或Integer包装类型，由于byte,short,char都可以隐含转换为int，所以，这些类型以及这些类型的包装类型也是可以的。显然，long和String类型都不符合switch的语法规规定，并且不能被隐式转换成int类型，所以，它们不能作用于switch语句中。

switch语句能否作用在String上说错了，Java1.7之后已经支持这种写法了！

6、`short s1= 1; s1 = (s1+1是int类型，而等号左边的是short类型，所以需要强转) 1 + 1;`有什么错? `short s1 = 1; s1 += 1;`有什么错?(没有错)

对于`short s1= 1; s1 = s1 + 1;`由于`s1+1`运算时会自动提升表达式的类型，所以结果是int型，再赋值给`short`类型`s1`时，编译器将报告需要强制转换类型的错误。

对于`short s1 = 1; s1 += 1;`由于`+=`是java语言规定的运算符，java编译器会对它进行特殊处理，因此可以正确编译。

7、char型变量中能不能存储一个中文汉字?为什么?

`char`型变量是用来存储Unicode编码的字符的，unicode编码字符集中包含了汉字，所以，`char`型变量中当然可以存储汉字啦。不过，如果某个特殊的汉字没有被包含在unicode编码字符集中，那么，这个`char`型变量中就不能存储这个特殊汉字。补充说明：unicode编码占用两个字节，所以，`char`类型的变量也是占用两个字节。

8、用最有效率的方法算出2乘以8等于几?

`2<< 3,` (左移三位)因为将一个数左移n位，就相当于乘以了2的n次方，那么，一个数乘以8只要将其左移3位即可，而位运算cpu直接支持的，效率最高，所以，2乘以8等于几的最效率的方法是`2<< 3`。

9、使用final关键字修饰一个变量时，是引用不能变，还是引用的对象不能变?

使用`final`关键字修饰一个变量时，是指引用变量不能变，引用变量所指向的对象中的内容还是可以改变的。例如，对于如下语句：

```
final StringBuffer a=new StringBuffer("immutable");
```

执行如下语句将报告编译期错误：

```
a=new StringBuffer("");
```

但是，执行如下语句则可以通过编译：

```
a.append(" broken!");
```

有人在定义方法的参数时，可能想采用如下形式来阻止方法内部修改传进来的参数对象：

```
public void method(final StringBuffer param){  
}
```

实际上，这是办不到的，在该方法内部仍然可以增加如下代码来修改参数对象：

```
param.append("a");
```

10.静态变量和实例变量的区别?

在语法定义上的区别：静态变量前要加`static`关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以使用，静态变量则可以直接使用类名来引用。

例如，对于下面的程序，无论创建多少个实例对象，永远都只分配了一个staticVar变量，并且每创建一个实例对象，这个staticVar就会加1；但是，每创建一个实例对象，就会分配一个instanceVar，即可能分配多个instanceVar，并且每个instanceVar的值都只自加了1次。

```
int arr[][] = {{1,2,3},{4,5,6,7},{9}};
boolean found = false;
for(int i=0;i<arr.length&&!found;i++) {
    for(int j=0;j<arr[i].length;j++){
        System.out.println("i=" + i + ",j=" + j);
        if(arr[i][j] == 5) {
            found = true;
            break;
        }
    }
}
```

11、是否可以从一个static方法内部发出对非static方法的调用？

不可以。因为非static方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而static方法调用时不需要创建对象，可以直接调用。也就是说，当一个static方法被调用时，可能还没有创建任何实例对象，如果从一个static方法中发出对非static方法的调用，那个非static方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个static方法内部发出对非static方法的调用。

12、Integer与int的区别

int是java提供的8种原始数据类型之一。Java为每个原始类型提供了封装类，Integer是java为int提供的封装类。int的默认值为0，而Integer的默认值为null，即Integer可以区分出未赋值和值为0的区别，int则无法表达出未赋值的情况。

例如：要想表达出没有参加考试和考试成绩为0的区别，则只能使用Integer。在JSP开发中，Integer的默认为null，所以用el表达式在文本框中显示时，值为空白字符串，而int默认的默认值为0，所以用el表达式在文本框中显示时，结果为0，所以，int不适合作为web层的表单数据的类型。

在Hibernate中，如果将OID定义为Integer类型，那么Hibernate就可以根据其值是否为null而判断一个对象是否是临时的，如果将OID定义为了int类型，还需要在hbm映射文件中设置其unsaved-value属性为0。

另外，Integer提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数，Integer中还定义了表示整数的最大值和最小值的常量。

13、Math.round(11.5)等于多少？Math.round(-11.5)等于多少？

Math类中提供了三个与取整有关的方法：ceil、floor、round，这些方法的作用与它们的英文名称的含义相对应。

例如，ceil的英文意义是天花板，该方法就表示向上取整，Math.ceil(11.3)的结果为12,Math.ceil(-11.3)的结果是-11；floor的英文意义是地板，该方法就表示向下取整，Math.floor(11.6)的结果为11,Math.floor(-11.6)的结果是-12；最难掌握的是round方法，它表示“四舍五入”，算法为Math.floor(x+0.5)，即将原来的数字加上0.5后再向下取整，所以，Math.round(11.5)的结果为12，Math.round(-11.5)的结果为-11。

这里有一些笔误，floor的英文意义是地板，该方法就表示向下取整，Math.floor(11.6)的结果为11,Math.floor(-11.6)的结果是-12；

14、Overload和Override的区别？Overloaded的方法是否可以改变返回值的类型？

Overload是重载的意思，Override是覆盖的意思，也就是重写。

重载Overload表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同（即参数个数或类型不同）。

重写Override表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了，这也是面向对象编程的多态性的一种表现。子类覆盖父类的方法时，只能比父类抛出更少的异常，或者是抛出父类抛出的异常的子异常，因为子类可以解决父类的一些问题，不能比父类有更多的问题。子类方法的访问权限只能比父类的更大，不能更小。如果父类的方法是private类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。

至于Overloaded的方法是否可以改变返回值的类型这个问题，要看你到底想问什么呢？这个题目很模糊。如果几个Overloaded的方法的参数列表不一样，它们的返回者类型当然也可以不一样。但我估计你想问的问题是：如果两个方法的参数列表完全一样，是否可以让它们的返回值不同来实现重载Overload。这是不行的，我们可以用反证法来说明这个问题，因为我们有时候调用一个方法时也可以不定义返回结果变量，即不要关心其返回结果，例如，我们调用map.remove(key)方法时，虽然remove方法有返回值，但是我们通常都不会定义接收返回结果的变量，这时候假设该类中有两个名称和参数列表完全相同的方法，仅仅是返回类型不同，java就无法确定编程者到底是想调用哪个方法了，因为它无法通过返回结果类型来判断。

override可以翻译为覆盖，从字面就可以知道，它是覆盖了一个方法并且对其重写，以求达到不同的作用。对我们来说最熟悉的覆盖就是对接口方法的实现，在接口中一般只是对方法进行了声明，而我们在实现时，就需要实现接口声明的所有方法。除了这个典型的用法以外，我们在继承中也可能会在子类覆盖父类中的方法。在覆盖要注意以下的几点：

- 1、覆盖的方法的标志必须要和被覆盖的方法的标志完全匹配，才能达到覆盖的效果；
- 2、覆盖的方法的返回值必须和被覆盖的方法的返回一致；
- 3、覆盖的方法所抛出的异常必须和被覆盖方法的所抛出的异常一致，或者是其子类；
- 4、被覆盖的方法不能为private，否则在其子类中只是新定义了一个方法，并没有对其进行覆盖。

Overload对我们来说可能比较熟悉，可以翻译为重载，它是指我们可以定义一些名称相同的方法，通过定义不同的输入参数来区分这些方法，然后再调用时，VM就会根据不同的参数样式，来选择合适的方法执行。在使用重载要注意以下的几点：

- 1、在使用重载时只能通过不同的参数样式。例如，不同的参数类型，不同的参数个数，不同的参数顺序（当然，同一方法内的几个参数类型必须不一样，例如可以是fun(int,float)，但是不能为fun(int,int)）；
- 2、不能通过访问权限、返回类型、抛出的异常进行重载；
- 3、方法的异常类型和数目不会对重载造成影响；

4、对于继承来说，如果某一方法在父类中是访问权限是private，那么就不能在子类对其进行重载，如果定义的话，也只是定义了一个新方法，而不会达到重载的效果。

15、接口是否可继承接口?抽象类是否可实现(implements)接口?抽象类是否可继承具体类(concreteclass)?抽象类中是否可以有静态的main方法?

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。抽象类中可以有静态的main方法。

备注：只要明白了接口和抽象类的本质和作用，这些问题都很好回答，你想想，如果你是java语言的设计者，你是否会提供这样的支持，如果不提供的话，有什么理由吗？如果你没有道理不提供，那答案就是肯定的了。

只要记住抽象类与普通类的唯一区别就是不能创建实例对象和允许有abstract方法。

16、Java中实现多态的机制是什么？

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

17、abstractclass和interface语法上有什么区别？

1. 抽象类可以有构造方法，接口中不能有构造方法。
2. 抽象类中可以有普通成员变量，接口中没有普通成员变量
3. 抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。
4. 抽象类中的抽象方法的访问类型可以是public, protected和（默认类型,虽然eclipse下不报错，但应该也不行），但接口中的抽象方法只能是public类型的，并且默认即为public abstract类型。
5. 抽象类中可以包含静态方法，接口中不能包含静态方法
6. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是public static final类型，并且默认即为public static final类型。
7. 一个类可以实现多个接口，但只能继承一个抽象类。

18、abstract的method是否可同时是static,是否可同时是native，是否可同时是synchronized?

abstract的method不可以是static的，因为抽象的方法是要被子类实现的，而static与子类扯不上关系！

native方法表示该方法要用另外一种依赖平台的编程语言实现的，不存在着被子类实现的问题，所以，它也不能是抽象的，不能与abstract混用。例如，FileOutputStream类要硬件打交道，底层的实现用的是操作系统相关的api实现；例如，在windows用c语言实现的，所以，查看jdk的源代码，可以发现 FileOutputStream的open方法的定义如下：

```
private native void open(String name) throws FileNotFoundException;
```

如果我们要用java调用别人写的c语言函数，我们是无法直接调用的，我们需要按照java的要求写一个c语言的函数，又我们的这个c语言函数去调用别人的c语言函数。由于我们的c语言函数是按java的要求来写的，我们这个c语言函数就可以与java对接上，java那边的对接方式就是定义出与我们这个c函数相对应的方法，java中对应的方法不需要写具体的代码，但需要在前面声明native。

关于synchronized与abstract合用的问题，我觉得也不行，因为在我几年的学习和开发中，从来没见过到过这种情况，并且我觉得synchronized应该是作用在一个具体的方法上才有意义。而且，方法上的synchronized同步所使用的同步锁对象是this，而抽象方法上无法确定this是什么。

19、内部类可以引用它的包含类的成员吗？有没有什么限制？

完全可以。如果不是静态内部类，那没有什么限制！

如果你把静态嵌套类当作内部类的一种特例，那在这种情况下不可以访问外部类的普通成员变量，而只能访问外部类中的静态成员，例如，下面的代码：

```
class Outer
{
    static int x;
    static class Inner
    {
        void test()
        {
            System.out.println(x);
        }
    }
}
```

20、String s = "Hello";s = s + "world!";这两行代码执行后，原始的String对象中的内容到底变了没有？

没有。因为String被设计成不可变(immutable)类，所以它的所有对象都是不可变对象。在这段代码中，s原先指向一个String对象，内容是 "Hello"，然后我们对s进行了+操作，那么s所指向的那个对象是否发生了改变呢？答案是没有。这时，s不指向原来那个对象了，而指向了另一个 String对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是s这个引用变量不再指向它了。

通过上面的说明，我们很容易导出另一个结论，如果经常对字符串进行各种各样的修改，或者说，不可预见的修改，那么使用String来代表字符串的话会引起很大的内存开销。因为String对象建立之后不能再改变，所以对于每一个不同的字符串，都需要一个String对象来表示。这时，应该考虑使用 StringBuffer类，它允许修改，而不是每个不同的字符串都要生成一个新的对象。并且，这两种类的对象转换十分容易。

同时，我们还可以知道，如果要使用内容相同的字符串，不必每次都new一个String。例如我们要在构造器中对一个名叫s的String引用变量进行初始化，把它设置为初始值，应当这样做：

```
public class Demo {  
    private String s;  
    ...  
    public Demo {  
        s = "Initial value";  
    }  
    ...  
}
```

而非

```
s = new String("Initial value");
```

后者每次都会调用构造器，生成新对象，性能低下且内存开销大，并且没有意义，因为String对象不可改变，所以对于内容相同的字符串，只要一个String对象来表示就可以了。也就是说，多次调用上面的构造器创建多个对象，他们的 String类型属性s都指向同一个对象。

上面的结论还基于这样一个事实：对于字符串常量，如果内容相同，Java认为它们代表同一个String对象。而用关键字new调用构造器，总是会创建一个新的对象，无论内容是否相同。

至于为什么要把String类设计成不可变类，是它的用途决定的。其实不只String，很多Java标准类库中的类都是不可变的。在开发一个系统的时候，我们有时候也需要设计不可变类，来传递一组相关的值，这也是面向对象思想的体现。不可变类有一些优点，比如因为它的对象是只读的，所以多线程并发访问也不会有任何问题。当然也有一些缺点，比如每个不同的状态都要一个对象来代表，可能会造成性能上的问题。所以Java标准类库还提供了一个可变版本，即StringBuffer。

21、ArrayList和Vector的区别

这两个类都实现了List接口（List接口继承了Collection接口），他们都是有序集合，即存储在这两个集合中的元素的位置都是有顺序的，相当于一种动态的数组，我们以后可以按位置索引号取出某个元素，并且其中的数据是允许重复的，这是与HashSet之类的集合的最大不同处，HashSet之类的集合不可以按索引号去检索其中的元素，也不允许有重复的元素。

ArrayList与Vector的区别主要包括两个方面：.

(1) 同步性：

Vector是线程安全的，也就是说它的方法之间是线程同步的，而ArrayList是线程不安全的，它的方法之间是线程不同步的。如果只有一个线程会访问到集合，那最好是使用ArrayList，因为它不考虑线程安全，效率会高些；如果有多个线程会访问到集合，那最好是使用Vector，因为不需要我们自己再去考虑和编写线程安全的代码。

(2) 数据增长：

ArrayList与Vector都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时，就需要增加ArrayList与Vector的存储空间，每次要增加存储空间时，不是只增加一个存储单元，而是增加多个存储单元，每次增加的存储单元的个数在内存空间利用与程序效率之间要取得一定的平衡。Vector默认增长为原来两倍，而ArrayList的增长策略在文档中没有明确规定（从源代码看到的是增长为原来的1.5倍）。ArrayList与Vector都可以设置初始的空间大小，Vector还可以设置增长的空间大小，而ArrayList没有提供设置增长空间的方法。

总结：即Vector增长原来的一倍，ArrayList增加原来的0.5倍。

22、HashMap和Hashtable的区别

HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，主要区别在于HashMap允许空（null）键值（key），由于非线程安全，在只有一个线程访问的情况下，效率要高于Hashtable。

HashMap允许将null作为一个entry的key或者value，而Hashtable不允许。

HashMap把Hashtable的contains方法去掉了，改成containsValue和containsKey。因为contains方法容易让人引起误解。

Hashtable继承自Dictionary类，而HashMap是Java1.2引进的Map interface的一个实现。

最大的不同是，Hashtable的方法是Synchronized的，而HashMap不是，在多个线程访问Hashtable时，不需要自己为它的方法实现同步，而HashMap就必须为之提供同步。

就HashMap与HashTable主要从三方面来说。

一.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

二.同步性:Hashtable是线程安全的，也就是说是同步的，而HashMap是线程不安全的，不是同步的

三.值：只有HashMap可以让你将空值作为一个表的条目的key或value

23、List和Map区别？

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合，List中存储的数据是有顺序，并且允许重复；Map中存储的数据是没有顺序的，其键是不能重复的，它的值是可以有重复的。

24、List, Set, Map是否继承自Collection接口？

List, Set是，Map不是

25、List、Map、Set三个接口，存取元素时，各有什么特点？

（这样的题比较考水平，两个方面的水平：一是要真正明白这些内容，二是要有较强的总结和表述能力。）

首先，List与Set具有相似性，它们都是单列元素的集合，所以，它们有一个共同的父接口，叫Collection。Set里面不允许有重复的元素，即不能有两个相等（注意，不是仅仅是相同）的对象，即假设Set集合中有了一个A对象，现在我要向Set集合再存入一个B对象，但B对象与A对象equals相等，则B对象存储不进去，所以，Set集合的add方法有一个boolean的返回值，当集合中没有某个元素，此时add方法可成功加入该元素时，则返回true，当集合含有与某个元素equals相等的元素时，此时add方法无法加入该元素，返回结果为false。Set取元素时，不能细说要取第几个，只能以Iterator接口取得所有的元素，再逐一遍历各个元素。

List表示有先后顺序的集合，注意，不是那种按年龄、按大小、按价格之类的排序。当我们多次调用add(Obj)方法时，每次加入的对象就像火车站买票有排队顺序一样，按先来后到的顺序排序。有时候，也可以插队，即调用add(int index, Obj e)方法，就可以指定当前对象在集合中的存放位置。一个对象可以被反复存储进List中，每调用一次add方法，这个对象就被插入进集合中一次，其实，并不是把这个对象本身存储进了集合中，而是在集合中用一个索引变量指向这个对象，当这个对象被add多次时，即相当于集合中有多个索引指向了这个对象，如图x所示。List除了可以用Iterator接口取得所有的元素，再逐一遍历各个元素之外，还可以调用get(index i)来明确说明取第几个。

Map与List和Set不同，它是双列的集合，其中有put方法，定义如下：put(obj key,obj value)，每次存储时，要存储一对key/value，不能存储重复的key，这个重复的规则也是按equals比较相等。取则可以根据key获得相应的value，即get(Object key)返回值为key所对应的value。另外，也可以获得所有的key的结合，还可以获得所有的value的结合，还可以获得key和value组合成的Map.Entry对象的集合。

List以特定次序来持有元素，可有重复元素。Set无法拥有重复元素，内部排序。Map保存key-value值，value可多值。

26、说出ArrayList,Vector,LinkedList的存储性能和特性

ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢，Vector由于使用了synchronized方法（线程安全），通常性能上较ArrayList差。而LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，索引就变慢了，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

LinkedList也是线程不安全的，LinkedList提供了一些方法，使得LinkedList可以被当作堆栈和队列来使用。

27、去掉一个Vector集合中重复的元素

```
Vector newVector = new Vector();
For (int i=0;i<vector.size();i++)
{
    Object obj = vector.get(i);
    if(!newVector.contains(obj));
        newVector.add(obj);
}
```

还有一种简单的方式，利用了Set不允许重复元素：

```
HashSetset = new HashSet(vector);
```

28、Collection和Collections的区别。

Collection是集合类的上级接口，继承他的接口主要有Set和List。

Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

29、Set里的元素是不能重复的，那么用什么方法来区分重复与否呢？是用==还是equals()？它们有何区别？

Set里的元素是不能重复的，元素重复与否是使用equals()方法进行判断的。

==和equal区别也是考烂了的题，这里说一下：

`==`操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用`==`操作符。

`equals`方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。

比如：两条new语句创建了两个对象，然后用`a/b`这两个变量分别指向了其中一个对象，这是两个不同的对象，它们的首地址是不同的，即`a`和`b`中存储的数值是不相同的，所以，表达式`a==b`将返回`false`，而这两个对象中的内容是相同的，所以，表达式`a.equals(b)`将返回`true`。

30、你所知道的集合类都有哪些？主要方法？

最常用的集合类是 List 和 Map。 List的具体实现包括 ArrayList和 Vector，它们是可变大小的列表，比较适合构建、存储和操作任何类型对象的元素列表。 List适用于按数值索引访问元素的情形。

Map 提供了一个更通用的元素存储方法。 Map集合类用于存储元素对（称作“键”和“值”），其中每个键映射到一个值。

它们都有增删改查的方法。

对于set，大概的方法是add,remove, contains等

对于map，大概的方法就是put,remove, contains等

List类会有`get(int index)`这样的方法，因为它可以按顺序取元素，而set类中没有`get(int index)`这样的方法。 List和set都可以迭代出所有元素，迭代时先要得到一个`iterator`对象，所以，set和list类都有一个`iterator`方法，用于返回那个`iterator`对象。 map可以返回三个集合，一个是返回所有的key的集合，另外一个返回的是所有value的集合，再一个返回的key和value组合成的EntrySet对象的集合，map也有get方法，参数是key，返回值是key对应的value，这个自由发挥，也不是考记方法的能力，这些编程过程中会有提示，结合他们三者的不同说一下用法就行。

31、`String s = new String("xyz");`创建了几个StringObject？是否可以继承String类？

两个或一个都有可能，“xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量“xyz”不管出现多少遍，都是缓冲区中的那一个。 NewString每写一遍，就创建一个新的对象，它使用常量“xyz”对象的内容来创建出一个新String对象。如果以前就用过‘xyz’，那么这里就不会创建“xyz”了，直接从缓冲区拿，这时创建了一个StringObject；但如果以前没有用过“xyz”，那么此时就会创建一个对象并放入缓冲区，这种情况它创建两个对象。至于String类是否继承，答案是否定的，因为String默认final修饰，是不可继承的。

32、String和StringBuffer的区别

JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，即包含多个字符的字符串数据。这个String类提供了数值不可改变的字符串。而这个StringBuffer类提供的字符串可以进行修改。当你知道字符数据要改变的时候你就可以使用StringBuffer。典型地，你可以使用StringBuffers来动态构造字符数据。

33、下面这条语句一共创建了多少个对象：String

```
s="a"+"b"+"c"+"d";
```

对于如下代码：

```
String s1 = "a";
String s2 = s1 + "b";
String s3 = "a" + "b";
System.out.println(s2 == "ab");
System.out.println(s3 == "ab");
```

第一条语句打印的结果为false，第二条语句打印的结果为true，这说明javac编译可以对字符串常量直接相加的表达式进行优化，不必要等到运行期再去进行加法运算处理，而是在编译时去掉其中的加号，直接将其编译成一个这些常量相连的结果。

题目中的第一行代码被编译器在编译时优化后，相当于直接定义了一个"abcd"的字符串，所以，上面的代码应该只创建了一个String对象。写如下两行代码，

对于如下代码：

```
String s = "a" + "b" + "c" + "d";
System.out.println(s == "abcd");
```

最终打印的结果应该为true。

34、try {}里有一个return语句，那么紧跟在这个try后的finally{}里的code会不会被执行，什么时候被执行，在return前还是后？

我们知道finally{}中的语句是一定会执行的，那么这个可能正常脱口而出就是return之前，return之后可能就出了这个方法了，鬼知道跑哪里去了，但更准确的应该是在return中间执行，请看下面程序代码的运行结果：

```
public classTest {
    public static void main(String[] args) {
        System.out.println(newTest().test());
    }
    static int test() {
        int x = 1;
        try {
            return x;
        }
        finally {
            ++x;
        }
    }
}
```

-----执行结果 -----

1

运行结果是1，为什么呢？主函数调用子函数并得到结果的过程，好比主函数准备一个空罐子，当子函数要返回结果时，先把结果放在罐子里，然后再将程序逻辑返回到主函数。所谓返回，就是子函数说，我不运行了，你主函数继续运行吧，这没什么结果可言，结果是在说这话之前放进罐子里的。

35、final, finally, finalize的区别。

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。内部类要访问局部变量，局部变量必须定义成final类型。

finally是异常处理语句结构的一部分，表示总是执行。

finalize是Object类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。但是JVM不保证此方法总被调用

36、运行时异常与一般异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java编译器要求方法必须声明抛出可能发生的非运行时异常，但是并不要求必须声明抛出未被捕获的运行时异常。

37、error和exception有什么区别？

error 表示恢复是不可能但很困难的情况下的一种严重问题。比如说内存溢出。不可能指望程序能处理这样的情况。exception表示一种设计或实现问题。也就是说，它表示如果程序运行正常，从不会发生的情况。

38、简单说说Java中的异常处理机制的简单原理和应用。

异常是指java程序运行时（非编译）所发生的非正常情况或错误，与现实生活中的事件很相似，现实生活中的事件可以包含事件发生的时间、地点、人物、情节等信息，可以用一个对象来表示，Java使用面向对象的方式来处理异常，它把程序中发生的每个异常也都分别封装到一个对象来表示的，该对象中包含有异常的信息。

Java对异常进行了分类，不同类型的异常分别用不同的Java类表示，所有异常的根类为java.lang.Throwable， Throwable下面又派生了两个子类：

Error和Exception， Error表示应用程序本身无法克服和恢复的一种严重问题，程序只有奔溃了，例如，说内存溢出和线程死锁等系统问题。

Exception表示程序还能够克服和恢复的问题，其中又分为系统异常和普通异常：

系统异常是软件本身缺陷所导致的问题，也就是软件开发人员考虑不周所导致的问题，软件使用者无法克服和恢复这种问题，但在这种问题下还可以让软件系统继续运行或者让软件挂掉，例如，数组脚本越界（ArrayIndexOutOfBoundsException），空指针异常（NullPointerException）、类转换异常（ClassCastException）；

普通异常是运行环境的变化或异常所导致的问题，是用户能够克服的问题，例如，网络断线，硬盘空间不够，发生这样的异常后，程序不应该死掉。

java为系统异常和普通异常提供了不同的解决方案，编译器强制普通异常必须try..catch处理或用throws声明继续抛给上层调用方法处理，所以普通异常也称为checked异常，而系统异常可以处理也可以不处理，所以，编译器不强制用try..catch处理或用throws声明，所以系统异常也称为unchecked异常。

39、Java 中堆和栈有什么区别？

JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享。

栈：在函数中定义的一些基本类型的变量和对象的引用变量都是在函数的栈内存中分配，当在一段代码块定义一个变量时，Java 就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java 会自动释放掉为该变量分配的内存空间，该内存空间可以立即被另作它用。

堆：堆内存用来存放由 new 创建的对象和数组，在堆中分配的内存，由 Java 虚拟机的自动垃圾回收器来管理。在堆中产生了一个数组或者对象之后，还可以在栈中定义一个特殊的变量，让栈中的这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或者对象，引用变量就相当于是为数组或者对象起的一个名称。

40、能将 int 强制转换为 byte 类型的变量吗？如果该值大于 byte 类型的范围，将会出现什么现象？

我们可以做强制转换，但是 Java 中 int 是 32 位的，而 byte 是 8 位的，所以，如果强制转化，int 类型的高 24 位将会被丢弃，因为byte 类型的范围是从 -128 到 128。这里笔误：-128到127

41、a.hashCode() 有什么用？与 a.equals(b) 有什么关系？

hashCode() 方法对应用对象整型的 hash 值。它常用于基于 hash 的集合类，如 Hashtable、HashMap、LinkedHashMap 等等。它与 equals() 方法关系特别紧密。根据 Java 规范，两个使用 equal() 方法来判断相等的对象，必须具有相同的 hash code。

42、字节流与字符流的区别

要把一段二进制数据数据逐一输出到某个设备中，或者从某个设备中逐一读取一段二进制数据，不管输入输出设备是什么，我们要用统一的方式来完成这些操作，用一种抽象的方式进行描述，这个抽象描述方式起名为IO流，对应的抽象类为OutputStream和InputStream，不同的实现类就代表不同的输入和输出设备，它们都是针对字节进行操作的。

计算机中的一切最终都是二进制的字节形式存在。对于经常用到的中文字符，首先要得到其对应的字节，然后将字节写入到输出流。读取时，首先读到的是字节，可是我们要把它显示为字符，我们需要将字节转换成字符。由于这样的需求很广泛，Java专门提供了字符流包装类。

底层设备永远只接受字节数据，有时候要写字符串到底层设备，需要将字符串转成字节再进行写入。字符流是字节流的包装，字符流则是直接接受字符串，它内部将串转成字节，再写入底层设备，这为我们向IO设备写入或读取字符串提供了一点点方便。

字符串向字节转换时，要注意编码的问题，因为字符串转成字节数组，其实是转成该字符的某种编码的字节形式，读取也是反之的道理。

43、什么是java序列化，如何实现java序列化？或者请解释Serializable接口的作用。

我们有时候将一个java对象变成字节流的形式传出去或者从一个字节流中恢复成一个java对象，例如，要将java对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个java对象变成某个格式的字节流再传输。

但是，jre本身就提供了这种支持，我们可以调用OutputStream的writeObject方法来做，如果要让java帮我们做，要被传输的对象必须实现Serializable接口，这样，javac编译时就会进行特殊处理，编译的类才可以被writeObject方法操作，这就是所谓的序列化。需要被序列化的类必须实现Serializable接口，该接口是一个mini接口，其中没有需要实现方法，implements Serializable只是为了标注该对象是可被序列化的。

例如，在web开发中，如果对象被保存在了Session中，tomcat在重启时要把Session对象序列化到硬盘，这个对象就必须实现Serializable接口。如果对象要经过分布式系统进行网络传输，被传输的对象就必须实现Serializable接口。

44、描述一下JVM加载class文件的原理机制？

JVM中类的装载是由ClassLoader和它的子类来实现的，Java ClassLoader是一个重要的Java运行时系统组件。它负责在运行时查找和装入类文件的类。

45、heap和stack有什么区别。

java的内存分为两类，一类是栈内存，一类是堆内存。栈内存是指程序进入一个方法时，会为这个方法单独分配一块私属存储空间，用于存储这个方法内部的局部变量，当这个方法结束时，分配给这个方法的栈会释放，这个栈中的变量也将随之释放。

堆是与栈作用不同的内存，一般用于存放不在当前方法栈中的那些数据，例如，使用new创建的对象都放在堆里，所以，它不会随方法的结束而消失。方法中的局部变量使用final修饰后，放在堆中，而不是栈中。

46、GC是什么？为什么要有GC？

GC是垃圾收集的意思（Garbage Collection），内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。

47、垃圾回收的优点和原理，并考虑2种回收机制。

Java语言中一个显著的特点就是引入了垃圾回收机制，使C++程序员最头疼的内存管理的问题迎刃而解，它使得Java程序员在编写程序的时候不再需要考虑内存管理。由于垃圾回收机制，Java中的对象不再有“作用域”的概念，只有对象的引用才有“作用域”。

垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。

回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

48、垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通知虚拟机进行垃圾回收？

对于GC来说，当程序员创建对象时，GC就开始监控这个对象的地址、大小以及使用情况。通常，GC采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当GC确定一些对象为"不可达"时，GC就有责任回收这些内存空间。

程序员可以手动执行System.gc()，通知GC运行，但是Java语言规范并不保证GC一定会执行。

49、Java 中， throw 和 throws 有什么区别

throw 用于抛出 java.lang.Throwable 类的一个实例化对象，意思是说你可以通过关键字 throw 抛出一个Exception，如：

```
throw new IllegalArgumentException("XXXXXXXXXX")
```

而throws 的作用是作为方法声明和签名的一部分，方法被抛出相应的异常以便调用者能处理。Java 中，任何未处理的受检查异常强制在 throws 子句中声明。

50、java中会存在内存泄漏吗，请简单描述。

先解释什么是内存泄漏：所谓内存泄露就是指一个不再被程序使用的对象或变量一直被占据在内存中。java中有垃圾回收机制，它可以保证当对象不再被引用的时候，对象将自动被垃圾回收器从内存中清除掉。

由于Java使用有向图的方式进行垃圾回收管理，可以消除引用循环的问题，例如有两个对象，相互引用，只要它们和根进程不可达，那么GC也是可以回收它们的。

java中的内存泄露的情况：长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是java中内存泄露的发生场景，通俗地说，就是程序员可能创建了一个对象，以后一直不再使用这个对象，这个对象却一直被引用，即这个对象无用但是却无法被垃圾回收器回收的，这就是java中可能出现内存泄露的情况，例如，缓存系统，我们加载了一个对象放在缓存中(例如放在一个全局map对象中)，然后一直不再使用它，这个对象一直被缓存引用，但却不再被使用。

51、说一说Servlet的生命周期？

Servlet有良好的生存期的定义，包括加载和实例化、初始化、处理请求以及服务结束。这个生存期由javax.servlet.Servlet接口的init(),service()和destroy方法表达。

Servlet被服务器实例化后，容器运行其init方法，请求到达时运行其service方法，service方法自动派遣运行与请求对应的doXXX方法（doGet, doPost）等，当服务器决定将实例销毁的时候调用其destroy方法。

web容器加载servlet，生命周期开始。通过调用servlet的init()方法进行servlet的初始化。通过调用service()方法实现，根据请求的不同调用不同的do***()方法。结束服务，web容器调用servlet的destroy()方法。

52、Servlet API中forward()与redirect()的区别？

1.从地址栏显示来说

forward是服务器请求资源,服务器直接访问目标地址的URL,把那个URL的响应内容读取过来,然后把这些内容再发给浏览器.浏览器根本不知道服务器发送的内容从哪里来的,所以它的地址栏还是原来的地址.

redirect是服务端根据逻辑,发送一个状态码,告诉浏览器重新去请求那个地址.所以地址栏显示的是新的URL.所以redirect等于客户端向服务器端发出两次request,同时也接受两次response.

2.从数据共享来说

forward:转发页面和转发到的页面可以共享request里面的数据.

redirect:不能共享数据.

redirect不仅可以重定向到当前应用程序的其他资源,还可以重定向到同一个站点上的其他应用程序中的资源,甚至是使用绝对URL重定向到其他站点的资源.

forward方法只能在同一个Web应用程序内的资源之间转发请求.forward 是服务器内部的一种操作.

redirect 是服务器通知客户端,让客户端重新发起请求.

所以,你可以说 redirect 是一种间接的请求,但是你不能说"一个请求是属于forward还是redirect "

3.从运用地方来说

forward:一般用于用户登陆的时候,根据角色转发到相应的模块.

redirect:一般用于用户注销登陆时返回主页面和跳转到其它的网站等.

4.从效率来说

forward:高.

redirect:低.

53、request.getAttribute()和request.getParameter()有何区别？

1, request.getParameter()取得是通过容器的实现来取得通过类似post, get等方式传入的数据。

request.setAttribute()和getAttribute()只是在web容器内部流转,仅仅是请求处理阶段。

2, getAttribute是返回对象,getParameter返回字符串

3, getAttribute()一向是和setAttribute()一起使用的,只有先用setAttribute()设置之后,才能够通过getAttribute()来获得值,它们传递的是Object类型的数据。而且必须在同一个request对象中使用才有效。,而getParameter()是接收表单的get或者post提交过来的参数

54, jsp静态包含和动态包含的区别

1、<%@include file="xxx.jsp"%>为jsp中的编译指令,其文件的包含是发生在jsp向servlet转换的时期,而<jsp:include page="xxx.jsp">是jsp中的动作指令,其文件的包含是发生在编译时期,也就是将java文件编译为class文件的时期

2、使用静态包含只会产生一个class文件,而使用动态包含会产生多个class文件

3、使用静态包含,包含页面和被包含页面的request对象为同一对象,因为静态包含只是将被包含的页面的内容复制到包含的页面中去;而动态包含包含页面和被包含页面不是同一个页面,被包含的页面的request对象可以取到的参数范围要相对大些,不仅可以取到传递到包含页面的参数,同样也能取得在包含页面向下传递的参数

55, MVC的各个部分都有那些技术来实现?如何实现?

MVC是Model - View - Controller的简写。Model代表的是应用的业务逻辑（通过JavaBean, EJB组件实现），View是应用的表示面（由JSP页面产生），Controller是提供应用的处理过程控制（一般是一个Servlet），通过这种设计模型把应用逻辑，处理过程和显示逻辑分成不同的组件实现。这些组件可以进行交互和重用。

56, jsp有哪些内置对象?作用分别是什么?

JSP共有以下9个内置的对象：

- 1, request 用户端请求，此请求会包含来自GET/POST请求的参数
- 2, response 网页传回用户端的回应
- 3, pageContext 网页的属性是在这里管理
- 4, session 与请求有关的会话期
- 5, application servlet 正在执行的内容
- 6, out 用来传送回应的输出
- 7, config servlet的构架部件
- 8, page JSP网页本身
- 9, exception 针对错误网页，未捕捉的例外

57, Http中, get和post方法的区别

- 1, Get是向服务器发索取数据的一种请求，而Post是向服务器提交数据的一种请求
- 2, Get是获取信息，而不是修改信息，类似数据库查询功能一样，数据不会被修改
- 3, Get请求的参数会跟在url后进行传递，请求的数据会附在URL之后，以?分割URL和传输数据，参数之间以&相连,%XX中的XX为该符号以16进制表示的ASCII，如果数据是英文字母/数字，原样发送，如果是空格，转换为+，如果是中文/其他字符，则直接把字符串用BASE64加密。
- 4, Get传输的数据有大小限制，因为GET是通过URL提交数据，那么GET可提交的数据量就跟URL的长度有直接关系了，不同的浏览器对URL的长度的限制是不同的。
- 5, GET请求的数据会被浏览器缓存起来，用户名和密码将明文出现在URL上，其他人可以查到历史浏览记录，数据不太安全。

在服务器端，用Request.QueryString来获取Get方式提交来的数据

Post请求则作为http消息的实际内容发送给web服务器，数据放置在HTML Header内提交，Post没有限制提交的数据。Post比Get安全，当数据是中文或者不敏感的数据，则用get，因为使用get，参数会显示在地址，对于敏感数据和不是中文字符的数据，则用post。

6, POST表示可能修改改变服务器上的资源的请求，在服务器端，用Post方式提交的数据只能用Request.Form来获取。

(仅供参考，如果有更好的回答，欢迎探讨)

58, 什么是cookie? Session和cookie有什么区别?

Cookie是会话技术,将用户的信息保存到浏览器的对象.

区别:

- (1)cookie数据存放在客户的浏览器上, session数据放在服务器上
- (2)cookie不是很安全, 别人可以分析存放在本地的COOKIE并进行COOKIE欺骗, 如果主要考虑到安全应当使用session
- (3)session会在一定时间内保存在服务器上. 当访问增多, 会比较占用你服务器的性能, 如果主要考虑到减轻服务器性能方面, 应当使用COOKIE
- (4)单个cookie在客户端的限制是3K, 就是说一个站点在客户端存放的COOKIE不能3K.

结论:

将登陆信息等重要信息存放在SESSION;其他信息如果需要保留, 可以放在COOKIE中。

59, jsp和servlet的区别、共同点、各自应用的范围?

JSP是Servlet技术的扩展, 本质上就是Servlet的简易方式。JSP编译后是“类servlet”。

Servlet和JSP最主要的不同点在于: Servlet的应用逻辑是在Java文件中, 并且完全从表示层中的HTML里分离开来。而JSP的情况是Java和HTML可以组合成一个扩展名为.jsp的文件。

JSP侧重于视图, Servlet主要用于控制逻辑。在struts框架中,JSP位于MVC设计模式的视图层, 而Servlet位于控制层。

60, tomcat容器是如何创建servlet类实例? 用到了什么原理?

当容器启动时, 会读取在webapps目录下所有的web应用中的web.xml文件, 然后对xml文件进行解析, 并读取servlet注册信息。然后, 将每个应用中注册的servlet类都进行加载, 并通过反射的方式实例化。(有时候也是在第一次请求时实例化)

在servlet注册时加上1如果为正数, 则在一开始就实例化, 如果不写或为负数, 则第一次请求实例化。

61, JDBC访问数据库的基本步骤是什么?

- 1, 加载驱动
- 2, 通过DriverManager对象获取连接对象Connection
- 3, 通过连接对象获取会话
- 4, 通过会话进行数据的增删改查, 封装对象
- 5, 关闭资源

62, 说说PreparedStatement和Statement的区别

- 1, 效率: 预编译会话比普通会话对象, 数据库系统不会对相同的sql语句不会再次编译
- 2, 安全性: 可以有效的避免sql注入攻击! sql注入攻击就是从客户端输入一些非法的特殊字符, 而使服务器端在构造sql语句的时候仍然能够正确构造, 从而收集程序和服务器的信息和数据。
比如: "select * from t_user where userName = " + userName + " and password = " + password + ""

如果用户名和密码输入的是'1' or '1'='1'; 则生产的sql语句是:

“select * from t_user where userName = '1' or '1' ='1' and password ='1' or '1'='1'”这个语句中的where部分没有起到对数据筛选的作用。

63，说说事务的概念，在JDBC编程中处理事务的步骤。

- 1 事务是作为单个逻辑工作单元执行的一系列操作。
- 2, 一个逻辑工作单元必须有四个属性，称为原子性、一致性、隔离性和持久性(ACID)属性，只有这样才能成为一个事务
3. 事务处理步骤：
 - 3, conn.setAutoCommit(false);设置提交方式为手工提交
 - 4, conn.commit()提交事务
 - 5, 出现异常，回滚 conn.rollback();

64，数据库连接池的原理。为什么要使用连接池。

- 1, 数据库连接是一件费时的操作，连接池可以使多个操作共享一个连接。
- 2, 数据库连接池的基本思想就是为数据库连接建立一个“缓冲池”。预先在缓冲池中放入一定数量的连接，当需要建立数据库连接时，只需从“缓冲池”中取出一个，使用完毕之后再放回去。我们可以通过设定连接池最大连接数来防止系统无尽的与数据库连接。更为重要的是我们可以通过连接池的管理机制监视数据库的连接的数量、使用情况，为系统开发、测试及性能调整提供依据。
- 3, 使用连接池是为了提高对数据库连接资源的管理

65，JDBC的脏读是什么？哪种数据库隔离级别能防止脏读？

当我们使用事务时，有可能会出现这样的情况，有一行数据刚更新，与此同时另一个查询读到了这个刚更新的值。这样就导致了脏读，因为更新的数据还没有进行持久化，更新这行数据的业务可能会进行回滚，这样这个数据就是无效的。数据库的TRANSACTIONREADCOMMITTED, TRANSACTIONREPEATABLEREAD, 和TRANSACTION_SERIALizable隔离级别可以防止脏读。

66，什么是幻读，哪种隔离级别可以防止幻读？

幻读是指一个事务多次执行一条查询返回的却是不同的值。假设一个事务正根据某个条件进行数据查询，然后另一个事务插入了一行满足这个查询条件的数据。之后这个事务再次执行了这条查询，返回的结果集中会包含刚插入的那条新数据。这行新数据被称为幻行，而这种现象就叫做幻读。

只有TRANSACTION_SERIALizable隔离级别才能防止产生幻读。

67，JDBC的DriverManager是用来做什么的？

JDBC的DriverManager是一个工厂类，我们通过它来创建数据库连接。当JDBC的Driver类被加载进来时，它会自己注册到DriverManager类里面
然后我们会把数据库配置信息传成DriverManager.getConnection()方法，DriverManager会使用注册到它里面的驱动来获取数据库连接，并返回给调用的程序。

68，execute, executeQuery, executeUpdate的区别是什么？

1, Statement的execute(String query)方法用来执行任意的SQL查询，如果查询的结果是一个ResultSet，这个方法就返回true。如果结果不是ResultSet，比如insert或者update查询，它就会返回false。我们可以通过它的getResultSet方法来获取ResultSet，或者通过getUpdateCount()方法来获取更新的记录条数。

2, Statement的executeQuery(String query)接口用来执行select查询，并且返回ResultSet。即使查询不到记录返回的ResultSet也不会为null。我们通常使用executeQuery来执行查询语句，这样的话如果传进来的是insert或者update语句的话，它会抛出错误信息为“executeQuery method can not be used for update”的java.util.SQLException。,

3, Statement的executeUpdate(String query)方法用来执行insert或者update/delete (DML) 语句，或者什么也不返回，对于DDL语句，返回值是int类型，如果是DML语句的话，它就是更新的条数，如果是DDL的话，就返回0。

只有当你不确定是什么语句的时候才应该使用execute()方法，否则应该使用executeQuery或者executeUpdate方法。

69, SQL查询出来的结果分页展示一般怎么做？

Oracle:

```
select * from
(select *,rownum as tempid from student ) t
where t.tempid between " + pageSize*(pageNumber-1) + " and " + pageSize*pageNumber
```

MySQL:

```
select * from students limit " + pageSize*(pageNumber-1) + "," + pageSize;
```

sql server: *

```
select top " + pageSize + " * from students where id not in +
(select top " + pageSize * (pageNumber-1) + id from students order by id) +
"order by id;*
```

70, JDBC的ResultSet是什么？

在查询数据库后会返回一个ResultSet，它就像是查询结果集的一张数据表。

ResultSet对象维护了一个游标，指向当前的数据行。开始的时候这个游标指向的是第一行。如果调用了ResultSet的next()方法游标会下移一行，如果没有更多的数据了，next()方法会返回false。可以在for循环中用它来遍历数据集。

默认的ResultSet是不能更新的，游标也只能往下移。也就是说你只能从第一行到最后一行遍历一遍。不过也可以创建可以回滚或者可更新的ResultSet

当生成ResultSet的Statement对象要关闭或者重新执行或是获取下一个ResultSet的时候，ResultSet对象也会自动关闭。

可以通过ResultSet的getter方法，传入列名或者从1开始的序号来获取列数据。

71, 谈谈你对Struts的理解。

1. struts是一个按MVC模式设计的Web层框架，其实它就是一个Servlet，这个Servlet名为ActionServlet，或是ActionServlet的子类。我们可以在web.xml文件中将符合某种特征的所有请求交给这个Servlet处理，这个Servlet再参照一个配置文件将各个请求分别分配给不同的action去处理。

(struts的配置文件可以有多个，可以按模块配置各自的配置文件，这样可以防止配置文件的过度膨胀)

2. ActionServlet把请求交给action去处理之前，会将请求参数封装成一个formbean对象（就是一个java类，这个类中的每个属性对应一个请求参数），

3. 要说明的是，ActionServlet把formbean对象传递给action的execute方法之前，可能会调用formbean的validate方法进行校验，只有校验通过后才将这个formbean对象传递给action的execute方法，否则，它将返回一个错误页面，这个错误页面由input属性指定。

4. action执行完后要返回显示的结果视图，这个结果视图是用一个ActionForward对象来表示的，actionForward对象通过struts-config.xml配置文件中的配置关联到某个jsp页面，因为程序中使用的是在struts-config.xml配置文件为jsp页面设置的逻辑名，这样可以实现action程序代码与返回的jsp页面名称的解耦。

(以上，也可以结合自己使用感受谈自己的看法)

72、谈谈你对Hibernate的理解。

1. 面向对象设计的软件内部运行过程可以理解成就是在不断创建各种新对象、建立对象之间的关系，调用对象的方法来改变各个对象的状态和对象消亡的过程，不管程序运行的过程和操作怎么样，本质上都是要得到一个结果，程序上一个时刻和下一个时刻的运行结果的差异就表现在内存中的对象状态发生了变化。

2. 为了在关机和内存空间不够的状况下，保持程序的运行状态，需要将内存中的对象状态保存到持久化设备和从持久化设备中恢复出对象的状态，通常都是保存到关系数据库来保存大量对象信息。从Java程序的运行功能上来讲，保存对象状态的功能相比系统运行的其他功能来说，应该是一个很不起眼的附属功能，java采用jdbc来实现这个功能，这个不起眼的功能却要编写大量的代码，而做的事情仅仅是保存对象和恢复对象，并且那些大量的jdbc代码并没有什么技术含量，基本上是采用一套例行公事的标准代码模板来编写，是一种苦活和重复性的工作。

3. 通过数据库保存java程序运行时产生的对象和恢复对象，其实就是实现了java对象与关系数据库记录的映射关系，称为ORM（即Object RelationMapping），人们可以通过封装JDBC代码来实现了这种功能，封装出来的产品称之为ORM框架，Hibernate就是其中的一种流行ORM框架。使用Hibernate框架，不用写JDBC代码，仅仅是调用一个save方法，就可以将对象保存到关系数据库中，仅仅是调用一个get方法，就可以从数据库中加载出一个对象。

4. 使用Hibernate的基本流程是：配置Configuration对象、产生SessionFactory、创建session对象，启动事务，完成CRUD操作，提交事务，关闭session。

5. 使用Hibernate时，先要配置hibernate.cfg.xml文件，其中配置数据库连接信息和方言等，还要为每个实体配置相应的hbm.xml文件，hibernate.cfg.xml文件中需要登记每个hbm.xml文件。

6. 在应用Hibernate时，重点要了解Session的缓存原理，级联，延迟加载和hql查询。

(以上，也可以结合自己使用DBC时的繁琐谈hibernate的感受)

73、谈谈你对Spring的理解。

1. Spring是实现了工厂模式的工厂类（在这里有必要解释清楚什么是工厂模式），这个类名为BeanFactory（实际上是一个接口），在程序中通常BeanFactory的子类ApplicationContext。Spring相当于一个大的工厂类，在其配置文件中通过元素配置用于创建实例对象的类名和实例对象的属性。

2. Spring提供了对IOC良好支持，IOC是一种编程思想，是一种架构艺术，利用这种思想可以很好地实现模块之间的解耦，IOC也称为DI（Dependency Injection）。

\3. Spring提供了对AOP技术的良好封装， AOP称为面向切面编程，就是系统中有很多各不相干的方法，在这些众多方法中要加入某种系统功能的代码，例如，加入日志，加入权限判断，加入异常处理，这种应用称为AOP。

实现AOP功能采用的是代理技术，客户端程序不再调用目标，而调用代理类，代理类与目标类对外具有相同的方法声明，有两种方式可以实现相同的方法声明，一是实现相同的接口，二是作为目标的子类。

在JDK中采用Proxy类产生动态代理的方式为某个接口生成实现类，如果要为某个类生成子类，则可以用CGLIB。在生成的代理类的方法中加入系统功能和调用目标类的相应方法，系统功能的代理以Advice对象进行提供，显然要创建出代理对象，至少需要目标类和Advice类。spring提供了这种支持，只需要在spring配置文件中配置这两个元素即可实现代理和aop功能。

(以上，也可以结合自己使用感受谈自己的看法)

74，谈谈Struts的优缺点

优点：

1. 实现MVC模式，结构清晰，使开发者只关注业务逻辑的实现。
2. 有丰富的tag可以用，Struts的标记库(Taglib)，如能灵活动用，则能大大提高开发效率
2. 页面导航使系统的脉络更加清晰。通过一个配置文件，即可把握整个系统各部分之间的联系，这对于后期的维护有着莫大的好处。尤其是当另一批开发者接手这个项目时，这种优势体现得更加明显。
3. 提供Exception处理机制。
4. 数据库链接池管理
5. 支持I18N

缺点：

一，转到展示层时，需要配置forward，如果有十个展示层的jsp，需要配置十次struts，而且还不包括有时候目录、文件变更，需要重新修改forward，注意，每次修改配置之后，要求重新部署整个项目，而tomcate这样的服务器，还必须重新启动服务器

二，Struts的Action必需是thread - safe方式，它仅仅允许一个实例去处理所有的请求。所以action用到的所有资源都必需统一同步，这个就引起了线程安全的问题。

三，测试不方便。Struts的每个Action都同Web层耦合在一起，这样它的测试依赖于Web容器，单元测试也很难实现。不过有一个Junit的扩展工具Struts TestCase可以实现它的单元测试。

四，类型的转换。Struts的FormBean把所有的数据都作为String类型，它可以使用工具Commons-Beanutils进行类型转化。但它的转化都是在Class级别，而且转化的类型是不可配置的。类型转化时的错误信息返回给用户也是非常困难的。

五，对Servlet的依赖性过强。Struts处理Action时必需要依赖ServletRequest和ServletResponse，所有它摆脱不了Servlet容器。

六，前端表达式语言方面。Struts集成了JSTL，所以它主要使用JSTL的表达式语言来获取数据。可是JSTL的表达式语言在Collection和索引属性方面处理显得很弱。

七，对Action执行的控制困难。Struts创建一个Action，如果想控制它的执行顺序将会非常困难。甚至你要重新去写Servlet来实现你的这个功能需求。

八，对Action执行前和后的处理。Struts处理Action的时候是基于class的hierarchies，很难在action处处理前和后进行操作。

九，对事件支持不够。在struts中，实际是一个表单Form对应一个Action类(或DispatchAction)，换一句话说：在Struts中实际是一个表单只能对应一个事件，struts这种事件方式称为application event，application event和component event相比是一种粗粒度的事件

75，iBatis与Hibernate有什么不同？

相同点：屏蔽jdbc api的底层访问细节，使用我们不用与jdbc api打交道，就可以访问数据。

jdbc api编程流程固定，还将sql语句与java代码混杂在了一起，经常需要拼凑sql语句，细节很繁琐。

ibatis的好处：屏蔽jdbc api的底层访问细节；将sql语句与java代码进行分离；提供了将结果集自动封装为实体对象和对象的集合的功能，queryForList返回对象集合，用queryForObject返回单个对象；提供了自动将实体对象的属性传递给sql语句的参数。

Hibernate是一个全自动的orm映射工具，它可以自动生成sql语句，ibatis需要我们自己在xml配置文件中写sql语句，hibernate要比ibatis功能负责和强大很多。因为hibernate自动生成sql语句，我们无法控制该语句，我们就无法去写特定的高效率的sql。对于一些不太复杂的sql查询，hibernate可以很好帮我们完成，但是，对于特别复杂的查询，hibernate就很难适应了，这时候用ibatis就是不错的选择，因为ibatis还是由我们自己写sql语句。

76，在hibernate进行多表查询每个表中各取几个字段，也就是说查询出来的结果集没有一个实体类与之对应如何解决？

解决方案一：按照Object[]数据取出数据，然后自己组bean

解决方案二：对每个表的bean写构造函数，比如表一要查出field1,field2两个字段，那么有一个构造函数就是Bean(type1filed1,type2

field2)，然后在hql里面就可以直接生成这个bean了。

77，介绍一下Hibernate的二级缓存

按照以下思路来回答：

- (1) 首先说清楚什么是缓存
- (2) 再说有了hibernate的Session就是一级缓存，即有了一级缓存，为什么还要有二级缓存
- (3) 最后再说如何配置Hibernate的二级缓存。

1，缓存就是把以前从数据库中查询出来和使用过的对象保存在内存中（一个数据结构中），这个数据结构通常是或类似HashMap，当以后要使用某个对象时，先查询缓存中是否有这个对象，如果有则使用缓存中的对象，如果没有则去查询数据库，并将查询出来的对象保存在缓存中，以便下次使用。

2，Hibernate的Session就是一种缓存，我们通常将之称为Hibernate的一级缓存，当想使用session从数据库中查询出一个对象时，Session也是先从自己内部查看是否存在这个对象，存在则直接返回，不存在才去访问数据库，并将查询的结果保存在自己内部。

由于Session代表一次会话过程，一个Session与一个数据库连接相关联，所以Session最好不要长时间保持打开，通常仅用于一个事务当中，在事务结束时就应关闭。并且Session是线程不安全的，被多个线程共享时容易出现问题。通常只有那种全局意义上的缓存才是真正的缓存应用，才有较大的缓存价值，因此，Hibernate的Session这一级缓存的缓存作用并不明显，应用价值不大。Hibernate的二级缓存就是要为Hibernate配置一种全局缓存，让多个线程和多个事务都可以共享这个缓存。我们希望的是一个人使用过，其他人也可以使用，session没有这种效果。

3，二级缓存是独立于Hibernate的软件部件，属于第三方的产品，多个厂商和组织都提供有缓存产品，例如，EHCache和OSCache等等。在Hibernate中使用二级缓存，首先就要在hibernate.cfg.xml配置文件中配置使用哪个厂家的缓存产品，接着需要配置该缓存产品自己的配置文件，最后要配置Hibernate中的哪些实体对象要纳入到二级缓存的管理中。明白了二级缓存原理和有了这个思路后，很容易配置起Hibernate的二级缓存。

扩展知识：一个SessionFactory可以关联一个二级缓存，也即一个二级缓存只能负责缓存一个数据库中的数据，当使用Hibernate的二级缓存后，注意不要有其他的应用或SessionFactory来更改当前数据库中的数据，这样缓存的数据就会与数据库中的实际数据不一致。

78，JDO是什么？

JDO是Java对象持久化的新的规范，为java data object的简称，也是一个用于存取某种数据仓库中的对象的标准化API。JDO提供了透明的对象存储，因此对开发人员来说，存储数据对象完全不需要额外的代码（如 JDBC API 的使用）。这些繁琐的例行工作已经转移到 JDO 产品提供商身上，使开发人员解脱出来，从而集中时间和精力在业务逻辑上。另外，JDO 很灵活，因为它可以在任何数据底层上运行。

比较：JDBC只是面向关系数据库（RDBMS）JDO更通用，提供到任何数据底层的存储功能，比如关系数据库、文件、XML以及对象数据库（ODBMS）等等，使得应用可移植性更强。

79，Hibernate的一对多和多对一双向关联的区别？

一对多关联映射和多对一关联映射实现的基本原理都是一样的，既是在多的一端加入一个外键指向另一端外键，而主要的区别就是维护端不同。

它们的区别在于维护的关系不同：

一对多关联映射是指在加载一的一端数据的同时加载多的一端的数据
多对一关联映射是指在加载多的一端数据的同时加载一的一端的数据。

80，Hibernate是如何延迟加载？

1. Hibernate2 延迟加载实现：a) 实体对象 b) 集合（Collection）

2. Hibernate3 提供了属性的延迟加载功能 当 Hibernate 在查询数据的时候，数据并没有存在与内存中，当程序真正对数据的操作时，对象才存在与内存中，就实现了延迟加载，他节省了服务器的内存开销，从而提高了服务器的性能。

81，使用Spring框架的好处是什么？

轻量： Spring 是轻量的，基本的版本大约2MB。

控制反转： Spring通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。

面向切面的编程(AOP)： Spring支持面向切面的编程，并且把应用业务逻辑和系统服务分开。

容器： Spring 包含并管理应用中对象的生命周期和配置。

MVC框架： Spring的WEB框架是个精心设计的框架，是Web框架的一个很好的替代品。

事务管理： Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。

异常处理：Spring 提供方便的API把具体技术相关的异常（比如由JDBC， Hibernate or JDO抛出的）转化为一致的unchecked 异常。

82. ApplicationContext通常的实现是什么？

FileSystemXmlApplicationContext：此容器从一个XML文件中加载beans的定义， XML Bean 配置文件的全路径名必须提供给它的构造函数。

ClassPathXmlApplicationContext：此容器也从一个XML文件中加载beans的定义，这里，你需要正确设置classpath因为这个容器将在classpath里找bean配置。

WebXmlApplicationContext：此容器加载一个XML文件，此文件定义了一个WEB应用的所有bean。

83，什么是Spring的依赖注入**？有哪些方法进行依赖注入**

依赖注入，是IOC的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC容器）负责把他们组装起来。

构造器依赖注入：构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。

Setter方法注入：Setter方法注入是容器通过调用无参构造器或无参static工厂方法实例化bean之后，调用该bean的setter方法，即实现了基于setter的依赖注入。

84，什么是Spring beans？

Spring beans 是那些形成Spring应用的主干的java对象。它们被Spring IOC容器初始化，装配，和管理。这些beans通过容器中配置的元数据创建。比如，以XML文件中的形式定义。

Spring 框架定义的beans都是单件beans。在bean tag中有个属性“singleton”，如果它被赋为TRUE，bean 就是单件，否则就是一个 prototype bean。默认是TRUE，所以所有在Spring框架中的beans 缺省都是单件。

85，解释Spring支持的几种bean的作用域。

Spring框架支持以下五种bean的作用域：

singleton：bean在每个Spring ioc 容器中只有一个实例。

prototype：一个bean的定义可以有多个实例。

request：每次http请求都会创建一个bean，该作用域仅在基于web的 Spring ApplicationContext情形下有效。

session：在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

global-session：在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

缺省的Spring bean 的作用域是Singleton.

86，解释Spring框架中bean的生命周期。

- 1, Spring容器从XML文件中读取bean的定义，并实例化bean。
- 2, Spring根据bean的定义填充所有的属性。
- 3, 如果bean实现了BeanNameAware 接口，Spring传递bean 的ID 到 setBeanName方法。
- 4, 如果Bean 实现了 BeanFactoryAware 接口，Spring传递beanfactory 给setBeanFactory 方法。
- 5, 如果有任何与bean相关联的BeanPostProcessors，Spring会在postProcesserBeforeInitialization()方法内调用它们。
- 6, 如果bean实现IntializingBean了，调用它的afterPropertySet方法，如果bean声明了初始化方法，调用此初始化方法。
- 7, 如果有BeanPostProcessors 和bean 关联，这些bean的postProcessAfterInitialization() 方法将被调用。
- 8, 如果bean实现了 DisposableBean，它将调用destroy()方法。

87，在Spring中如何注入一个java集合？

Spring提供以下几种集合的配置元素：

- 类型用于注入一列值，允许有相同的值。
- 类型用于注入一组值，不允许有相同的值。
- 类型用于注入一组键值对，键和值都可以为任意类型。
- 类型用于注入一组键值对，键和值都只能为String类型。

88，解释不同方式的自动装配。

有五种自动装配的方式，用来指导Spring容器用自动装配方式进行依赖注入。

- no:** 默认的方式是不进行自动装配，通过显式设置ref 属性来进行装配。
- byName:** 通过参数名自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byname，之后容器试图匹配、装配和该bean的属性具有相同名字的bean。
- byType:** 通过参数类型自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byType，之后容器试图匹配、装配和该bean的属性具有相同类型的bean。如果有多个bean符合条件，则抛出错误。
- constructor:** 这个方式类似于byType，但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
- autodetect:** 首先尝试使用constructor来自动装配，如果无法工作，则使用byType方式。

89，Spring框架的事务管理有哪些优点？

它为不同的事务API如JTA, JDBC, Hibernate, JPA和DO, 提供一个不变的编程模式。

它为编程式事务管理提供了一套简单的API而不是一些复杂的事务API如

它支持声明式事务管理。

它和Spring各种数据访问抽象层很好得集成。

90. 什么是基于Java的Spring注解配置? 给一些注解的例子.

基于Java的配置, 允许你在少量的Java注解的帮助下, 进行你的大部分Spring配置而非通过XML文件。

以@Configuration注解为例, 它用来标记类可以当做一个bean的定义, 被Spring IOC容器使用。另一个例子是@Bean注解, 它表示此方法将要返回一个对象, 作为一个bean注册进Spring应用上下文。

91, 什么是ORM?

对象关系映射 (Object-Relational Mapping, 简称ORM) 是一种为了解决程序的面向对象模型与数据库的关系模型互不匹配问题的技术;

简单的说, ORM是通过使用描述对象和数据库之间映射的元数据 (在Java中可以用XML或者是注解), 将程序中的对象自动持久化到关系数据库中或者将关系数据库表中的行转换成Java对象, 其本质上就是将数据从一种形式转换到另外一种形式。

92, Hibernate中SessionFactory是线程安全的吗? Session是线程安全的吗 (两个线程能够共享同一个Session吗) ?

SessionFactory对应Hibernate的一个数据存储的概念, 它是线程安全的, 可以被多个线程并发访问。SessionFactory一般只会在启动的时候构建。对于应用程序, 最好将SessionFactory通过单例模式进行封装以便于访问。

Session是一个轻量级非线程安全的对象 (线程间不能共享session), 它表示与数据库进行交互的一个工作单元。Session是由SessionFactory创建的, 在任务完成之后它会被关闭。Session是持久层服务对外提供的主要接口。

Session会延迟获取数据库连接 (也就是在需要的时候才会获取)。为了避免创建太多的session, 可以使用ThreadLocal将session和当前线程绑定在一起, 这样可以让同一个线程获得的总是同一个session。Hibernate 3中SessionFactory的getCurrentSession()方法就可以做到。

93, Session的save()、update()、merge()、lock()、 saveOrUpdate()和persist()方法分别是做什么的? 有什么区别?

Hibernate的对象有三种状态: 瞬时态 (transient)、持久态 (persistent) 和游离态 (detached)。

瞬时态的实例可以通过调用save()、persist()或者saveOrUpdate()方法变成持久态;

游离态的实例可以通过调用 update()、saveOrUpdate()、lock()或者replicate()变成持久态。save()和persist()将会引发SQL的INSERT语句, 而update()或merge()会引发UPDATE语句。

save()和update()的区别在于一个是将瞬时态对象变成持久态，一个是将游离态对象变为持久态。merge()方法可以完成save()和update()方法的功能，它的意图是将新的状态合并到已有的持久化对象上或创建新的持久化对象。

对于persist()方法，按照官方文档的说明：

- 1、persist()方法把一个瞬时态的实例持久化，但是并不保证标识符被立刻填入到持久化实例中，标识符的填入可能被推遲到flush的时间；
- 2、persist()方法保证当它在一个事务外部被调用的时候并不触发一个INSERT语句，当需要封装一个长会话流程的时候，persist()方法是很有必要的；
- 3、save()方法不保证第2条，它要返回标识符，所以它会立即执行INSERT语句，不管是在事务内部还是外部。至于lock()方法和update()方法的区别，update()方法是把一个已经更改过的脱管状态的对象变成持久状态；lock()方法是把一个没有更改过的脱管状态的对象变成持久状态。

94，阐述Session加载实体对象的过程。

- 1、Session在调用数据库查询功能之前，首先会在一级缓存中通过实体类型和主键进行查找，如果一级缓存查找命中且数据状态合法，则直接返回；
- 2、如果一级缓存没有命中，接下来Session会在当前NonExists记录（相当于一个查询黑名单，如果出现重复的无效查询可以迅速做出判断，从而提升性能）中进行查找，如果NonExists中存在同样的查询条件，则返回null；
- 3、如果一级缓存查询失败查询二级缓存，如果二级缓存命中直接返回；
- 4、如果之前的查询都未命中，则发出SQL语句，如果查询未发现对应记录则将此次查询添加到Session的NonExists中加以记录，并返回null；
- 5、根据映射配置和SQL语句得到ResultSet，并创建对应的实体对象；
- 6、将对象纳入Session（一级缓存）的管理；
- 7、如果有对应的拦截器，则执行拦截器的onLoad方法；
- 8、如果开启并设置了要使用二级缓存，则将数据对象纳入二级缓存；
- 9、返回数据对象。

95，MyBatis中使用#和\$书写占位符有什么区别？

#将传入的数据都当成一个字符串，会对传入的数据自动加上引号；

\$将传入的数据直接显示生成在SQL中。

注意：使用\$占位符可能会导致SQL注入攻击，能用#的地方就不要使用\$，写order by子句的时候应该用\$而不是#。

96，解释一下MyBatis中命名空间（namespace）的作用。

在大型项目中，可能存在大量的SQL语句，这时候为每个SQL语句起一个唯一的标识（ID）就变得不容易了。为了解决这个问题，在MyBatis中，可以为每个映射文件起一个唯一的命名空间，这样定义在这个映射文件中的每个SQL语句就成了定义在这个命名空间中的一个ID。只要我们能够保证每个命名空间中这个ID是唯一的，即使在不同映射文件中的语句ID相同，也不会再产生冲突了。

97、MyBatis中的动态SQL是什么意思？

对于一些复杂的查询，我们可能会指定多个查询条件，但是这些条件可能存在也可能不存在，如果不使用持久层框架我们可能需要自己拼装SQL语句，不过MyBatis提供了动态SQL的功能来解决这个问题。MyBatis中用于实现动态SQL的元素主要有：

- if - choose / when / otherwise - trim - where - set - foreach

用法举例：

```
<select id="foo" parameterType="Blog" resultType="Blog">
    select * from t_blog where 1 = 1
    <if test="title != null">
        and title = #{title}
    </if>
    <if test="content != null">
        and content = #{content}
    </if>
    <if test="owner != null">
        and owner = #{owner}
    </if>
</select>
```

98， JDBC编程有哪些不足之处， MyBatis是如何解决这些问题的？

1、JDBC：数据库链接创建、释放频繁造成系统资源浪费从而影响系统性能，如果使用数据库链接池可解决此问题。

MyBatis：在SqlMapConfig.xml中配置数据链接池，使用连接池管理数据库链接。

2、JDBC：Sql语句写在代码中造成代码不易维护，实际应用sql变化的可能较大，sql变动需要改变java代码。

MyBatis：将Sql语句配置在XXXXmapper.xml文件中与java代码分离。

3、JDBC：向sql语句传参数麻烦，因为sql语句的where条件不一定，可能多也可能少，占位符需要和参数一一对应。

MyBatis：Mybatis自动将java对象映射至sql语句。

4、JDBC：对结果集解析麻烦，sql变化导致解析代码变化，且解析前需要遍历，如果能将数据库记录封装成pojo对象解析比较方便。

MyBatis：Mybatis自动将sql执行结果映射至java对象。

99， MyBatis与Hibernate有哪些不同？

1、Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。

2、Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件则需要自定义多套sql映射文件，工作量大。

Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的缺点是学习门槛高，要

精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

(这里也可以结合自己的理解说，别说到收不住)

100，简单的说一下MyBatis的一级缓存和二级缓存？

Mybatis首先去缓存中查询结果集，如果没有则查询数据库，如果有则从缓存取出返回结果集就不走数据库。Mybatis内部存储缓存使用一个HashMap，key为hashCode+sqlId+Sql语句。value为从查询出来映射生成的java对象

Mybatis的二级缓存即查询缓存，它的作用域是一个mapper的namespace，即在同一个namespace中查询sql可以从缓存中获取数据。二级缓存是可以跨SqlSession的。

基本表结构：

```
student(sno,sname,sage,ssex)学生表  
course(cno,cname,tno) 课程表  
sc(sno,cno,score) 成绩表  
teacher(tno,tname) 教师表
```

101，查询课程1的成绩比课程2的成绩高的所有学生的学号

```
select a.sno from  
(select sno,score from sc where cno=1) a,  
(select sno,score from sc where cno=2) b  
where a.score>b.score and a.sno=b.sno
```

102，查询平均成绩大于60分的同学的学号和平均成绩*

```
select a.sno as "学号", avg(a.score) as "平均成绩"  
from  
(select sno,score from sc) a  
group by sno having avg(a.score)>60
```

103，查询所有同学的学号、姓名、选课数、总成绩

```
select a.sno as 学号, b.sname as 姓名,  
count(a.cno) as 选课数, sum(a.score) as 总成绩  
from sc a, student b  
where a.sno = b.sno  
group by a.sno, b.sname
```

或者：

```
select student.sno as 学号, student.sname as 姓名,  
count(sc.cno) as 选课数, sum(score) as 总成绩  
from student left Outer join sc on student.sno = sc.sno  
group by student.sno, sname
```

104, 查询姓“张”的老师的个数

```
select count(distinct(tname)) from teacher where tname like '张%'
```

或者：

```
select tname as "姓名", count(distinct(tname)) as "人数"  
from teacher  
where tname like '张%'  
group by tname
```

105, 查询没学过“张三”老师课的同学的学号、姓名

```
select student.sno, student.sname from student  
where sno not in (select distinct(sc.sno) from sc, course, teacher  
where sc.cno=course.cno and teacher.tno=course.tno and teacher.tname='张三')
```

106, 查询同时学过课程1和课程2的同学的学号、姓名

```
select sno, sname from student  
where sno in (select sno from sc where sc.cno = 1)  
and sno in (select sno from sc where sc.cno = 2)
```

或者：

```
select c.sno, c.sname from  
(select sno from sc where sc.cno = 1) a,  
(select sno from sc where sc.cno = 2) b,  
student c  
where a.sno = b.sno and a.sno = c.sno
```

或者：

```
select student.sno, student.sname from student, sc where student.sno=sc.sno and sc.cno=1  
and exists( select * from sc as sc_2 where sc_2.sno=sc.sno and sc_2.cno=2)
```

107, 查询学过“李四”老师所教所有课程的所有同学的学号、姓名

```
select a.sno, a.sname from student a, sc b  
where a.sno = b.sno and b.cno in  
(select c.cno from course c, teacher d where c.tno = d.tno and d.tname = '李四')
```

或者：

```
select a.sno, a.sname from student a, sc b,  
(select c.cno from course c, teacher d where c.tno = d.tno and d.tname = '李四') e  
where a.sno = b.sno and b.cno = e.cno
```

108, 查询课程编号1的成绩比课程编号2的成绩高的所有同学的学号、姓名

```
select a.sno, a.sname from student a,
(select sno, score from sc where cno = 1) b,
(select sno, score from sc where cno = 2) c
where b.score > c.score and b.sno = c.sno and a.sno = b.sno
```

109, 查询所有课程成绩小于60分的同学的学号、姓名

```
select sno,sname from student
where sno not in (select distinct sno from sc where score > 60)
```

110, 查询至少有一门课程与学号为1的同学所学课程相同的同学的学号和姓名

```
select distinct a.sno, a.sname
from student a, sc b
where a.sno <> 1 and a.sno=b.sno and
b.cno in (select cno from sc where sno = 1)
```

或者：

```
select s.sno,s.sname
from student s,
(select sc.sno
from sc
where sc.cno in (select sc1.cno from sc sc1 where sc1.sno=1)and sc.sno<>1
group by sc.sno)r1
where r1.sno=s.sno
```

111、把“sc”表中“王五”所教课的成绩都更改为此课程的平均成绩

```
update sc set score = (select avg(sc_2.score) from sc sc_2 where sc_2.cno=sc.cno)
from course,teacher where course.cno=sc.cno and course.tno=teacher.tno and teacher.tname='王五'
```

112、查询和编号为2的同学学习的课程完全相同的其他同学学号和姓名

这一题分两步查：

1,

```
select sno
from sc
where sno <> 2
group by sno
having sum(cno) = (select sum(cno) from sc where sno = 2)
```

```
2,
select b.sno, b.sname
from sc a, student b
where b.sno <> 2 and a.sno = b.sno
group by b.sno, b.sname
having sum(cno) = (select sum(cno) from sc where sno = 2)
```

113、删除学习“王五”老师课的sc表记录

```
delete sc from course, teacher
where course.cno = sc.cno and course.tno = teacher.tno and tname = '王五'
```

**114、向sc表中插入一些记录，这些记录要求符合以下条件：

114、将没有课程3成绩同学的该成绩补齐，其成绩取所有学生的课程2的平均成绩

```
insert sc select sno, 3, (select avg(score) from sc where cno = 2)
from student
where sno not in (select sno from sc where cno = 3)
```

115、按平均分从高到低显示所有学生的如下统计报表：

```
-- 学号,企业管理,马克思,UML,数据库,物理,课程数,平均分**
select sno as 学号
,max(case when cno = 1 then score end) AS 企业管理
,max(case when cno = 2 then score end) AS 马克思
,max(case when cno = 3 then score end) AS UML
,max(case when cno = 4 then score end) AS 数据库
,max(case when cno = 5 then score end) AS 物理
,count(cno) AS 课程数
,avg(score) AS 平均分
FROM sc
GROUP by sno
ORDER by avg(score) DESC
```

116、查询各科成绩最高分和最低分：

以如下形式显示：课程号，最高分，最低分

```
select cno as 课程号, max(score) as 最高分, min(score) 最低分
from sc group by cno
```

```
select course.cno as '课程号'
,MAX(score) as '最高分'
,MIN(score) as '最低分'
from sc,course
where sc.cno=course.cno
group by course.cno
```

117、按各科平均成绩从低到高和及格率的百分数从高到低顺序

```

SELECT t.cno AS 课程号,
max(course cname) AS 课程名,
isnull(AVG(score), 0) AS 平均成绩
100 * SUM(CASE WHEN isnull(score, 0) >= 60 THEN 1 ELSE 0 END) / count(1) AS 及格率
FROM sc t, course
where t.cno = course.cno
GROUP BY t.cno
ORDER BY 及格率 desc

```

118、查询如下课程平均成绩和及格率的百分数(用"1行"显示):

企业管理 (001) , 马克思 (002) , UML (003) , 数据库 (004)

```

select
avg(case when cno = 1 then score end) as 平均分1,
avg(case when cno = 2 then score end) as 平均分2,
avg(case when cno = 3 then score end) as 平均分3,
avg(case when cno = 4 then score end) as 平均分4,
100 * sum(case when cno = 1 and score > 60 then 1 else 0 end) / sum(case when cno = 1 then 1 else 0 end) as 及格率1,
100 * sum(case when cno = 2 and score > 60 then 1 else 0 end) / sum(case when cno = 2 then 1 else 0 end) as 及格率2,
100 * sum(case when cno = 3 and score > 60 then 1 else 0 end) / sum(case when cno = 3 then 1 else 0 end) as 及格率3,
100 * sum(case when cno = 4 and score > 60 then 1 else 0 end) / sum(case when cno = 4 then 1 else 0 end) as 及格率4
from sc

```

119、查询不同老师所教不同课程平均分,从高到低显示

```

select max(c.tname) as 教师, max(b cname) 课程, avg(a.score) 平均分
from sc a, course b, teacher c
where a.cno = b.cno and b.tno = c.tno
group by a.cno
order by 平均分 desc

```

或者:

```

select r.tname as '教师', r.rname as '课程', AVG(score) as '平均分'
from sc,
(select t.tname, c.cno as rcso, c cname as rname
from teacher t, course c
where t.tno=c.tno)r
where sc.cno=r.rcso
group by sc.cno, r.tname, r.rname
order by AVG(score) desc

```

**120、查询如下课程成绩均在第3名到第6名之间的学生的成绩:

120、[学生ID],[学生姓名],企业管理,马克思,UML,数据库,平均成绩

```

*select top 6 max(a.sno) 学号, max(b.sname) 姓名,
max(case when cno = 1 then score end) as 企业管理,
max(case when cno = 2 then score end) as 马克思,
max(case when cno = 3 then score end) as UML,
max(case when cno = 4 then score end) as 数据库,

```

```
avg(score) as 平均分  
from sc a, student b  
where a.sno not in  
  
(select top 2 sno from sc where cno = 1 order by score desc)  
and a.sno not in (select top 2 sno from sc where cno = 2 order by scoredesc)  
and a.sno not in (select top 2 sno from sc where cno = 3 order by scoredesc)  
and a.sno not in (select top 2 sno from sc where cno = 4 order by scoredesc)  
and a.sno = b.sno  
group by a.sno
```

121，什么是线程？

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要100毫秒，那么用十个线程完成改任务只需10毫秒。

122，线程和进程有什么区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。每个线程都拥有单独的栈内存用来存储本地数据。

123，如何在Java中实现线程？

两种方式：java.lang.Thread 类的实例就是一个线程但是它需要调用java.lang.Runnable接口来执行，由于线程类本身就是调用的Runnable接口所以你可以继承java.lang.Thread 类或者直接调用Runnable接口来重写run()方法实现线程。

124，Java 关键字volatile 与 synchronized 作用与区别？

1, volatile

它所修饰的变量不保留拷贝，直接访问主内存中的。

在Java内存模型中，有main memory，每个线程也有自己的memory(例如寄存器)。为了性能，一个线程会在自己的memory中保持要访问的变量的副本。这样就会出现同一个变量在某个瞬间，在一个线程的memory中的值可能与另外一个线程memory中的值，或者main memory中的值不一致的情况。一个变量声明为volatile，就意味着这个变量是随时会被其他线程修改的，因此不能将它cache在线程memory中。

2, synchronized

当它用来修饰一个方法或者一个代码块的时候，能够保证在同一时刻最多只有一个线程执行该段代码。

一、当两个并发线程访问同一个对象object中的这个synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而，当一个线程访问object的一个synchronized(this)同步代码块时，另一个线程仍然可以访问该object中的非synchronized(this)同步代码块。

三、尤其关键的是，当一个线程访问object的一个synchronized(this)同步代码块时，其他线程对object中所有其它synchronized(this)同步代码块的访问将被阻塞。

四、当一个线程访问object的一个synchronized(this)同步代码块时，它就获得了这个object的对象锁。结果，其它线程对该object对象所有同步代码部分的访问都被暂时阻塞。

五、以上规则对其它对象锁同样适用。

125，有哪些不同的线程生命周期？

当我们在Java程序中新建一个线程时，它的状态是*New*。当我们调用线程的start()方法时，状态被改变为*Runnable*。线程调度器会为*Runnable*线程池中的线程分配CPU时间并且将它们的状态改变为*Running*。其他的线程状态还有*Waiting*, *Blocked* 和*Dead*。

126，你对线程优先级的理解是什么？

每一个线程都是有优先级的，一般来说，高优先级的线程在运行时会具有优先权，但这依赖于线程调度的实现，这个实现是和操作系统相关的(OS dependent)。我们可以定义线程的优先级，但是这并不能保证高优先级的线程会在低优先级的线程前执行。线程优先级是一个int变量(从1-10)，1代表最低优先级，10代表最高优先级。

127，什么是死锁(Deadlock)? 如何分析和避免死锁?

死锁是指两个以上的线程永远阻塞的情况，这种情况产生至少需要两个以上的线程和两个以上的资源。

分析死锁，我们需要查看Java应用程序的线程转储。我们需要找出那些状态为BLOCKED的线程和他们等待的资源。每个资源都有一个唯一的id，用这个id我们可以找出哪些线程已经拥有了它的对象锁。

避免嵌套锁，只在需要的地方使用锁和避免无限期等待是避免死锁的通常办法。

128，什么是线程安全？Vector是一个线程安全类吗？

如果你的代码所在的进程中多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector是用同步方法来实现线程安全的，而和它相似的ArrayList不是线程安全的。

129，Java中如何停止一个线程？

Java提供了很丰富的API但没有为停止线程提供API。JDK 1.0本来有一些像stop(), suspend() 和 resume() 的控制方法但是由于潜在的死锁威胁因此在后续的JDK版本中他们被弃用了，之后Java API的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当run() 或者 call() 方法执行完的时候线程会自动结束，如果要手动结束一个线程，你可以用volatile 布尔变量来退出run()方法的循环或者是取消任务来中断线程。

130, 什么是ThreadLocal?

ThreadLocal用于创建线程的本地变量，我们知道一个对象的所有线程会共享它的全局变量，所以这些变量不是线程安全的，我们可以使用同步技术。但是当我们不想使用同步的时候，我们可以选择ThreadLocal变量。

每个线程都会拥有他们自己的Thread变量，它们可以使用get()\set()方法去获取他们的默认值或者在
线程内部改变他们的值。ThreadLocal实例通常是希望它们同线程状态关联起来是private static属性。

131, Sleep()、suspend()和wait()之间有什么区别?

Thread.sleep()使当前线程在指定的时间处于“非运行”(Not Runnable)状态。线程一直持有对象的监视器。比如一个线程当前在一个同步块或同步方法中，其它线程不能进入该块或方法中。如果另一线程调用了interrupt()方法，它将唤醒那个“睡眠的”线程。

注意：sleep()是一个静态方法。这意味着只对当前线程有效，一个常见的错误是调用t.sleep()，(这里的t是一个不同于当前线程的线程)。即便是执行t.sleep()，也是当前线程进入睡眠，而不是t线程。
t.suspend()是过时的方法，使用suspend()导致线程进入停滞状态，该线程会一直持有对象的监视器，
suspend()容易引起死锁问题。

object.wait()使当前线程出于“不可运行”状态，和sleep()不同的是wait是object的方法而不是thread。调用object.wait()时，线程先要获取这个对象的对象锁，当前线程必须在锁对象保持同步，把当前线程添加到等待队列中，随后另一线程可以同步同一个对象锁来调用object.notify()，这样将唤醒原来等待中的线程，然后释放该锁。基本上wait()/notify()与sleep()/interrupt()类似，只是前者需要获取对象锁。

132, 什么是线程饿死，什么是活锁?

当所有线程阻塞，或者由于需要的资源无效而不能处理，不存在非阻塞线程使资源可用。Java API中线程活锁可能发生在以下情形：

- 1, 当所有线程在程序中执行Object.wait(0)，参数为0的wait方法。程序将发生活锁直到在相应的对象上有线程调用Object.notify()或者Object.notifyAll()。
- 2, 当所有线程卡在无限循环中。

133, 什么是Java Timer类? 如何创建一个有特定时间间隔的任务?

java.util.Timer是一个工具类，可以用于安排一个线程在未来的某个特定时间执行。Timer类可以用安排一次性任务或者周期任务。

java.util.TimerTask是一个实现了Runnable接口的抽象类，我们需要去继承这个类来创建我们自己的定时任务并使用Timer去安排它的执行。

134, Java中的同步集合与并发集合有什么区别?

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。

在Java1.5之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。

Java5介绍了并发集合像ConcurrentHashMap，不仅提供线程安全还用锁分离和 内部分区等现代技术提高了可扩展性。

135，同步方法和同步块，哪个是更好的选择？

同步块是更好的选择，因为它不会锁住整个对象（当然你也可以让它锁住整个对象）。同步方法会锁住整个对象，哪怕这个类中有多个不相关联的同步块，这通常会导致他们停止执行并需要等待获得这个对象上的锁。

136，什么是线程池？为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。

为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。

从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

137，Java中invokeAndWait 和 invokeLater有什么区别？

这两个方法是Swing API 提供给Java开发者用来从当前线程而不是事件派发线程更新GUI组件用的。InvokeAndWait()同步更新GUI组件，比如一个进度条，一旦进度更新了，进度条也要做出相应改变。如果进度被多个线程跟踪，那么就调用invokeAndWait()方法请求事件派发线程对组件进行相应更新。而invokeLater()方法是异步调用更新组件的。

138，多线程中的忙循环是什么？

忙循环就是程序员用循环让一个线程等待，不像传统方法wait(), sleep() 或 yield() 它们都放弃了CPU控制，而忙循环不会放弃CPU，它就是在运行一个空循环。这么做的目的是为了保留CPU缓存。

在多核系统中，一个等待线程醒来的時候可能会在另一个内核运行，这样会重建缓存。为了避免重建缓存和减少等待重建的时间就可以使用它了。

139. Java中的泛型是什么？使用泛型的好处是什么？

泛型是Java SE 1.5的新特性，泛型的本质是参数化类型，也就是说所操作的数据类型被指定为一个参数。

好处：

- 1、类型安全，提供编译期间的类型检测
- 2、前后兼容
- 3、泛化代码，代码可以更多的重复利用
- 4、性能较高，用GJ(泛型JAVA)编写的代码可以为java编译器和虚拟机带来更多的类型信息，这些信息对java程序做进一步优化提供条件。

140, Java的泛型是如何工作的？什么是类型擦除？如何工作？

- 1、类型检查：在生成字节码之前提供类型检查
- 2、类型擦除：所有类型参数都用他们的限定类型替换，包括类、变量和方法（类型擦除）
- 3、如果类型擦除和多态性发生了冲突时，则在子类中生成桥方法解决
- 4、如果调用泛型方法的返回类型被擦除，则在调用该方法时插入强制类型转换

类型擦除：

所有类型参数都用他们的限定类型替换：

比如T->Object ? extends BaseClass->BaseClass

如何工作：

泛型是通过类型擦除来实现的，编译器在编译时擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。例如 List在运行时仅用一个List来表示。这样做的目的，是确保能和Java 5之前的版本开发二进制类库进行兼容。你无法在运行时访问到类型参数，因为编译器已经把泛型类型转换成了原始类型。根据你对这个泛型问题的回答情况，你会得到一些后续提问，比如为什么泛型是由类型擦除来实现的或者给你展示一些会导致编译器出错的错误泛型代码。

141, 你可以把List传递给一个接受List参数的方法吗？

对任何一个不太熟悉泛型的人来说，这个Java泛型题目看起来令人疑惑，因为乍看起来String是一种Object，所以 List应当可以用在需要List的地方，但是事实并非如此。真这样做的话会导致编译错误。如果你再深一步考虑，你会发现Java这样做是有意义的，因为List可以存储任何类型的对象包括String, Integer等等，而List却只能用来存储String s。

```
List objectList;  
List stringList;  
objectList = stringList; //compilation error incompatible types
```

142, 如何阻止Java中的类型未检查的警告？

如果你把泛型和原始类型混合起来使用，例如下列代码，java 5的javac编译器会产生类型未检查的警告，例如

```
List rawList = newArrayList()
```

注意: Hello.java使用了未检查或称为不安全的操作;

这种警告可以使用@SuppressWarnings("unchecked")注解来屏蔽。

143, Java中List和原始类型List之间的区别？

原始类型和带参数类型之间的主要区别是，在编译时编译器不会对原始类型进行类型安全检查，却会对带参数的类型进行检查，通过使用Object作为类型，可以告知编译器该方法可以接受任何类型的对象，比如String或Integer。

这道题的考察点在于对泛型中原始类型的正确理解。它们之间的第二点区别是，你可以把任何带参数的类型传递给原始类型List，但却不能把List传递给接受List的方法，因为会产生编译错误。

144，编写一段泛型程序来实现LRU缓存？

对于喜欢Java编程的人来说这相当于是一次练习。给你个提示，LinkedHashMap可以用来实现固定大小的LRU缓存，当LRU缓存已经满了的时候，它会把最老的键值对移出缓存。

LinkedHashMap提供了一个称为removeEldestEntry()的方法，该方法会被put()和putAll()调用来删除最老的键值对。当然，如果你已经编写了一个可运行的JUnit测试，你也可以随意编写你自己的实现代码。

145，Array中可以用泛型吗？

这可能是Java泛型面试题中最简单的一个了，当然前提是你要知道Array事实上并不支持泛型，这也是为什么Joshua Bloch在Effective Java一书中建议使用List来代替Array，因为List可以提供编译期的类型安全保证，而Array却不能。

146，如何编写一个泛型方法，让它能接受泛型参数并返回泛型类型？

编写泛型方法并不困难，你需要用泛型类型来替代原始类型，比如使用T, E or K,V等被广泛认可的类型占位符。最简单的情况下，一个泛型方法可能会像这样：

```
public V put(K key, V value) {  
    return cache.put(key,value);  
}
```

147，C++模板和java泛型之间有何不同？

java泛型实现根植于“类型消除”这一概念。当源代码被转换为Java虚拟机字节码时，这种技术会消除参数化类型。有了Java泛型，我们可以做的事情也并没有真正改变多少；他只是让代码变得漂亮些。鉴于此，Java泛型有时也被称为“语法糖”。

这和C++模板截然不同。在C++中，模板本质上就是一套宏指令集，只是换了个名头，编译器会针对每种类型创建一份模板代码的副本。

由于架构设计上的差异，Java泛型和C++模板有很多不同点：

C++模板可以使用int等基本数据类型。Java则不行，必须转而使用Integer。

在Java中，可以将模板的参数类型限定为某种特定类型。

在C++中，类型参数可以实例化，但Java不支持。

在Java中，类型参数不能用于静态方法(?)和变量，因为它们会被不同类型参数指定的实例共享。在C++，这些类时不同的，因此类型参数可以用于静态方法和静态变量。

在Java中，不管类型参数是什么，所有的实例变量都是同一类型。类型参数会在运行时被抹去。在C++中，类型参数不同，实例变量也不同。

148, AJAX有哪些有点和缺点?

优点:

- 1、最大的一点是页面无刷新，用户的体验非常好。
- 2、使用异步方式与服务器通信，具有更加迅速的响应能力。
- 3、可以把以前一些服务器负担的工作转嫁到客户端，利用客户端闲置的能力来处理，减轻服务器和带宽的负担，节约空间和宽带租用成本。并且减轻服务器的负担，ajax的原则是“按需取数据”，可以最大程度的减少冗余请求，和响应对服务器造成的负担。
- 4、基于标准化的并被广泛支持的技术，不需要下载插件或者小程序。

缺点:

- 1、ajax不支持浏览器back按钮。
- 2、安全问题 AJAX暴露了与服务器交互的细节。
- 3、对搜索引擎的支持比较弱。
- 4、破坏了程序的异常机制。
- 5、不容易调试。

149, AJAX应用和传统Web应用有什么不同?

在传统的Javascript编程中，如果想得到服务器端数据库或文件上的信息，或者发送客户端信息到服务器，需要建立一个HTML form然后GET或者POST数据到服务器端。用户需要点击“Submit”按钮来发送或者接受数据信息，然后等待服务器响应请求，页面重新加载。

因为服务器每次都会返回一个新的页面，所以传统的web应用有可能很慢而且用户交互不友好。

使用AJAX技术，就可以使Javascript通过XMLHttpRequest对象直接与服务器进行交互。

通过HTTP Request，一个web页面可以发送一个请求到web服务器并且接受web服务器返回的信息(不用重新加载页面)，展示给用户的还是同一个页面，用户感觉页面刷新，也看不到到javascript后台进行的发送请求和接受响应，体验非常好。

150, Ajax的实现流程是怎样的?

- (1)创建XMLHttpRequest对象,也就是创建一个异步调用对象.
- (2)创建一个新的HTTP请求,并指定该HTTP请求的方法、URL及验证信息.
- (3)设置响应HTTP请求状态变化的函数.
- (4)发送HTTP请求.
- (5)获取异步调用返回的数据.
- (6)使用JavaScript和DOM实现局部刷新.

具体一点:

1, 创建XNLHttpRequest对象

(不考虑ie) XMLHttpRequest request = new XMLHttpRequest () ;

2, 创建新的Http请求

```
XMLHttpRequest.open (method,url,flag,name,password) ;
```

3, 设置响应Http请求变化的函数

```
XMLHttpRequest.onreadystatechange=getData;

function getData(){
    if(XMLHttpRequest.readyState==4){
        获取数据
    }
}
```

4, 发送http请求

```
XMLHttpRequest.send(data);
```

5, 获取异步调用返回的对象

```
, function(data){
    //异步提交后，交互成功，返回的data便是异步调用返回的对象，该对象是一个string类型的
}
```

6, 使用js、DOM实现局部刷新

```
myDiv.innerHTML="这是刷新后的数据"
```

151, 简单说一下数据库的三范式?

第一范式：数据库表的每一个字段都是不可分割的

第二范式：数据库表中的非主属性只依赖于主键

第三范式：不存在非主属性对关键字的传递函数依赖关系

152, Java集合框架是什么？说出一些集合框架的优点？

每种编程语言中都有集合，最初的Java版本包含几种集合类：Vector、Stack、HashTable和Array。

随着集合的广泛使用，Java1.2提出了囊括所有集合接口、实现和算法的集合框架。在保证线程安全的情况下使用泛型和并发集合类，Java已经经历了很久。它还包括在Java并发包中，阻塞接口以及它们的实现。

集合框架的部分优点如下：

- (1) 使用核心集合类降低开发成本，而非实现我们自己的集合类。
- (2) 随着使用经过严格测试的集合框架类，代码质量会得到提高。
- (3) 通过使用JDK附带的集合类，可以降低代码维护成本。
- (4) 复用性和可操作性。

153, Java集合框架的基础接口有哪些?

Collection为集合层级的根接口。一个集合代表一组对象，这些对象即为它的元素。Java平台不提供这个接口任何直接的实现。

Set是一个不能包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。

List是一个有序集合，可以包含重复元素。你可以通过它的索引来访问任何元素。List更像长度动态变换的数组。

Map是一个将key映射到value的对象。一个Map不能包含重复的key：每个key最多只能映射一个value。

一些其它的接口有Queue、Dequeue、SortedSet、SortedMap和ListIterator。

154, 集合框架中的泛型有什么优点?

Java1.5引入了泛型，所有的集合接口和实现都大量地使用它。泛型允许我们为集合提供一个可以容纳的对象类型。

因此，如果你添加其它类型的任何元素，它会在编译时报错。这避免了在运行时出现ClassCastException，因为你将会在编译时得到报错信息。

泛型也使得代码整洁，我们不需要使用显式转换和instanceOf操作符。它也给运行时带来好处，因为不会产生类型检查的字节码指令。

155, Enumeration和Iterator接口的区别?

Enumeration的速度是Iterator的两倍，也使用更少的内存。Enumeration是非常基础的，也满足了基础的需要。

但是，与Enumeration相比，Iterator更加安全，因为当一个集合正在被遍历的时候，它会阻止其它线程去修改集合。

迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者从集合中移除元素，而Enumeration不能做到。为了使它的功能更加清晰，迭代器方法名已经经过改善。

156, Iterator和ListIterator之间有什么区别?

- 1, 我们可以使用Iterator来遍历Set和List集合，而ListIterator只能遍历List。
- 2, Iterator只可以向前遍历，而ListIterator可以双向遍历。
- 3, ListIterator从Iterator接口继承，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

157, 我们如何对一组对象进行排序?

如果我们需要对一个对象数组进行排序，我们可以使用Arrays.sort()方法。如果我们需要排序一个对象列表，我们可以使用Collection.sort()方法。

两个类都有用于自然排序（使用Comparable）或基于标准的排序（使用Comparator）的重载方法sort()。

Collections内部使用数组排序方法，所有它们两者都有相同的性能，只是Collections需要花时间将列表转换为数组。

158，与Java集合框架相关的有哪些最好的实践？

1，根据需要选择正确的集合类型。比如，如果指定了大小，我们会选用Array而非ArrayList。如果我们想根据插入顺序遍历一个Map，我们需要使用TreeMap。如果我们不想重复，我们应该使用Set。

2，一些集合类允许指定初始容量，所以如果我们能够估计到存储元素的数量，我们可以使用它，就避免了重新哈希或大小调整。

3，基于接口编程，而非基于实现编程，它允许我们后来轻易地改变实现。

4，总是使用类型安全的泛型，避免在运行时出现ClassCastException。

5，使用JDK提供的不可变类作为Map的key，可以避免自己实现hashCode()和equals()。

6，尽可能使用Collections工具类，或者获取只读、同步或空的集合，而非编写自己的实现。它将会提供代码重用性，它有着更好的稳定性和可维护性。

159，什么是事务？

事务是恢复和并发控制的基本单位

事务的四个基本特征

原子性，一致性，隔离性，持久性

原子性和一致性差不多，意思是要么全部成功，要么就失败

一致性是说，从一个一致性状态到另一个一致性状态

隔离性是说一个事务执行的过程中不能被另一个事务干扰

持久性也就是事务一旦提交，他对数据库中数据的改变就应该是永久的，不能变的（这里只是面试简单的说一下理解，详细理解问度娘）

160，说说你开发中遇到过什么难题啊？怎么解决的？

卒.....

161，Java内存模型是什么？

Java内存模型规定和指引Java程序在不同的内存架构、CPU和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。这个关系定义了一些规则让程序员在并发编程时思路更清晰。比如，先行发生关系确保了：

线程内的代码能够按先后顺序执行，这被称为程序次序规则。

对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。

前一个对volatile的写操作在后一个volatile的读操作之前，也叫volatile变量规则。

一个线程内的任何操作必需在这个线程的start()调用之后，也叫作线程启动规则。

一个线程的所有操作都会在线程终止之前，线程终止规则。

一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。

可传递性

更多介绍可以移步并发编程网：

(深入理解Java内存模型系列文章：<http://ifeve.com/java-memory-model-0/>)

162，Java中interrupted 和isInterrupted方法的区别？

interrupted() 和 isInterrupted() 的主要区别是前者会将中断状态清除而后者不会。Java多线程的中断机制是用内部标识来实现的，调用Thread.interrupt() 来中断一个线程就会设置中断标识为true。当中断线程调用静态方法Thread.interrupted() 来检查中断状态时，中断状态会被清零。

非静态方法isInterrupted() 用来查询其它线程的中断状态且不会改变中断状态标识。简单的说就是任何抛出InterruptedException异常的方法都会将中断状态清零。无论如何，一个线程的中断状态都有可能被其它线程调用中断来改变。

163，Java中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在Java1.5之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5介绍了并发集合像ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。

不管是同步集合还是并发集合他们都支持线程安全，他们之间主要的区别体现在性能和可扩展性，还有他们如何实现的线程安全上。

同步HashMap, Hashtable, HashSet, Vector, ArrayList 相比他们并发的实现

(ConcurrentHashMap, CopyOnWriteArrayList, CopyOnWriteHashSet) 会慢得多。造成如此慢的主要原因是锁，同步集合会把整个Map或List锁起来，而并发集合不会。并发集合实现线程安全是通过使用先进的和成熟的技术像锁剥离。

比如ConcurrentHashMap 会把整个Map划分成几个片段，只对相关的几个片段上锁，同时允许多线程访问其他未上锁的片段。

同样的，CopyOnWriteArrayList 允许多个线程以非同步的方式读，当有线程写的时候它会将整个List复制一个副本给它。

如果在读多写少这种对并发集合有利的条件下使用并发集合，这会比使用同步集合更具有可伸缩性。

164，什么是线程池？为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）

线程池的作用，就是在调用线程的时候初始化一定数量的线程，有线程过来的时候，先检测初始化的线程还有空的没有，没有就再看当前运行中的线程数是不是已经达到了最大数，如果没有，就新分配一个线程去处理。

就像餐馆中吃饭一样，从里面叫一个服务员出来；但如果已经达到了最大数，就相当于服务员已经用尽了，那没得办法，另外的线程就只有等了，直到有新的“服务员”为止。

线程池的优点就是可以管理线程，有一个高度中枢，这样程序才不会乱，保证系统不会因为大量的并发而因为资源不足挂掉。

165，Java中活锁和死锁有什么区别？

活锁：一个线程通常会有会响应其他线程的活动。如果其他线程也会响应另一个线程的活动，那么就有可能发生活锁。同死锁一样，发生活锁的线程无法继续执行。然而线程并没有阻塞——他们在忙于响应对方无法恢复工作。这就相当于两个在走廊相遇的人：甲向他自己的左边靠想让乙过去，而乙向他的右边靠想让甲过去。可见他们阻塞了对方。甲向他的右边靠，而乙向他的左边靠，他们还是阻塞了对方。

死锁：两个或更多线程阻塞着等待其它处于死锁状态的线程所持有的锁。死锁通常发生在多个线程同时但以不同的顺序请求同一组锁的时候，死锁会让你的程序挂起无法完成任务。

166，如何避免死锁？

死锁的发生必须满足以下四个条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源，在未使用完之前，不能强行剥夺。

循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

三种用于避免死锁的技术：

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）

死锁检测

（死锁原因及如何避免更深理解移步：<http://blog.csdn.net/ls5718/article/details/51896159>）

167，notify()和notifyAll()有什么区别？

1, notify()和notifyAll()都是Object对象用于通知处在等待该对象的线程的方法。

2, void notify(): 唤醒一个正在等待该对象的线程。

3, void notifyAll(): 唤醒所有正在等待该对象的线程。

两者最大的区别在于：

notifyAll使所有原来在该对象上等待被notify的线程统统退出wait的状态，变成等待该对象上的锁，一旦该对象被解锁，他们就会去竞争。

notify他只是选择一个wait状态线程进行通知，并使它获得该对象上的锁，但不惊动其他同样在等待被该对象notify的线程们，当第一个线程运行完毕以后释放对象上的锁，此时如果该对象没有再次使用notify语句，即便该对象已经空闲，其他wait状态等待的线程由于没有得到该对象的通知，继续处在wait状态，直到这个对象发出一个notify或notifyAll，它们等待的是被notify或notifyAll，而不是锁。

168, 什么是可重入锁 (ReentrantLock) ?

Java.util.concurrent.lock 中的 Lock 框架是锁定的一个抽象，它允许把锁定的实现作为Java 类，而不是作为语言的特性来实现。这就为Lock 的多种实现留下了空间，各种实现可能有不同的调度算法、性能特性或者锁定语义。 ReentrantLock 类实现了Lock，它拥有与synchronized 相同的并发性和内存语义，但是添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。此外，它还提供了在激烈争用情况下更佳的性能。（换句话说，当许多线程都想访问共享资源时，JVM可以花更少的时候来调度线程，把更多时间用在执行线程上。）

Reentrant 锁意味着什么呢？简单来说，它有一个与锁相关的获取计数器，如果拥有锁的某个线程再次得到锁，那么获取计数器就加1，然后锁需要被释放两次才能获得真正释放。这模仿了synchronized 的语义；如果线程进入由线程已经拥有的监控器保护的synchronized 块，就允许线程继续进行，当线程退出第二个（或者后续）synchronized块的时候，不释放锁，只有线程退出它进入的监控器保护的第一个synchronized 块时，才释放锁。

169, 读写锁可以用于什么应用场景？

读写锁可以用于“多读少写”的场景，读写锁支持多个读操作并发执行，写操作只能由一个线程来操作。ReadWriteLock对向数据结构相对不频繁地写入，但是有多个任务要经常读取这个数据结构的这类情况进行了优化。ReadWriteLock使得你可以同时有多个读取者，只要它们都不试图写入即可。如果写锁已经被其他任务持有，那么任何读取者都不能访问，直至这个写锁被释放为止。

ReadWriteLock 对程序性能的提高主要受制于如下几个因素：

1, 数据被读取的频率与被修改的频率相比较的结果。

2, 读取和写入的时间

3, 有多少线程竞争

4, 是否在多处理机器上运行

30 个 Java 集合面试问题及答案

1. Java集合框架是什么？说出一些集合框架的优点？

每种编程语言中都有集合，最初的Java版本包含几种集合类：Vector、Stack、HashTable和Array。

随着集合的广泛使用，Java1.2提出了囊括所有集合接口、实现和算法的集合框架。在保证线程安全的情况下使用泛型和并发集合类，Java已经经历了很久。它还包括在Java并发包中，阻塞接口以及它们的实现。

集合框架的部分优点如下：

- (1) 使用核心集合类降低开发成本，而非实现我们自己的集合类。
- (2) 随着使用经过严格测试的集合框架类，代码质量会得到提高。
- (3) 通过使用JDK附带的集合类，可以降低代码维护成本。
- (4) 复用性和可操作性。

2.集合框架中的泛型有什么优点？

- 1. Java1.5引入了泛型，所有的集合接口和实现都大量地使用它。
- 2. 泛型允许我们为集合提供一个可以容纳的对象类型，因此，如果你添加其它类型的任何元素，它会在编译时报错。
- 3. 这避免了在运行时出现ClassCastException，因为你将会在编译时得到报错信息。
- 4. 泛型也使得代码整洁，我们不需要使用显式转换和instanceOf操作符。
- 5. 它也给运行时带来好处，因为不会产生类型检查的字节码指令。

3. Java集合框架的基础接口有哪些？

Collection为集合层级的根接口。一个集合代表一组对象，这些对象即为它的元素。Java平台不提供这个接口任何直接的实现。

Set是一个不能包含重复元素的集合。这个接口对数学集合抽象进行建模，被用来代表集合，就如一副牌。

List是一个有序集合，可以包含重复元素。你可以通过它的索引来访问任何元素。List更像长度动态变换的数组。

Map是一个将key映射到value的对象。一个Map不能包含重复的key：每个key最多只能映射一个value。

一些其它的接口有Queue、Dequeue、SortedSet、SortedMap和ListIterator。

4.为何Collection不从Cloneable和Serializable接口继承?

Collection接口指定一组对象，对象即为它的元素。如何维护这些元素由Collection的具体实现决定。例如，一些如List的Collection实现允许重复的元素，而其它的如Set就不允许。

很多Collection实现有一个公有的clone方法。然而，把它放到集合的所有实现中也是没有意义的。这是因为Collection是一个抽象表现。重要的是实现。

当与具体实现打交道的时候，克隆或序列化的语义和含义才发挥作用。所以，具体实现应该决定如何对它进行克隆或序列化，或它是否可以被克隆或序列化。

在所有的实现中授权克隆和序列化，最终导致更少的灵活性和更多的限制。特定的实现应该决定它是否可以被克隆和序列化。

5.为何Map接口不继承Collection接口?

尽管Map接口和它的实现也是集合框架的一部分，但Map不是集合，集合也不是Map。因此，Map继承Collection毫无意义，反之亦然。

如果Map继承Collection接口，那么元素去哪儿？Map包含key-value对，它提供抽取key或value列表集合的方法，但是它不适合“一组对象”规范。

6.Iterator是什么?

Iterator接口提供遍历任何Collection的接口。我们可以从一个Collection中使用迭代器方法来获取迭代器实例。迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者在迭代过程中移除元素。

7 Enumeration和Iterator接口的区别?

Enumeration的速度是Iterator的两倍，也使用更少的内存。Enumeration是非常基础的，也满足了基础的需要。但是，与Enumeration相比，Iterator更加安全，因为当一个集合正在被遍历的时候，它会阻止其它线程去修改集合。

迭代器取代了Java集合框架中的Enumeration。迭代器允许调用者从集合中移除元素，而Enumeration不能做到。为了使它的功能更加清晰，迭代器方法名已经经过改善。

8.为何没有像Iterator.add()这样的方法，向集合中添加元素?

语义不明，已知的是，Iterator的协议不能确保迭代的次序。然而要注意，ListIterator没有提供一个add操作，它要确保迭代的顺序。

9.为何迭代器没有一个方法可以直接获取下一个元素，而不需要移动游标？

它可以在当前Iterator的顶层实现，但是它用得很少，如果将它加到接口中，每个继承都要去实现它，这没有意义。

10.Iterator和ListIterator之间有什么区别？

- (1) 我们可以使用Iterator来遍历Set和List集合，而ListIterator只能遍历List。
- (2) Iterator只可以向前遍历，而ListIterator可以双向遍历。
- (3) ListIterator从Iterator接口继承，然后添加了一些额外的功能，比如添加一个元素、替换一个元素、获取前面或后面元素的索引位置。

11.通过迭代器fail-fast属性，你明白了什么？

每次我们尝试获取下一个元素的时候，Iterator fail-fast属性检查当前集合结构里的任何改动。如果发现任何改动，它抛出ConcurrentModificationException。Collection中所有Iterator的实现都是按fail-fast来设计的（ConcurrentHashMap和CopyOnWriteArrayList这类并发集合类除外）。

12.fail-fast与fail-safe有什么区别？

Iterator的fail-fast属性与当前的集合共同起作用，因此它不会受到集合中任何改动的影响。Java.util包中的所有集合类都被设计为fail-fast的，

而java.util.concurrent中的集合类都为fail-safe的。

Fall—fast迭代器抛出ConcurrentModificationException，

fall—safe迭代器从不抛出ConcurrentModificationException。

13.在迭代一个集合的时候，如何避免？

ConcurrentModificationException？

在遍历一个集合的时候我们可以使用并发集合类来避免ConcurrentModificationException，比如使用CopyOnWriteArrayList，而不是ArrayList。

14.为何Iterator接口没有具体的实现？

Iterator接口定义了遍历集合的方法，但它的实现则是集合实现类的责任。每个能够返回用于遍历的Iterator的集合类都有它自己的Iterator实现内部类。

这就允许集合类去选择迭代器是fail-fast还是fail-safe的。比如，ArrayList迭代器是fail-fast的，而CopyOnWriteArrayList迭代器是fail-safe的。

15.UnsupportedOperationException是什么？

UnsupportedOperationException是用于表明操作不支持的异常。在JDK类中已被大量运用，在集合框架java.util.Collections.UnmodifiableCollection将会在所有add和remove操作中抛出这个异常。

16.hashCode()和equals()方法有何重要性？

HashMap使用Key对象的hashCode()和equals()方法去决定key-value对的索引。

当我们试着从HashMap中获取值的时候，这些方法也会被用到。如果这些方法没有被正确地实现，在这种情况下，两个不同Key也许会产生相同的hashCode()和equals()输出，HashMap将会认为它们是相同的，然后覆盖它们，而非把它们存储到不同的地方。

同样的，所有不允许存储重复数据的集合类都使用hashCode()和equals()去查找重复，所以正确实现它们非常重要。equals()和hashCode()的实现应该遵循以下规则：

- 1.如果o1.equals(o2)，那么o1.hashCode() == o2.hashCode()总是为true的。
- 2.如果o1.hashCode() == o2.hashCode()，并不意味着o1.equals(o2)会为true。

17.Map接口提供了哪些不同的集合视图？

Map接口提供三个集合视图：

1) Set keyset(): 返回map中包含的所有key的一个Set视图。集合是受map支持的，map的变化会在集合中反映出来，反之亦然。当一个迭代器正在遍历一个集合时，若map被修改了（除迭代器自身的移除操作以外），迭代器的结果会变为未定义。集合支持通过Iterator的Remove、Set.remove、removeAll、retainAll和clear操作进行元素移除，从map中移除对应的映射。

它不支持add和addAll操作。

2) Collection values(): 返回一个map中包含的所有value的一个Collection视图。这个collection受map支持的，map的变化会在collection中反映出来，反之亦然。当一个迭代器正在遍历一个collection时，若map被修改了（除迭代器自身的移除操作以外），迭代器的结果会变为未定义。集合支持通过Iterator的Remove、Set.remove、removeAll、retainAll和clear操作进行元素移除，从map中移除对应的映射。它不支持add和addAll操作。

3) Set<Map.Entry<K,V>> entrySet(): 返回一个map中包含的所有映射的一个集合视图。这个集合受map支持的，map的变化会在collection中反映出来，反之亦然。当一个迭代器正在遍历一个集合时，若map被修改了（除迭代器自身的移除操作，以及对迭代器返回的entry进行setValue外），迭代器的结果会变为未定义。集合支持通过Iterator的Remove、Set.remove、removeAll、retainAll和clear操作进行元素移除，从map中移除对应的映射。它不支持add和addAll操作。

18.HashMap和HashTable有何不同？

- (1) HashMap允许key和value为null，而HashTable不允许。
- (2) HashTable是同步的，而HashMap不是。所以HashMap适合单线程环境，HashTable适合多线程环境。
- (3) 在Java1.4中引入了LinkedHashMap，HashMap的一个子类，假如你想要遍历顺序，你很容易从HashMap转向LinkedHashMap，但是HashTable不是这样的，它的顺序是不可预知的。
- (4) HashMap提供对key的Set进行遍历，因此它是fail-fast的，但HashTable提供对key的Enumeration进行遍历，它不支持fail-fast。
- (5) HashTable被认为是个遗留的类，如果你寻求在迭代的时候修改Map，你应该使用ConcurrentHashMap。

19.如何决定选用HashMap还是TreeMap？

对于在Map中插入、删除和定位元素这类操作，HashMap是最好的选择。然而，假如你需要对一个有序的key集合进行遍历，TreeMap是更好的选择。基于你的collection的大小，也许向HashMap中添加元素会更快，将map换为TreeMap进行有序key的遍历。

20.ArrayList和Vector有何异同点？

ArrayList和Vector很多时候都很类似。

- (1) 两者都是基于索引的，内部由一个数组支持。
- (2) 两者维护插入的顺序，我们可以根据插入顺序来获取元素。
- (3) ArrayList和Vector的迭代器实现都是fail-fast的。
- (4) ArrayList和Vector两者允许null值，也可以使用索引值对元素进行随机访问。

以下是ArrayList和Vector的不同点。

- (1) Vector是同步的，而ArrayList不是。然而，如果你寻求在迭代的时候对列表进行改变，你应该使用CopyOnWriteArrayList。
- (2) ArrayList比Vector快，它因为有同步，不会过载。
- (3) ArrayList更加通用，因为我们可以使用Collections工具类轻易地获取同步列表和只读列表。

21.Array和ArrayList有何区别？什么时候更适合用Array？

Array可以容纳基本类型和对象，而ArrayList只能容纳对象。

Array是指定大小的，而ArrayList大小是固定的。

Array没有提供ArrayList那么多功能，比如addAll、removeAll和iterator等。尽管ArrayList明显是更好的选择，但也有些时候Array比较好用。

- (1) 如果列表的大小已经指定，大部分情况下是存储和遍历它们。
- (2) 对于遍历基本数据类型，尽管Collections使用自动装箱来减轻编码任务，在指定大小的基本类型的列表上工作也会变得很慢。
- (3) 如果你要使用多维数组，使用`[][]`比`List<List<>>`更容易。

22.ArrayList和LinkedList有何区别？

ArrayList和LinkedList两者都实现了List接口，但是它们之间有些不同。

- 1) ArrayList是由Array所支持的基于一个索引的数据结构，所以它提供对元素的随机访问，复杂度为O(1)，但LinkedList存储一系列的节点数据，每个节点都与前一个和下一个节点相连接。所以，尽管有使用索引获取元素的方法，内部实现是从起始点开始遍历，遍历到索引的节点然后返回元素，时间复杂度为O(n)，比ArrayList要慢。
- 2) 与ArrayList相比，在LinkedList中插入、添加和删除一个元素会更快，因为在一个元素被插入到中间的时候，不会涉及改变数组的大小，或更新索引。
- 3) LinkedList比ArrayList消耗更多的内存，因为LinkedList中的每个节点存储了前后节点的引用。

23.哪些集合类提供对元素的随机访问？

ArrayList、HashMap、TreeMap和HashTable类提供对元素的随机访问。

24.哪些集合类是线程安全的?

Vector、HashTable、Properties和Stack是同步类，所以它们是线程安全的，可以在多线程环境下使用。Java1.5并发API包括一些集合类，允许迭代时修改，因为它们都工作在集合的克隆上，所以它们在多线程环境中是安全的。

25.并发集合类是什么?

Java1.5并发包 (java.util.concurrent) 包含线程安全集合类，允许在迭代时修改集合。迭代器被设计为 fail-fast 的，会抛出ConcurrentModificationException。一部分类为：CopyOnWriteArrayList、ConcurrentHashMap、CopyOnWriteArraySet。

26.队列和栈是什么，列出它们的区别?

栈和队列两者都被用来预存储数据。java.util.Queue是一个接口，它的实现类在Java并发包中。队列允许先进先出 (FIFO) 检索元素，但并非总是这样。Deque接口允许从两端检索元素。栈与队列很相似，但它允许对元素进行后进先出 (LIFO) 进行检索。Stack是一个扩展自Vector的类，而Queue是一个接口。

27.Collections类是什么?

Java.util.Collections是一个工具类仅包含静态方法，它们操作或返回集合。

它包含操作集合的多态算法，返回一个由指定集合支持的新集合和其它一些内容。这个类包含集合框架算法的方法，比如折半搜索、排序、混编和逆序等。

28.Comparable和Comparator接口有何区别?

Comparable和Comparator接口被用来对对象集合或者数组进行排序。Comparable接口被用来提供对象的自然排序，我们可以使用它来提供基于单个逻辑的排序。

Comparator接口被用来提供不同的排序算法，我们可以选择需要使用的Comparator来对给定的对象集合进行排序。

29.我们如何对一组对象进行排序?

如果我们需要对一个对象数组进行排序，我们可以使用Arrays.sort()方法。如果我们需要排序一个对象列表，我们可以使用Collection.sort()方法。

两个类都有用于自然排序（使用Comparable）或基于标准的排序（使用Comparator）的重载方法sort()。Collections内部使用数组排序方法，所有它们两者都有相同的性能，只是Collections需要花时间将列表转换为数组。

30.当一个集合被作为参数传递给一个函数时，如何才可以确保函数不能修改它？

在作为参数传递之前，我们可以使用Collections.unmodifiableCollection(Collection c)方法创建一个只读集合，

这将确保改变集合的任何操作都会抛出UnsupportedOperationException。

Java 多线程面试题及答案

1、多线程有什么用？

一个可能在很多人看来很扯淡的一个问题：我会用多线程就好了，还管它有什么用？在我看来，这个回答更扯淡。所谓“知其然知其所以然”，“会用”只是“知其然”，“为什么用”才是“知其所以然”，只有达到“知其然知其所以然”的程度才可以说是把一个知识点运用自如。OK，下面说说我对这个问题的看法：

1) 发挥多核CPU的优势

随着工业的进步，现在的笔记本、台式机乃至商用的应用服务器至少也都是双核的，4核、8核甚至16核的也都不少见，如果是单线程的程序，那么在双核CPU上就浪费了50%，在4核CPU上就浪费了75%。

单核CPU上所谓的“多线程”那是假的多线程，同一时间处理器只会处理一段逻辑，只不过线程之间切换得比较快，看着像多个线程“同时”运行罢了。多核CPU上的多线程才是真正的多线程，它能让你的多段逻辑同时工作，多线程，可以真正发挥出多核CPU的优势来，达到充分利用CPU的目的。

2) 防止阻塞

从程序运行效率的角度来看，单核CPU不但不会发挥出多线程的优势，反而会因为在单核CPU上运行多线程导致线程上下文的切换，而降低程序整体的效率。但是单核CPU我们还是要应用多线程，就是为了防止阻塞。试想，如果单核CPU使用单线程，那么只要这个线程阻塞了，比方说远程读取某个数据吧，对端迟迟未返回又没有设置超时时间，那么你的整个程序在数据返回回来之前就停止运行了。多线程可以防止这个问题，多条线程同时运行，哪怕一条线程的代码执行读取数据阻塞，也不会影响其它任务的执行。

3) 便于建模

这是另外一个没有这么明显的优点了。假设有一个大的任务A，单线程编程，那么就要考虑很多，建立整个程序模型比较麻烦。但是如果把这个大的任务A分解成几个小任务，任务B、任务C、任务D，分别建立程序模型，并通过多线程分别运行这几个任务，那就简单很多了。

2、创建线程的方式

比较常见的一个问题了，一般就是两种：

1) 继承Thread类

2) 实现Runnable接口

至于哪个好，不用说肯定是后者好，因为实现接口的方式比继承类的方式更灵活，也能减少程序之间的耦合度，**面向接口编程**也是设计模式6大原则的核心。

3、start()方法和run()方法的区别

只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

4、Runnable接口和Callable接口的区别

有点深的问题了，也看出一个Java程序员学习知识的广度。

Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行run()方法中的代码而已；Callable接口中的call()方法是有返回值的，是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。

这其实是很有一个特性，因为**多线程相比单线程更难、更复杂的一个重要原因就是因为多线程充满着未知性**，某条线程是否执行了？某条线程执行了多久？某条线程执行的时候我们期望的数据是否已经赋值完毕？无法得知，我们能做的只是等待这条多线程的任务执行完毕而已。而Callable+Future/FutureTask却可以获取多线程运行的结果，可以在等待时间太长没获取到需要的数据的情况下取消该线程的任务，真的是非常有用。

5、CyclicBarrier和CountDownLatch的区别

两个看上去有点像的类，都在java.util.concurrent下，都可以用来表示代码运行到某个点上，二者的区别在于：

- 1) CyclicBarrier的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch则不是，某线程运行到某个点上之后，只是给某个数值-1而已，该线程继续运行。
- 2) CyclicBarrier只能唤起一个任务，CountDownLatch可以唤起多个任务。
- 3) CyclicBarrier可重用，CountDownLatch不可重用，计数值为0该CountDownLatch就不可再用了。

6、volatile关键字的作用

一个非常重要的问题，是每个学习、应用多线程的Java程序员都必须掌握的。理解volatile关键字的作用的前提是要理解Java内存模型，这里就不讲Java内存模型了，可以参见第31点，volatile关键字的作用主要有两个：

- 1) 多线程主要围绕可见性和原子性两个特性而展开，使用volatile关键字修饰的变量，保证了其在多线程之间的可见性，即每次读取到volatile变量，一定是最新的数据。

2) 代码底层执行不像我们看到的高级语言---Java程序这么简单，它的执行是**Java代码-->字节码-->根据字节码执行对应的C/C++代码-->C/C++代码被编译成汇编语言-->和硬件电路交互**，现实中，为了获取更好的性能JVM可能会对指令进行重排序，多线程下可能会出现一些意想不到的问题。使用volatile则会对禁止语义重排序，当然这也一定程度上降低了代码执行效率。

从实践角度而言，volatile的一个重要作用就是和CAS结合，保证了原子性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger，更多详情请点击[这里](#)进行学习。

7、什么是线程安全

又是一个理论的问题，各式各样的答案有很多，我给出一个个人认为解释地最好的：**如果你的代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么你的代码就是线程安全的。**

这个问题有值得一提的地方，就是线程安全也是有几个级别的：

1) 不可变

像String、Integer、Long这些，都是final类型的类，任何一个线程都改变不了它们的值，要改变除非新创建一个，因此这些不可变对象不需要任何同步手段就可以直接在多线程环境下使用

2) 绝对线程安全

不管运行时环境如何，调用者都不需要额外的同步措施。要做到这一点通常需要付出许多额外的代价，Java中标注自己是线程安全的类，实际上绝大多数都不是线程安全的，不过绝对线程安全的类，Java中也有，比方说CopyOnWriteArrayList、CopyOnWriteArraySet

3) 相对线程安全

相对线程安全也就是我们通常意义上所说的线程安全，像Vector这种，add、remove方法都是原子操作，不会被打断，但也仅限于此，如果有个线程在遍历某个Vector、有个线程同时在add这个Vector，99%的情况下都会出现ConcurrentModificationException，也就是**fail-fast机制**。

4) 线程非安全

这个就没什么好说的了，ArrayList、LinkedList、HashMap等都是线程非安全的类，点击[这里](#)了解为什么不安全。

8、Java中如何获取到线程dump文件

死循环、死锁、阻塞、页面打开慢等问题，打线程dump是最好的解决问题的途径。所谓线程dump也就是线程堆栈，获取到线程堆栈有两步：

- 1) 获取到线程的pid，可以通过使用jps命令，在Linux环境下还可以使用ps -ef | grep java
- 2) 打印线程堆栈，可以通过使用jstack pid命令，在Linux环境下还可以使用kill -3 pid

另外提一点，Thread类提供了一个getStackTrace()方法也可以用于获取线程堆栈。这是一个实例方法，因此此方法是和具体线程实例绑定的，每次获取获取到的是具体某个线程当前运行的堆栈。

9、一个线程如果出现了运行时异常会怎么样

如果这个异常没有被捕获的话，这个线程就停止执行了。另外重要的一点是：**如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放**

10、如何在两个线程之间共享数据

通过在线程之间共享对象就可以了，然后通过wait/notify/notifyAll、await/signal/signalAll进行唤起和等待，比方说阻塞队列BlockingQueue就是为线程之间共享数据而设计的

11、sleep方法和wait方法有什么区别

这个问题常问，sleep方法和wait方法都可以用来放弃CPU一定的时间，不同点在于如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

12、生产者消费者模型的作用是什么

这个问题很理论，但是很重要：

- 1) 通过平衡生产者的生产能力和消费者的消费能力来提升整个系统的运行效率，这是生产者消费者模型最重要的作用
- 2) 解耦，这是生产者消费者模型附带的作用，解耦意味着生产者和消费者之间的联系少，联系越少越可以独自发展而不需要收到相互的制约

13、ThreadLocal有什么用

简单说ThreadLocal就是一种以空间换时间的做法，在每个Thread里面维护了一个以开地址法实现的ThreadLocal.ThreadLocalMap，把数据进行隔离，数据不共享，自然就没有线程安全方面的问题了

14、为什么wait()方法和notify()/notifyAll()方法要在同步块中被调用

这是JDK强制的，wait()方法和notify()/notifyAll()方法在调用前都必须先获得对象的锁

15、wait()方法和notify()/notifyAll()方法在放弃对象监视器时有什么区别

wait()方法和notify()/notifyAll()方法在放弃对象监视器的时候的区别在于：wait()方法立即释放对象监视器，notify()/notifyAll()方法则会等待线程剩余代码执行完毕才会放弃对象监视器。

16、为什么要使用线程池

避免频繁地创建和销毁线程，达到线程对象的重用。另外，使用线程池还可以根据项目灵活地控制并发的数目。

17、怎么检测一个线程是否持有对象监视器

我也是在网上看到一道多线程面试题才知道有方法可以判断某个线程是否持有对象监视器：Thread类提供了一个holdsLock(Object obj)方法，当且仅当对象obj的监视器被某条线程持有的时候才会返回true，注意这是一个static方法，这意味着“某条线程”指的是当前线程。

18、synchronized和ReentrantLock的区别

synchronized是和if、else、for、while一样的关键字，ReentrantLock是类，这是二者的本质区别。既然ReentrantLock是类，那么它就提供了比synchronized更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock比synchronized的扩展性体现在几点上：

- (1) ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁
- (2) ReentrantLock可以获取各种锁的信息
- (3) ReentrantLock可以灵活地实现多路通知

另外，二者的锁机制其实也是不一样的。ReentrantLock底层调用的是Unsafe的park方法加锁，synchronized操作的应该是对象头中mark word，这点我不能确定。

19、ConcurrentHashMap的并发度是什么

ConcurrentHashMap的并发度就是segment的大小，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最大优势，任何情况下，Hashtable能同时有两条线程获取Hashtable中的数据吗？

20、ReadWriteLock是什么

首先明确一下，不是说ReentrantLock不好，只是ReentrantLock某些时候有局限。如果使用ReentrantLock，可能本身是为了防止线程A在写数据、线程B在读数据造成的数据不一致，但这样，如果线程C在读数据、线程D也在读数据，读数据是不会改变数据的，没有必要加锁，但是还是加锁了，降低了程序的性能。

因为这个，才诞生了读写锁ReadWriteLock。ReadWriteLock是一个读写锁接口，ReentrantReadWriteLock是ReadWriteLock接口的一个具体实现，实现了读写的分离，**读锁是共享的，写锁是独占的**，读和读之间不会互斥，读和写、写和读、写和写之间才会互斥，提升了读写的性能。

21、FutureTask是什么

这个其实前面有提到过，FutureTask表示一个异步运算的任务。FutureTask里面可以传入一个Callable的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中。

22、Linux环境下如何查找哪个线程使用CPU最长

这是一个比较偏实践的问题，这种问题我觉得挺有意义的。可以这么做：

- (1) 获取项目的pid，jps或者ps -ef | grep java，这个前面有讲过

(2) top -H -p pid, 顺序不能改变

这样就可以打印出当前的项目，每条线程占用CPU时间的百分比。注意这里打出的是LWP，也就是操作系统原生线程的线程号，我笔记本没有部署Linux环境下的Java工程，因此没有办法截图演示，网友朋友们如果公司是使用Linux环境部署项目的话，可以尝试一下。

使用"top -H -p pid"+"jps pid"可以很容易地找到某条占用CPU高的线程的线程堆栈，从而定位占用CPU高的原因，一般是因为不当的代码操作导致了死循环。

最后提一点，"top -H -p pid"打出来的LWP是十进制的，"jps pid"打出来的本地线程号是十六进制的，转换一下，就能定位到占用CPU高的线程的当前线程堆栈了。

23、Java编程写一个会导致死锁的程序

第一次看到这个题目，觉得这是一个非常好的问题。很多人都知道死锁是怎么一回事儿：线程A和线程B相互等待对方持有的锁导致程序无限死循环下去。当然也仅限于此了，问一下怎么写一个死锁的程序就不知道了，这种情况说白了就是不懂什么是死锁，懂一个理论就完事儿了，实践中碰到死锁的问题基本上是看不出来的。

真正理解什么是死锁，这个问题其实不难，几个步骤：

- 1) 两个线程里面分别持有两个Object对象：lock1和lock2。这两个lock作为同步代码块的锁；
- 2) 线程1的run()方法中同步代码块先获取lock1的对象锁，Thread.sleep(xxx)，时间不需要太多，50毫秒差不多了，然后接着获取lock2的对象锁。这么做主要是为了防止线程1启动一下子就连续获得了lock1和lock2两个对象的对象锁
- 3) 线程2的run()方法中同步代码块先获取lock2的对象锁，接着获取lock1的对象锁，当然这时lock1的对象锁已经被线程1锁持有，线程2肯定是要等待线程1释放lock1的对象锁的

这样，线程1"睡觉"睡完，线程2已经获取了lock2的对象锁了，线程1此时尝试获取lock2的对象锁，便被阻塞，此时一个死锁就形成了。代码就不写了，占的篇幅有点多，Java多线程7：死锁这篇文章里面有，就是上面步骤的代码实现。

24、怎么唤醒一个阻塞的线程

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

25、不可变对象对多线程有什么帮助

前面有提到过的一个问题，不可变对象保证了对象的内存可见性，对不可变对象的读取不需要进行额外的同步手段，提升了代码执行效率。

26、什么是多线程的上下文切换

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程。

27、如果你提交任务时，线程池队列已满，这时会发生什么

这里区分一下：

- 1) 如果使用的是无界队列LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为LinkedBlockingQueue可以近乎认为是一个无穷大的队列，可以无限存放任务
- 2) 如果使用的是有界队列比如ArrayBlockingQueue，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，会根据maximumPoolSize的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue继续满，那么则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy

28、Java中用到的线程调度算法是什么

抢占式。一个线程用完CPU之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。

29、Thread.sleep(0)的作用是什么

这个问题和上面那个问题是相关的，我就连在一起了。由于Java采用抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级比较低的线程也能获取到CPU控制权，可以使用Thread.sleep(0)手动触发一次操作系统分配时间片的操作，这也是平衡CPU控制权的一种操作。

30、什么是自旋

很多synchronized里面的代码只是一些很简单的代码，执行时间非常快，此时等待的线程都加锁可能是一种不太值得的操作，因为线程阻塞涉及到用户态和内核态切换的问题。既然synchronized里面的代码执行得非常快，不妨让等待锁的线程不要被阻塞，而是在synchronized的边界做忙循环，这就是自旋。如果做了多次忙循环发现还没有获得锁，再阻塞，这样可能是一种更好的策略。

31、什么是Java内存模型

Java内存模型定义了一种多线程访问Java内存的规范。Java内存模型要完整讲不是这里几句话能说清楚的，我简单总结一下Java内存模型的几部分内容：

- 1) Java内存模型将内存分为了**主内存**和**工作内存**。类的状态，也就是类之间共享的变量，是存储在主内存中的，每次Java线程用到这些主内存中的变量的时候，会读一次主内存中的变量，并让这些内存存在自己的工作内存中有一份拷贝，运行自己线程代码的时候，用到这些变量，操作的都是自己工作内存中的那一份。在线程代码执行完毕之后，会将最新的值更新到主内存中去
- 2) 定义了几个原子操作，用于操作主内存和工作内存中的变量
- 3) 定义了volatile变量的使用规则
- 4) happens-before，即先行发生原则，定义了操作A必然先行发生于操作B的一些规则，比如在同一个线程内控制流前面的代码一定先行发生于控制流后面的代码、一个释放锁unlock的动作一定先行发生于后面对于同一个锁进行锁定lock的动作等等，只要符合这些规则，则不需要额外做同步措施，如果某段代码不符合所有的happens-before规则，则这段代码一定是线程非安全的

32、什么是CAS

CAS，全称为Compare and Swap，即比较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS一定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是一个不会变的值A，只要某次CAS操作失败，永远都不可能成功。更多CAS详情请点击[这里](#)学习。

33、什么是乐观锁和悲观锁

1) 乐观锁：就像它的名字一样，对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，将**比较-替换**这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，那么就应该有相应的重试逻辑。

2) 悲观锁：还是像它的名字一样，对于并发间操作产生的线程安全问题持悲观状态，悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像synchronized，不管三七二十一，直接上了锁就操作资源了。

34、什么是AQS

简单说一下AQS，AQS全称为AbstractQueuedSynchronizer，翻译过来应该是抽象队列同步器。

如果说java.util.concurrent的基础是CAS的话，那么AQS就是整个Java并发包的核心了，ReentrantLock、CountDownLatch、Semaphore等等都用到了它。AQS实际上以双向队列的形式连接所有的Entry，比方说ReentrantLock，所有等待的线程都被放在一个Entry中并连成双向队列，前面一个线程使用ReentrantLock好了，则双向队列实际上的第一个Entry开始运行。

AQS定义了对双向队列所有的操作，而只开放了tryLock和tryRelease方法给开发者使用，开发者可以根据自己的实现重写tryLock和tryRelease方法，以实现自己的并发功能。

35、单例模式的线程安全性

老生常谈的问题了，首先要说的是单例模式的线程安全意味着：**某个类的实例在多线程环境下只会被创建一次出来**。单例模式有很多种的写法，我总结一下：

- 1) 饿汉式单例模式的写法：线程安全
- 2) 懒汉式单例模式的写法：非线程安全
- 3) 双检锁单例模式的写法：线程安全

36、Semaphore有什么作用

Semaphore就是一个信号量，它的作用是**限制某段代码块的并发数**。Semaphore有一个构造函数，可以传入一个int型整数n，表示某段代码最多只有n个线程可以访问，如果超出了n，那么请等待，等到某个线程执行完毕这段代码块，下一个线程再进入。由此可以看出如果Semaphore构造函数中传入的int型整数n=1，相当于变成了一个synchronized了。

37、Hashtable的size()方法中明明只有一条语句"return count"，为什么还要做同步？

这是我之前的一个困惑，不知道大家有没有想过这个问题。某个方法中如果有两条语句，并且都在操作同一个类变量，那么在多线程环境下不加锁，势必会引发线程安全问题，这很好理解，但是size()方法明明只有一条语句，为什么还要加锁？

关于这个问题，在慢慢地工作、学习中，有了理解，主要原因有两点：

- 1) 同一时间只能有一条线程执行固定类的同步方法，但是对于类的非同步方法，可以多条线程同时访问。所以，这样就有问题了，可能线程A在执行Hashtable的put方法添加数据，线程B则可以正常调用size()方法读取Hashtable中当前元素的个数，那读取到的值可能不是最新的，可能线程A添加了完了数据，但是没有对size++，线程B就已经读取size了，那么对于线程B来说读取到的size一定是不准确的。而给size()方法加了同步之后，意味着线程B调用size()方法只有在线程A调用put方法完毕之后才可以调用，这样就保证了线程安全性
- 2) CPU执行代码，执行的不是Java代码，这点很关键，一定得记住。Java代码最终是被翻译成机器码执行的，机器码才是真正可以和硬件电路交互的代码。即使你看到Java代码只有一行，甚至你看到java代码编译之后生成的字节码也只有一行，也不意味着对于底层来说这句语句的操作只有一个。一句"return count"假设被翻译成了三句汇编语句执行，一句汇编语句和其机器码做对应，完全可能执行完第一句，线程就切换了。

38、线程类的构造方法、静态块是被哪个线程调用的

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。

如果说上面的说法让你感到困惑，那么我举个例子，假设Thread2中new了Thread1，main函数中new了Thread2，那么：

- 1) Thread2的构造方法、静态块是main线程调用的，Thread2的run()方法是Thread2自己调用的
- 2) Thread1的构造方法、静态块是Thread2调用的，Thread1的run()方法是Thread1自己调用的

39、同步方法和同步块，哪个是更好的选择

同步块，这意味着同步块之外的代码是异步执行的，这比同步整个方法更提升代码的效率。请知道一条原则：**同步的范围越小越好**。

借着这一条，我额外提一点，虽说同步的范围越小越好，但是在Java虚拟机中还是存在着一种叫做锁粗化的优化方法，这种方法就是把同步范围变大。这是有用的，比方说StringBuffer，它是一个线程安全的类，自然最常用的append()方法是一个同步方法，我们写代码的时候会反复append字符串，这意味着要进行反复的加锁->解锁，这对性能不利，因为这意味着Java虚拟机在这条线程上要反复地在内核态和用户态之间进行切换，因此Java虚拟机会将多次append方法调用的代码进行一个锁粗化的操作，将多次的append的操作扩展到append方法的头尾，变成一个大的同步块，这样就减少了加锁-->解锁的次数，有效地提升了代码执行的效率。

40、高并发、任务执行时间短的业务怎样使用线程池？并发不高、任务执行时间长的业务怎样使用线程池？并发高、业务执行时间长的业务怎样使用线程池？

这是我在并发编程网上看到的一个问题，把这个问题放在最后一个，希望每个人都能看到并且思考一下，因为这个问题非常好、非常实际、非常专业。关于这个问题，个人看法是：

- 1) 高并发、任务执行时间短的业务，线程池线程数可以设置为CPU核数+1，减少线程上下文的切换
- 2) 并发不高、任务执行时间长的业务要区分开看：
 - a) 假如是业务时间长集中在IO操作上，也就是IO密集型的任务，因为IO操作并不占用CPU，所以不要让所有的CPU闲下来，可以加大线程池中的线程数目，让CPU处理更多的业务
 - b) 假如是业务时间长集中在计算操作上，也就是计算密集型任务，这个就没办法了，和（1）一样吧，线程池中的线程数设置得少一些，减少线程上下文的切换
 - c) 并发高、业务执行时间长，解决这种类型任务的关键不在于线程池而在于整体架构的设计，看看这些业务里面某些数据是否能做缓存是第一步，增加服务器是第二步，至于线程池的设置，设置参考其他有关线程池的文章。最后，业务执行时间长的问题，也可能需要分析一下，看看能不能使用中间件对任务进行拆分和解耦。

整理 69 道 Spring 面试题和答案

1. 什么是spring？

Spring 是个java企业级应用的开源开发框架。Spring主要用来开发Java应用，但是有些扩展是针对构建J2EE平台的web应用。Spring 框架目标是简化Java企业级应用开发，并通过POJO为基础的编程模型促进良好的编程习惯。

2. 使用Spring框架的好处是什么？

- **轻量**: Spring 是轻量的，基本的版本大约2MB
- **控制反转**: Spring通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们
- **面向切面的编程(AOP)**: Spring支持面向切面的编程，并且把应用业务逻辑和系统服务分开
- **容器**: Spring 包含并管理应用中对象的生命周期和配置
- **MVC框架**: Spring的WEB框架是个精心设计的框架，是Web框架的一个很好的替代品
- **事务管理**: Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务 (JTA)
- **异常处理**: Spring 提供方便的API把具体技术相关的异常（比如由JDBC, Hibernate or JDO抛出的）转化为一致的unchecked 异常

3. Spring由哪些模块组成？

以下是Spring 框架的基本模块：

- Core module
- Bean module
- Context module
- Expression Language module

- JDBC module
- ORM module
- OXM module
- Java Messaging Service(JMS) module
- Transaction module
- Web module
- Web-Servlet module
- Web-Struts module
- Web-Portlet module

4. 核心容器（应用上下文）模块

这是基本的Spring模块，提供spring框架的基础功能，BeanFactory是任何以spring为基础的应用的核心。Spring框架建立在此模块之上，它使Spring成为一个容器。

5. BeanFactory – BeanFactory 实现举例

Bean工厂是工厂模式的一个实现，提供了控制反转功能，用来把应用的配置和依赖从正真的应用代码中分离。最常用的BeanFactory实现是XmlBeanFactory类。

6. XMLBeanFactory

最常用的就是org.springframework.beans.factory.xml.XmlBeanFactory，它根据XML文件中的定义加载beans。该容器从XML文件读取配置元数据并用它去创建一个完全配置的系统或应用。

7. 解释AOP模块

AOP模块用于发给我们的Spring应用做面向切面的开发，很多支持由AOP联盟提供，这样就确保了Spring和其他AOP框架的共通性。这个模块将元数据编程引入Spring。

8. 解释JDBC抽象和DAO模块

通过使用JDBC抽象和DAO模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用Spring的AOP模块给Spring应用中的对象提供事务管理服务。

9. 解释对象/关系映射集成模块

Spring 通过提供ORM模块，支持我们在直接JDBC之上使用一个对象/关系映射(ORM)工具，Spring 支持集成主流的ORM框架，如Hibernate,JDO和 iBATIS SQL Maps。Spring的事务管理同样支持以上所有ORM框架及JDBC。

10.解释WEB 模块

Spring的WEB模块是构建在application context 模块基础之上，提供一个适合web应用的上下文。这个模块也包括支持多种面向web的任务，如透明地处理多个文件上传请求和程序级请求参数的绑定到你的业务对象。它也有对Jakarta Struts的支持。

12.Spring配置文件

Spring配置文件是个XML 文件，这个文件包含了类信息，描述了如何配置它们，以及如何相互调用。

13.什么是Spring IOC 容器？

Spring IOC 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。

14.IOC的优点是什么？

IOC 或 依赖注入把应用的代码量降到最低。它使应用容易测试，单元测试不再需要单例和JNDI查找机制。最小的代价和最小的侵入性使松散耦合得以实现。IOC容器支持加载服务时的饿汉式初始化和懒加载。

15.ApplicationContext通常的实现是什么？

- **FileSystemXmlApplicationContext**：此容器从一个XML文件中加载beans的定义，XML Bean 配置文件的全路径名必须提供给它的构造函数。
- **ClassPathXmlApplicationContext**：此容器也从一个XML文件中加载beans的定义，这里，你需要正确设置classpath因为这个容器将在classpath里找bean配置。
- **WebXmlApplicationContext**：此容器加载一个XML文件，此文件定义了一个WEB应用的所有 bean。

16.Bean 工厂和 Application contexts 有什么区别？

Application contexts提供一种方法处理文本消息，一个通常的做法是加载文件资源（比如镜像），它们可以向注册为监听器的bean发布事件。另外，在容器或容器内的对象上执行的那些不得不由bean工厂以程序化方式处理的操作，可以在Application contexts中以声明的方式处理。Application contexts实现了MessageSource接口，该接口的实现以可插拔的方式提供获取本地化消息的方法。

17.一个Spring的应用看起来象什么？

- 一个定义了一些功能的接口
- 这实现包括属性，它的Setter，getter方法和函数等
- Spring AOP
- Spring 的XML 配置文件
- 使用以上功能的客户端程序

18.什么是Spring的依赖注入？

依赖注入，是IOC的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC容器）负责把他们组装起来。

19.有哪些不同类型的IOC（依赖注入）方式？

- **构造器依赖注入：**构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。
- **Setter方法注入：**Setter方法注入是容器通过调用无参构造器或无参static工厂方法实例化bean之后，调用该bean的setter方法，即实现了基于setter的依赖注入。

20.哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入？

你两种依赖方式都可以使用，构造器注入和Setter方法注入。最好的解决方案是用构造器参数实现强制依赖，setter方法实现可选依赖。

21.什么是Spring beans？

Spring beans 是那些形成Spring应用的主干的java对象。它们被Spring IOC容器初始化，装配，和管理。这些beans通过容器中配置的元数据创建。比如，以XML文件中的形式定义。

Spring 框架定义的beans都是单件beans。在bean tag中有个属性"singleton"，如果它被赋为TRUE，bean 就是单件，否则就是一个 prototype bean。默认是TRUE，所以所有在Spring框架中的beans 缺省都是单件。点击[这里](#)一图Spring Bean的生命周期。

22.一个 Spring Bean 定义 包含什么？

一个Spring Bean 的定义包含容器必知的所有配置元数据，包括如何创建一个bean，它的生命周期详情及它的依赖。

23.如何给Spring 容器提供配置元数据？

这里有三种重要的方法给Spring 容器提供配置元数据。

XML配置文件。

基于注解的配置。

基于java的配置。

24.你怎样定义类的作用域？

当定义一个在Spring里，我们还能给这个bean声明一个作用域。它可以通过bean 定义中的scope属性来定义。如，当Spring要在需要的时候每次生产一个新的bean实例，bean的scope属性被指定为prototype。另一方面，一个bean每次使用的时候必须返回同一个实例，这个bean的scope 属性 必须设为 singleton。

25.解释Spring支持的几种bean的作用域

Spring框架支持以下五种bean的作用域：

- **singleton** : bean在每个Spring ioc 容器中只有一个实例。
- **prototype**: 一个bean的定义可以有多个实例。
- **request**: 每次http请求都会创建一个bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。
- **session**: 在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。
- **global-session**: 在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

缺省的Spring bean 的作用域是Singleton。

26.Spring框架中的单例bean是线程安全的吗？

不，Spring框架中的单例bean不是线程安全的。

27.解释Spring框架中bean的生命周期

- Spring容器从XML文件中读取bean的定义，并实例化bean。
- Spring根据bean的定义填充所有的属性。
- 如果bean实现了BeanNameAware接口，Spring传递bean的ID到setBeanName方法。
- 如果Bean实现了BeanFactoryAware接口，Spring传递beanfactory给setBeanFactory方法。
- 如果有任何与bean相关联的BeanPostProcessors，Spring会在postProcesserBeforeInitialization()方法内调用它们。
- 如果bean实现IntializingBean了，调用它的afterPropertySet方法，如果bean声明了初始化方法，调用此初始化方法。
- 如果有BeanPostProcessors和bean关联，这些bean的postProcessAfterInitialization()方法将被调用。
- 如果bean实现了DisposableBean，它将调用destroy()方法。

28.哪些是重要的bean生命周期方法？你能重载它们吗？

有两个重要的bean生命周期方法，第一个是setup，它是在容器加载bean的时候被调用。第二个方法是teardown，它是在容器卸载类的时候被调用。

The bean标签有两个重要的属性（init-method和destroy-method）。用它们你可以自己定制初始化和注销方法。它们也有相应的注解（@PostConstruct和@PreDestroy）。

29.什么是Spring的内部bean？

当一个bean仅被用作另一个bean的属性时，它能被声明为一个内部bean，为了定义inner bean，在Spring的基于XML的配置元数据中，可以在或元素内使用元素，内部bean通常是匿名的，它们的Scope一般是prototype。

30.在Spring中如何注入一个java集合？

Spring提供以下几种集合的配置元素：

- 类型用于注入一列值，允许有相同的值。
- 类型用于注入一组值，不允许有相同的值。
- 类型用于注入一组键值对，键和值都可以为任意类型。
- 类型用于注入一组键值对，键和值都只能为String类型。

31.什么是bean装配？

装配，或bean 装配是指在Spring 容器中把bean组装到一起，前提是容器需要知道bean的依赖关系，如何通过依赖注入来把它们装配到一起。

32.什么是bean的自动装配？

Spring 容器能够自动装配相互合作的bean，这意味着容器不需要和配置，能通过Bean工厂自动处理bean之间的协作。

33.解释不同方式的自动装配

有五种自动装配的方式，可以用来指导Spring容器用自动装配方式来进行依赖注入

- **no**: 默认的方式是不进行自动装配，通过显式设置ref 属性来进行装配。
- **byName**: 通过参数名 自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byname，之后容器试图匹配、装配和该bean的属性具有相同名字的bean。
- **byType**: 通过参数类型自动装配，Spring容器在配置文件中发现bean的autowire属性被设置成byType，之后容器试图匹配、装配和该bean的属性具有相同类型的bean。如果有多个bean符合条件，则抛出错误。
- **constructor**: 这个方式类似于byType，但是要提供给构造器参数，如果没有确定的带参数的构造器参数类型，将会抛出异常。
- **autodetect**: 首先尝试使用constructor来自动装配，如果无法工作，则使用byType方式。

34.自动装配有哪些局限性？

自动装配的局限性是：

- **重写**: 你仍需用 和 配置来定义依赖，意味着总要重写自动装配。
- **基本数据类型**: 你不能自动装配简单的属性，如基本数据类型，String字符串，和类。
- **模糊特性**: 自动装配不如显式装配精确，如果有可能，建议使用显式装配。

35.你可以在Spring中注入一个null 和一个空字符串吗？

可以。

36.什么是基于Java的Spring注解配置？给一些注解的例子

基于Java的配置，允许你在少量的Java注解的帮助下，进行你的大部分Spring配置而非通过XML文件。

以@Configuration注解为例，它用来标记类可以当做一个bean的定义，被Spring IOC容器使用。另一个例子是@Bean注解，它表示此方法将要返回一个对象，作为一个bean注册进Spring应用上下文。点击[这里](#)学习JAVA几大元注解。

37.什么是基于注解的容器配置？

相对于XML文件，注解型的配置依赖于通过字节码元数据装配组件，而非尖括号的声明。

开发者通过在相应的类，方法或属性上使用注解的方式，直接组件类中进行配置，而不是使用xml表述bean的装配关系。

38.怎样开启注解装配？

注解装配在默认情况下是不开启的，为了使用注解装配，我们必须在Spring配置文件中配置[context:annotation-config](#)元素。

39.@Required 注解

这个注解表明bean的属性必须在配置的时候设置，通过一个bean定义的显式的属性值或通过自动装配，若@Required注解的bean属性未被设置，容器将抛出BeanInitializationException。

40.@Autowired 注解

@Autowired注解提供了更细粒度的控制，包括在何处以及如何完成自动装配。它的用法和@Required一样，修饰setter方法、构造器、属性或者具有任意名称和/或多个参数的PN方法。

41.@Qualifier 注解

当有多个相同类型的bean却只有一个需要自动装配时，将@Qualifier注解和@Autowired注解结合使用以消除这种混淆，指定需要装配的确切的bean。

42.在Spring框架中如何更有效地使用JDBC？

使用SpringJDBC框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写statements和queries从数据存取数据，JDBC也可以在Spring框架提供的模板类的帮助下更有效地被使用，这个模板叫JdbcTemplate（例子见[这里](#)here）

43.JdbcTemplate

JdbcTemplate 类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

44.Spring对DAO的支持

Spring对数据访问对象（DAO）的支持旨在简化它和数据访问技术如JDBC，Hibernate or JDO 结合使用。这使我们可以方便切换持久层。编码时也不用担心会捕获每种技术特有的异常。

45.使用Spring通过什么方式访问Hibernate？

在Spring中有两种方式访问Hibernate：

- 控制反转 Hibernate Template和 Callback
- 继承 HibernateDAOsupport提供一个AOP 拦截器

46.Spring支持的ORM

Spring支持以下ORM：

- Hibernate
- iBatis
- JPA (Java Persistence API)
- TopLink
- JDO (Java Data Objects)
- OJB

47.如何通过HibernateDaoSupport将Spring和Hibernate结合起来？

用Spring的 SessionFactory 调用 LocalSessionFactory。集成过程分三步：

- 配置the Hibernate SessionFactory
- 继承HibernateDaoSupport实现一个DAO
- 在AOP支持的事务中装配

48.Spring支持的事务管理类型

Spring支持两种类型的事务管理：

- **编程式事务管理**: 这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。
- **声明式事务管理**: 这意味着你可以将业务代码和事务管理分离，你只需用注解和XML配置来管理事务。

49.Spring框架的事务管理有哪些优点？

- 它为不同的事务API 如 JTA, JDBC, Hibernate, JPA 和JDO, 提供一个不变的编程模式。
- 它为编程式事务管理提供了一套简单的API而不是一些复杂的事务API如
- 它支持声明式事务管理。
- 它和Spring各种数据访问抽象层很好得集成。

50.你更倾向用那种事务管理类型？

大多数Spring框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此更符合一个无侵入的轻量级容器的思想。声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

51.解释AOP

面向切面的编程，或AOP，是一种编程技术，允许程序模块化横向切割关注点，或横切典型的责任划分，如日志和事务管理。

52.Aspect 切面

AOP核心就是切面，它将多个类的通用行为封装成可重用的模块，该模块含有一组API提供横切功能。比如，一个日志模块可以被称作日志的AOP切面。根据需求的不同，一个应用程序可以有若干切面。在Spring AOP中，切面通过带有@Aspect注解的类实现。

53.在Spring AOP 中，关注点和横切关注的区别是什么？

关注点是应用中一个模块的行为，一个关注点可能会被定义成一个我们想实现的一个功能。

横切关注点是一个关注点，此关注点是整个应用都会使用的功能，并影响整个应用，比如日志、安全和数据传输，几乎应用的每个模块都需要的功能。因此这些都属于横切关注点。

54.连接点

连接点代表一个应用程序的某个位置，在这个位置我们可以插入一个AOP切面，它实际上是个应用程序执行Spring AOP的位置。

55.通知

通知是个在方法执行前或执行后要做的动作，实际上是程序执行时要通过Spring AOP框架触发的代码段。

Spring切面可以应用五种类型的通知：

- **before:** 前置通知，在一个方法执行前被调用
- **after:** 在方法执行之后调用的通知，无论方法执行是否成功
- **after-returning:** 仅当方法成功完成后执行的通知
- **after-throwing:** 在方法抛出异常退出时执行的通知
- **around:** 在方法执行之前和之后调用的通知

56.切入点

切入点是一个或一组连接点，通知将在这些位置执行。可以通过表达式或匹配的方式指明切入点。

57.什么是引入？

引入允许我们在已存在的类中增加新的方法和属性。

58.什么是目标对象？

被一个或者多个切面所通知的对象。它通常是一个代理对象。也指被通知（advised）对象。

59.什么是代理？

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。

60.有几种不同类型的自动代理？

BeanNameAutoProxyCreator

DefaultAdvisorAutoProxyCreator

Metadata autoproxying

61.什么是织入。什么是织入应用的不同点？

织入是将切面和到其他应用类型或对象连接或创建一个被通知对象的过程。

织入可以在编译时，加载时，或运行时完成。

62.解释基于XML Schema方式的切面实现

在这种情况下，切面由常规类以及基于XML的配置实现。

63.解释基于注解的切面实现

在这种情况下(基于@AspectJ的实现)，涉及到的切面声明的风格与带有java5标注的普通java类一致。

64.什么是Spring的MVC框架？

Spring 配备构建Web 应用的全功能MVC框架。Spring可以很便捷地和其他MVC框架集成，如Struts，Spring 的MVC框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

65.DispatcherServlet

Spring的MVC框架是围绕DispatcherServlet来设计的，它用来处理所有的HTTP请求和响应。

66.WebApplicationContext

WebApplicationContext 继承了 ApplicationContext 并增加了一些WEB应用必备的特有功能，它不同于一般的 ApplicationContext，因为它能处理主题，并找到被关联的servlet。

67.什么是Spring MVC框架的控制器？

控制器提供一个访问应用程序的行为，此行为通常通过服务接口实现。控制器解析用户输入并将其转换为一个由视图呈现给用户的模型。Spring用一个非常抽象的方式实现了一个控制层，允许用户创建多种用途的控制器。

68.@Controller 注解

该注解表明该类扮演控制器的角色，Spring不需要你继承任何其他控制器基类或引用Servlet API。

69.@RequestMapping 注解

该注解是用来映射一个URL到一个类或一个特定的方法上。

41道 SpringBoot 面试题以及答案

1. 什么是springboot ?

用来简化spring应用的初始搭建以及开发过程 使用特定的方式来进行配置 (properties或yml文件)

创建独立的spring引用程序 main方法运行

嵌入的Tomcat 无需部署war文件

简化maven配置

自动配置spring添加对应功能starter自动化配置

答：spring boot来简化spring应用开发，约定大于配置，去繁从简，just run就能创建一个独立的，产品级别的应用

2. Springboot 有哪些优点？

-快速创建独立运行的spring项目与主流框架集成

- 使用嵌入式的servlet容器，应用无需打包成war包
- starters自动依赖与版本控制
- 大量的自动配置，简化开发，也可修改默认值
- 准生产环境的运行应用监控
- 与云计算的天然集成

3. 如何重新加载Spring Boot上的更改，而无需重新启动服务器？

这可以使用DEV工具来实现。通过这种依赖关系，您可以节省任何更改，嵌入式tomcat将重新启动。

Spring Boot有一个开发工具（DevTools）模块，它有助于提高开发人员的生产力。Java开发人员面临的一个主要挑战是将文件更改自动部署到服务器并自动重启服务器。

开发人员可以重新加载Spring Boot上的更改，而无需重新启动服务器。这将消除每次手动部署更改的需要。Spring Boot在发布它的第一个版本时没有这个功能。

这是开发人员最重要的功能。DevTools模块完全满足开发人员的需求。该模块将在生产环境中被禁用。它还提供H2数据库控制台以更好地测试应用程序。

```
org.springframework.boot  
spring-boot-devtools  
true
```

4. Spring Boot、Spring MVC 和 Spring 有什么区别？

1、Spring

Spring最重要的特征是依赖注入。所有 SpringModules 不是依赖注入就是 IOC 控制反转。

当我们恰当的使用 DI 或者是 IOC 的时候，我们可以开发松耦合应用。松耦合应用的单元测试可以很容易的进行。

2、Spring MVC

Spring MVC 提供了一种分离式的方法来开发 Web 应用。通过运用像 DispatcherServlet, MoudlAndView 和 ViewResolver 等一些简单的概念，开发 Web 应用将会变得非常简单。

3、SpringBoot

Spring 和 SpringMVC 的问题在于需要配置大量的参数。

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <property name="prefix">  
        <value>/WEB-INF/views/</value>  
    </property>  
    <property name="suffix">  
        <value>.jsp</value>  
    </property>  
</bean>  
  
<mvc:resources mapping="/webjars/**" location="/webjars/" />
```

Spring Boot 通过一个自动配置和启动的项来解决这个问题。为了更快的构建产品就绪应用程序，Spring Boot 提供了一些非功能性特征。

5. 什么是自动配置？

Spring 和 SpringMVC 的问题在于需要配置大量的参数。

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix">
        <value>/WEB-INF/views/</value>
    </property>
    <property name="suffix">
        <value>.jsp</value>
    </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/" />
```

https://blog.csdn.net/Dome_

我们能否带来更多的智能？当一个 MVC JAR 添加到应用程序中的时候，我们能否自动配置一些 beans？

Spring 查看（CLASSPATH 上可用的框架）已存在的应用程序的配置。在此基础上，Spring Boot 提供了配置应用程序和框架所需要的基本配置。这就是自动配置。

6 .什么是 Spring Boot Stater ?

启动器是一套方便的依赖没描述符，它可以放在自己的程序中。你可以一站式的获取你所需要的 Spring 和相关技术，而不需要依赖描述符的通过示例代码搜索和复制黏贴的负载。

例如，如果你想使用 Sping 和JPA 访问数据库，只需要你的项目包含 spring-boot-starter-data-jpa 依赖项，你就可以完美进行。

7 .能否举一个例子来解释更多 Staters 的内容？

让我们来思考一个 Stater 的例子 -Spring Boot Stater Web。

如果你想开发一个 web 应用程序或者是公开 REST 服务的应用程序。Spring Boot Start Web 是首选。让我们使用 Spring Initializr 创建一个 Spring Boot Start Web 的快速项目。

Spring Boot Start Web 的依赖项

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

下面的截图是添加进我们应用程序的不同的依赖项

Maven Dependencies

- ▶ spring-boot-starter-web-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/starter/web/1.4.4.RELEASE/spring-boot-starter-web-1.4.4.RELEASE.jar
- ▶ spring-boot-starter-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/starter/1.4.4.RELEASE/spring-boot-starter-1.4.4.RELEASE.jar
- ▶ spring-boot-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/1.4.4.RELEASE/spring-boot-1.4.4.RELEASE.jar
- ▶ spring-boot-autoconfigure-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/autoconfigure/1.4.4.RELEASE/spring-boot-autoconfigure-1.4.4.RELEASE.jar
- ▶ spring-boot-starter-logging-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/starter/logging/1.4.4.RELEASE/spring-boot-starter-logging-1.4.4.RELEASE.jar
- ▶ logback-classic-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-classic/1.1.9/logback-classic-1.1.9.jar
- ▶ logback-core-1.1.9.jar - /Users/rangaraokaranam/.m2/repository/ch/qos/logback/logback-core/1.1.9/logback-core-1.1.9.jar
- ▶ slf4j-api-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/slf4j-api/1.7.22/slf4j-api-1.7.22.jar
- ▶ jcl-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jcl-over-slf4j/1.7.22/jcl-over-slf4j-1.7.22.jar
- ▶ jul-to-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/jul-to-slf4j/1.7.22/jul-to-slf4j-1.7.22.jar
- ▶ log4j-over-slf4j-1.7.22.jar - /Users/rangaraokaranam/.m2/repository/org/slf4j/log4j-over-slf4j/1.7.22/log4j-over-slf4j-1.7.22.jar
- ▶ spring-core-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/core/4.3.6.RELEASE/spring-core-4.3.6.RELEASE.jar
- ▶ snakeyaml-1.17.jar - /Users/rangaraokaranam/.m2/repository/org/yaml/snakeyaml/1.17/snakeyaml-1.17.jar
- ▶ spring-boot-starter-tomcat-1.4.4.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/boot/starter/tomcat/1.4.4.RELEASE/spring-boot-starter-tomcat-1.4.4.RELEASE.jar
- ▶ tomcat-embed-core-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/embed/core/8.5.11/tomcat-embed-core-8.5.11.jar
- ▶ tomcat-embed-el-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/embed/el/8.5.11/tomcat-embed-el-8.5.11.jar
- ▶ tomcat-embed-websocket-8.5.11.jar - /Users/rangaraokaranam/.m2/repository/org/apache/tomcat/embed/websocket/8.5.11/tomcat-embed-websocket-8.5.11.jar
- ▶ hibernate-validator-5.2.4.Final.jar - /Users/rangaraokaranam/.m2/repository/org/hibernate/validator/5.2.4.Final/hibernate-validator-5.2.4.Final.jar
- ▶ validation-api-1.1.0.Final.jar - /Users/rangaraokaranam/.m2/repository/javax/validation/validation-api/1.1.0.Final/validation-api-1.1.0.Final.jar
- ▶ jboss-logging-3.3.0.Final.jar - /Users/rangaraokaranam/.m2/repository/org/jboss/logging/jboss-logging/3.3.0.Final/jboss-logging-3.3.0.Final.jar
- ▶ classmate-1.3.3.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/classmate/1.3.3/classmate-1.3.3.jar
- ▶ jackson-databind-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/databind/2.8.6/jackson-databind-2.8.6.jar
- ▶ jackson-annotations-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/annotations/2.8.6/jackson-annotations-2.8.6.jar
- ▶ jackson-core-2.8.6.jar - /Users/rangaraokaranam/.m2/repository/com/fasterxml/jackson/core/2.8.6/jackson-core-2.8.6.jar
- ▶ spring-web-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/web/4.3.6.RELEASE/spring-web-4.3.6.RELEASE.jar
- ▶ spring-aop-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/aop/4.3.6.RELEASE/spring-aop-4.3.6.RELEASE.jar
- ▶ spring-beans-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/beans/4.3.6.RELEASE/spring-beans-4.3.6.RELEASE.jar
- ▶ spring-context-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/context/4.3.6.RELEASE/spring-context-4.3.6.RELEASE.jar
- ▶ spring-webmvc-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/webmvc/4.3.6.RELEASE/spring-webmvc-4.3.6.RELEASE.jar
- ▶ spring-expression-4.3.6.RELEASE.jar - /Users/rangaraokaranam/.m2/repository/org/springframework/expression/4.3.6.RELEASE/spring-expression-4.3.6.RELEASE.jar

https://blog.csdn.net/Dorne_

依赖项可以被分为：

- Spring - core, beans, context, aop
- Web MVC - (Spring MVC)
- Jackson - for JSON Binding
- Validation - Hibernate,Validation API
- Embedded Servlet Container - Tomcat
- Logging - logback,slf4j

任何经典的 Web 应用程序都会使用所有这些依赖项。Spring Boot Starter Web 预先打包了这些依赖项。

作为一个开发者，我不需要再担心这些依赖项和它们的兼容版本。

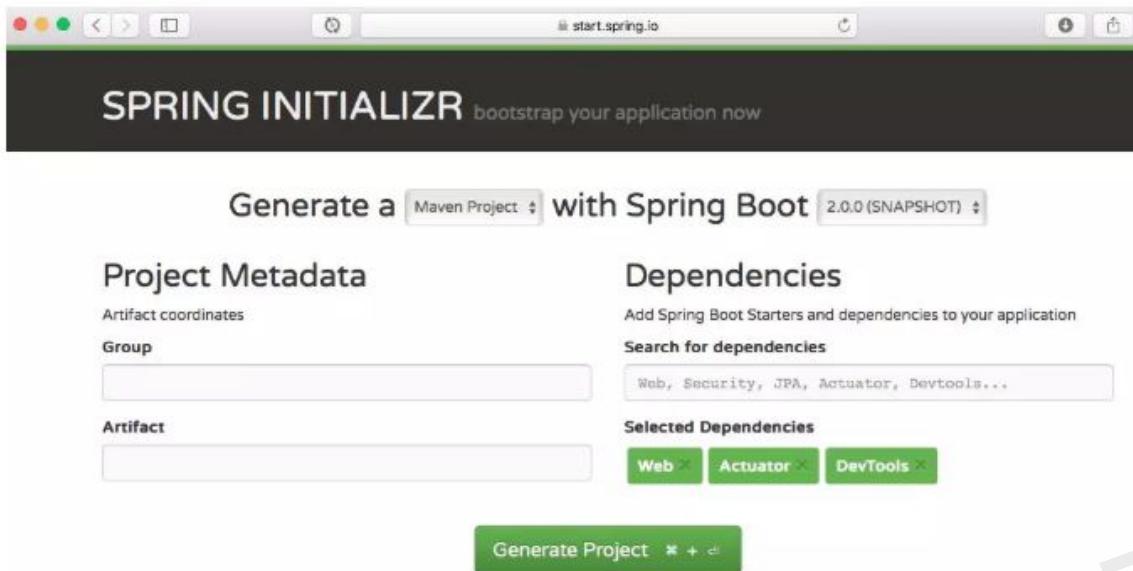
8.Spring Boot 还提供了其它的哪些 Starter Project Options?

Spring Boot 也提供了其它的启动器项目包括，包括用于开发特定类型应用程序的典型依赖项。

- spring-boot-starter-web-services - SOAP Web Services;
- spring-boot-starter-web - Web 和 RESTful 应用程序;
- spring-boot-starter-test - 单元测试和集成测试;
- spring-boot-starter-jdbc - 传统的 JDBC;
- spring-boot-starter-hateoas - 为服务添加 HATEOAS 功能;
- spring-boot-starter-security - 使用 SpringSecurity 进行身份验证和授权;
- spring-boot-starter-data-jpa - 带有 Hibernat 的 Spring Data JPA;
- spring-boot-starter-data-rest - 使用 Spring Data REST 公布简单的 REST 服务;

9. 创建一个 Spring Boot Project 的最简单的方法是什么？

Spring Initializr是启动 Spring Boot Projects 的一个很好的工具。



就像上图中所展示的一样，我们需要做一下几步：

- 1、登录 Spring Initializr，按照以下方式进行选择：
- 2、选择 com.in28minutes.springboot 为组
- 3、选择 studet-services 为组件
- 4、选择下面的依赖项
 - Web
 - Actuator
 - DevTools
- 5、点击生 GenerateProject
- 6、将项目导入 Eclipse。文件 - 导入 - 现有的 Maven 项目

10. Spring Initializr 是创建 Spring Boot Projects 的唯一方法吗？

不是的。

Spring Initializr 让创建 Spring Boot 项目变的很容易，但是，你也可以通过设置一个 maven 项目并添加正确的依赖项来开始一个项目。

在我们的 Spring 课程中，我们使用两种方法来创建项目。

第一种方法是 start.spring.io 。

另外一种方法是在项目的标题为“Basic Web Application”处进行手动设置。

手动设置一个 maven 项目

这里有几个重要的步骤：

- 1、在 Eclipse 中，使用文件 - 新建 Maven 项目来创建一个新项目
 - 2、添加依赖项。
 - 3、添加 maven 插件。
 - 4、添加 Spring Boot 应用程序类。
- 到这里，准备工作已经做好！

11.为什么我们需要 spring-boot-maven-plugin?

spring-boot-maven-plugin 提供了一些像 jar 一样打包或者运行应用程序的命令。

- 1、spring-boot:run 运行你的 SpringBooty 应用程序。
- 2、spring-boot: repackage 重新打包你的 jar 包或者是 war 包使其可执行
- 3、spring-boot: start 和 spring-boot: stop 管理 Spring Boot 应用程序的生命周期（也可以说是为了集成测试）。
- 4、spring-boot:build-info 生成执行器可以使用的构造信息。

12.如何使用 SpringBoot 自动重装我的应用程序?

使用 Spring Boot 开发工具。

把 Spring Boot 开发工具添加进入你的项目是简单的。

把下面的依赖项添加至你的 Spring Boot Project pom.xml 中

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
```

重启应用程序，然后就可以了。

同样的，如果你想自动装载页面，有可以看看 FiveReload

<http://www.logicbig.com/tutorials/spring-framework/spring-boot/boot-livereload/>.

在我测试的时候，发现了 LiveReload 漏洞，如果你测试时也发现了，请一定要告诉我们。

13.Spring Boot中的监视器是什么？

Spring boot actuator是spring启动框架中的重要功能之一。Spring boot监视器可帮助您访问生产环境中正在运行的应用程序的当前状态。

有几个指标必须在生产环境中进行检查和监控。即使一些外部应用程序可能正在使用这些服务来向相关人员触发警报消息。监视器模块公开了一组可直接作为HTTP URL访问的REST端点来检查状态。

14.什么是YAML?

YAML是一种人类可读的数据序列化语言。它通常用于配置文件。

与属性文件相比，如果我们想要在配置文件中添加复杂的属性，YAML文件就更加结构化，而且更少混淆。可以看出YAML具有分层配置数据。

15.springboot自动配置的原理

在spring程序main方法中 添加@SpringBootApplication或者@EnableAutoConfiguration

会自动去maven中读取每个starter中的spring.factories文件 该文件里配置了所有需要被创建spring容器中的bean

16.springboot读取配置文件的方式

springboot默认读取配置文件为application.properties或者是application.yml

17.springboot集成mybatis的过程

添加mybatis的starter maven依赖

```
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>1.3.2</version>
</dependency>
```

在mybatis的接口中 添加@Mapper注解

在application.yml配置数据源信息

18.什么是嵌入式服务器？我们为什么要使用嵌入式服务器呢？

思考一下在你的虚拟机上部署应用程序需要些什么。

第一步：安装 Java

第二步：安装 Web 或者是应用程序的服务器 (Tomcat/WebSphere/WebLogic 等等)

第三步：部署应用程序 war 包

如果我们想简化这些步骤，应该如何做呢？

让我们来思考如何使服务器成为应用程序的一部分？

你只需要一个安装了 Java 的虚拟机，就可以直接在上面部署应用程序了，

这个想法是嵌入式服务器的起源。

当我们创建一个可以部署的应用程序的时候，我们将会把服务器（例如，tomcat）嵌入到可部署的服务器中。

例如，对于一个 Spring Boot 应用程序来说，你可以生成一个包含 Embedded Tomcat 的应用程序 jar。你就可以像运行正常 Java 应用程序一样来运行 web 应用程序了。

嵌入式服务器就是我们的可执行单元包含服务器的二进制文件（例如，tomcat.jar）。

19.如何在 Spring Boot 中添加通用的 JS 代码?

在源文件夹下，创建一个名为 static 的文件夹。然后，你可以把你的静态的内容放在这里面。

例如，myapp.js 的路径是 resources\static\js\myapp.js

你可以参考它在 jsp 中的使用方法：

```
<script src="/js/myapp.js"></script>
```

错误：HAL browser gives me unauthorized error - Full authentication is required to access this resource.

该如何来修复这个错误呢？

```
{  
    "timestamp": 1488656019562,  
    "status": 401,  
    "error": "Unauthorized",  
    "message": "Full authentication is required to access this resource.",  
    "path": "/beans"  
}
```

两种方法：

方法 1：关闭安全验证

application.properties

```
management.security.enabled:FALSE
```

方法二：在日志中搜索密码并传递至请求标头中

20.什么是 Spring Data?

来自：//projects.spring.io/spring-data/

Spring Data 的使命是在保证底层数据存储特殊性的前提下，为数据访问提供一个熟悉的，一致性的，基于 Spring 的编程模型。这使得使用数据访问技术，关系数据库和非关系数据库，map-reduce 框架以及基于云的数据服务变得很容易。

为了让它更简单一些，Spring Data 提供了不受底层数据源限制的 Abstractions 接口。

下面来举一个例子：

```
interface TodoRepository extends CrudRepository<Todo, Long> {
```

```
H
```

你可以定义一简单的库，用来插入，更新，删除和检索代办事项，而不需要编写大量的代码。

21. 什么是 Spring Data REST?

Spring Data TEST 可以用来发布关于 Spring 数据库的 HATEOAS RESTful 资源。

下面是一个使用 JPA 的例子：

```
@RepositoryRestResource(collectionResourceRel = "todos", path = "todos")
public interface TodoRepository extends PagingAndSortingRepository<Todo, Long> {
}
```

不需要写太多代码，我们可以发布关于 Spring 数据库的 RESTful API。

下面展示的是一些关于 TEST 服务器的例子

```
POST:URL:http://localhost:8080/todosuse Header:Content-Type:Type:application/jsonRequest Content
```

代码如下：

```
{
  "user": "Jill",
  "desc": "Learn Hibernate",
  "done": false
}
```

响应内容：

```
{
  "user": "Jill",
  "desc": "Learn Hibernate",
  "done": false,
  "_links": {
    "self": {
      "href": "http://localhost:8080/todos/1"
    },
    "todo": {
      "href": "http://localhost:8080/todos/1"
    }
  }
}
```

https://blog.csdn.net/Dome_

响应包含新创建资源的 href。

22.path="users", collectionResourceRel="users" 如何与 Spring Data Rest 一起使用?

```
@RepositoryRestResource(collectionResourceRel = "users", path = "users")  
public interface UserRestRepository extends PagingAndSortingRepository<User, Long>{  
}
```

path- 这个资源要导出的路径段。

collectionResourceRel- 生成指向集合资源的链接时使用的 rel 值。在生成 HATEOAS 链接时使用。

23.当 Spring Boot 应用程序作为 Java 应用程序运行时，后台会发什么？

如果你使用 Eclipse IDE, Eclipse maven 插件确保依赖项或者类文件的改变一经添加，就会被编译并在目标文件中准备好！在这之后，就和其它的 Java 应用程序一样了。

当你启动 java 应用程序的时候，spring boot 自动配置文件就会魔法般的启用了。

当 Spring Boot 应用程序检测到你正在开发一个 web 应用程序的时候，它就会启动 tomcat。

24.我们能否在 spring-boot-starter-web 中用 jetty 代替 tomcat？

在 spring-boot-starter-web 移除现有的依赖项，并把下面这些添加进去。

```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-web</artifactId>  
    <exclusions>  
        <exclusion>  
            <groupId>org.springframework.boot</groupId>  
            <artifactId>spring-boot-starter-tomcat</artifactId>  
        </exclusion>  
    </exclusions>  
</dependency>  
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-jetty</artifactId>  
</dependency>
```

https://blog.csdn.net/Dome_

25.如何使用 Spring Boot 生成一个 WAR 文件？

推荐阅读：

<https://spring.io/guides/gs/convert-jar-to-war/>

下面有 spring 说明文档直接的链接地址：

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#build-tool-plugins-maven-packaging>

26.如何使用 Spring Boot 部署到不同的服务器？

你需要做下面两个步骤：

在一个项目中生成一个 war 文件。

将它部署到你最喜欢的服务器（websphere 或者 Weblogic 或者 Tomcat and so on）。

第一步：这本入门指南应该有所帮助：

<https://spring.io/guides/gs/convert-jar-to-war/>

第二步：取决于你的服务器。

27.RequestMapping 和 GetMapping 的不同之处在哪里？

RequestMapping 具有类属性的，可以进行 GET,POST,PUT 或者其它的注释中具有的请求方法。

GetMapping 是 GET 请求方法中的一个特例。它只是 RequestMapping 的一个延伸，目的是为了提高清晰度。

28.为什么我们不建议在实际的应用程序中使用 Spring Data Rest？

我们认为 Spring Data Rest 很适合快速原型制造！在大型应用程序中使用需要谨慎。

通过 Spring Data REST 你可以把你的数据实体作为 RESTful 服务直接发布。

当你设计 RESTful 服务器的时候，最佳实践表明，你的接口应该考虑到两件重要的事情：

你的模型范围。

你的客户。

通过 With Spring Data REST，你不需要再考虑这两个方面，只需要作为 TEST 服务发布实体。

这就是为什么我们建议使用 Spring Data Rest 在快速原型构造上面，或者作为项目的初始解决方法。对于完整演变项目来说，这并不是一个好的注意。

29.在 Spring Initializer 中，如何改变一个项目的包名字？

好消息是你可以定制它。点击链接“转到完整版本”。你可以配置你想要修改的包名称！

30.JPA 和 Hibernate 有哪些区别？

简而言之

JPA 是一个规范或者接口

Hibernate 是 JPA 的一个实现

当我们使用 JPA 的时候，我们使用 javax.persistence 包中的注释和接口时，不需要使用 hibernate 的导入包。

我们建议使用 JPA 注释，因为哦我们没有将其绑定到 Hibernate 作为实现。后来（我知道 - 小于百分之一的几率），我们可以使用另一种 JPA 实现。

31. 使用 Spring Boot 启动连接到内存数据库 H2 的 JPA 应用程序需要哪些依赖项？

在 Spring Boot 项目中，当你确保下面的依赖项都在类路里面的时候，你可以加载 H2 控制台。

web 启动器

h2

jpa 数据启动器

其它的依赖项在下面：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

https://blog.csdn.net/Gume_

需要注意的一些地方：

一个内部数据内存只在应用程序执行期间存在。这是学习框架的有效方式。

这不是你希望的真是世界应用程序的方式。

在问题“如何连接一个外部数据库？”中，我们解释了如何连接一个你所选择的数据库。

32. 如何不通过任何配置来选择 Hibernate 作为 JPA 的默认实现？

因为 Spring Boot 是自动配置的。

下面是我们添加的依赖项：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

spring-boot-stater-data-jpa 对于 Hibernate 和 JPA 有过渡依赖性。

当 Spring Boot 在类路径中检测到 Hibernate 中，将会自动配置它为默认的 JPA 实现。

33. 我们如何连接一个像 MySQL 或者 Oracle 一样的外部数据库？

让我们以 MySQL 为例来思考这个问题：

第一步 - 把 mysql 连接器的依赖项添加至 pom.xml

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

第二步 - 从 pom.xml 中移除 H2 的依赖项

或者至少把它作为测试的范围。

```
<!--
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
-->
```

https://blog.csdn.net/Demo_

第三步 - 安装你的 MySQL 数据库

更多的来看看这里 - <https://github.com/in28minutes/jpa-with-hibernate#installing-and-setting-up-mysql>

第四步 - 配置你的 MySQL 数据库连接

配置 application.properties

```
spring.jpa.hibernate.ddl-auto=none
spring.datasource.url=jdbc:mysql://localhost:3306/todo_example
spring.datasource.username=todouser
spring.datasource.password=YOUR_PASSWORD
```

第五步 - 重新启动，你就准备好了！

就是这么简单！

34. 你能否举一个以 `ReadOnly` 为事务管理的例子？

当你从数据库读取内容的时候，你想把事物中的用户描述或者是其它描述设置为只读模式，以便于 Hebernate 不需要再次检查实体的变化。这是非常高效的。

35. Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？

启动类上面的注解是`@SpringBootApplication`，它也是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

`@SpringBootConfiguration`: 组合了 `@Configuration` 注解，实现配置文件的功能。

`@EnableAutoConfiguration`: 打开自动配置的功能，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能：

`@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })`。

`@ComponentScan`: Spring组件扫描。

36. 开启 Spring Boot 特性有哪几种方式？

- 1) 继承`spring-boot-starter-parent`项目
- 2) 导入`spring-boot-dependencies`项目依赖

37. Spring Boot 需要独立的容器运行吗？

可以不需要，内置了 Tomcat/ Jetty 等容器。

38. 运行 Spring Boot 有哪几种方式？

- 1) 打包用命令或者放到容器中运行
- 2) 用 Maven/ Gradle 插件运行
- 3) 直接执行 `main` 方法运行

39. 你如何理解 Spring Boot 中的 Starters？

Starters可以理解为启动器，它包含了一系列可以集成到应用里面的依赖包，你可以一站式集成 Spring 及其他技术，而不需要到处找示例代码和依赖包。如你想使用 Spring JPA 访问数据库，只要加入 `spring-boot-starter-data-jpa` 启动器依赖就能使用了。

40.Spring Boot 支持哪些日志框架？推荐和默认的日志框架是哪个？

Spring Boot 支持 Java Util Logging, Log4j2, Logback 作为日志框架，如果你使用 Starters 启动器，Spring Boot 将使用 Logback 作为默认日志框架。

41.SpringBoot 实现热部署有哪几种方式？

主要有两种方式：

- 1、Spring Loaded
- 2、Spring-boot-devtools

Spring AOP 16道面试题及答案

1、描述一下Spring AOP

Spring AOP(Aspect Oriented Programming, 面向切面编程)是OOPs(面向对象编程)的补充，它也提供了模块化。在面向对象编程中，关键的单元是对象，AOP的关键单元是切面，或者说关注点（可以简单地理解为你程序中的独立模块）。一些切面可能有集中的代码，但是有些可能被分散或者混杂在一起，例如日志或者事务。这些分散的切面被称为横切关注点。一个横切关注点是一个可以影响到整个应用的关注点，而且应该被尽量地集中到代码的一个地方，例如事务管理、权限、日志、安全等。

AOP让你可以使用简单可插拔的配置，在实际逻辑执行之前、之后或周围动态添加横切关注点。这让代码在当下和将来都变得易于维护。如果你是使用XML来使用切面的话，要添加或删除关注点，你不用重新编译完整的源代码，而仅仅需要修改配置文件就可以了。

Spring AOP通过以下两种方式来使用。但是最广泛使用的方式是Spring AspectJ 注解风格(Spring AspectJ Annotation Style)。

- 使用AspectJ 注解风格
- 使用Spring XML 配置风格

2、在Spring AOP中关注点和横切关注点有什么不同？

关注点是我们想在应用的模块中实现的行为。关注点可以被定义为：我们想实现以解决特定业务问题的方法。比如，在所有电子商务应用中，不同的关注点（或者模块）可能是库存管理、航运管理、用户管理等。

横切关注点是贯穿整个应用程序的关注点。像日志、安全和数据转换，它们在应用的每一个模块都是必须的，所以他们是一种横切关注点。

3、AOP有哪些可用的实现？

基于Java的主要AOP实现有，如下。

- AspectJ
- Spring AOP
- JBoss AOP

在维基百科上你可以找到一个AOP实现的大列表。

Implementations [edit]

The following programming languages have implemented AOP, within the language, or as an external library:

- .NET Framework languages (C# / VB.NET)^[17]
 - PostSharp^[18] is a commercial AOP implementation with a free but limited edition.
 - Unity, It provides an API to facilitate proven practices in core areas of programming including data access, security, logging, exception handling and others.
- ActionScript^[19]
- Ada^[19]
- AutoHotkey^[20]
- C / C++^[21]
- COBOL^[22]
- The Cocoa Objective-C frameworks^[23]
- ColdFusion^[24]
- Common Lisp^[25]
- Delphi^[26]^[27]^[28]
- Delphi Prism^[29]
- e (IEEE 1647)
- Emacs Lisp^[30]
- Groovy
- Haskell^[31]
- Java^[32]
 - AspectJ
- JavaScript^[33]
- Logtalk^[34]
- Lua^[35]
- make^[36]
- Matlab^[37]
- ML^[38]
- Perl^[39]
- PHP^[40]
- Prolog^[41]
- Python^[42]
- Racket^[43]
- Ruby^[44]^[45]^[46]
- Squeak Smalltalk^[47]^[48]
- UML 2.0^[49]
- XML^[50]

4、Spring中有哪些不同的通知类型

通知(advice)是你在你的程序中想要应用在其他模块中的横切关注点的实现。Advice主要有以下5种类型。

前置通知(Before Advice): 在连接点之前执行的Advice，不过除非它抛出异常，否则没有能力中断执行流。使用 @Before注解使用这个Advice。

返回之后通知(After Retuning Advice): 在连接点正常结束之后执行的Advice。例如，如果一个方法没有抛出异常正常返回。通过 @AfterReturning`关注使用它。

抛出（异常）后执行通知(After Throwing Advice): 如果一个方法通过抛出异常来退出的话，这个Advice就会被执行。通常 @AfterThrowing `注解来使用。

后置通知(After Advice): 无论连接点是通过什么方式退出的(正常返回或者抛出异常)都会执行在结束后执行这些Advice。通过@After注解使用。

围绕通知(Around Advice): 围绕连接点执行的Advice，就你一个方法调用。这是最强大的Advice。通过@Around注解使用。

5、Spring AOP 代理是什么？

代理是使用非常广泛的设计模式。简单来说，代理是一个看其他像另一个对象的对象，但它添加了一些特殊的功能。

Spring AOP是基于代理实现的。AOP 代理是一个由 AOP 框架创建的用于在运行时实现切面协议的对象。

Spring AOP默认认为 AOP 代理使用标准的 JDK 动态代理。这使得任何接口（或者接口的集合）可以被代理。Spring AOP 也可以使用 CGLIB 代理。这对代理类而不是接口是必须的。

如果业务对象没有实现任何接口那么默认使用CGLIB。

6、引介(Introduction)是什么？

引介让一个切面可以声明被通知的对象实现了任何他们没有真正实现的额外接口，而且为这些对象提供接口的实现。

使用 @DeclareParents `注解来生成一个引介。

更多详情，请参考官方文档。

7、连接点(Joint Point)和切入点(Point cut)是什么？

连接点是程序执行的一个点。例如，一个方法的执行或者一个异常的处理。在 Spring AOP 中，一个连接点总是代表一个方法执行。举例来说，所有定义在你的 EmployeeManager 接口中的方法都可以被认为是一个连接点，如果你在这些方法上使用横切关注点的话。

切入点是一个匹配连接点的断言或者表达式。Advice 与切入点表达式相关联，并在切入点匹配的任何连接点处运行（比如，表达式 execution(* EmployeeManager.getEmployeeById(..)) 可以匹配 EmployeeManager 接口的 getEmployeeById()）。由切入点表达式匹配的连接点的概念是 AOP 的核心。Spring 默认使用 AspectJ 切入点表达式语言。

8、什么是织入(weaving)?

Spring AOP 框架仅支持有限的几个 AspectJ 切入点的类型，它允许将切面运用到在 IoC 容器中声明的 bean 上。如果你想使用额外的切入点类型或者将切面应用到在 Spring IoC 容器外部创建的类，那么你必须在你的 Spring 程序中使用 AspectJ 框架，并且使用它的织入特性。

织入是将切面与外部的应用类型或者类连接起来以创建通知对象(advised object)的过程。这可以在编译时(比如使用 AspectJ 编译器)、加载时或者运行时完成。Spring AOP 跟其他纯 Java AOP 框架一样，只在运行时执行织入。在协议上，AspectJ 框架支持编译时和加载时织入。

AspectJ 编译时织入是通过一个叫做ajc特殊的 AspectJ 编译器完成的。它可以将切面织入到你的 Java 源码文件中，然后输出织入后的二进制 class 文件。它也可以将切面织入你的编译后的 class 文件或者 Jar 文件。这个过程叫做后编译时织入(post-compile-time weaving)。在 Spring IoC 容器中声明你的类之前，你可以为它们运行编译时和后编译时织入。Spring 完全没有被包含到织入的过程中。更多关于编译时和后编译时织入的信息，请查阅 AspectJ 文档。

AspectJ 加载时织入(load-time weaving, LTW)在目标类被类加载器加载到VM时触发。对于一个被织入的对象，需要一个特殊的类加载器来增强目标类的字节码。AspectJ 和 Spring 都提供了加载时织入器以为类加载添加加载时织入的能力。你只需要简单的配置就可以打开这个加载时织入器。

9.什么是Aspect?

Aspect由PointCut和Advice组成

它即包含了横切逻辑的定义，也包括了连接点的定义

SpringAOP就是负责实施切面的框架，它将切面所定义的横切逻辑编辑到切面指定的连接点中

AOP的工作重心在于如何将增强编织目标对象的连接点中，这里包含两个工作：

1.如何通过PointCut和Advice定位到特定的JoinPoint上

2.如何Advice中编写切面编程

可以简单地认为，使用@Aspect注解的类就是切面

10.什么是JoinPoint?

JoinPoint,切点，程序运行中的一些时间点，例如：

一个方法的执行

或者是一个异常处理

在SpringAOP中，JoinPoint总是方法的执行点

11.什么是PointCut?

PointCut，匹配JoinPoint的谓词 (a predicate that matches join points)

简单来说，PointCut是匹配JoinPoint的条件

Advice是和特定PointCut关联的，并且在PointCut相匹配的JoinPoint中执行，即
Advice=>PointCut=>JoinPoint.

在Spring中，所有的方法都可以认为JoinPoint，但是我们并不希望在所有方法上都添加Advice。而 PointCut的作用，就是提供一组规则 (使用 AspectJ PointCut expression language来描述) 来匹配 JoinPoint，给满足规则的JoinPoint添加Advice

是不是觉得有点绕，实际场景下，其实也不会这么问（一般面试都是根据逻辑说知识）

12.有哪些类型Advice？

Before 这些类型的Advice在JoinPoint方法之前执行，并使用@Before注解进行配置

After Returning这些类型的Advice在连接点方法正常执行后执行，并使用@AfterReturning注解进行配置

AfterThrowing这些类型的Advice仅在JoinPoint方法通过抛出异常退出使用@AfterThrowing注解进行配置

AfterFinally这些类型的Advice在连接点方法之后指定，无论方法退出时正常还是异常返回，并使用@After注解标记进行配置

Around这些类型的Advice在连接点之前和之后执行，并使用@Around注解进行配置

13.什么是 Target？

Target，织入Advice的目标对象。目标对象也被称为 Advised Object

因为SpringAOP使用运行时代理的方式来实现Aspect因此AdvisedObject总是一个代理对象（Proxy Object）

注意Advised Object指的不是原来的对象，而是织入Advice后所产生的代理对象

advice+Target Object = Advised Object = Proxy。

14.AOP有哪些实现方式

实现AOP的技术，主要分为两类：

静态代理指使用AOP框架停工的命令进行编译，从而在编译阶段就可以生成AOP代理类，因此也称为AOP代理类，因此也称为编译时增强

编译时编织（特殊编译器实现）

类加载时编织（特殊的类加载器实现）

动态代理在运行时在内存中临时生成AOP动态代理类，因此也被称为运行时增强，目前Spring中使用了两种代理库

JDK动态代理

CGLIB

15.Spring什么时候使用JDK动态代理，什么时候使用CGLIB呢？

Spring AOP部分使用JDK动态代理或则CGLIB来为目标对象创建代理（建议尽量使用JDK的动态代理）如果被代理的目标对象实现了至少一个接口，则会使用JDK动态代理。所有该目标类型实现的接口都被该代理

若该目标对象没有实现任何接口，则创建一个CGLIB代理

如果你希望强制使用CGLIB代理，（例如希望代理目标的所有方法，而不只是实现自定义接口）那么也可以，但是需要考虑一下两个方法：

无法通知（advise）Final方法，因为他们不能被覆盖

你需要将CGLIB二进制发型包放在classpath下面

为什么Spring默认使用JDK的动态代理呢？（个人猜测）

使用JDK原始支持，减少依赖

JDK8开始后，JDK代理的性能差距CGLIB的性能不会太多

SpringAOP中的动态代理主要两种方式

JDK动态代理

JDK动态代理通过反射来接收被代理的类，并且要求被代理的类必须实现一个借口，JDK动态代理的核心是InvocationHandler接口和Proxy类

CGLIB动态代理

如果目标没有实现接口，那么SpringAOP会选择使用CGLIB来动态代理目标类，当然Spring也支持配置强制使用CGLIB动态代理

CGLIB是一个代码生成的类库，可以在运行时动态的生成某个类的子类，注意，CGLIB是通过集成的方式做的动态代理，因此如果某个类被标记为final，那么它无法使用CGLIB做动态代理

16.SpringAOP 和 AspectjAOP有什么区别？

代理方式不同

SpringAOP基于动态代理

AspectjAOP基于静态代理实现方式

PointCut支持力度不同

Spring AOP仅支持方法级别的PointCut

AspectjAOP提供了完全的AOP支持，它还支持属性级别的PointCut

30道 Spring Cloud 面试题及答案

1: 什么是Spring Cloud?

spring cloud 是一系列框架的有序集合。它利用 spring boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 spring boot 的开发风格做到一键启动和部署。

2: 使用Spring Cloud有什么优势？

使用Spring Boot开发分布式微服务时，我们面临以下问题

1. 与分布式系统相关的复杂性-这种开销包括网络问题，延迟开销，带宽问题，安全问题。
2. 服务发现-服务发现工具管理群集中的流程和服务如何查找和互相交谈。它涉及一个服务目录，在该目录中注册服务，然后能够查找并连接到该目录中的服务。
3. 冗余-分布式系统中的冗余问题。
4. 负载平衡--负载平衡改善跨多个计算资源的工作负荷，诸如计算机，计算机集群，网络链路，中央处理单元，或磁盘驱动器的分布。
5. 性能-问题 由于各种运营开销导致的性能问题。
6. 部署复杂性-Devops技能的要求。

问题三：服务注册和发现是什么意思？Spring Cloud如何实现？

当我们开始一个项目时，我们通常在属性文件中进行所有的配置。随着越来越多的服务开发和部署，添加和修改这些属性变得更加复杂。有些服务可能会下降，而某些位置可能会发生变化。手动更改属性可能会产生问题。Eureka服务注册和发现可以在这种情况下提供帮助。由于所有服务都在Eureka服务器上注册并通过调用Eureka服务器完成查找，因此无需处理服务地点的任何更改和处理。

问题四：负载平衡的意义什么？

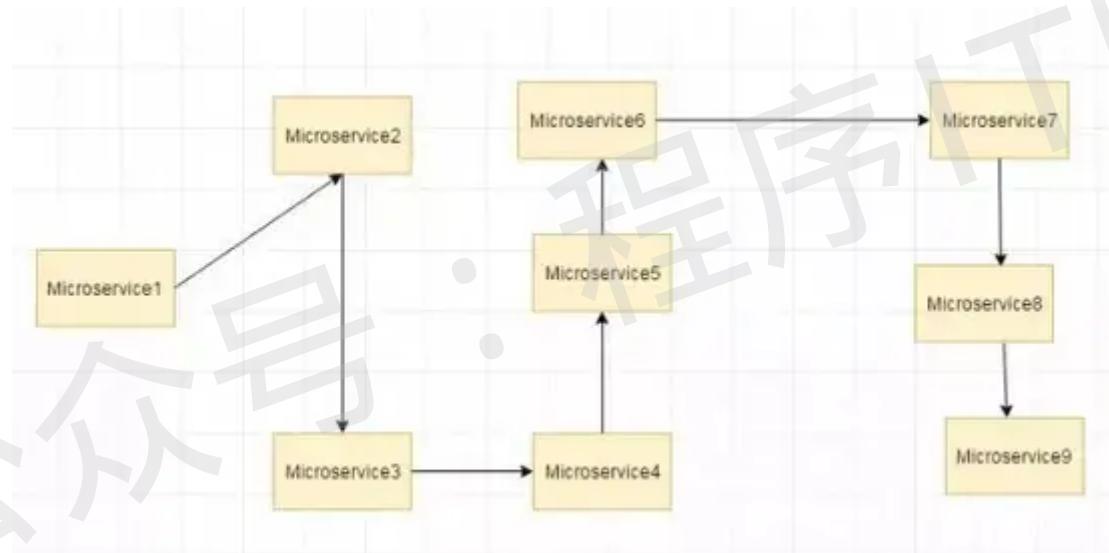
在计算中，负载平衡可以改善跨计算机，计算机集群，网络链接，中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载平衡旨在优化资源使用，最大化吞吐量，最小化响应时间并避免任何单一资源的过载。使用多个组件进行负载平衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件，例如多层交换机或域名系统服务器进程。

5：什么是Hystrix？它如何实现容错？

Hystrix是一个延迟和容错库，旨在隔离远程系统，服务和第三方库的访问点，当出现故障是不可避免的故障时，停止级联故障并在复杂的分布式系统中实现弹性。

通常对于使用微服务架构开发的系统，涉及到许多微服务。这些微服务彼此协作。

思考以下微服务



假设如果上图中的微服务9失败了，那么使用传统方法我们将传播一个异常。但这仍然会导致整个系统崩溃。

随着微服务数量的增加，这个问题变得更加复杂。微服务的数量可以高达1000.这是hystrix出现的地方。我们将使用Hystrix在这种情况下的Fallback方法功能。我们有两个服务employee-consumer使用由employee-consumer公开的服务。

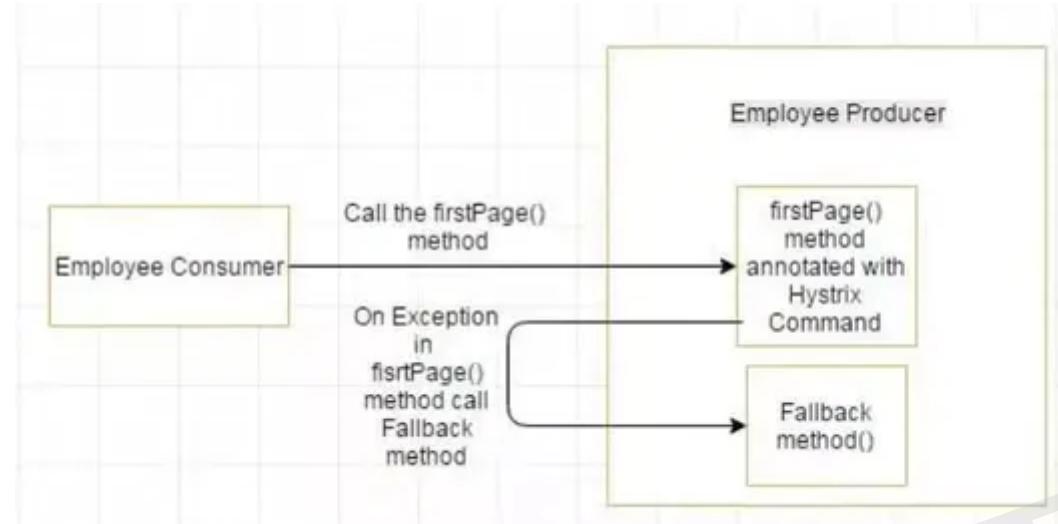
简化图如下所示



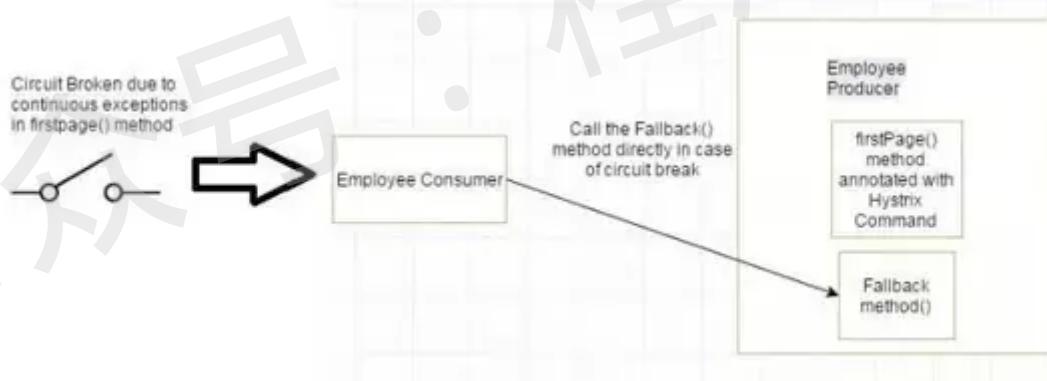
现在假设由于某种原因，employee-producer公开的服务会抛出异常。我们在这种情况下使用Hystrix定义了一个回退方法。这种后备方法应该具有与公开服务相同的返回类型。如果暴露服务中出现异常，则回退方法将返回一些值。

6: 什么是Hystrix断路器？我们需要它吗？

由于某些原因，employee-consumer公开服务会引发异常。在这种情况下使用Hystrix我们定义了一个回退方法。如果在公开服务中发生异常，则回退方法返回一些默认值。



如果firstPage method() 中的异常继续发生，则Hystrix电路将中断，并且员工使用者将一起跳过firstPage方法，并直接调用回退方法。断路器的目的是给第一页方法或第一页方法可能调用的其他方法留出时间，并导致异常恢复。可能发生的情况是，在负载较小的情况下，导致异常的问题有更好的恢复机会。



问题七：什么是Netflix Feign？它的优点是什么？

Feign是受到Retrofit, JAXRS-2.0和WebSocket启发的java客户端联编程序。Feign的第一个目标是将约束分母的复杂性统一到http apis，而不考虑其稳定性。在employee-consumer的例子中，我们使用了employee-producer使用REST模板公开的REST服务。

但是我们必须编写大量代码才能执行以下步骤

1. 使用功能区进行负载平衡。
2. 获取服务实例，然后获取基本URL。
3. 利用REST模板来使用服务。前面的代码如下

```
@Controller  
public class ConsumerControllerClient {  
    @Autowired
```

```

private LoadBalancerClient loadBalancer;
public void getEmployee() throws RestClientException, IOException {
    ServiceInstance serviceInstance=loadBalancer.choose("employee-
producer");
    System.out.println(serviceInstance.getUri());
    String baseUrl=serviceInstance.getUri().toString();
    baseUrl=baseUrl+"/employee";
    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<String> response=null;
    try{
        response=restTemplate.exchange(baseUrl,
            HttpMethod.GET, getHeaders(),String.class);
    }
    catch (Exception ex)
    {
        System.out.println(ex);
    }
    System.out.println(response.getBody());
}

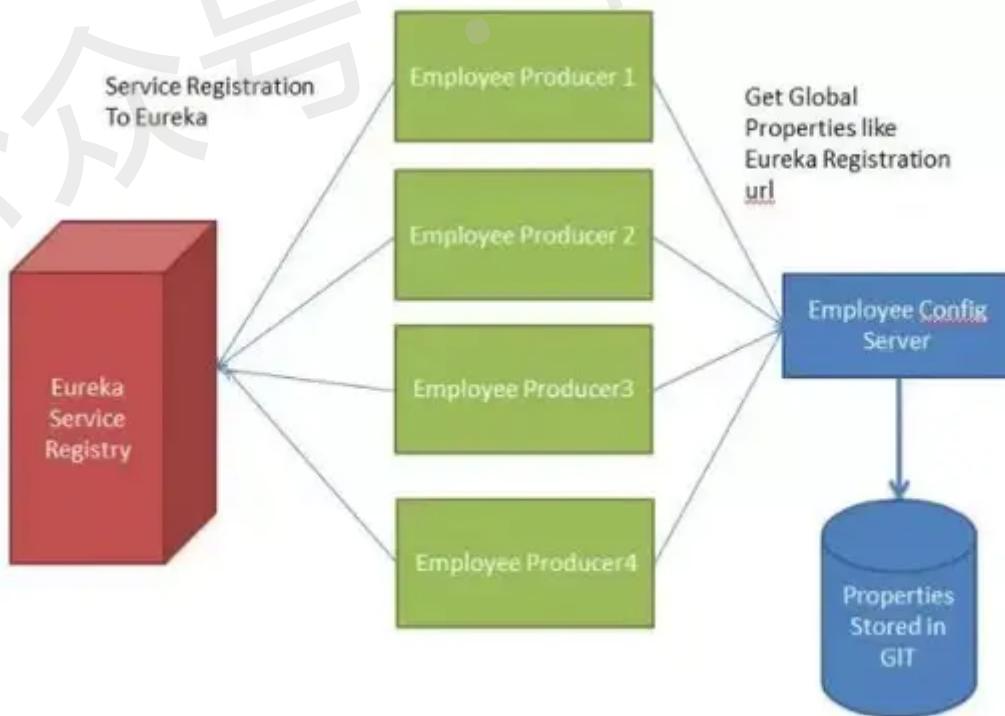
```

之前的代码，有像NullPointerException这样的例外的机会，并不是最优的。我们将看到如何使用Netflix Feign使呼叫变得更加轻松和清洁。如果Netflix Ribbon依赖关系也在类路径中，那么Feign默认也会负责负载平衡。

问题八：什么是Spring Cloud Bus？我们需要它吗？

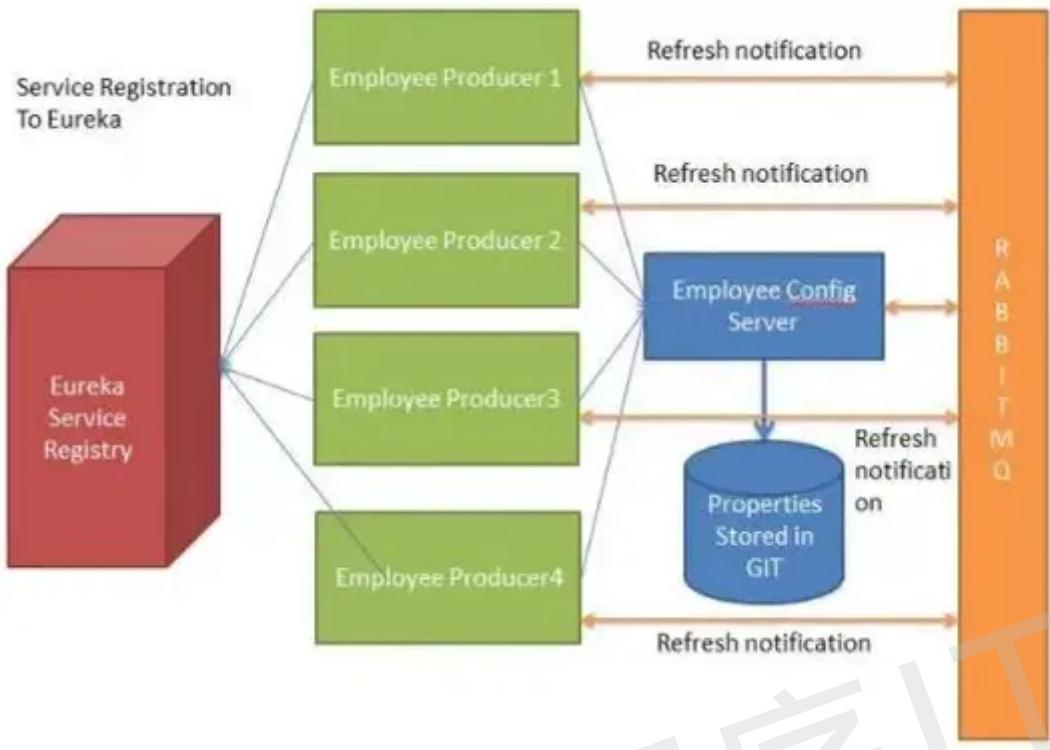
考虑以下情况：我们有多个应用程序使用Spring Cloud Config读取属性，而Spring Cloud Config从GIT读取这些属性。

下面的例子中多个员工生产者模块从Employee Config Module获取Eureka注册的财产。



如果假设GIT中的Eureka注册属性更改为指向另一台Eureka服务器，会发生什么情况。在这种情况下，我们将不得不重新启动服务以获取更新的属性。

还有另一种使用执行器端点/刷新的方式。但是我们将不得不为每个模块单独调用这个url。例如，如果Employee Producer1部署在端口8080上，则调用 `http://localhost:8080/refresh`。同样对于Employee Producer2 `http://localhost:8081/refresh`等等。这又很麻烦。这就是Spring Cloud Bus发挥作用的地方。



Spring Cloud Bus提供了跨多个实例刷新配置的功能。因此，在上面的示例中，如果我们刷新Employee Producer1，则会自动刷新所有其他必需的模块。如果我们有多个微服务启动并运行，这特别有用。这是通过将所有微服务连接到单个消息代理来实现的。无论何时刷新实例，此事件都会订阅到侦听此代理的所有微服务，并且它们也会刷新。可以通过使用端点/总线/刷新来实现对任何单个实例的刷新。

9. spring cloud 断路器的作用是什么？

在分布式架构中，断路器模式的作用也是类似的，当某个服务单元发生故障（类似用电器发生短路）之后，通过断路器的故障监控（类似熔断保险丝），向调用方返回一个错误响应，而不是长时间的等待。这样就不会使得线程因调用故障服务被长时间占用不释放，避免了故障在分布式系统中的蔓延。

10. spring cloud 的核心组件有哪些？

- Eureka：服务注册于发现。
- Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。
- Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。
- Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。
- Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

11. SpringCloud和Dubbo

SpringCloud和Dubbo都是现在主流的微服务架构

SpringCloud是Apache旗下的Spring体系下的微服务解决方案

Dubbo是阿里系的分布式服务治理框架

从技术维度上,其实SpringCloud远远的超过Dubbo,Dubbo本身只是实现了服务治理,而SpringCloud现

在以及有21个子项目以后还会更多
所以其实很多人都会说Dubbo和SpringCloud是不公平的
但是由于RPC以及注册中心元数据等原因,在技术选型的时候我们只能二者选其一,所以我们常常为用他俩来对比
服务的调用方式Dubbo使用的是RPC远程调用,而SpringCloud使用的是 Rest API,其实更符合微服务官方的定义
服务的注册中心来看,Dubbo使用了第三方的ZooKeeper作为其底层的注册中心,实现服务的注册和发现, SpringCloud使用Spring Cloud Netflix Eureka实现注册中心,当然SpringCloud也可以使用ZooKeeper实现,但一般我们不会这样做
服务网关,Dubbo并没有本身的实现,只能通过其他第三方技术的整合,而SpringCloud有Zuul路由网关,作为路由服务器,进行消费者的请求分发, SpringCloud还支持断路器,与git完美集成分布式配置文件支持版本控制,事务总线实现配置文件的更新与服务自动装配等等一系列的微服务架构要素

从技术选型上讲~

目前国内的分布式系统选型主要还是Dubbo毕竟国产,而且国内工程师的技术熟练程度高,并且Dubbo在其他维度上的缺陷可以由其他第三方框架进行集成进行弥补
而SpringCloud目前是国外比较流行,当然我觉得国内的市场也会慢慢的偏向SpringCloud,就连刘军作为Dubbo重启的负责人也发表过观点,Dubbo的发展方向是积极适应SpringCloud生态,并不是起冲突

Rest和RPC对比

其实如果仔细阅读过微服务提出者马丁福勒的论文的话可以发现其定义的服务间通信机制就是Http
Rest

RPC最主要的缺陷就是服务提供方和调用方式之间依赖太强,我们需要为每一个微服务进行接口的定义,并通过持续继承发布,需要严格的版本控制才不会出现服务提供和调用之间因为版本不同而产生的冲突
而REST是轻量级的接口,服务的提供和调用不存在代码之间的耦合,只是通过一个约定进行规范,但也有可能出现文档和接口不一致而导致的服务集成问题,但可以通过swagger工具整合,是代码和文档一体化解决,所以REST在分布式环境下比RPC更加灵活

这也是为什么当当网的DubboX在对Dubbo的增强中增加了对REST的支持的原因

文档质量和社区活跃度

SpringCloud社区活跃度远高于Dubbo,毕竟由于梁飞团队的原因导致Dubbo停止更新迭代五年,而中小型公司无法承担技术开发的成本导致Dubbo社区严重低落,而SpringCloud异军突起,迅速占领了微服务的市场,背靠Spring混的风生水起

Dubbo经过多年的积累文档相当成熟,对于微服务的架构体系各个公司也有稳定的现状

12. SpringBoot和SpringCloud

SpringBoot是Spring推出用于解决传统框架配置文件冗余,装配组件繁杂的基于Maven的解决方案,旨在快速搭建单个微服务

而SpringCloud专注于解决各个微服务之间的协调与配置,服务之间的通信,熔断,负载均衡等技术维度并相同,并且SpringCloud是依赖于SpringBoot的,而SpringBoot并不是依赖与SpringCloud,甚至还可以和Dubbo进行优秀的整合开发

总结:

- SpringBoot专注于快速方便的开发单个个体的微服务
- SpringCloud是关注全局的微服务协调整理治理框架,整合并管理各个微服务,为各个微服务之间提供,配置管理,服务发现,断路器,路由,事件总线等集成服务
- SpringBoot不依赖于SpringCloud, SpringCloud依赖于SpringBoot, 属于依赖关系
- SpringBoot专注于快速,方便的开发单个的微服务个体, SpringCloud关注全局的服务治理框架

13. 微服务之间是如何独立通讯的

1. 远程过程调用 (Remote Procedure Invocation) :

也就是我们常说的服务的注册与发现

直接通过远程过程调用来访问别的service。

优点:

简单, 常见, 因为没有中间件代理, 系统更简单

缺点:

只支持请求/响应的模式, 不支持别的, 比如通知、请求/异步响应、发布/订阅、发布/异步响应

降低了可用性, 因为客户端和服务端在请求过程中必须都是可用的

2. 消息:

使用异步消息来做服务间通信。服务间通过消息管道来交换消息, 从而通信。

优点:

把客户端和服务端解耦, 更松耦合

提高可用性, 因为消息中间件缓存了消息, 直到消费者可以消费

支持很多通信机制比如通知、请求/异步响应、发布/订阅、发布/异步响应

缺点:

消息中间件有额外的复杂

14. 负载均衡的意义是什么?

在计算中, 负载均衡可以改善跨计算机, 计算机集群, 网络链接, 中央处理单元或磁盘驱动器等多种计算资源的工作负载分布。负载均衡旨在优化资源使用, 最大吞吐量, 最小响应时间并避免任何单一资源的过载。使用多个组件进行负载均衡而不是单个组件可能会通过冗余来提高可靠性和可用性。负载平衡通常涉及专用软件或硬件, 例如多层交换机或域名系统服务进程。

15. springcloud如何实现服务的注册?

1. 服务发布时, 指定对应的服务名, 将服务注册到 注册中心(eureka zookeeper)

2. 注册中心加@EnableEurekaServer, 服务用@EnableDiscoveryClient, 然后用ribbon或feign进行服务直接的调用发现。

16. 什么是服务熔断?什么是服务降级

在复杂的分布式系统中, 微服务之间的相互调用, 有可能出现各种各样的原因导致服务的阻塞, 在高并发场景下, 服务的阻塞意味着线程的阻塞, 导致当前线程不可用, 服务器的线程全部阻塞, 导致服务器崩溃, 由于服务之间的调用关系是同步的, 会对整个微服务系统造成服务雪崩

为了解决某个微服务的调用响应时间过长或者不可用进而占用越来越多的系统资源引起雪崩效应就需要进行服务熔断和服务降级处理。

所谓的服务熔断指的是某个服务故障或异常一起类似显示世界中的“保险丝”当某个异常条件被触发就直接熔断整个服务, 而不是一直等到此服务超时。

服务熔断就是相当于我们电闸的保险丝, 一旦发生服务雪崩的, 就会熔断整个服务, 通过维护一个自己的线程池, 当线程达到阈值的时候就启动服务降级, 如果其他请求继续访问就直接返回fallback的默认值

17. 微服务的优缺点分别是什么?说下你在项目开发中碰到的坑

优点

- 每一个服务足够内聚,代码容易理解
- 开发效率提高,一个服务只做一件事
- 微服务能够被小团队单独开发
- 微服务是松耦合的,是有功能意义的服务
- 可以用不同的语言开发,面向接口编程
- 易于与第三方集成
- 微服务只是业务逻辑的代码,不会和HTML,CSS或者其他界面组合

开发中,两种开发模式

前后端分离
全栈工程师

- 可以灵活搭配,连接公共库/连接独立库

缺点

- 分布式系统的责任性
- 多服务运维难度,随着服务的增加,运维的压力也在增大
- 系统部署依赖
- 服务间通信成本
- 数据一致性
- 系统集成测试
- 性能监控

18. 你所知道的微服务技术栈?

- 维度(springcloud)
- 服务开发: springboot spring springmvc
- 服务配置与管理:Netflix公司的Archaius, 阿里的Diamond
- 服务注册与发现:Eureka, Zookeeper
- 服务调用:Rest RPC gRpc
- 服务熔断器:Hystrix
- 服务负载均衡:Ribbon Nginx
- 服务接口调用:Fegin
- 消息队列:Kafka Rabbitmq activemq
- 服务配置中心管理:SpringCloudConfig
- 服务路由 (API网关) Zuul
- 事件消息总线:SpringCloud Bus

19. Eureka和ZooKeeper都可以提供服务注册与发现的功能,请说说两个的区别

1.ZooKeeper保证的是CP,Eureka保证的是AP

ZooKeeper在选举期间注册服务瘫痪,虽然服务最终会恢复,但是选举期间不可用的

Eureka各个节点是平等关系,只要有一台Eureka就可以保证服务可用,而查询到的数据并不是最新的

自我保护机制会导致

Eureka不再从注册列表移除因长时间没收到心跳而应该过期的服务

Eureka仍然能够接受新服务的注册和查询请求,但是不会被同步到其他节点(高可用)

当网络稳定时,当前实例新的注册信息会被同步到其他节点中(最终一致性)

Eureka可以很好的应对因网络故障导致部分节点失去联系的情况,而不会像ZooKeeper一样使得整个注册系统瘫痪

2.ZooKeeper有Leader和Follower角色,Eureka各个节点平等

3.ZooKeeper采用过半数存活原则,Eureka采用自我保护机制解决分区问题

4.Eureka本质上是一个工程,而ZooKeeper只是一个进程

20. eureka自我保护机制是什么?

当Eureka Server 节点在短时间内丢失了过多实例的连接时（比如网络故障或频繁启动关闭客户端）节点会进入自我保护模式，保护注册信息，不再删除注册数据，故障恢复时，自动退出自我保护模式。

21 什么是Ribbon?

ribbon是一个负载均衡客户端，可以很好的控制http和tcp的一些行为。feign默认集成了ribbon。

22. 什么是feign? 它的优点是什么?

1.feign采用的是基于接口的注解

2.feign整合了ribbon，具有负载均衡的能力

3.整合了Hystrix，具有熔断的能力

使用:

1.添加pom依赖。

2.启动类添加@EnableFeignClients

3.定义一个接口@FeignClient(name="xxx")指定调用哪个服务

23. Ribbon和Feign的区别?

1.Ribbon都是调用其他服务的，但方式不同。

2.启动类注解不同，Ribbon是@RibbonClient feign的是@EnableFeignClients

3.服务指定的位置不同，Ribbon是在@RibbonClient注解上声明，Feign则是在定义抽象方法的接口中使用@FeignClient声明。

4.调用方式不同，Ribbon需要自己构建http请求，模拟http请求然后使用RestTemplate发送给其他服务，步骤相当繁琐。Feign需要将调用的方法定义成抽象方法即可。

24. 什么是Spring Cloud Bus?

spring cloud bus 将分布式的节点用轻量的消息代理连接起来，它可以用于广播配置文件的更改或者服务直接的通讯，也可用于监控。

如果修改了配置文件，发送一次请求，所有的客户端便会重新读取配置文件。

使用:

1.添加依赖

2.配置rabbitmq

25. 什么是Hystrix?

防雪崩利器，具备服务降级，服务熔断，依赖隔离，监控（Hystrix Dashboard）

服务降级:

双十一 提示 哎哟喂，被挤爆了。app秒杀 网络开小差了，请稍后再试。

优先核心服务，非核心服务不可用或弱可用。通过HystrixCommand注解指定。

fallbackMethod(回退函数)中具体实现降级逻辑。

26. springcloud断路器作用?

当一个服务调用另一个服务由于网络原因或自身原因出现问题，调用者就会等待被调用者的响应 当更多的服务请求到这些资源导致更多的请求等待，发生连锁效应（雪崩效应）

断路器有完全打开状态：一段时间内 达到一定的次数无法调用 并且多次监测没有恢复的迹象 断路器完全打开 那么下次请求就不会请求到该服务

半开：短时间内 有恢复迹象 断路器会将部分请求发给该服务，正常调用时 断路器关闭

关闭：当服务一直处于正常状态 能正常调用

27. 什么是SpringCloudConfig？

在分布式系统中，由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以需要分布式配置中心组件。在Spring Cloud中，有分布式配置中心组件spring cloud config，它支持配置服务放在配置服务的内存中（即本地），也支持放在远程Git仓库中。在spring cloud config 组件中，分两个角色，一是config server，二是config client。

使用：

- 1、添加pom依赖
- 2、配置文件添加相关配置
- 3、启动类添加注解@EnableConfigServer

28. 架构？

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统

在Spring Cloud微服务系统中，一种常见的负载均衡方式是，客户端的请求首先经过负载均衡（zuul、Nginx），再到达服务网关（zuul集群），然后再到具体的服。，服务统一注册到高可用的服务注册中心集群，服务的所有的配置文件由配置服务管理，配置服务的配置文件放在git仓库，方便开发人员随时改配置。

最全 50 道 Redis 面试题及答案

1. 什么是Redis？

Redis本质上是一个Key-Value类型的内存数据库，很像memcached，整个数据库统统加载在内存当中进行操作，定期通过异步操作把数据库数据flush到硬盘上进行保存。因为是纯内存操作，Redis的性能非常出色，每秒可以处理超过 10万次读写操作，是已知性能最快的Key-Value DB。

Redis的出色之处不仅仅是性能，Redis最大的魅力是支持保存多种数据结构，此外单个value的最大限制是1GB，不像 memcached只能保存1MB的数据，因此Redis可以用来实现很多有用的功能，比方说用他的List来做FIFO双向链表，实现一个轻量级的高性能消息队列服务，用他的Set可以做高性能的tag系统等等。另外Redis也可以对存入的Key-Value设置expire时间，因此也可以被当作一个功能加强版的memcached来用。

Redis的主要缺点是数据库容量受到物理内存的限制，不能用作海量数据的高性能读写，因此Redis适合的场景主要局限在较小数据量的高性能操作和运算上。

2. Redis相比memcached有哪些优势？

(1) memcached所有的值均是简单的字符串，redis作为其替代者，支持更为丰富的数据类型

(2) redis的速度比memcached快很多

(3) redis可以持久化其数据

3、Redis支持哪几种数据类型？

String、List、Set、Sorted Set、hashes

4、Redis主要消耗什么物理资源？

内存。

5、Redis的全称是什么？

Remote Dictionary Server。

6、Redis有哪几种数据淘汰策略？

noeviction: 返回错误当内存限制达到并且客户端尝试执行会让更多内存被使用的命令（大部分的写入指令，但DEL和几个例外）

allkeys-lru: 尝试回收最少使用的键（LRU），使得新添加的数据有空间存放。

volatile-lru: 尝试回收最少使用的键（LRU），但仅限于在过期集合的键，使得新添加的数据有空间存放。

allkeys-random: 回收随机的键使得新添加的数据有空间存放。

volatile-random: 回收随机的键使得新添加的数据有空间存放，但仅限于在过期集合的键。

volatile-ttl: 回收在过期集合的键，并且优先回收存活时间（TTL）较短的键，使得新添加的数据有空间存放。

7、Redis官方为什么不提供Windows版本？

因为目前Linux版本已经相当稳定，而且用户量很大，无需开发windows版本，反而会带来兼容性等问题。

8、一个字符串类型的值能存储最大容量是多少？

512M

9、为什么Redis需要把所有数据放到内存中？

Redis为了达到最快的读写速度将数据都读到内存中，并通过异步的方式将数据写入磁盘。所以redis具有快速和数据持久化的特征。如果不将数据放在内存中，磁盘I/O速度为严重影响redis的性能。在内存越来越便宜的今天，redis将会越来越受欢迎。

如果设置了最大使用的内存，则数据已有记录数达到内存限值后不能继续插入新值。

10、Redis集群方案应该怎么做？都有哪些方案？

1.twemproxy，大概概念是，它类似于一个代理方式，使用方法和普通redis无任何区别，设置好它下属的多个redis实例后，使用时在本需要连接redis的地方改为连接twemproxy，它会以一个代理的身份接收请求并使用一致性hash算法，将请求转接到具体redis，将结果再返回twemproxy。使用方式简便（相对redis只需修改连接端口），对旧项目扩展的首选。问题：twemproxy自身单端口实例的压力，使用一致性hash后，对redis节点数量改变时候的计算值的改变，数据无法自动移动到新的节点。

2.codis，目前用的最多的集群方案，基本和twemproxy一致的效果，但它支持在节点数量改变情况下，旧节点数据可恢复到新hash节点。

3.redis cluster3.0自带的集群，特点在于他的分布式算法不是一致性hash，而是hash槽的概念，以及自身支持节点设置从节点。具体看官方文档介绍。

4.在业务代码层实现，起几个毫无关联的redis实例，在代码层，对key进行hash计算，然后去对应的redis实例操作数据。这种方式对hash层代码要求比较高，考虑部分包括，节点失效后的替代算法方案，数据震荡后的自动脚本恢复，实例的监控，等等。

11、Redis集群方案什么情况下会导致整个集群不可用？

有A, B, C三个节点的集群，在没有复制模型的情况下，如果节点B失败了，那么整个集群就会以为缺少5501-11000这个范围的槽而不可用。

12、MySQL里有2000w数据，redis中只存20w的数据，如何保证redis中的数据都是热点数据？

redis内存数据集大小上升到一定大小的时候，就会施行数据淘汰策略。

13、Redis有哪些适合的场景？

(1) 、会话缓存 (Session Cache)

最常用的一种使用Redis的情景是会话缓存 (session cache)。用Redis缓存会话比其他存储（如Memcached）的优势在于：Redis提供持久化。当维护一个不是严格要求一致性的缓存时，如果用户的购物车信息全部丢失，大部分人都会不高兴的，现在，他们还会这样吗？

幸运的是，随着Redis这些年的改进，很容易找到怎么恰当的使用Redis来缓存会话的文档。甚至广为人知的商业平台Magento也提供Redis的插件。

(2) 、全页缓存 (FPC)

除基本的会话token之外，Redis还提供很简便的FPC平台。回到一致性问题，即使重启了Redis实例，因为有磁盘的持久化，用户也不会看到页面加载速度的下降，这是一个极大改进，类似PHP本地FPC。

再次以Magento为例，Magento提供一个插件来使用Redis作为全页缓存后端。

此外，对WordPress的用户来说，Pantheon有一个非常好的插件 wp-redis，这个插件能帮助你以最快的速度加载你曾浏览过的页面。

(3) 、队列

Redis在内存存储引擎领域的一大优点是提供list 和 set 操作，这使得Redis能作为一个很好的消息队列平台来使用。Redis作为队列使用的操作，就类似于本地程序语言（如Python）对list的push/pop操作。

如果你快速的在Google中搜索“Redis queues”，你马上就能找到大量的开源项目，这些项目的目的就是利用Redis创建非常好的后端工具，以满足各种队列需求。例如，Celery有一个后台就是使用Redis作为broker，你可以从这里去查看。

(4) , 排行榜/计数器

Redis在内存中对数字进行递增或递减的操作实现的非常好。集合 (Set) 和有序集合 (Sorted Set) 也使得我们在执行这些操作的时候变的非常简单，Redis只是正好提供了这两种数据结构。所以，我们要从排序集合中获取到排名最靠前的10个用户-我们称之为“user_scores”，我们只需要像下面一样执行即可：

当然，这是假定你是根据你用户的分数做递增的排序。如果你想返回用户及用户的分数，你需要这样执行：

```
ZRANGE user_scores 0 10 WITHSCORES
```

Agora Games就是一个很好的例子，用Ruby实现的，它的排行榜就是使用Redis来存储数据的，你可以在这里看到。

(5) 、发布/订阅

最后（但肯定不是最不重要的）是Redis的发布/订阅功能。发布/订阅的使用场景确实非常多。我已看见人们在社交网络连接中使用，还可作为基于发布/订阅的脚本触发器，甚至用Redis的发布/订阅功能来建立聊天系统！（不，这是真的，你可以去核实）。

14、Redis支持的Java客户端都有哪些？官方推荐用哪个？

Redisson、Jedis、lettuce等等，官方推荐使用Redisson。

15、Redis和Redisson有什么关系？

Redisson是一个高级的分布式协调Redis客服端，能帮助用户在分布式环境中轻松实现一些Java的对象（Bloom filter, BitSet, Set, SetMultimap, ScoredSortedSet, SortedSet, Map, ConcurrentMap, List, ListMultimap, Queue, BlockingQueue, Deque, BlockingDeque, Semaphore, Lock, ReadWriteLock, AtomicLong, CountDownLatch, Publish / Subscribe, HyperLogLog）。

16、Jedis与Redisson对比有什么优缺点？

Jedis是Redis的Java实现的客户端，其API提供了比较全面的Redis命令的支持；Redisson实现了分布式和可扩展的Java数据结构，和Jedis相比，功能较为简单，不支持字符串操作，不支持排序、事务、管道、分区等Redis特性。Redisson的宗旨是促进使用者对Redis的关注分离，从而让使用者能够将精力更集中地放在处理业务逻辑上。

17、Redis如何设置密码及验证密码？

设置密码：config set requirepass 123456

授权密码：auth 123456

18、说说Redis哈希槽的概念？

Redis集群没有使用一致性hash,而是引入了哈希槽的概念，Redis集群有16384个哈希槽，每个key通过CRC16校验后对16384取模来决定放置哪个槽，集群的每个节点负责一部分hash槽。

19、Redis集群的主从复制模型是怎样的？

为了使在部分节点失败或者大部分节点无法通信的情况下集群仍然可用，所以集群使用了主从复制模型，每个节点都会有N-1个复制品。

20、Redis集群会有写操作丢失吗？为什么？

Redis并不能保证数据的强一致性，这意味着在实际中集群在特定的条件下可能会丢失写操作。

21、Redis集群之间是如何复制的？

异步复制

22、Redis集群最大节点个数是多少？

16384个。

23、Redis集群如何选择数据库？

Redis集群目前无法做数据库选择， 默认在0数据库。

24、怎么测试Redis的连通性？

ping

25、Redis中的管道有什么用？

一次请求/响应服务器能实现处理新的请求即使旧的请求还未被响应。这样就可以将多个命令发送到服务器，而不用等待回复，最后在一个步骤中读取该答复。

这就是管道（pipelining），是一种几十年来广泛使用的技术。例如许多POP3协议已经实现支持这个功能，大大加快了从服务器下载新邮件的过程。

26、怎么理解Redis事务？

事务是一个单独的隔离操作：事务中的所有命令都会序列化、按顺序地执行。事务在执行的过程中，不会被其他客户端发送来的命令请求所打断。

事务是一个原子操作：事务中的命令要么全部被执行，要么全部都不执行。

27、Redis事务相关的命令有哪几个？

MULTI、EXEC、DISCARD、WATCH

28、Redis key的过期时间和永久有效分别怎么设置？

EXPIRE和PERSIST命令。

29、Redis如何做内存优化？

尽可能使用散列表（hashes），散列表（是说散列表里面存储的数少）使用的内存非常小，所以你应该尽可能的将你的数据模型抽象到一个散列表里面。比如你的web系统中有一个用户对象，不要为这个用户的名称，姓氏，邮箱，密码设置单独的key,而是应该把这个用户的所有信息存储到一张散列表里面.

30、Redis回收进程如何工作的？

1. 一个客户端运行了新的命令，添加了新的数据。
2. Redis检查内存使用情况，如果大于maxmemory的限制，则根据设定好的策略进行回收。
3. 一个新的命令被执行，等等。
4. 所以我们不断地穿越内存限制的边界，通过不断达到边界然后不断地收回回到边界以下。

如果一个命令的结果导致大量内存被使用（例如很大的集合的交集保存到一个新的键），不用多久内存限制就会被这个内存使用量超越。

31、Redis回收使用的是什么算法？

LRU算法

32、Redis如何做大量数据插入？

Redis2.6开始redis-cli支持一种新的被称之为pipe mode的新模式用于执行大量数据插入工作。

33、为什么要做Redis分区？

分区可以让Redis管理更大的内存，Redis将可以使用所有机器的内存。如果没有分区，你最多只能使用一台机器的内存。分区使Redis的计算能力通过简单地增加计算机得到成倍提升，Redis的网络带宽也会随着计算机和网卡的增加而成倍增长。

34、你知道有哪些Redis分区实现方案？

- 客户端分区就是在客户端就已经决定数据会被存储到哪个redis节点或者从哪个redis节点读取。大多数客户端已经实现了客户端分区。
- 代理分区意味着客户端将请求发送给代理，然后代理决定去哪个节点写数据或者读数据。代理根据分区规则决定请求哪些Redis实例，然后根据Redis的响应结果返回给客户端。redis和memcached的一种代理实现就是Twemproxy
- 查询路由(Query routing)的意思是客户端随机地请求任意一个redis实例，然后由Redis将请求转发给正确的Redis节点。Redis Cluster实现了一种混合形式的查询路由，但并不是直接将请求从一个redis节点转发到另一个redis节点，而是在客户端的帮助下直接redirected到正确的redis节点。

35、Redis分区有什么缺点？

- 涉及多个key的操作通常不会被支持。例如你不能对两个集合求交集，因为他们可能被存储到不同的Redis实例（实际上这种情况也有办法，但是不能直接使用交集指令）。
- 同时操作多个key，则不能使用Redis事务。
- 分区使用的粒度是key，不能使用一个非常长的排序key存储一个数据集（The partitioning granularity is the key, so it is not possible to shard a dataset with a single huge key like a very big sorted set）。
- 当使用分区的时候，数据处理会非常复杂，例如为了备份你必须从不同的Redis实例和主机同时收集RDB / AOF文件。
- 分区时动态扩容或缩容可能非常复杂。Redis集群在运行时增加或者删除Redis节点，能做到最大程度对用户透明地数据再平衡，但其他一些客户端分区或者代理分区方法则不支持这种特性。然而，有一种预分片的技术也可以较好的解决这个问题。

36、Redis持久化数据和缓存怎么做扩容？

- 如果Redis被当做缓存使用，使用一致性哈希实现动态扩容缩容。
- 如果Redis被做一个持久化存储使用，必须使用固定的keys-to-nodes映射关系，节点的数量一旦确定不能变化。否则的话（即Redis节点需要动态变化的情况），必须使用可以在运行时进行数据再平衡的一套系统，而当前只有Redis集群可以做到这样。

37、分布式Redis是前期做还是后期规模上来了再做好？为什么？

既然Redis是如此的轻量（单实例只使用1M内存），为防止以后的扩容，最好的办法就是一开始就启动较多实例。即便你只有一台服务器，你也可以一开始就让Redis以分布式的方式运行，使用分区，在同一台服务器上启动多个实例。

一开始就多设置几个Redis实例，例如32或者64个实例，对大多数用户来说这操作起来可能比较麻烦，但是从长久来看做这点牺牲是值得的。

这样的话，当你的数据不断增长，需要更多的Redis服务器时，你需要做的就是仅仅将Redis实例从一台服务迁移到另外一台服务器而已（而不用考虑重新分区的问题）。一旦你添加了另一台服务器，你需要将你一半的Redis实例从第一台机器迁移到第二台机器。

38、Twemproxy是什么？

Twemproxy是Twitter维护的（缓存）代理系统，代理Memcached的ASCII协议和Redis协议。它是单线程程序，使用c语言编写，运行起来非常快。它是采用Apache 2.0 license的开源软件。Twemproxy支持自动分区，如果其代理的其中一个Redis节点不可用时，会自动将该节点排除（这将改变原来的key-instances的映射关系，所以你应该仅在把Redis当缓存时使用Twemproxy）。Twemproxy本身不存在单点问题，因为你可以启动多个Twemproxy实例，然后让你的客户端去连接任意一个Twemproxy实例。Twemproxy是Redis客户端和服务器端的一个中间层，由它来处理分区功能应该不算复杂，并且应该算比较可靠的。

39、支持一致性哈希的客户端有哪些？

Redis-rb、Predis等。

40、Redis与其他key-value存储有什么不同？

1. Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
2. Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，应为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

41、Redis的内存占用情况怎么样？

给你举个例子：100万个键值对（键是0到999999值是字符串“hello world”）在我的32位的Mac笔记本上用了100MB。同样的数据放到一个key里只需要16MB，这是因为键值有一个很大的开销。在Memcached上执行也是类似的结果，但是相对Redis的开销要小一点点，因为Redis会记录类型信息引用计数等等。

当然，大键值对时两者比例要好很多。

64位的系统比32位的需要更多的内存开销，尤其是键值对都较小时，这是因为64位的系统里指针占用了8个字节。但是，当然，64位系统支持更大的内存，所以为了运行大型的Redis服务器或多或少的需要使用64位的系统。

42、都有哪些办法可以降低Redis的内存使用情况呢？

如果你使用的是32位的Redis实例，可以好好利用Hash,list,sorted set,set等集合类型数据，因为通常情况下很多小的Key-Value可以用更紧凑的方式存放到一起。

43、查看Redis使用情况及状态信息用什么命令？

info

44、Redis的内存用完了会发生什么？

如果达到设置的上限，Redis的写命令会返回错误信息（但是读命令还可以正常返回。）或者你可以将Redis当缓存来使用配置淘汰机制，当Redis达到内存上限时会冲刷掉旧的内容。

45、Redis是单线程的，如何提高多核CPU的利用率？

可以在同一个服务器部署多个Redis的实例，并把他们当作不同的服务器来使用，在某些时候，无论任何一个服务器是不够的，所以，如果你想使用多个CPU，你可以考虑一下分片（shard）。

46、一个Redis实例最多能存放多少的keys？List、Set、Sorted Set他们最多能存放多少元素？

理论上Redis可以处理多达232的keys，并且在实际中进行了测试，每个实例至少存放了2亿5千万的keys。我们正在测试一些较大的值。

任何list、set、和sorted set都可以放232个元素。

换句话说，Redis的存储极限是系统中的可用内存值。

47、Redis常见性能问题和解决方案？

- (1) Master最好不要做任何持久化工作，如RDB内存快照和AOF日志文件
- (2) 如果数据比较重要，某个Slave开启AOF备份数据，策略设置为每秒同步一次
- (3) 为了主从复制的速度和连接的稳定性，Master和Slave最好在同一个局域网内
- (4) 尽量避免在压力很大的主库上增加从库
- (5) 主从复制不要用图状结构，用单向链表结构更为稳定，即：Master <- Slave1 <- Slave2 <- Slave3...

这样的结构方便解决单点故障问题，实现Slave对Master的替换。如果Master挂了，可以立刻启用Slave1做Master，其他不变。

48、Redis提供了哪几种持久化方式？

1. RDB持久化方式能够在指定的时间间隔能对你的数据进行快照存储。
2. AOF持久化方式记录每次对服务器写的操作，当服务器重启的时候会重新执行这些命令来恢复原始的数据，AOF命令以redis协议追加保存每次写的操作到文件末尾。Redis还能对AOF文件进行后台重写，使得AOF文件的体积不至于过大。
3. 如果你只希望你的数据在服务器运行的时候存在，你也可以不使用任何持久化方式。
4. 你也可以同时开启两种持久化方式，在这种情况下，当redis重启的时候会优先载入AOF文件来恢复原始的数据，因为在通常情况下AOF文件保存的数据集要比RDB文件保存的数据集要完整。
5. 最重要的事情是了解RDB和AOF持久化方式的不同，让我们以RDB持久化方式开始。

49、如何选择合适的持久化方式？

一般来说，如果想达到足以媲美PostgreSQL的数据安全性，你应该同时使用两种持久化功能。如果你非常关心你的数据，但仍然可以承受数分钟以内的数据丢失，那么你可以只使用RDB持久化。

有很多用户都只使用AOF持久化，但并不推荐这种方式：因为定时生成RDB快照(snapshot)非常便于进行数据库备份，并且RDB恢复数据集的速度也要比AOF恢复的速度要快，除此之外，使用RDB还可以避免之前提到的AOF程序的bug。

50、修改配置不重启Redis会实时生效吗？

针对运行实例，有许多配置选项可以通过 CONFIG SET 命令进行修改，而无需执行任何形式的重启。从 Redis 2.2 开始，可以从 AOF 切换到 RDB 的快照持久性或其他方式而不需要重启 Redis。检索 'CONFIG GET *' 命令获取更多信息。

但偶尔重新启动是必须的，如为升级 Redis 程序到新的版本，或者当你需要修改某些目前 CONFIG 命令还不支持的配置参数的时候。

Mybatis 28道面试题及答案

1、什么是Mybatis?

- 1.Mybatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高。
- 2.MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO 映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
- 3.通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

2、Mybatis的优点：

- 1.基于SQL语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在XML里，解除sql与程序代码的耦合，便于统一管理；提供XML标签，支持编写动态SQL语句，并可重用。
- 2.与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；
- 3.很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持）。
- 4.能够与Spring很好的集成；
- 5.提供映射标签，支持对象与数据库的ORM字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

3.MyBatis框架的缺点：

- 1.SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。
- 2.SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

4、MyBatis框架适用场合：

- 1.MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。
- 2.对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis将是不错的选择。

5、MyBatis与Hibernate有哪些不同？

- 1.Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句。
- 2.Mybatis直接编写原生态sql，可以严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频繁，一旦需求变化要求迅速输出成果。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套sql映射文件，工作量大。

3.Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用hibernate开发可以节省很多代码，提高效率。

6、#{ }和\${ }的区别是什么？

{}是预编译处理， \${ }是字符串替换。

Mybatis在处理#{ }时，会将sql中的#{ }替换为?号，调用PreparedStatement的set方法来赋值；

Mybatis在处理\${ }时，就是把\${ }替换成变量的值。

使用#{ }可以有效的防止SQL注入，提高系统安全性。

7、当实体类中的属性名和表中的字段名不一样，怎么办？

第1种：通过在查询的sql语句中定义字段名的别名，让字段名的别名和实体类的属性名一致。

```
<select id = "selectorder" parameterType = "int" resultType = "me.gacl.domain.order">
    select order_id id, order_no orderno ,order_price price from orders where order_id=#{id};
</select>
```

第2种：通过来映射字段名和实体类属性名的一一对应的关系。

```
<select id="getOrder" parameterType="int" resultMap="orderresultmap">
    select * from orders where order_id=#{id}
</select>

<resultMap type="me.gacl.domain.order" id="orderresultmap">
    <!-用id属性来映射主键字段->
    <id property="id" column="order_id">
        <!-用result属性来映射非主键字段，property为实体类属性名，column为数据表中的属性->
        <result property = "orderno" column = "order_no"/>
        <result property="price" column="order_price" />
    </resultMap>
```

8、模糊查询like语句该怎么写？

第1种：在Java代码中添加sql通配符。

```
string wildcardname = "%smi%";
list<name> names = mapper.selectlike(wildcardname);

<select id="selectlike">
    select * from foo where bar like #{value}
</select>
```

第2种：在sql语句中拼接通配符，会引起sql注入

```
string wildcardname = "smi";
list<name> names = mapper.selectlike(wildcardname);

<select id="selectlike">
    select * from foo where bar like "%"#{value}%""
</select>
```

9、通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

Dao接口即Mapper接口。接口的全限名，就是映射文件中的namespace的值；接口的方法名，就是映射文件中Mapper的Statement的id值；接口方法内的参数，就是传递给sql的参数。

Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MapperStatement。在Mybatis中，每一个、、、标签，都会被解析为一个MapperStatement对象。

举例：`com.mybatis3.mappers.StudentDao.findStudentById`，可以唯一找到namespace为`com.mybatis3.mappers.StudentDao`下面 id 为 `findStudentById` 的 MapperStatement。

Mapper接口里的方法，是不能重载的，因为是使用 全限名+方法名 的保存和寻找策略。Mapper接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Mapper接口生成代理对象 proxy，代理对象会拦截接口方法，转而执行MapperStatement所代表的sql，然后将sql执行结果返回。

10、Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页。可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

11、Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式？**

第一种是使用 标签，逐一定义数据库列名和对象属性名之间的映射关系。

第二种是使用sql列的别名功能，将列的别名书写为对象属性名。

有了列名与属性名的映射关系后，Mybatis通过反射创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

12、如何执行批量插入？

首先，创建一个简单的insert语句：

```
<insert id="insertname">
  insert into names (name) values (#{value})
</insert>
```

然后在java代码中像下面这样执行批处理插入:

```
list < string > names = new ArrayList();
names.add("fred");
names.add("barney");
names.add("betty");
names.add("wilma");
// 注意这里 executortype.batch
SqlSession sqlsession = sqlSessionFactory.openSession(executortype.batch);
try {
    NameMapper mapper = sqlsession.getMapper(NameMapper.class);
    for (String name: names) {
        mapper.insertName(name);
    }
    sqlsession.commit();
} catch (Exception e) {
    e.printStackTrace();
    sqlSession.rollback();
    throw e;
}
finally {
    sqlsession.close();
}
```

13、如何获取自动生成的(主)键值?

insert 方法总是返回一个int值，这个值代表的是插入的行数。

如果采用自增长策略，自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。

示例：

```
<insert id="insertname" usegeneratedkeys="true" keyproperty="id">
  insert into names (name) values (#{name})
</insert>
Name name = new Name();
name.setName("fred");

int rows = mapper.insertName(name);
// 完成后，id已经被设置到对象中
System.out.println("rows inserted = " + rows);
System.out.println("generated key value = " + name.getId());
```

14、Mybatis动态sql有什么用？执行原理？有哪些动态sql？

Mybatis动态sql可以在Xml映射文件内，以标签的形式编写动态sql，执行原理是根据表达式的值完成逻辑判断并动态拼接sql的功能。

Mybatis提供了9种动态sql标签：

trim|where|set|foreach|if|choose|when|otherwise|bind。

15、Xml映射文件中，除了常见的select|insert|update|delete标签之外，还有哪些标签？

答：加上动态sql的9个标签，其中为sql片段标签，通过标签引入sql片段，为不支持自增的主键生成策略标签。

16、Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复？

不同的Xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；

原因就是namespace+id是作为Map <String,MapperStatement>的key使用的，如果没有namespace，就剩下id，那么，id重复会导致数据互相覆盖。有了namespace，自然id就可以重复，namespace不同，namespace+id自然也就不同。

17、为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具。

18、MyBatis实现一对一有几种方式？具体怎么操作的？

有联合查询和嵌套查询，联合查询是几个表联合查询，只查询一次，通过在resultMap里面配置association节点配置一对一的类就可以完成；

嵌套查询是先查一个表，根据这个表里面的结果的外键id，去再另外一个表里面查询数据，也是通过association配置，但另外一个表的查询通过select属性配置。

19、MyBatis实现一对多有几种方式，怎么操作的？

有联合查询和嵌套查询。联合查询是几个表联合查询，只查询一次，通过在resultMap里面的collection节点配置一对多的类就可以完成；嵌套查询是先查一个表，根据这个表里面的结果的外键id，去再另外一个表里面查询数据，也是通过配置collection，但另外一个表的查询通过select节点配置。

20、Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

答：Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true | false。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的。

21、Mybatis的一级、二级缓存：

- 1) 一级缓存：基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。
- 2) 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态)，可在它的映射文件中配置；
- 3) 对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存 Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

22、什么是MyBatis的接口绑定？有哪些实现方式？

接口绑定，就是在MyBatis中任意定义接口，然后把接口里面的方法和SQL语句绑定，我们直接调用接口方法就可以，这样比起原来SqlSession提供的方法我们可以有更加灵活的选择和设置。

接口绑定有两种实现方式，一种是通过注解绑定，就是在接口的方法上面加上 @Select、@Update 等注解，里面包含SQL语句来绑定；另外一种就是通过xml里面写SQL来绑定，在这种情况下，要指定xml映射文件里面的namespace必须为接口的全路径名。当SQL语句比较简单时候，用注解绑定，当SQL语句比较复杂时候，用xml绑定，一般用xml绑定的比较多。

23、使用MyBatis的mapper接口调用时有哪些要求？

1. Mapper接口方法名和mapper.xml中定义的每个sql的id相同；
2. Mapper接口方法的输入参数类型和mapper.xml中定义的每个sql 的parameterType的类型相同；
3. Mapper接口方法的输出参数类型和mapper.xml中定义的每个sql的结果resultType的类型相同；
4. Mapper.xml文件中的namespace即是mapper接口的类路径。

24、简述Mybatis的插件运行原理，以及如何编写一个插件。

答：Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

编写插件：实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

MySQL 40题面试题及答案

1、数据库三大范式你能说一下吗？

第一范式: 1NF是对属性的原子性约束，要求字段具有原子性，不可再分解；(只要是关系型数据库都满足1NF)

第二范式: 2NF是在满足第一范式的前提下，非主键字段不能出现部分依赖主键；解决：消除复合主键就可避免出现部分以来，可增加单列关键字。

第三范式: 3NF是在满足第二范式的前提下，非主键字段不能出现传递依赖，比如某个字段a依赖于主键，而一些字段依赖字段a，这就是传递依赖。解决：将一个实体信息的数据放在一个表内实现。

2、sql语句分类：

DDL: 数据定义语言 (create drop)

DML: 数据操作语句 (insert update delete)

DQL: 数据查询语句 (select)

DCL: 数据控制语句，进行授权和权限回收 (grant revoke)

TPL: 数据事务语句 (commit callback savapoint)

3、SQL 的 select 语句完整的执行顺序

- 1、from 子句组装来自不同数据源的数据；
- 2、where 子句基于指定的条件对记录行进行筛选；
- 3、group by 子句将数据划分为多个分组；
- 4、使用聚集函数进行计算；
- 5、使用 having 子句筛选分组；
- 6、计算所有的表达式；
- 7、select 的字段；
- 8、使用 order by 对结果集进行排序。

4、delete、drop、truncate区别

truncate 和 delete只删除数据，不删除表结构 ,drop删除表结构，并且释放所占的空间。

删除数据的速度, **drop > truncate > delete**

delete属于DML语言，需要事务管理，commit之后才能生效。drop和truncate属于DDL语言，操作立刻生效，不可回滚。

使用场合：

当你不再需要该表时，用 drop;

当你仍要保留该表，但要删除所有记录时，用 truncate;

当你要删除部分记录时 (always with a where clause), 用 delete.

注意：对于有主外键关系的表，不能使用truncate而应该使用不带where子句的delete语句，由于truncate不记录在日志中，不能够激活触发器。

5、char和varchar的区别？

`char` 是一种固定长度的字符串类型，（如果数据类型不足固定长度的话，会自动用0补齐）
`varchar` 是一种可变长度的字符串类型，（如果数据类型不足长度的话，自动将长度缩放成我们所输入的数据类型的长度）

6、事务是什么？事务的四大特性是什么？

事务是一组原子性的 SQL 语句，或者说一个独立的工作单元。如果数据库引擎能够成功地对数据库应用该组操作的全部语句，那么就执行该组查询。如果其中任何一条语句因为崩溃或其他原因无法执行，那么所有的语句都不会执行。也就是说，事务内的语句，要么全部执行成功，要么全部执行失败。

原子性：不可分割的操作单元，事务中所有操作，要么全部成功；要么撤回到执行事务之前的状态；

一致性：如果在执行事务之前数据库是一致的，那么在执行事务之后数据库也还是一致的；

隔离性：事务操作之间彼此独立和透明互不影响。事务独立运行。这通常使用锁来实现。一个事务处理后的结果，影响了其他事务，那么其他事务会撤回。事务的100%隔离，需要牺牲速度。

持久性：事务一旦提交，其结果就是永久的。即便发生系统故障，也能恢复。

7、事务的隔离级别，mysql默认的隔离级别是什么？

读未提交(Read uncommitted)，一个事务可以读取另一个未提交事务的数据，最低级别，任何情况都无法保证。

读已提交(Read committed)，一个事务要等另一个事务提交后才能读取数据，可避免脏读的发生。

可重复读(Repeatable read)，就是在开始读取数据（事务开启）时，不再允许修改操作，可避免脏读、不可重复读的发生。

串行(Serializable)，是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

MySQL的默认隔离级别是Repeatable read。

8、索引是什么？它是如何加快查询性能的？

索引是对数据库表中一列或多列的值进行排序的一种数据结构，也就是说索引是一种数据结构。数据库在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中通过二分法等高效率算法一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询性能。类似新华字典的目录，如果没有目录的话，我们想要查找一个汉字的话，就必须检索整本字典，而正因为有了目录，我们只要知道我们所要查找的偏旁或者拼音首字母，就可以快速地定位到我们想要查找汉字的所在页码。

9、MySQL主要的索引类型

普通索引：是最基本的索引，它没有任何限制；

唯一索引：索引列的值必须唯一，但允许有空值。如果是组合索引，则列值的组合必须唯一；

主键索引：是一种特殊的唯一索引，一个表只能有一个主键，不允许有空值；

组合索引：指多个字段上创建的索引，只有在查询条件中使用了创建索引时的第一个字段，索引才会被使用。使用组合索引时遵循最左前缀集合；

全文索引：主要用来查找文本中的关键字，而不是直接与索引中的值相比较，MySQL中MyISAM支持全文索引而InnoDB不支持；

10、说一说什么是外键？它的优缺点是什么？

外键指的是外键约束，目的是保持数据一致性，完整性，控制存储在外键表中的数据，使两张表形成关联，外键只能引用外表中列的值；

优点：由数据库自身保证数据一致性，完整性，更可靠，因为程序很难100%保证数据的完整性，而用外键即使在数据库服务器当机或者出现其他问题的时候，也能够最大限度的保证数据的一致性和完整性。有主外键的数据库设计可以增加ER图的可读性，这点在数据库设计时非常重要。外键在一定程度上说明的业务逻辑，会使设计周到具体全面。

缺点：可以用触发器或应用程序保证数据的完整性；过分强调或者说使用外键会平添开发难度，导致表过多，更改业务困难，扩展困难等问题；不用外键时数据管理简单，操作方便，性能高（导入导出等操作，在insert, update, delete 数据的时候更快）；

11、在什么时候你会选择使用外键，为什么？

在我的业务逻辑非常简单，业务一旦确定不会轻易更改，表结构简单，业务量小的时候我会选择使用外键。因为当不符合以上条件的时候，外键会影响业务的扩展和修改，当数据量庞大时，会严重影响增删改查的效率。

12、说一说你能想到的sql语句优化，至少五种

- 1) 避免select *，将需要查找的字段列出来；
- 2) 使用连接（join）来代替子查询；
- 3) 拆分大的delete或insert语句；
- 4) 使用limit对查询结果的记录进行限定；
- 5) 用 exists 代替 in 是一个好的选择；
- 6) 用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤；
- 7) 不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引尽量避免在where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描；
- 8) 尽量避免在 where 子句中使用 or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描；
- 9) 尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描；

13、MyISAM与InnoDB的区别？

- 1) InnoDB 支持事务； MyISAM 不支持事务；
- 2) InnoDB 支持行级锁； MyISAM 支持表级锁；
- 3) InnoDB 支持 MVCC(多版本并发控制)； MyISAM 不支持；
- 4) InnoDB 支持外键， MyISAM 不支持；
- 5) InnoDB 不支持全文索引； MyISAM 支持；
- 6) InnoDB 不保存表的总行数，执行 select count(*) from table 时需要全表扫描； MyISAM 用一个变量保存表的总行数，查总行数速度很快；
- 7) InnoDB 是聚集索引，数据文件是和索引绑在一起的，必须要有主键，通过主键索引效率很高。辅助索引需要两次查询，先查询到主键，再通过主键查询到数据。主键太大，其他索引也会很大；
- MyISAM 是非聚集索引，数据文件是分离的，索引保存的是数据文件的指针，主键索引和辅助索引是独立的。

14. 索引是个什么样的数据结构呢？

索引的数据结构和具体存储引擎的实现有关,在MySQL中使用较多的索引有Hash索引,B+树索引等,而我们经常使用的InnoDB存储引擎的默认索引实现为:B+树索引.

15. Hash索引和B+树所有有什么区别或者说优劣呢?

首先要知道Hash索引和B+树索引的底层实现原理:

hash索引底层就是hash表,进行查找时,调用一次hash函数就可以获取到相应的键值,之后进行回表查询获得实际数据.B+树底层实现是多路平衡查找树.对于每一次的查询都是从根节点出发,查找到叶子节点方可以获得所查键值,然后根据查询判断是否需要回表查询数据.

那么可以看出他们有以下的不同:

- hash索引进行等值查询更快(一般情况下),但是却无法进行范围查询.

因为在hash索引中经过hash函数建立索引之后,索引的顺序与原顺序无法保持一致,不能支持范围查询.而B+树的所有节点皆遵循(左节点小于父节点,右节点大于父节点,多叉树也类似),天然支持范围.

- hash索引不支持使用索引进行排序,原理同上.
- hash索引不支持模糊查询以及多列索引的最左前缀匹配.原理也是因为hash函数的不可预测.AAAA和AAAAAB的索引没有相关性.
- hash索引任何时候都避免不了回表查询数据,而B+树在符合某些条件(聚簇索引,覆盖索引等)的时候可以只通过索引完成查询.
- hash索引虽然在等值查询上较快,但是不稳定.性能不可预测,当某个键值存在大量重复的时候,发生hash碰撞,此时效率可能极差.而B+树的查询效率比较稳定,对于所有的查询都是从根节点到叶子节点,且树的高度较低.

因此,在大多数情况下,直接选择B+树索引可以获得稳定且较好的查询速度.而不需要使用hash索引.

16. 上面提到了B+树在满足聚簇索引和覆盖索引的时候不需要回表查询数据,什么是聚簇索引?

在B+树的索引中,叶子节点可能存储了当前的key值,也可能存储了当前的key值以及整行的数据,这就是聚簇索引和非聚簇索引.在InnoDB中,只有主键索引是聚簇索引,如果没有主键,则挑选一个唯一键建立聚簇索引.如果没有唯一键,则隐式的生成一个键来建立聚簇索引.

当查询使用聚簇索引时,在对应的叶子节点,可以获取到整行数据,因此不用再次进行回表查询.

17. 非聚簇索引一定会回表查询吗?

不一定,这涉及到查询语句所要求的字段是否全部命中了索引,如果全部命中了索引,那么就不必再进行回表查询.

举个简单的例子,假设我们在员工表的年龄上建立了索引,那么当进行select age from employee where age < 20的查询时,在索引的叶子节点上,已经包含了age信息,不会再次进行回表查询.

18. 在建立索引的时候,都有哪些需要考虑的因素呢?

建立索引的时候一般要考虑到字段的使用频率,经常作为条件进行查询的字段比较适合.如果需要建立联合索引的话,还需要考虑联合索引中的顺序.此外也要考虑其他方面,比如防止过多的所有对表造成太大的压力.这些都和实际的表结构以及查询方式有关.

19. 联合索引是什么?为什么需要注意联合索引中的顺序?

MySQL可以使用多个字段同时建立一个索引,叫做联合索引.在联合索引中,如果想要命中索引,需要按照建立索引时的字段顺序挨个使用,否则无法命中索引.

具体原因为:

MySQL使用索引时需要索引有序,假设现在建立了"name,age,school"的联合索引,那么索引的排序为:先按照name排序,如果name相同,则按照age排序,如果age的值也相等,则按照school进行排序.

当进行查询时,此时索引仅仅按照name严格有序,因此必须首先使用name字段进行等值查询,之后对于匹配到的列而言,其按照age字段严格有序,此时可以使用age字段用做索引查找,,,以此类推.因此在建立联合索引的时候应该注意索引列的顺序,一般情况下,将查询需求频繁或者字段选择性高的列放在前面.此外可以根据特例的查询或者表结构进行单独的调整.

20. 创建的索引有没有被使用到?或者说怎么才可以知道这条语句运行很慢的原因?

MySQL提供了explain命令来查看语句的执行计划,MySQL在执行某个语句之前,会将该语句过一遍查询优化器,之后会拿到对语句的分析,也就是执行计划,其中包含了许多信息.可以通过其中和索引有关的信息来分析是否命中了索引,例如possible_key,key,key_len等字段,分别说明了此语句可能会使用的索引,实际使用的索引以及使用的索引长度.

21. 那么在哪些情况下会发生针对该列创建了索引但是在查询的时候并没有使用呢?

- 使用不等于查询,
- 列参与了数学运算或者函数
- 在字符串like时左边是通配符.类似于'%aaa'.
- 当mysql分析全表扫描比使用索引快的时候不使用索引.
- 当使用联合索引,前面一个条件为范围查询,后面的即使符合最左前缀原则,也无法使用索引.

以上情况,MySQL无法使用索引.

22. 什么是事务?

理解什么是事务最经典的就是转账的栗子,相信大家也都了解,这里就不再说一边了.

事务是一系列的操作,他们要符合ACID特性.最常见的理解就是:事务中的操作要么全部成功,要么全部失败.但是只是这样还不够的.

23. ACID是什么?可以详细说一下吗?

A=Atomicity

原子性,就是上面说的,要么全部成功,要么全部失败.不可能只执行一部分操作.

C=Consistency

系统(数据库)总是从一个一致性的状态转移到另一个一致性的状态,不会存在中间状态.

I=Isolation

隔离性:通常来说:一个事务在完全提交之前,对其他事务是不可见的.注意前面的通常来说加了红色,意味着有例外情况.

D=Durability

持久性,一旦事务提交,那么就永远是这样子了,哪怕系统崩溃也不会影响到这个事务的结果.

24. 同时有多个事务在进行会怎么样呢?

多事务的并发进行一般会造成以下几个问题:

- 脏读: A事务读取到了B事务未提交的内容,而B事务后面进行了回滚.
- 不可重复读: 当设置A事务只能读取B事务已经提交的部分,会造成在A事务内的两次查询,结果竟然不一样,因为在此期间B事务进行了提交操作.
- 幻读: A事务读取了一个范围的内容,而同时B事务在此期间插入了一条数据,造成"幻觉".

25. 怎么解决这些问题呢?MySQL的事务隔离级别了解吗?

MySQL的四种隔离级别如下:

- 未提交读(READ UNCOMMITTED)

这就是上面所说的例外情况了,这个隔离级别下,其他事务可以看到本事务没有提交的部分修改.因此会造成脏读的问题(读取到了其他事务未提交的部分,而之后该事务进行了回滚).

这个级别的性能没有足够大的优势,但是又有很多的问题,因此很少使用.

- 已提交读(READ COMMITTED)

其他事务只能读取到本事务已经提交的部分.这个隔离级别有 不可重复读的问题,在同一个事务内的两次读取,拿到的结果竟然不一样,因为另外一个事务对数据进行了修改.

- REPEATABLE READ(可重复读)

可重复读隔离级别解决了上面不可重复读的问题(看名字也知道),但是仍然有一个新问题,就是 幻读,当你读取id> 10 的数据行时,对涉及到的所有行加上了读锁,此时例外一个事务新插入了一条id=11的数据,因为是新插入的,所以不会触发上面的锁的排斥,那么进行本事务进行下一次的查询时会发现有一条id=11的数据,而上次的查询操作并没有获取到,再进行插入就会有主键冲突的问题.

- SERIALIZABLE(可串行化)

这是最高的隔离级别,可以解决上面提到的所有问题,因为他强制将所有的操作串行执行,这会导致并发性能极速下降,因此也不是很常用.

26. InnoDB使用的是哪种隔离级别呢?

InnoDB默认使用的是可重复读隔离级别.

27. 对MySQL的锁了解吗?

当数据库有并发事务的时候,可能会产生数据的不一致,这时候需要一些机制来保证访问的次序,锁机制就是这样的一个机制.

就像酒店的房间,如果大家随意进出,就会出现多人抢夺同一个房间的情况,而在房间上装上锁,申请到钥匙的人才可以入住并且将房间锁起来,其他人只有等他使用完毕才可以再次使用.

28. MySQL都有哪些锁呢?像上面那样子进行锁定岂不是有点阻碍并发效率了?

从锁的类别上来讲,有共享锁和排他锁.

共享锁: 又叫做读锁. 当用户要进行数据的读取时,对数据加上共享锁.共享锁可以同时加上多个.

排他锁: 又叫做写锁. 当用户要进行数据的写入时,对数据加上排他锁.排他锁只可以加一个,他和其他的排他锁,共享锁都相斥.

用上面的例子来说就是用户的行为有两种,一种是来看房,多个用户一起看房是可以接受的. 一种是真正的入住一晚,在这期间,无论是想入住的还是想看房的都不可以.

锁的粒度取决于具体的存储引擎,InnoDB实现了行级锁,页级锁,表级锁.

他们的加锁开销从大大小，并发能力也是从大到小。

29. 为什么要尽量设定一个主键？

主键是数据库确保数据行在整张表唯一性的保障，即使业务上本张表没有主键，也建议添加一个自增长的ID列作为主键。设定了主键之后，在后续的删改查的时候可能更加快速以及确保操作数据范围安全。

30. 主键使用自增ID还是UUID？

推荐使用自增ID，不要使用UUID。

因为在InnoDB存储引擎中，主键索引是作为聚簇索引存在的，也就是说，主键索引的B+树叶子节点上存储了主键索引以及全部的数据（按照顺序），如果主键索引是自增ID，那么只需要不断向后排列即可；如果是UUID，由于到来的ID与原来的大小不确定，会造成非常多的数据插入、数据移动，然后导致产生很多的内存碎片，进而造成插入性能的下降。

总之，在数据量大一些的情况下，用自增主键性能会好一些。

图片来源于《高性能MySQL》：其中默认后缀为使用自增ID，_uuid为使用UUID为主键的测试，测试了插入100w行和300w行的性能。

表 5-1：向InnoDB表插入数据的测试结果

表名	行数	时间（秒）	索引大小（MB）
userinfo	1 000 000	137	342
userinfo_uuid	1 000 000	180	544
userinfo	3 000 000	1233	1036
userinfo_uuid	3 000 000	4525	1707

关于主键是聚簇索引，如果没有主键，InnoDB会选择一个唯一键来作为聚簇索引，如果没有唯一键，会生成一个隐式的主键。

If you define a PRIMARY KEY on your table, InnoDB uses it as the clustered index.

If you do not define a PRIMARY KEY for your table, MySQL picks the first UNIQUE index that has only NOT NULL columns as the primary key and InnoDB uses it as the clustered index.

31. 字段为什么要求定义为not null？

MySQL官网这样介绍：

NULL columns require additional space in the row to record whether their values are NULL.
For MyISAM tables, each NULL column takes one bit extra, rounded up to the nearest byte.

null值会占用更多的字节，且会在程序中造成很多与预期不符的情况。

32. 如果要存储用户的密码散列，应该使用什么字段进行存储？

密码散列、盐、用户身份证号等固定长度的字符串应该使用char而不是varchar来存储，这样可以节省空间且提高检索效率。

33. MySQL支持哪些存储引擎？

MySQL支持多种存储引擎，比如InnoDB、MyISAM、Memory、Archive等等。在大多数的情况下，直接选择使用InnoDB引擎都是最合适的，InnoDB也是MySQL的默认存储引擎。

1. InnoDB和MyISAM有什么区别?

- InnoDB支持事物, 而MyISAM不支持事物
- InnoDB支持行级锁, 而MyISAM支持表级锁
- InnoDB支持MVCC, 而MyISAM不支持
- InnoDB支持外键, 而MyISAM不支持
- InnoDB不支持全文索引, 而MyISAM支持。

34. MySQL中的varchar和char有什么区别.**

char是一个定长字段,假如申请了 `char(10)` 的空间,那么无论实际存储多少内容,该字段都占用10个字符,而varchar是变长的,也就是说申请的只是最大长度,占用的空间为实际字符长度+1,最后一个字符存储使用了多长的空间.

在检索效率上来讲,`char > varchar`,因此在使用中,如果确定某个字段的值的长度,可以使用char,否则应该尽量使用varchar.例如存储用户MD5加密后的密码,则应该使用char.

35. varchar(10)和int(10)代表什么含义?

varchar的10代表了申请的空间长度,也是可以存储的数据的最大长度,而int的10只是代表了展示的长度,不足10位以0填充.也就是说,int(1)和int(10)所能存储的数字大小以及占用的空间都是相同的,只是在展示时按照长度展示.

36. MySQL的binlog有几种录入格式?分别有什么区别?

有三种格式,statement,row和mixed.

- statement模式下,记录单元为语句.即每一个sql造成的影响会记录.由于sql的执行是有上下文的,因此在保存的时候需要保存相关的信息,同时还有一些使用了函数之类的语句无法被记录复制.
- row级别下,记录单元为每一行的改动,基本是可以全部记下来但是由于很多操作,会导致大量行的改动(比如`alter table`),因此这种模式的文件保存的信息太多,日志量太大.
- mixed. 一种折中的方案,普通操作使用statement记录,当无法使用statement的时候使用row.

此外,新版的MySQL中对row级别也做了一些优化,当表结构发生变化的时候,会记录语句而不是逐行记录.

37. 超大分页怎么处理?

超大的分页一般从两个方向上来解决.

- 数据库层面,这也是我们主要集中关注的(虽然收效没那么大),类似于 `select * from table where age > 20 limit 1000000,10` 这种查询其实也是有可以优化的余地的. 这条语句需要load1000000数据然后基本上全部丢弃,只取10条当然比较慢. 当时我们可以修改为 `select * from table where id in (select id from table where age > 20 limit 1000000,10)`. 这样虽然也load了一百万的数据,但是由于索引覆盖,要查询的所有字段都在索引中,所以速度会很快. 同时如果ID连续的好,我们还可以 `select * from table where id > 1000000 limit 10`,效率也是不错的,优化的可能性有许多种,但是核心思想都一样,就是减少load的数据.
- 从需求的角度减少这种请求....主要是不做类似的需求(直接跳转到几百万页之后的具体某一页.只允许逐页查看或者按照给定的路线走,这样可预测,可缓存)以及防止ID泄漏且连续被人恶意攻击.

解决超大分页,其实主要是靠缓存,可预测性的提前查到内容,缓存至redis等k-V数据库中,直接返回即可.

在阿里巴巴《Java开发手册》中,对超大分页的解决办法是类似于上面提到的第一种.

7. 【推荐】利用延迟关联或者子查询优化超多分页场景。

说明：MySQL 并不是跳过 offset 行，而是取 offset+N 行，然后返回放弃前 offset 行，返回 N 行，那当 offset 特别大的时候，效率就非常的低下，要么控制返回的总页数，要么对超过特定阈值的页数进行 SQL 改写。

正例：先快速定位需要获取的 id 段，然后再关联：

```
SELECT a.* FROM 表1 a, (select id from 表1 where 条件 LIMIT 100000,20 ) b where a.id=b.id
```

38. 关心过业务系统里面的sql耗时吗？统计过慢查询吗？对慢查询都怎么优化过？

在业务系统中，除了使用主键进行的查询，其他的我都会在测试库上测试其耗时，慢查询的统计主要由运维在做，会定期将业务中的慢查询反馈给我们。

慢查询的优化首先要搞明白慢的原因是什么？是查询条件没有命中索引？是 load 了不需要的数据列？还是数据量太大？

所以优化也是针对这三个方向来的，

- 首先分析语句，看看是否 load 了额外的数据，可能是查询了多余的行并且抛弃掉了，可能是加载了许多结果中并不需要的列，对语句进行分析以及重写。
- 分析语句的执行计划，然后获得其使用索引的情况，之后修改语句或者修改索引，使得语句可以尽可能的命中索引。
- 如果对语句的优化已经无法进行，可以考虑表中的数据量是否太大，如果是的话可以进行横向或者纵向的分表。

39. 上面提到横向分表和纵向分表，可以分别举一个适合他们的例子吗？**

横向分表是按行分表。假设我们有一张用户表，主键是自增 ID 且同时是用户的 ID。数据量较大，有 1 亿多条，那么此时放在一张表里的查询效果就不太理想。我们可以根据主键 ID 进行分表，无论是按尾号分，或者按 ID 的区间分都是可以的。假设按照尾号 0-99 分为 100 个表，那么每张表中的数据就仅有 100w。这时的查询效率无疑是完全可以满足要求的。

纵向分表是按列分表。假设我们现在有一张文章表，包含字段 id-摘要-内容。而系统中的展示形式是刷新出一个列表，列表中仅包含标题和摘要，当用户点击某篇文章进入详情时才需要正文内容。此时，如果数据量大，将内容这个很大且不经常使用的列放在一起会拖慢原表的查询速度。我们可以将上面的表分为两张。id-摘要，id-内容。当用户点击详情，那主键再来取一次内容即可。而增加的存储量只是很小的主键字段。代价很小。

当然，分表其实和业务的关联度很高，在分表之前一定要做好调研以及 benchmark，不要按照自己的猜想盲目操作。

40. 什么是存储过程？有哪些优缺点？

存储过程是一些预编译的 SQL 语句。1、更加直白的理解：存储过程可以说是一个记录集，它是由一些 T-SQL 语句组成的代码块，这些 T-SQL 语句代码像一个方法一样实现一些功能（对单表或多表的增删改查），然后再给这个代码块取一个名字，在用到这个功能的时候调用他就行了。2、存储过程是一个预编译的代码块，执行效率比较高，一个存储过程替代大量 T_SQL 语句，可以降低网络通信量，提高通信速率，可以在一定程度上确保数据安全。

但是，在互联网项目中，其实是不太推荐存储过程的，比较出名的就是阿里的《Java 开发手册》中禁止使用存储过程，我个人的理解是，在互联网项目中，迭代太快，项目的生命周期也比较短，人员流动相比于传统的项目也更加频繁，在这样的情况下，存储过程的管理确实是没有那么方便，同时，复用性也没有写在服务层那么好。

MongoDB常见面试题及答案

1.什么是NoSQL数据库？NoSQL和RDBMS有什么区别？在哪些情况下使用和不使用NoSQL数据库？

NoSQL是非关系型数据库，NoSQL = Not Only SQL。

关系型数据库采用的结构化的数据，NoSQL采用的是键值对的方式存储数据。

在处理非结构化/半结构化的大数据时；在水平方向上进行扩展时；随时应对动态增加的数据项时可以优先考虑使用NoSQL数据库。

在考虑数据库的成熟度；支持；分析和商业智能；管理及专业性等问题时，应优先考虑关系型数据库。

2.非关系型数据库有哪些？

Membase、MongoDB、Hypertable**

3.MySQL和MongoDB之间最基本的的区别是什么？

关系型数据库与非关系型数据库的区别，即数据存储结构的不同。

4.MongoDB的特点是什么？

- (1) 面向文档
- (2) 高性能
- (3) 高可用
- (4) 易扩展
- (5) 丰富的查询语言

5. MongoDB支持存储过程吗？如果支持的话，怎么用？

MongoDB支持存储过程，它是javascript写的，保存在db.system.js表中。

6.如何理解MongoDB中的GridFS机制，MongoDB为何使用GridFS来存储文件？

GridFS是一种将大型文件存储在MongoDB中的文件规范。使用GridFS可以将大文件分隔成多个小文档存放，这样我们能够有效的保存大文档，而且解决了BSON对象有限制的问题。

7.为什么MongoDB的数据文件很大？

MongoDB采用的预分配空间的方式来防止文件碎片。

8.当更新一个正在被迁移的块（Chunk）上的文档时会发生什么？

更新操作会立即发生在旧的块（Chunk）上，然后更改才会在所有权转移前复制到新的分片上。

9.MongoDB在A:{B,C}上建立索引，查询A:{B,C}和A:{C,B}都会使用索引吗？

不会，只会在A:{B,C}上使用索引。

10.如果一个分片（Shard）停止或很慢的时候，发起一个查询会怎样？

如果一个分片停止了，除非查询设置了“Partial”选项，否则查询会返回一个错误。如果一个分片响应很慢，MongoDB会等待它的响应。

11、为什么我们要使用MongoDB？

1、特点：

高性能、易部署、易使用，存储数据非常方便。主要功能特性有：
面向集合存储，易存储对象类型的数据。
模式自由。
支持动态查询。
支持完全索引，包含内部对象。
支持查询。
支持复制和故障恢复。
使用高效的二进制数据存储，包括大型对象（如视频等）。
自动处理碎片，以支持云计算层次的扩展性。
支持Python, PHP, Ruby, Java, C, C#, Javascript, Perl及C++语言的驱动程序，社区中也提供了对Erlang及.NET等平台的驱动程序。
文件存储格式为BSON（一种JSON的扩展）。
可通过网络访问。

2、功能：

面向集合的存储：适合存储对象及JSON形式的数据。
动态查询：Mongo支持丰富的查询表达式。查询指令使用JSON形式的标记，可轻易查询文档中内嵌的对象及数组。
完整的索引支持：包括文档内嵌对象及数组。Mongo的查询优化器会分析查询表达式，并生成一个高效的查询计划。
查询监视：Mongo包含一个监视工具用于分析数据库操作的性能。
复制及自动故障转移：Mongo数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目标是提供冗余及自动故障转移。
高效的传统存储方式：支持二进制数据及大型对象（如照片或图片）
自动分片以支持云级别的伸缩性：自动分片功能支持水平的数据库集群，可动态添加额外的机器。

3、适用场合：

网站数据：Mongo非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。
缓存：由于性能很高，Mongo也适合作为信息基础设施的缓存层。在系统重启之后，由Mongo搭建的持久化缓存层可以避免下层的数据源过载。
大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。
高伸缩性的场景：Mongo非常适合由数十或数百台服务器组成的数据库。Mongo的路线图中已经包含对MapReduce引擎的内置支持。
用于对象及JSON数据的存储：Mongo的BSON数据格式非常适合文档化格式的存储及查询。

12、MongoDB要注意的问题

- 1 因为MongoDB是全索引的，所以它直接把索引放在内存中，因此最多支持2.5G的数据。如果是64位的会更多。
- 2 因为没有恢复机制，因此要做好数据备份
- 3 因为默认监听地址是127.0.0.1，因此要进行身份验证，否则不够安全；如果是自己使用，建议配置成localhost主机名

4 通过GetLastError确保变更。 (这个不懂，实际中没用过)

13、MongoDB结构介绍

1、MongoDB中存储的对象是BSON，是一种类似JSON的二进制文件，它是由许多的键值对组成。

如下所示

```
{  
    "name" : "huangz",  
    "age" : 20,  
    "sex" : "male"  
}  
  
{  
    "name" : "jack",  
    "class" : 3,  
    "grade" : 3  
}
```

2、数据库的整体结构组成如下：

键值对-》文档-》集合-》数据库

MongoDB的文件单个大小不超过4M，但是新版本后可提升到16M

3、MongoDB中的key命名规则如下：

'\0'不能使用

带有'.'号，'_'号和'\$'号前缀的Key被保留

大小写有区别，Age不同于age

同一个文档不能有相同的Key

除了上面几条规则外，其他所有UTF-8字符都可以使用

14、常用命令

1 #进入数据库

```
use admin
```

2 #增加或修改密码

```
db.addUser('Diana','123')
```

db.addUser('Diana','123',true) 参数分别为 用户名、密码、是否只读

3 #查看用户列表

```
db.system.users.find()
```

4 #用户认证

```
db.auth('Diana','123')
```

5 #删除用户

```
db.removeUser('Diana')
```

6 #查看所有用户

```
show users
```

7 #查看所有数据库

```
show dbs
```

8 #查看所有的collection集合

```
show collections
```

9 #查看各个collection的状态

```
db.printcollectionstats()
```

10 #查看主从复制状态

```
db.printReplicationInfo()
```

11 #修复数据库

```
db.repairDatabase()
```

12 #设置profiling,0:off 1:slow 2 all

```
db.setProfilingLevel(1)
```

13 #查看profiling

```
show profiling
```

14 #拷贝数据库

```
db.copyDatabase('Dianatest','Dianatest1')
```

```
db.copyDatabase('Dianatest','temp','127.0.0.1')
```

15 #删除集合collection

```
db.Dianatest.drop()
```

16 #删除当前数据库

```
db.dropDatabase()
```

15、MongoDB增删改命令

1 #存储嵌套的对象

```
db.foo.save({'name':'Diana', 'age':25, 'address':  
{'city':'changchun', 'Province':'Jilin'}})
```

2 #存储数组对象

```
db.foo.save({'name':Diana, 'age':25, 'address':['Jilin Province', 'Liaoning  
Province']})
```

3 #根据query条件修改, 如果不存在则插入, 允许修改多条记录

```
db.foo.update({'age':25}, {'$set':{'name':'Diana'}}, upsert=true, multi=true)
```

4 #删除name='Diana'的记录

```
db.foo.remove({'name':'Diana'})
```

5 #删除所有的记录

```
db.foo.remove()
```

16、索引

1 #增加索引|:1 asc -1 desc

```
db.foo.createIndex({firstname:1, lastname:-1}, {unieap:true})
```

2 #索引子对象(不懂)

```
db.foo.createIndex({'A1.Em':1})
```

3 #查看索引信息

```
db.foo.getIndexes()
```

```
db.foo.getIndexKeys()
```

4 #根据索引名删除索引(不懂)

```
db.foo.dropIndex('A1.Em_1')
```

17、查询

条件操作符

```
1 $gt ---- >
2 $lt ---- <
3 $gte ---- >=
4 $lte ---- <=
5 $ne ---- != 、 <>
6 $in ---- in
7 $nin ---- not in
8 $all ---- all
9 $or ---- or
10 $not ---- 反匹配
```

1 #查询所有记录

```
db.foo.find() -- select * from foo
```

2 #查询某列非重复的记录

```
db.foo.distinct('Diana') -- select distinct name from foo
```

3 #查询age = 22 的记录

```
db.foo.find({'age':22}) -- select * from foo where age = 22
```

4 #查询age > 22 的记录

```
db.foo.find({age:{$gt:22}}) -- select * from foo where age > 22
```

5 #查询age < 22 的记录

```
db.foo.find({age:{$lt:22}}) -- select * from foo where age < 22
```

6 #查询age <= 25的记录

```
db.foo.find({age:{$lte:25}})
```

7 #查询age >= 23 并且 age <=26的记录

```
db.foo.find({age:{$gte:23,$lte:26}})
```

8 #查询name中包含Diana的数据

```
db.foo.find({name:/Diana/}) -- select * from foo where name like '%Diana%'
```

9 #查询name中以Diana开头的数据

```
db.foo.find({name:/^Diana/}) -- select * from foo where name like 'Diana%'
```

10 #查询指定列name、 age的数据

```
db.foo.find({}, {name:1,age:1}) -- select name,age from foo
```

11 #查询制定列name、 age数据，并且age > 22

```
db.foo.find({age:{$gt:22}}, {name:1, age:1}) -- select name,age from foo where age >22
```

12 #按照年龄排序

```
升序: db.foo.find().sort({age:1}) 降序: db.foo.find().sort({age:-1})
```

13 #查询name=Diana.age=25的数据

```
db.foo.find({name:'Diana', age:22}) -- select * from foo where name='Diana' and age ='25'
```

14#查询前5条数据

```
db.foo.find().limit(5) -- select top 5 * from foo
```

15 #查询10条以后的数据

```
db.foo.find().skip(10) -- select * from foo where id not in (select top 10 * from foo);
```

16 #查询在5-10之间的数据

```
db.foo.find().limit(10).skip(5)
```

17 #or与查询

```
db.foo.find({$or:[{age:22},{age:25}]}) -- select * from foo where age=22 or age =25
```

18 #查询第一条数据

```
db.foo.findone()、db.foo.find().limit(1)-- select top 1 * from foo
```

19 #查询某个结果集的记录条数

```
db.foo.find({age:{$gte:25}}).count() -- select count(*) from foo where age >= 20
```

20 #按照某列 (判断存在 sex 这个字段的记录) 进行排序

```
db.foo.find({sex:{$exists:true}}).count() -- select count(sex) from foo
```

21 #查询age取模10等于0的数据

```
db.foo.find('this.age % 10 == 0')  
db.foo.find({age:{$mod:[10,0]}})
```

22 #匹配所有

```
db.foo.find({age:{$all:[22,25]}})
```

23 #查询不匹配 name=x* 带头的记录

```
db.foo.find({name:{$not:/^x.*/}})
```

24 #查询的记录不要 age 字段

```
db.foo.find({name:'Diana'}, {age:0})
```

25 #判断存在 name 这个字段的所有记录

```
db.foo.find({name:{$exists:true}})
```

18、管理

1 #查看collection数据大小

```
db.Dianatest.datasize()
```

2 #查看collection状态

```
db.Dianatest.stats()
```

3 #查询所有索引的大小

```
db.Dianatest.totalIndexsize()
```

Kafka常见23道面试题以答案

1、Kafka的用途有哪些？使用场景如何？

总结下来就几个字：异步处理、日常系统解耦、削峰、提速、广播

如果说具体一点例如：消息，网站活动追踪，监测指标，日志聚合，流处理，事件采集，提交日志等

2、Kafka中的ISR、AR又代表什么？ISR的伸缩又指什么

ISR:In-Sync Replicas 副本同步队列

AR:Assigned Replicas 所有副本

ISR是由leader维护，follower从leader同步数据有一些延迟（包括延迟时间replica.lag.time.max.ms和延迟条数replica.lag.max.messages两个维度，当前最新的版本0.10.x中只支持replica.lag.time.max.ms这个维度），任意一个超过阈值都会把follower剔除出ISR，存入OSR（Outof-Sync Replicas）列表，新加入的follower也会先存放在OSR中。AR=ISR+OSR。

3、Kafka中的HW、LEO、LSO、LW等分别代表什么？

HW:High Watermark 高水位，取一个partition对应的ISR中最小的LEO作为HW，consumer最多只能消费到HW所在的位置上一条信息。

LEO:LogEndOffset 当前日志文件中下一条待写信息的offset

HW/LEO这两个都是指最后一条的下一条的位置而不是指最后一条的位置。

LSO:Last Stable Offset 对未完成的事务而言，LSO 的值等于事务中第一条消息的位置(firstUnstableOffset)，对已完成的事务而言，它的值同 HW 相同

LW:Low Watermark 低水位, 代表 AR 集合中最小的 logStartOffset 值

4、Kafka中是怎么体现消息顺序性的？

kafka每个partition中的消息在写入时都是有序的，消费时，每个partition只能被每一个group中的一个消费者消费，保证了消费时也是有序的。

整个topic不保证有序。如果为了保证topic整个有序，那么将partition调整为1.

5、Kafka中的分区器、序列化器、拦截器是否了解？它们之间的处理顺序是什么？

拦截器->序列化器->分区器

6、Kafka生产者客户端中使用了几个线程来处理？分别是什么？

2个，主线程和Sender线程。主线程负责创建消息，然后通过分区器、序列化器、拦截器作用之后缓存到累加器RecordAccumulator中。Sender线程负责将RecordAccumulator中消息发送到kafka中。

7、“消费组中的消费者个数如果超过topic的分区，那么就会有消费者消费不到数据”这句话是否正确？如果不正确，那么有没有什么hack的手段？

不正确，通过自定义分区分配策略，可以将一个consumer指定消费所有partition。

8、消费者提交消费位移时提交的是当前消费到的最新消息的offset还是offset+1？

offset+1

9、有哪些情形会造成重复消费？

消费者消费后没有commit offset(程序崩溃/强行kill/消费耗时/自动提交偏移情况下unscrable)

10、那些情景下会造成消息漏消费？

消费者没有处理完消息 提交offset(自动提交偏移 未处理情况下程序异常结束)

11、KafkaConsumer是非线程安全的，那么怎么样实现多线程消费？

在每个线程中新建一个KafkaConsumer

单线程创建KafkaConsumer，多个处理线程处理消息（难点在于是否要考虑消息顺序性，offset的提交方式）

12、简述消费者与消费组之间的关系

消费者从属与消费组，消费偏移以消费组为单位。每个消费组可以独立消费主题的所有数据，同一消费组内消费者共同消费主题数据，每个分区只能被同一消费组内一个消费者消费。

13、当你使用kafka-topics.sh创建（删除）了一个topic之后，Kafka背后会执行什么逻辑？

创建：在zk上/brokers/topics/下节点 kafka broker会监听节点变化创建主题

删除：调用脚本删除topic会在zk上将topic设置待删除标志，kafka后台有定时的线程会扫描所有需要删除的topic进行删除

14、topic的分区数可不可以增加？如果可以怎么增加？如果不可以，那又是为什么？

可以

15、topic的分区数可不可以减少？如果可以怎么减少？如果不可以，那又是为什么？

不可以

16、创建topic时如何选择合适的分区数？

根据集群的机器数量和需要的吞吐量来决定适合的分区数

17、Kafka目前有那些内部topic，它们都有什么特征？各自的作用又是什么？

_consumer_offsets 以下划线开头，保存消费组的偏移

18、优先副本是什么？它有什么特殊的作用？

优先副本 会是默认的leader副本 发生leader变化时重选举会优先选择优先副本作为leader

19、Kafka有哪几处地方有分区分配的概念？简述大致的过程及原理

创建主题时

如果不手动指定分配方式 有两种分配方式

20、简述Kafka的日志目录结构

每个partition一个文件夹，包含四类文件.index .log .timeindex leader-epoch-checkpoint
.index .log .timeindex 三个文件成对出现 前缀为上一个segment的最后一个消息的偏移 log文件中保存了所有的消息 index文件中保存了稀疏的相对偏移的索引 timeindex保存的是时间索引
leader-epoch-checkpoint中保存了每一任leader开始写入消息时的offset 会定时更新
follower被选为leader时会根据这个确定哪些消息可用

21、如果我指定了一个offset，Kafka怎么查找到对应的消息？

通过文件名前缀数字x找到该绝对offset 对应消息所在文件

22、如果我指定了一个timestamp， Kafka怎么查找到对应的消 息？

原理同上 但是时间的因为消息体中不带有时间戳 所以不精确

23、聊一聊你对Kafka的Log Retention的理解

kafka留存策略包括 删除和压缩两种

删除: 根据时间和大小两个方式进行删除 大小是整个partition日志文件的大小

超过的会从老到新依次删除 时间指日志文件中的最大时间戳而非文件的最后修改时间

压缩: 相同key的value只保存一个 压缩过的是clean 未压缩的dirty 压缩之后的偏移量不连续 未压缩时连续

Dubbo面试40问及参考答案

1、Dubbo是什么？

Dubbo是阿里巴巴开源的基于 Java 的高性能 RPC 分布式服务框架，现已成为 Apache 基金会孵化项目。

面试官问你如果这个都不清楚，那下面的就没必要问了。

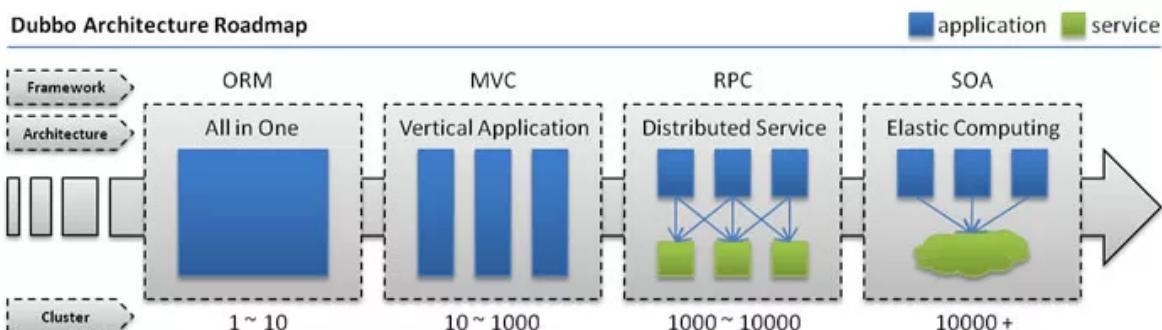
官网：<http://dubbo.apache.org>

2、为什么要用Dubbo？

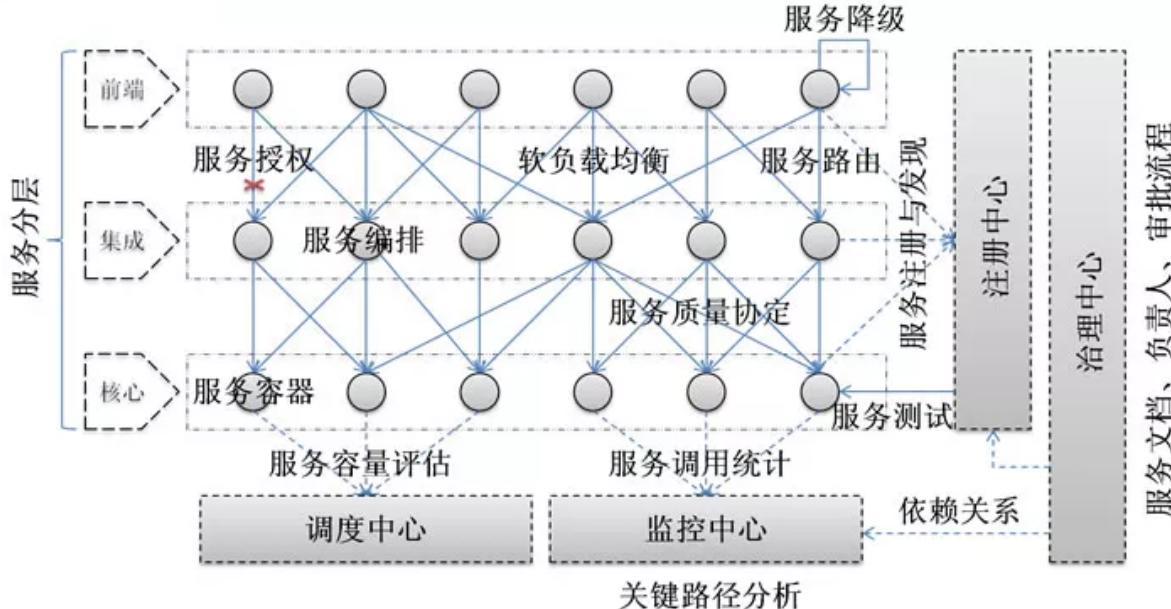
因为是阿里开源项目，国内很多互联网公司都在用，已经经过很多线上考验。内部使用了 Netty、Zookeeper，保证了高性能高可用性。

使用 Dubbo 可以将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，可用于提高业务复用灵活扩展，使前端应用能更快速的响应多变的市场需求。

下面这张图可以很清楚的诠释，最重要的一点是，分布式架构可以承受更大规模的并发流量。



下面是 Dubbo 的服务治理图。



3、Dubbo 和 Spring Cloud 有什么区别？

两个没关联，如果硬要说区别，有以下几点。

1) 通信方式不同

Dubbo 使用的是 RPC 通信，而 Spring Cloud 使用的是 HTTP RESTful 方式。

2) 组成部分不同

组件	Dubbo	Spring Cloud
服务注册中心	Zookeeper	Spring Cloud Netflix Eureka
服务监控	Dubbo-monitor	Spring Boot Admin
断路器	不完善	Spring Cloud Netflix Hystrix
服务网关	无	Spring Cloud Netflix Gateway
分布式配置	无	Spring Cloud Config
服务跟踪	无	Spring Cloud Sleuth
消息总线	无	Spring Cloud Bus
数据流	无	Spring Cloud Stream
批量任务	无	Spring Cloud Task
...

4、dubbo都支持什么协议，推荐用哪种？

- dubbo:// (推荐)

- rmi://
- hessian://
- http://
- webservice://
- thrift://
- memcached://
- redis://
- rest://

5、Dubbo需要 Web 容器吗？

不需要，如果硬要用 Web 容器，只会增加复杂性，也浪费资源。

6、Dubbo内置了哪几种服务容器？

- Spring Container
- Jetty Container
- Log4j Container

Dubbo 的服务容器只是一个简单的 Main 方法，并加载一个简单的 Spring 容器，用于暴露服务。

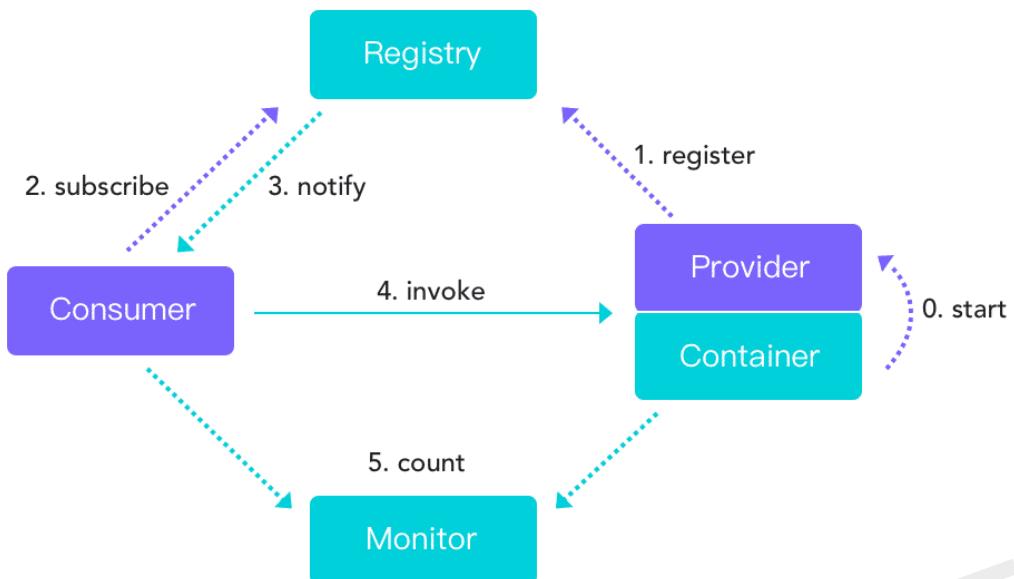
7、Dubbo里面有哪几种节点角色？

节点	角色说明
Provider	暴露服务的服务提供方
Consumer	调用远程服务的服务消费方
Registry	服务注册与发现的注册中心
Monitor	统计服务的调用次数和调用时间的监控中心
Container	服务运行容器

8、画一画服务注册与发现的流程图

Dubbo Architecture

..... init > async —> sync



该图来自 Dubbo 官网，供你参考，如果说你熟悉 Dubbo，面试官经常会让你画这个图，记好了。

9、Dubbo默认使用什么注册中心，还有别的选择吗？

推荐使用 Zookeeper 作为注册中心，还有 Redis、Multicast、Simple 注册中心，但不推荐。

10、Dubbo有哪几种配置方式？

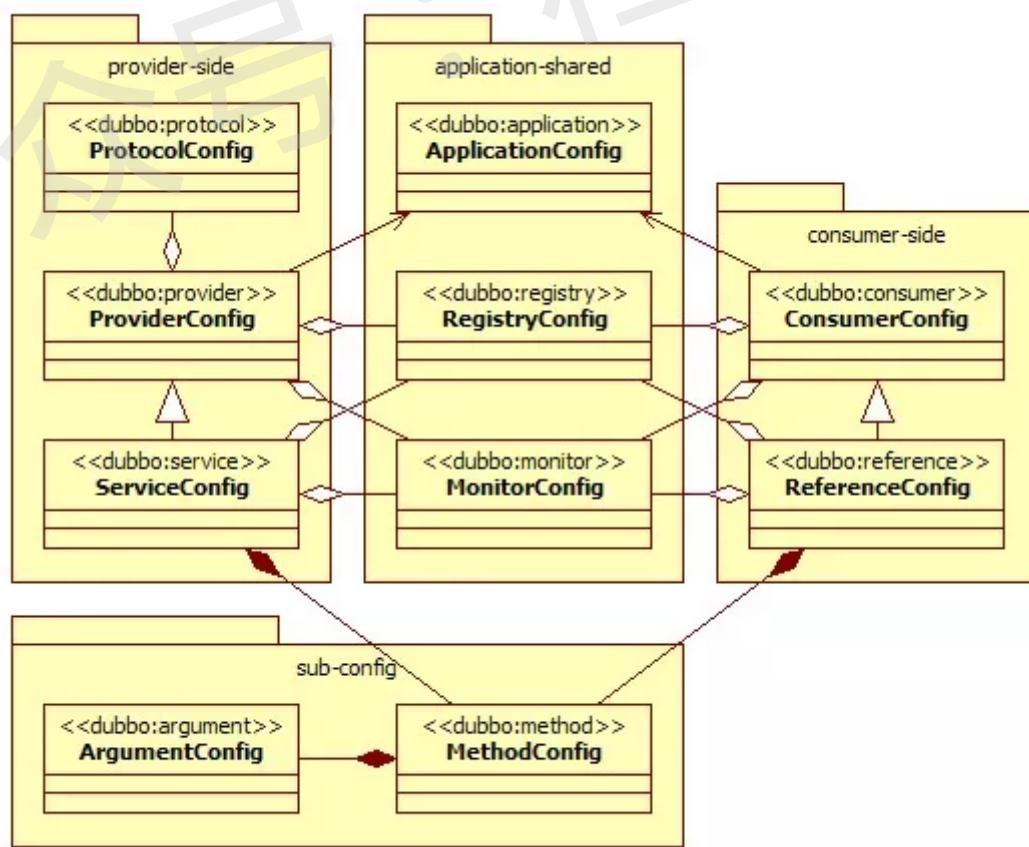
- 1) Spring 配置方式
- 2) Java API 配置方式

11、Dubbo 核心的配置有哪些？

我曾经面试就遇到过面试官让你写这些配置，我也是蒙逼。。

配置	配置说明
dubbo:service	服务配置
dubbo:reference	引用配置
dubbo:protocol	协议配置
dubbo:application	应用配置
dubbo:module	模块配置
dubbo:registry	注册中心配置
dubbo:monitor	监控中心配置
dubbo:provider	提供方配置
dubbo:consumer	消费方配置
dubbo:method	方法配置
dubbo:argument	参数配置

配置之间的关系见下图。



12、在 Provider 上可以配置的 Consumer 端的属性有哪些？

- 1) timeout: 方法调用超时
- 2) retries: 失败重试次数，默认重试 2 次
- 3) loadbalance: 负载均衡算法，默认随机
- 4) actives 消费者端，最大并发调用限制

13、Dubbo启动时如果依赖的服务不可用会怎样？

Dubbo 缺省会在启动时检查依赖的服务是否可用，不可用时会抛出异常，阻止 Spring 初始化完成， 默认 check="true"，可以通过 check="false" 关闭检查。

14、Dubbo推荐使用什么序列化框架，你知道的还有哪些？

推荐使用Hessian序列化，还有Duddo、FastJson、Java自带序列化。

15、Dubbo默认使用的是什么通信框架，还有别的选择吗？

Dubbo 默认使用 Netty 框架，也是推荐的选择，另外还集成有Mina、Grizzly。

16、Dubbo有哪几种集群容错方案，默认是哪种？

集群容错方案	说明
Failover Cluster	失败自动切换，自动重试其它服务器（默认）
Failstash Cluster	快速失败，立即报错，只发起一次调用
Failsafe Cluster	失败安全，出现异常时，直接忽略
Failback Cluster	失败自动恢复，记录失败请求，定时重发
Forking Cluster	并行调用多个服务器，只要一个成功即返回
Broadcast Cluster	广播逐个调用所有提供者，任意一个报错则报错

17、Dubbo有哪几种负载均衡策略，默认是哪种？

负载均衡策略	说明
Random LoadBalance	随机，按权重设置随机概率（默认）
RoundRobin LoadBalance	轮询，按公约后的权重设置轮询比率
LeastActive LoadBalance	最少活跃调用数，相同活跃数的随机
ConsistentHash LoadBalance	一致性 Hash，相同参数的请求总是发到同一提供者

18、注册了多个同一样的服务，如果测试指定的某一个服务呢？

可以配置环境点对点直连，绕过注册中心，将以服务接口为单位，忽略注册中心的提供者列表。

19、Dubbo支持服务多协议吗？

Dubbo 允许配置多协议，在不同服务上支持不同协议或者同一服务上同时支持多种协议。

20、当一个服务接口有多种实现时怎么做？

当一个接口有多种实现时，可以用 group 属性来分组，服务提供方和消费方都指定同一个 group 即可。

21、服务上线怎么兼容旧版本？

可以用版本号（version）过渡，多个不同版本的服务注册到注册中心，版本号不同的服务相互间不引用。这个和服务分组的概念有一点类似。

22、Dubbo可以对结果进行缓存吗？

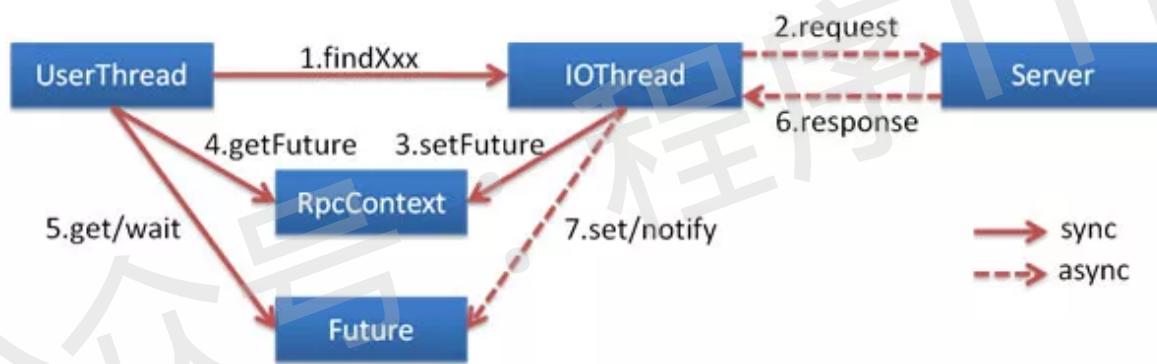
可以，Dubbo 提供了声明式缓存，用于加速热门数据的访问速度，以减少用户加缓存的工作量。

23、Dubbo服务之间的调用是阻塞的吗？

默认是同步等待结果阻塞的，支持异步调用。

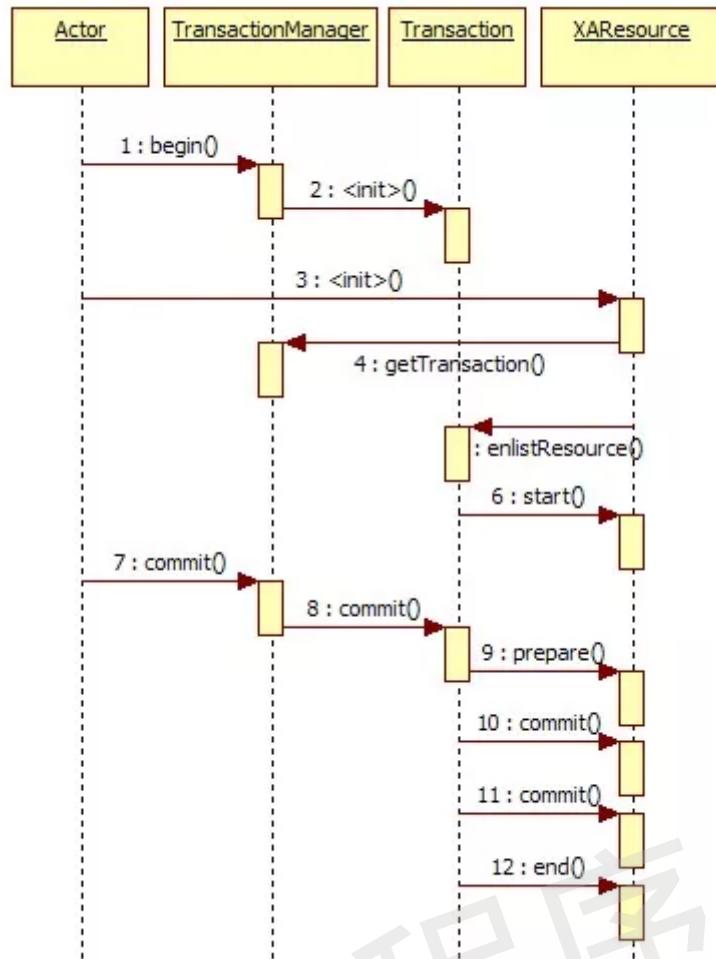
Dubbo 是基于 NIO 的非阻塞实现并行调用，客户端不需要启动多线程即可完成并行调用多个远程服务，相对多线程开销较小，异步调用会返回一个 Future 对象。

异步调用流程图如下。



24、Dubbo支持分布式事务吗？

目前暂时不支持，后续可能采用基于 JTA/XA 规范实现，如以图所示。



25、Dubbo telnet 命令能做什么？

dubbo 通过 telnet 命令来进行服务治理。

telnet localhost 8090

26、Dubbo支持服务降级吗？

Dubbo 2.2.0 以上版本支持。

27、Dubbo如何优雅停机？

Dubbo 是通过 JDK 的 ShutdownHook 来完成优雅停机的，所以如果使用 kill -9 PID 等强制关闭指令，是不会执行优雅停机的，只有通过 kill PID 时，才会执行。

28、服务提供者能实现失效踢出是什么原理？

服务失效踢出基于 Zookeeper 的临时节点原理。

29、如何解决服务调用链过长的问题？

Dubbo 可以使用 Pinpoint 和 Apache Skywalking(Incubator) 实现分布式服务追踪，当然还有其他很多方案。

30、服务读写推荐的容错策略是怎样的？

读操作建议使用 Failover 失败自动切换，默认重试两次其他服务器。

写操作建议使用 Failfast 快速失败，发一次调用失败就立即报错。

31、Dubbo必须依赖的包有哪些？

Dubbo 必须依赖 JDK，其他为可选。

32、Dubbo的管理控制台能做什么？

管理控制台主要包含：路由规则，动态配置，服务降级，访问控制，权重调整，负载均衡，等管理功能。

33、说说 Dubbo 服务暴露的过程。

Dubbo 会在 Spring 实例化完 bean 之后，在刷新容器最后一步发布 ContextRefreshEvent 事件的时候，通知实现了 ApplicationListener 的 ServiceBean 类进行回调 onApplicationEvent 事件方法，Dubbo 会在这个方法中调用 ServiceBean 父类 ServiceConfig 的 export 方法，而该方法真正实现了服务的（异步或者非异步）发布。

34、Dubbo 停止维护了吗？

2014 年开始停止维护过几年，17 年开始重新维护，并进入了 Apache 项目。

35、Dubbo 和 Dubbox 有什么区别？

Dubbox 是继 Dubbo 停止维护后，当当网基于 Dubbo 做的一个扩展项目，如加了服务可 Restful 调用，更新了开源组件等。

36、你还了解别的分布式框架吗？

别的还有 Spring cloud、Facebook 的 Thrift、Twitter 的 Finagle 等。

37、Dubbo 能集成 Spring Boot 吗？

可以的，项目地址如下。

<https://github.com/apache/incubator-dubbo-spring-boot-project>

38、在使用过程中都遇到了些什么问题？

Dubbo 的设计目的是为了满足高并发小数据量的 rpc 调用，在大数据量下的性能表现并不好，建议使用 rmi 或 http 协议。

39、你读过 Dubbo 的源码吗？

要了解 Dubbo 就必须看其源码，了解其原理，花点时间看下吧，网上也有很多教程，后续有时间我也会在公众号上分享 Dubbo 的源码。

40、你觉得用 Dubbo 好还是 Spring Cloud 好？

扩展性的问题，没有好坏，只有适合不适合，不过我好像更倾向于使用 Dubbo，Spring Cloud 版本升级太快，组件更新替换太频繁，配置太繁琐，还有很多我觉得是没有 Dubbo 顺手的地方.....

1. ZooKeeper是什么?

ZooKeeper是一个开放源码的分布式协调服务，它是集群的管理者，监视着集群中各个节点的状态根据节点提交的反馈进行下一步合理操作。最终，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

分布式应用程序可以基于Zookeeper实现诸如数据发布/订阅、负载均衡、命名服务、分布式协调/通知、集群管理、Master选举、分布式锁和分布式队列等功能。

Zookeeper保证了如下分布式一致性特性：

- 顺序一致性
- 原子性
- 单一视图
- 可靠性
- 实时性（最终一致性）

客户端的读请求可以被集群中的任意一台机器处理，如果读请求在节点上注册了监听器，这个监听器也是由所连接的zookeeper机器来处理。对于写请求，这些请求会同时发给其他zookeeper机器并且达成一致后，请求才会返回成功。因此，随着zookeeper的集群机器增多，读请求的吞吐会提高但是写请求的吞吐会下降。

有序性是zookeeper中非常重要的一个特性，所有的更新都是全局有序的，每个更新都有一个唯一的时间戳，这个时间戳称为zxid（Zookeeper Transaction Id）。而读请求只会相对于更新有序，也就是读请求的返回结果中会带有这个zookeeper最新的zxid。

2. ZooKeeper提供了什么？

- 1、文件系统
- 2、通知机制

3. Zookeeper文件系统

Zookeeper提供一个多层级的节点命名空间（节点称为znode）。与文件系统不同的是，这些节点都可以设置关联的数据，而文件系统中只有文件节点可以存放数据而目录节点不行。

Zookeeper为了保证高吞吐和低延迟，在内存中维护了这个树状的目录结构，这种特性使得Zookeeper不能用于存放大量的数据，每个节点的存放数据上限为1M。

4. ZAB协议？

ZAB协议是为分布式协调服务Zookeeper专门设计的一种支持崩溃恢复的原子广播协议。

ZAB协议包括两种基本的模式：崩溃恢复和消息广播。

当整个zookeeper集群刚刚启动或者Leader服务器宕机、重启或者网络故障导致不存在过半的服务器与Leader服务器保持正常通信时，所有进程（服务器）进入崩溃恢复模式，首先选举产生新的Leader服务器，然后集群中Follower服务器开始与新的Leader服务器进行数据同步，当集群中超过半数机器与该Leader服务器完成数据同步之后，退出恢复模式进入消息广播模式，Leader服务器开始接收客户端的事务请求生成事物提案来进行事物请求处理。

5. 四种类型的数据节点 Znode

- PERSISTENT-持久节点
除非手动删除，否则节点一直存在于Zookeeper上
- EPHEMERAL-临时节点
临时节点的生命周期与客户端会话绑定，一旦客户端会话失效（客户端与zookeeper连接断开不一

定会话失效），那么这个客户端创建的所有临时节点都会被移除。

- PERSISTENT_SEQUENTIAL-持久顺序节点
基本特性同持久节点，只是增加了顺序属性，节点名后边会追加一个由父节点维护的自增整型数字。
- EPHEMERAL_SEQUENTIAL-临时顺序节点
基本特性同临时节点，增加了顺序属性，节点名后边会追加一个由父节点维护的自增整型数字。

6. Zookeeper Watcher 机制 - 数据变更通知

Zookeeper允许客户端向服务端的某个Znode注册一个Watcher监听，当服务端的一些指定事件触发了这个Watcher，服务端会向指定客户端发送一个事件通知来实现分布式的通知功能，然后客户端根据Watcher通知状态和事件类型做出业务上的改变。

工作机制：

- 客户端注册watcher
- 服务端处理watcher
- 客户端回调watcher
-

Watcher特性总结：

1、一次性

无论是服务端还是客户端，一旦一个Watcher被触发，Zookeeper都会将其从相应的存储中移除。这样设计有效的减轻了服务端的压力，不然对于更新非常频繁的节点，服务端会不断的向客户端发送事件通知，无论对于网络还是服务端的压力都非常大。

2、客户端串行执行

客户端Watcher回调的过程是一个串行同步的过程。

3、轻量

1) Watcher通知非常简单，只会告诉客户端发生了事件，而不会说明事件的具体内容。

2) 客户端向服务端注册Watcher的时候，并不会把客户端真实的Watcher对象实体传递到服务端，仅仅是在客户端请求中使用boolean类型属性进行了标记。

4、watcher event异步发送watcher的通知事件从server发送到client是异步的，这就存在一个问题，不同的客户端和服务器之间通过socket进行通信，由于网络延迟或其他因素导致客户端在不通的时刻监听到事件，由于Zookeeper本身提供了ordering guarantee，即客户端监听事件后，才会感知它所监视znode发生了变化。所以我们使用Zookeeper不能期望能够监控到节点每次的变化。Zookeeper只能保证最终的一致性，而无法保证强一致性。

5、注册watcher getData、exists、getChildren

6、触发watcher create、delete、setData

7、当一个客户端连接到一个新的服务器上时，watch将会被以任意会话事件触发。当与一个服务器失去连接的时候，是无法接收到watch的。而当client重新连接时，如果需要的话，所有先前注册过的watch，都会被重新注册。通常这是完全透明的。只有在一个特殊情况下，watch可能会丢失：对于一个未创建的znode的exist watch，如果在客户端断开连接期间被创建了，并且随后在客户端连接上之前又删除了，这种情况下，这个watch事件可能会丢失。

7. 客户端注册Watcher实现

- 调用getData()/getChildren()/exist()三个API，传入Watcher对象
- 标记请求request，封装Watcher到WatchRegistration
- 封装成Packet对象，发服务端发送request
- 收到服务端响应后，将Watcher注册到ZKWatcherManager中进行管理
- 请求返回，完成注册。

8. 服务端处理Watcher实现

服务端接收Watcher并存储

接收到客户端请求，处理请求判断是否需要注册Watcher，需要的话将数据节点的节点路径和ServerCnxn（ServerCnxn代表一个客户端和服务端的连接，实现了Watcher的process接口，此时可以看成一个Watcher对象）存储在WatcherManager的WatchTable和watch2Paths中去。

Watcher触发

以服务端接收到 setData() 事务请求触发NodeDataChanged事件为例：

封装WatchedEvent

将通知状态（SyncConnected）、事件类型（NodeDataChanged）以及节点路径封装成一个 WatchedEvent 对象

查询Watcher

从WatchTable中根据节点路径查找Watcher

没找到；说明没有客户端在该数据节点上注册过Watcher

找到；提取并从WatchTable和Watch2Paths中删除对应Watcher（从这里可以看出Watcher在服务端是一次性的，触发一次就失效了）

调用process方法来触发Watcher

这里process主要就是通过ServerCnxn对应的TCP连接发送Watcher事件通知。

9. 客户端回调Watcher

客户端SendThread线程接收事件通知，交由EventThread线程回调Watcher。客户端的Watcher机制同样是一次性的，一旦被触发后，该Watcher就失效了。

10. ACL权限控制机制

UGO (User/Group/Others)

目前在Linux/Unix文件系统中使用，也是使用最广泛的权限控制方式。是一种粗粒度的文件系统权限控制模式。

ACL (Access Control List) 访问控制列表

包括三个方面：

权限模式 (Scheme)

- IP：从IP地址粒度进行权限控制
- Digest：最常用，用类似于 username:password 的权限标识来进行权限配置，便于区分不同应用来进行权限控制
- World：最开放的权限控制方式，是一种特殊的digest模式，只有一个权限标识“world:anyone”
- Super：超级用户

授权对象

- 授权对象指的是权限赋予的用户或一个指定实体，例如IP地址或是机器灯。

权限 Permission

- CREATE：数据节点创建权限，允许授权对象在该Znode下创建子节点
- DELETE：子节点删除权限，允许授权对象删除该数据节点的子节点
- READ：数据节点的读取权限，允许授权对象访问该数据节点并读取其数据内容或子节点列表等
- WRITE：数据节点更新权限，允许授权对象对该数据节点进行更新操作
- ADMIN：数据节点管理权限，允许授权对象对该数据节点进行ACL相关设置操作

11. Chroot特性

3.2.0版本后，添加了Chroot特性，该特性允许每个客户端为自己设置一个命名空间。如果一个客户端设置了Chroot，那么该客户端对服务器的任何操作，都将会被限制在其自己的命名空间下。

通过设置Chroot，能够将一个客户端应用于Zookeeper服务端的一颗子树相对应，在那些多个应用公用一个Zookeeper进群的场景下，对实现不同应用间的相互隔离非常有帮助。

12. 会话管理

分桶策略： 将类似的会话放在同一区块中进行管理，以便于Zookeeper对会话进行不同区块的隔离处理以及同一区块的统一处理。

分配原则： 每个会话的“下次超时时间点” (ExpirationTime)

计算公式：

$\text{ExpirationTime_} = \text{currentTime} + \text{sessionTimeout}$

$\text{ExpirationTime} = (\text{ExpirationTime_} / \text{ExpirationInterval} + 1) * \text{ExpirationInterval}$, $\text{ExpirationInterval}$ 是指 Zookeeper 会话超时检查时间间隔，默认 tickTime

13. 服务器角色

Leader

- 事务请求的唯一调度和处理者，保证集群事务处理的顺序性
- 集群内部各服务的调度者

Follower

- 处理客户端的非事务请求，转发事务请求给Leader服务器
- 参与事务请求Proposal的投票
- 参与Leader选举投票

Observer

- 3.3.0版本以后引入的一个服务器角色，在不影响集群事务处理能力的基础上提升集群的非事务处理能力
- 处理客户端的非事务请求，转发事务请求给Leader服务器
- 不参与任何形式的投票

14. Zookeeper 下 Server 工作状态

服务器具有四种状态，分别是LOOKING、FOLLOWING、LEADING、OBSERVING。

LOOKING： 寻找Leader状态。当服务器处于该状态时，它会认为当前集群中没有Leader，因此需要进入Leader选举状态。

FOLLOWING： 跟随者状态。表明当前服务器角色是Follower。

LEADING： 领导者状态。表明当前服务器角色是Leader。

OBSERVING： 观察者状态。表明当前服务器角色是Observer。

15. Leader 选举

Leader选举是保证分布式数据一致性的关键所在。当Zookeeper集群中的一台服务器出现以下两种情况之一时，需要进入Leader选举。

- (1) 服务器初始化启动。
- (2) 服务器运行期间无法和Leader保持连接。

下面就两种情况进行分析讲解。

1. 服务器启动时期的Leader选举

若进行Leader选举，则至少需要两台机器，这里选取3台机器组成的服务器集群为例。在集群初始化阶段，当有一台服务器Server1启动时，其单独无法进行和完成Leader选举，当第二台服务器Server2启动时，此时两台机器可以相互通信，每台机器都试图找到Leader，于是进入Leader选举过程。选举过程如下

(1) 每个Server发出一个投票。 由于是初始情况，Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票会包含所推举的服务器的myid和ZXID，使用(myid, ZXID)来表示，此时Server1的投票为(1, 0)，Server2的投票为(2, 0)，然后各自将这个投票发给集群中其他机器。

(2) 接受来自各个服务器的投票。 集群的每个服务器收到投票后，首先判断该投票的有效性，如检查是否是本轮投票、是否来自LOOKING状态的服务器。

(3) 处理投票。 针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下

- 优先检查ZXID。ZXID比较大的服务器优先作为Leader。
- 如果ZXID相同，那么就比较myid。myid较大的服务器作为Leader服务器。

对于Server1而言，它的投票是(1, 0)，接收Server2的投票为(2, 0)，首先会比较两者的ZXID，均为0，再比较myid，此时Server2的myid最大，于是更新自己的投票为(2, 0)，然后重新投票，对于Server2而言，其无须更新自己的投票，只是再次向集群中所有机器发出上一次投票信息即可。

(4) 统计投票。 每次投票后，服务器都会统计投票信息，判断是否已经有过半机器接受到相同的投票信息，对于Server1、Server2而言，都统计出集群中已经有两台机器接受了(2, 0)的投票信息，此时便认为已经选出了Leader。

(5) 改变服务器状态。 一旦确定了Leader，每个服务器就会更新自己的状态，如果是Follower，那么就变更为FOLLOWING，如果是Leader，就变更为LEADING。

2. 服务器运行时期的Leader选举

在Zookeeper运行期间，Leader与非Leader服务器各司其职，即便当有非Leader服务器宕机或新加入，此时也不会影响Leader，但是一旦Leader服务器挂了，那么整个集群将暂停对外服务，进入新一轮Leader选举，其过程和启动时期的Leader选举过程基本一致。假设正在运行的有Server1、Server2、Server3三台服务器，当前Leader是Server2，若某一时刻Leader挂了，此时便开始Leader选举。选举过程如下

(1) 变更状态。 Leader挂后，余下的非Observer服务器都会将自己的服务器状态变更为LOOKING，然后开始进入Leader选举过程。

(2) 每个Server会发出一个投票。 在运行期间，每个服务器上的ZXID可能不同，此时假定Server1的ZXID为123，Server3的ZXID为122；在第一轮投票中，Server1和Server3都会投自己，产生投票(1, 123)，(3, 122)，然后各自将投票发送给集群中所有机器。

(3) 接收来自各个服务器的投票。 与启动时过程相同。

(4) 处理投票。 与启动时过程相同，此时，Server1将会成为Leader。

(5) 统计投票。 与启动时过程相同。

(6) 改变服务器的状态。 与启动时过程相同。

2.2 Leader选举算法分析

在3.4.0后的Zookeeper的版本只保留了TCP版本的FastLeaderElection选举算法。当一台机器进入Leader选举时，当前集群可能会处于以下两种状态

- 集群中已经存在Leader。
- 集群中不存在Leader。

对于集群中已经存在Leader而言，此种情况一般都是某台机器启动得较晚，在其启动之前，集群已经在正常工作，对这种情况，该机器试图去选举Leader时，会被告知当前服务器的Leader信息，对于该机器而言，仅仅需要和Leader机器建立起连接，并进行状态同步即可。而在集群中不存在Leader情况下则会相对复杂，其步骤如下

(1) 第一次投票。无论哪种导致进行Leader选举，集群的所有机器都处于试图选举出一个Leader的状态，即LOOKING状态，LOOKING机器会向所有其他机器发送消息，该消息称为投票。投票中包含了SID（服务器的唯一标识）和ZXID（事务ID），(SID, ZXID)形式来标识一次投票信息。假定Zookeeper由5台机器组成，SID分别为1、2、3、4、5，ZXID分别为9、9、9、8、8，并且此时SID为2的机器是Leader机器，某一时刻，1、2所在机器出现故障，因此集群开始进行Leader选举。在第一次投票时，每台机器都会将自己作为投票对象，于是SID为3、4、5的机器投票情况分别为(3, 9), (4, 8), (5, 8)。

(2) 变更投票。每台机器发出投票后，也会收到其他机器的投票，每台机器会根据一定规则来处理收到的其他机器的投票，并以此来决定是否需要变更自己的投票，这个规则也是整个Leader选举算法的核心所在，其中术语描述如下

- vote_sid：接收到的投票中所推举Leader服务器的SID。
- vote_zxid：接收到的投票中所推举Leader服务器的ZXID。
- self_sid：当前服务器自己的SID。
- self_zxid：当前服务器自己的ZXID。

每次对收到的投票的处理，都是对(vote_sid, vote_zxid)和(self_sid, self_zxid)对比的过程。

- 规则一：如果vote_zxid大于self_zxid，就认可当前收到的投票，并再次将该投票发送出去。
- 规则二：如果vote_zxid小于self_zxid，那么坚持自己的投票，不做任何变更。
- 规则三：如果vote_zxid等于self_zxid，那么就对比两者的SID，如果vote_sid大于self_sid，那么就认可当前收到的投票，并再次将该投票发送出去。
- 规则四：如果vote_zxid等于self_zxid，并且vote_sid小于self_sid，那么坚持自己的投票，不做任何变更。

结合上面规则，给出下面的集群变更过程。

(3) 确定Leader。经过第二轮投票后，集群中的每台机器都会再次接收到其他机器的投票，然后开始统计投票，如果一台机器收到了超过半数的相同投票，那么这个投票对应的SID机器即为Leader。此时Server3将成为Leader。

由上面规则可知，通常那台服务器上的数据越新（ZXID会越大），其成为Leader的可能性越大，也就越能够保证数据的恢复。如果ZXID相同，则SID越大机会越大。

2.3 Leader选举实现细节

1. 服务器状态

服务器具有四种状态，分别是LOOKING、FOLLOWING、LEADING、OBSERVING。

LOOKING：寻找Leader状态。当服务器处于该状态时，它会认为当前集群中没有Leader，因此需要进入Leader选举状态。

FOLLOWING：跟随者状态。表明当前服务器角色是Follower。

LEADING：领导者状态。表明当前服务器角色是Leader。

OBSERVING：观察者状态。表明当前服务器角色是Observer。

2. 投票数据结构

每个投票中包含了两个最基本的信息，所推举服务器的SID和ZXID，投票（Vote）在Zookeeper中包含字段如下

id: 被推举的Leader的SID。

zxid: 被推举的Leader事务ID。

electionEpoch: 逻辑时钟, 用来判断多个投票是否在同一轮选举周期中, 该值在服务端是一个自增序列, 每次进入新一轮的投票后, 都会对该值进行加1操作。

peerEpoch: 被推举的Leader的epoch。

state: 当前服务器的状态。

3. QuorumCnxManager: 网络I/O

每台服务器在启动的过程中, 会启动一个QuorumPeerManager, 负责各台服务器之间的底层Leader选举过程中的网络通信。

(1) 消息队列。 QuorumCnxManager内部维护了一系列的队列, 用来保存接收到的、待发送的消息以及消息的发送器, 除接收队列以外, 其他队列都按照SID分组形成队列集合, 如一个集群中除了自身还有3台机器, 那么就会为这3台机器分别创建一个发送队列, 互不干扰。

- **recvQueue**: 消息接收队列, 用于存放那些从其他服务器接收到的消息。
- **queueSendMap**: 消息发送队列, 用于保存那些待发送的消息, 按照SID进行分组。
- **senderWorkerMap**: 发送器集合, 每个SenderWorker消息发送器, 都对应一台远程Zookeeper服务器, 负责消息的发送, 也按照SID进行分组。
- **lastMessageSent**: 最近发送过的消息, 为每个SID保留最近发送过的一个消息。

(2) 建立连接。 为了能够相互投票, Zookeeper集群中的所有机器都需要两两建立起网络连接。

QuorumCnxManager在启动时会创建一个ServerSocket来监听Leader选举的通信端口(默认为3888)。开启监听后, Zookeeper能够不断地接收到来自其他服务器的创建连接请求, 在接收到其他服务器的TCP连接请求时, 会进行处理。为了避免两台机器之间重复地创建TCP连接, Zookeeper只允许SID大的服务器主动和其他机器建立连接, 否则断开连接。在接收到创建连接请求后, 服务器通过对比自己和远程服务器的SID值来判断是否接收连接请求, 如果当前服务器发现自己的SID更大, 那么会断开当前连接, 然后自己主动和远程服务器建立连接。一旦连接建立, 就会根据远程服务器的SID来创建相应的消息发送器SendWorker和消息接收器RecvWorker, 并启动。

(3) 消息接收与发送。 消息接收: 由消息接收器RecvWorker负责, 由于Zookeeper为每个远程服务器都分配一个单独的RecvWorker, 因此, 每个RecvWorker只需要不断地从这个TCP连接中读取消息, 并将其保存到recvQueue队列中。消息发送: 由于Zookeeper为每个远程服务器都分配一个单独的SendWorker, 因此, 每个SendWorker只需要不断地从对应的消息发送队列中获取出一个消息发送即可, 同时将这个消息放入lastMessageSent中。在SendWorker中, 一旦Zookeeper发现针对当前服务器的消息发送队列为空, 那么此时需要从lastMessageSent中取出一个最近发送过的一个消息来进行再次发送, 这是为了解决接收方在消息接收前或者接收到消息后服务器挂了, 导致消息尚未被正确处理。同时, Zookeeper能够保证接收方在处理消息时, 会对重复消息进行正确的处理。

4. FastLeaderElection: 选举算法核心

- **外部投票**: 特指其他服务器发来的投票。
- **内部投票**: 服务器自身当前的投票。
- **选举轮次**: Zookeeper服务器Leader选举的轮次, 即logicalclock。
- **PK**: 对内部投票和外部投票进行对比来确定是否需要变更内部投票。

(1) 选票管理

- **sendqueue**: 选票发送队列, 用于保存待发送的选票。
- **recvqueue**: 选票接收队列, 用于保存接收到的外部投票。
- **WorkerReceiver**: 选票接收器。其会不断地从QuorumCnxManager中获取其他服务器发来的选举消息, 并将其转换成一个选票, 然后保存到recvqueue中, 在选票接收过程中, 如果发现该外部选票的选举轮次小于当前服务器的, 那么忽略该外部投票, 同时立即发送自己的内部投票。

- WorkerSender：选票发送器，不断地从sendqueue中获取待发送的选票，并将其传递到底层QuorumCnxManager中。

(2) 算法核心

上图展示了FastLeaderElection模块是如何与底层网络I/O进行交互的。Leader选举的基本流程如下

- 1. 自增选举轮次。** Zookeeper规定所有有效的投票都必须在同一轮次中，在开始新一轮投票时，会首先对logicalclock进行自增操作。
- 2. 初始化选票。** 在开始进行新一轮投票之前，每个服务器都会初始化自身的选票，并且在初始化阶段，每台服务器都会将自己推举为Leader。
- 3. 发送初始化选票。** 完成选票的初始化后，服务器就会发起第一次投票。Zookeeper会将刚刚初始化好的选票放入sendqueue中，由发送器WorkerSender负责发送出去。
- 4. 接收外部投票。** 每台服务器会不断地从recvqueue队列中获取外部选票。如果服务器发现无法获取到任何外部投票，那么就会立即确认自己是否和集群中其他服务器保持着有效的连接，如果没有连接，则马上建立连接，如果已经建立了连接，则再次发送自己当前的内部投票。
- 5. 判断选举轮次。** 在发送完初始化选票之后，接着开始处理外部投票。在处理外部投票时，会根据选举轮次来进行不同的处理。
 - 外部投票的选举轮次大于内部投票。若服务器自身的选举轮次落后于该外部投票对应服务器的选举轮次，那么就会立即更新自己的选举轮次(logicalclock)，并且清空所有已经收到的投票，然后使用初始化的投票来进行PK以确定是否变更内部投票。最终再将内部投票发送出去。
 - 外部投票的选举轮次小于内部投票。若服务器接收的外选票的选举轮次落后于自身的选举轮次，那么Zookeeper就会直接忽略该外部投票，不做任何处理，并返回步骤4。
 - 外部投票的选举轮次等于内部投票。此时可以开始进行选票PK。
- 6. 选票PK。** 在进行选票PK时，符合任意一个条件就需要变更投票。
 - 若外部投票中推举的Leader服务器的选举轮次大于内部投票，那么需要变更投票。
 - 若选举轮次一致，那么就对比两者的ZXID，若外部投票的ZXID大，那么需要变更投票。
 - 若两者的ZXID一致，那么就对比两者的SID，若外部投票的SID大，那么就需要变更投票。
- 7. 变更投票。** 经过PK后，若确定了外部投票优于内部投票，那么就变更投票，即使用外部投票的选票信息来覆盖内部投票，变更完成后，再次将这个变更后的内部投票发送出去。
- 8. 选票归档。** 无论是否变更了投票，都会将刚刚收到的那份外部投票放入选票集合recvset中进行归档。recvset用于记录当前服务器在本轮次的Leader选举中收到的所有外部投票（按照服务队的SID区别，如{(1, vote1), (2, vote2)...}）。
- 9. 统计投票。** 完成选票归档后，就可以开始统计投票，统计投票是为了统计集群中是否已经有过半的服务器认可了当前的内部投票，如果确定已经有过半服务器认可了该投票，则终止投票。否则返回步骤4。
- 10. 更新服务器状态。** 若已经确定可以终止投票，那么就开始更新服务器状态，服务器首选判断当前被过半服务器认可的投票所对应的Leader服务器是否是自己，若是自己，则将自己的服务器状态更新为LEADING，若不是，则根据具体情况来确定自己是FOLLOWING或是OBSERVING。

以上10个步骤就是FastLeaderElection的核心，其中步骤4-9会经过几轮循环，直到有Leader选举产生。

16. 数据同步

整个集群完成Leader选举之后，Learner（Follower和Observer的统称）回向Leader服务器进行注册。当Learner服务器想Leader服务器完成注册后，进入数据同步环节。

数据同步流程：（均以消息传递的方式进行）

i. Learner向Leader注册

ii. 数据同步

iii. 同步确认

Zookeeper的数据同步通常分为四类：

直接差异化同步 (DIFF同步)

先回滚再差异化同步 (TRUNC+DIFF同步)

仅回滚同步 (TRUNC同步)

全量同步 (SNAP同步)

在进行数据同步前，Leader服务器会完成数据同步初始化：

peerLastZxid：从learner服务器注册时发送的ACKEPOCH消息中提取lastZxid（该Learner服务器最后处理的ZXID）

minCommittedLog：Leader服务器Proposal缓存队列committedLog中最小ZXID

maxCommittedLog：Leader服务器Proposal缓存队列committedLog中最大ZXID

直接差异化同步 (DIFF同步)

场景：peerLastZxid介于minCommittedLog和maxCommittedLog之间

先回滚再差异化同步 (TRUNC+DIFF同步)

场景：当新的Leader服务器发现某个Learner服务器包含了一条自己没有的事务记录，那么就需要让该Learner服务器进行事务回滚-回滚到Leader服务器上存在的，同时也是最接近于peerLastZxid的ZXID
仅回滚同步 (TRUNC同步)

场景：peerLastZxid 大于 maxCommittedLog

全量同步 (SNAP同步)

场景一：peerLastZxid 小于 minCommittedLog

场景二：Leader服务器上没有Proposal缓存队列且peerLastZxid不等于lastProcessZxid

17. zookeeper是如何保证事务的顺序一致性的？

zookeeper采用了全局递增的事务Id来标识，所有的proposal（提议）都在被提出的时候加上了zxid，zxid实际上是一个64位的数字，高32位是epoch（时期；纪元；世；新时代）用来标识leader周期，如果有新的leader产生出来，epoch会自增，低32位用来递增计数。当新产生proposal的时候，会依据数据库的两阶段过程，首先会向其他的server发出事务执行请求，如果超过半数的机器都能执行并且能够成功，那么就会开始执行。

18. 分布式集群中为什么会有Master？

在分布式环境中，有些业务逻辑只需要集群中的某一台机器进行执行，其他的机器可以共享这个结果，这样可以大大减少重复计算，提高性能，于是就需要进行leader选举。

19. zk节点宕机如何处理？

Zookeeper本身也是集群，推荐配置不少于3个服务器。Zookeeper自身也要保证当一个节点宕机时，其他节点会继续提供服务。

如果是一个Follower宕机，还有2台服务器提供访问，因为Zookeeper上的数据是有多个副本的，数据并不会丢失；

如果是一个Leader宕机，Zookeeper会选举出新的Leader。

ZK集群的机制是只要超过半数的节点正常，集群就能正常提供服务。只有在ZK节点挂得太多，只剩一半或不到一半节点能工作，集群才失效。

所以

3个节点的cluster可以挂掉1个节点(leader可以得到2票>1.5)

2个节点的cluster就不能挂掉任何1个节点了(leader可以得到1票<=1)

20. zookeeper负载均衡和nginx负载均衡区别

zk的负载均衡是可以调控，nginx只是能调权重，其他需要可控的都需要自己写插件；但是nginx的吞吐量比zk大很多，应该说按业务选择用哪种方式。

21. Zookeeper有哪几种几种部署模式？

部署模式：单机模式、伪集群模式、集群模式。

22. 集群最少要几台机器，集群规则是怎样的？

集群规则为 $2N+1$ 台， $N>0$ ，即3台。

23. 集群支持动态添加机器吗？

其实就是水平扩容了，Zookeeper在这方面不太好。两种方式：

全部重启：关闭所有Zookeeper服务，修改配置之后启动。不影响之前客户端的会话。

逐个重启：在过半存活即可用的原则下，一台机器重启不影响整个集群对外提供服务。这是比较常用的方式。

3.5版本开始支持动态扩容。

24. Zookeeper对节点的watch监听通知是永久的吗？为什么不是永久的？

不是。官方声明：一个Watch事件是一个一次性的触发器，当被设置了Watch的数据发生了改变的时候，则服务器将这个改变发送给设置了Watch的客户端，以便通知它们。

为什么不是永久的，举个例子，如果服务端变动频繁，而监听的客户端很多情况下，每次变动都要通知到所有的客户端，给网络和服务器造成很大压力。

一般是客户端执行getData("/节点A",true)，如果节点A发生了变更或删除，客户端会得到它的watch事件，但是在之后节点A又发生了变更，而客户端又没有设置watch事件，就不再给客户端发送。

在实际应用中，很多情况下，我们的客户端不需要知道服务端的每一次变动，我只要最新的数据即可。

25. Zookeeper的java客户端都有哪些？

java客户端：zk自带的zkclient及Apache开源的Curator。

26. chubby是什么，和zookeeper比你怎么看？

chubby是google的，完全实现paxos算法，不开源。zookeeper是chubby的开源实现，使用zab协议，paxos算法的变种。

27. 说几个zookeeper常用的命令。

常用命令：ls get set create delete等。

28. ZAB和Paxos算法的联系与区别？

相同点：

两者都存在一个类似于Leader进程的角色，由其负责协调多个Follower进程的运行

Leader进程都会等待超过半数的Follower做出正确的反馈后，才会将一个提案进行提交

ZAB协议中，每个Proposal中都包含一个 epoch 值来代表当前的Leader周期，Paxos中名字为Ballot

不同点：

ZAB用来构建高可用的分布式数据主备系统（Zookeeper），Paxos是用来构建分布式一致性状态机系统。

29. Zookeeper的典型应用场景

Zookeeper是一个典型的发布/订阅模式的分布式数据管理与协调框架，开发人员可以使用它来进行分布式数据的发布和订阅。

通过对Zookeeper中丰富的数据节点进行交叉使用，配合Watcher事件通知机制，可以非常方便的构建一系列分布式应用中都会涉及的核心功能，如：

- 数据发布/订阅
- 负载均衡
- 命名服务
- 分布式协调/通知
- 集群管理
- Master选举
- 分布式锁
- 分布式队列

29.1 数据发布/订阅

介绍

数据发布/订阅系统，即所谓的配置中心，顾名思义就是发布者发布数据供订阅者进行数据订阅。

目的

- 动态获取数据（配置信息）
- 实现数据（配置信息）的集中式管理和数据的动态更新

设计模式

- Push 模式
- Pull 模式

数据（配置信息）特性

- 数据量通常比较小
- 数据内容在运行时会发生动态更新
- 集群中各机器共享，配置一致

如：机器列表信息、运行时开关配置、数据库配置信息等

基于Zookeeper的实现方式

- 数据存储：将数据（配置信息）存储到Zookeeper上的一个数据节点
- 数据获取：应用在启动初始化节点从Zookeeper数据节点读取数据，并在该节点上注册一个数据变更Watcher
- 数据变更：当变更数据时，更新Zookeeper对应节点数据，Zookeeper会将数据变更通知发到各客户端，客户端接到通知后重新读取变更后的数据即可。

29.2 负载均衡

zk的命名服务

命名服务是指通过指定的名字来获取资源或者服务的地址，利用zk创建一个全局的路径，这个路径就可以作为一个名字，指向集群中的集群，提供的服务的地址，或者一个远程的对象等等。

分布式通知和协调

对于系统调度来说：操作人员发送通知实际是通过控制台改变某个节点的状态，然后zk将这些变化发送给注册了这个节点的watcher的所有客户端。

对于执行情况汇报：每个工作进程都在某个目录下创建一个临时节点。并携带工作的进度数据，这样汇总的进程可以监控目录子节点的变化获得工作进度的实时的全局情况。

29.3 zk的命名服务（文件系统）

命名服务是指通过指定的名字来获取资源或者服务的地址，利用zk创建一个全局的路径，即是唯一的路径，这个路径就可以作为一个名字，指向集群中的集群，提供的服务的地址，或者一个远程的对象等等。

29.4 zk的配置管理（文件系统、通知机制）

程序分布式的部署在不同的机器上，将程序的配置信息放在zk的znode下，当有配置发生改变时，也就是znode发生变化时，可以通过改变zk中某个目录节点的内容，利用watcher通知给各个客户端，从而更改配置。

29.5 Zookeeper集群管理（文件系统、通知机制）

所谓集群管理无在乎两点：是否有机器退出和加入、选举master。

对于第一点，所有机器约定在父目录下创建临时目录节点，然后监听父目录节点的子节点变化消息。一旦有机器挂掉，该机器与zookeeper的连接断开，其所创建的临时目录节点被删除，所有其他机器都收到通知：某个兄弟目录被删除，于是，所有人都知道：它上船了。

新机器加入也是类似，所有机器收到通知：新兄弟目录加入，highcount又有了，对于第二点，我们稍微改变一下，所有机器创建临时顺序编号目录节点，每次选取编号最小的机器作为master就好。

29.6 Zookeeper分布式锁（文件系统、通知机制）

有了zookeeper的一致性文件系统，锁的问题变得容易。锁服务可以分为两类，一个是保持独占，另一个是控制时序。

对于第一类，我们将zookeeper上的一个znode看作是一把锁，通过createznode的方式来实现。所有客户端都去创建/distribute_lock节点，最终成功创建的那个客户端也即拥有了这把锁。用完删除掉自己创建的distribute_lock节点就释放出锁。

对于第二类，/distribute_lock已经预先存在，所有客户端在它下面创建临时顺序编号目录节点，和选master一样，编号最小的获得锁，用完删除，依次方便。

29.7 获取分布式锁的流程

clipboard.png

在获取分布式锁的时候在locker节点下创建临时顺序节点，释放锁的时候删除该临时节点。客户端调用createNode方法在locker下创建临时顺序节点，

然后调用getChildren("locker")来获取locker下面的所有子节点，注意此时不用设置任何Watcher。客户端获取到所有的子节点path之后，如果发现自己创建的节点在所有创建的子节点序号最小，那么就认为该客户端获取到了锁。如果发现自己创建的节点并非locker所有子节点中最小的，说明自己还没有获取到锁，此时客户端需要找到比自己小的那个节点，然后对其调用exist()方法，同时对其注册事件监听器。之后，让这个被关注的节点删除，则客户端的Watcher会收到相应通知，此时再次判断自己创建的节点是否是locker子节点中序号最小的，如果是则获取到了锁，如果不是则重复以上步骤继续获取到比自己小的一个节点并注册监听。当前这个过程中还需要许多的逻辑判断。

clipboard.png

代码的实现主要是基于互斥锁，获取分布式锁的重点逻辑在于BaseDistributedLock，实现了基于Zookeeper实现分布式锁的细节。

29.8 Zookeeper队列管理（文件系统、通知机制）

两种类型的队列：

- 1、同步队列，当一个队列的成员都聚齐时，这个队列才可用，否则一直等待所有成员到达。
- 2、队列按照FIFO方式进行入队和出队操作。

第一类，在约定目录下创建临时目录节点，监听节点数目是否是我们要求的数目。

第二类，和分布式锁服务中的控制时序场景基本原理一致，入列有编号，出列按编号。在特定的目录下创建PERSISTENT_SEQUENTIAL节点，创建成功时Watcher通知等待的队列，队列删除序列号最小的节点用以消费。此场景下Zookeeper的znode用于消息存储，znode存储的数据就是消息队列中的消息内容，SEQUENTIAL序列号就是消息的编号，按序取出即可。由于创建的节点是持久化的，所以不必担心队列消息的丢失问题。

RabbitMQ常见面试题及答案

1.rabbitmq 的使用场景有哪些？

- ①. 跨系统的异步通信，所有需要异步交互的地方都可以使用消息队列。就像我们除了打电话（同步）以外，还需要发短信，发电子邮件（异步）的通讯方式。
- ②. 多个应用之间的耦合，由于消息是平台无关和语言无关的，而且语义上也不再是函数调用，因此更适合作为多个应用之间的松耦合的接口。基于消息队列的耦合，不需要发送方和接收方同时在线。在企业应用集成（EAI）中，文件传输，共享数据库，消息队列，远程过程调用都可以作为集成的方法。
- ③. 应用内的同步变异步，比如订单处理，就可以由前端应用将订单信息放到队列，后端应用从队列里依次获得消息处理，高峰时的大量订单可以积压在队列里慢慢处理掉。由于同步通常意味着阻塞，而大量线程的阻塞会降低计算机的性能。
- ④. 消息驱动的架构（EDA），系统分解为消息队列，和消息制造者和消息消费者，一个处理流程可以根据需要拆成多个阶段（Stage），阶段之间用队列连接起来，前一个阶段处理的结果放入队列，后一个阶段从队列中获取消息继续处理。
- ⑤. 应用需要更灵活的耦合方式，如发布订阅，比如可以指定路由规则。
- ⑥. 跨局域网，甚至跨城市的通讯（CDN行业），比如北京机房与广州机房的应用程序的通信。

2.rabbitmq有哪些重要的角色？

RabbitMQ 中重要的角色有：生产者、消费者和代理：

生产者：消息的创建者，负责创建和推送数据到消息服务器；

消费者：消息的接收方，用于处理数据和确认消息；

代理：就是 RabbitMQ 本身，用于扮演“快递”的角色，本身不生产消息，只是扮演“快递”的角色。

3.rabbitmq 有哪些重要的组件？

ConnectionFactory（连接管理器）：应用程序与Rabbit之间建立连接的管理器，程序代码中使用。

Channel（信道）：消息推送使用的通道。

Exchange（交换器）：用于接受、分配消息。

Queue（队列）：用于存储生产者的消息。

RoutingKey（路由键）：用于把生成者的数据分配到交换器上。

BindingKey（绑定键）：用于把交换器的消息绑定到队列上。

4.rabbitmq 中 vhost 的作用是什么？

vhost 可以理解为虚拟 broker，即 mini-RabbitMQ server。其内部均含有独立的 queue、exchange 和 binding 等，但最重要的是，其拥有独立的权限系统，可以做到 vhost 范围的用户控制。当然，从 RabbitMQ 的全局角度，vhost 可以作为不同权限隔离的手段（一个典型的例子就是不同的应用可以跑在不同的 vhost 中）。

5.rabbitmq 怎么保证消息的稳定性？

提供了事务的功能。

通过将 channel 设置为 confirm (确认) 模式。

6.rabbitmq 怎么避免消息丢失？

消息持久化

ACK确认机制

设置集群镜像模式

消息补偿机制

7.要保证消息持久化成功的条件有哪些？

声明队列必须设置持久化 durable 设置为 true.

消息推送投递模式必须设置持久化，deliveryMode 设置为 2 (持久)。

消息已经到达持久化交换器。

消息已经到达持久化队列。

以上四个条件都满足才能保证消息持久化成功。

8.rabbitmq持久化有什么缺点？

持久化的缺点就是降低了服务器的吞吐量，因为使用的是磁盘而非内存存储，从而降低了吞吐量。可尽量使用 ssd 硬盘来缓解吞吐量的问题。

9.rabbitmq有几种广播类型？

三种广播模式：

fanout: 所有bind到此exchange的queue都可以接收消息（纯广播，绑定到RabbitMQ的接受者都能收到消息）；

direct: 通过routingKey和exchange决定的那个唯一的queue可以接收消息；

topic:所有符合routingKey(此时可以是一个表达式)的routingKey所bind的queue可以接收消息；

10.rabbitmq怎么实现延迟消息队列？

通过消息过期后进入死信交换器，再由交换器转发到延迟消费队列，实现延迟功能；

使用 RabbitMQ-delayed-message-exchange 插件实现延迟功能。

11.rabbitmq 集群有什么用？

集群主要有以下两个用途：

高可用：某个服务器出现问题，整个 RabbitMQ 还可以继续使用；

高容量：集群可以承载更多的消息量。

12.rabbitmq节点的类型有哪些？

磁盘节点：消息会存储到磁盘。

内存节点：消息都存储在内存中，重启服务器消息丢失，性能高于磁盘类型。

13.rabbitmq集群搭建需要注意哪些问题？

各节点之间使用“-link”连接，此属性不能忽略。

各节点使用的 erlang cookie 值必须相同，此值相当于“秘钥”的功能，用于各节点的认证。

整个集群中必须包含一个磁盘节点。

14.rabbitmq 每个节点是其他节点的完整拷贝吗？为什么？

不是，原因有以下两个：

存储空间的考虑：如果每个节点都拥有所有队列的完全拷贝，这样新增节点不但没有新增存储空间，反而增加了更多的冗余数据；

性能的考虑：如果每条消息都需要完整拷贝到每一个集群节点，那新增节点并没有提升处理消息的能力，最多是保持和单节点相同的性能甚至是更糟。

15.rabbitmq集群中唯一一个磁盘节点崩溃了会发生什么情况？

如果唯一磁盘的磁盘节点崩溃了，不能进行以下操作：

不能创建队列

不能创建交换器

不能创建绑定

不能添加用户

不能更改权限

不能添加和删除集群节点

唯一磁盘节点崩溃了，集群是可以保持运行的，但你不能更改任何东西。

16.rabbitmq 对集群节点停止顺序有要求吗？

RabbitMQ 对集群的停止的顺序是有要求的，应该先关闭内存节点，最后再关闭磁盘节点。如果顺序恰好相反的话，可能会造成消息的丢失。

Elasticsearch常见面试题及答案

1、Elasticsearch能够实现快速搜索的原因

ES的核心是倒排索引(其他搜索引擎也类似)；并且基于倒排索引，充分利用了缓存。这是可以快速搜索的主要原因。

另外对于搜索请求是分布式执行的，由多台机器同时执行，然后汇总，这也是可以快速搜索的一个主要原因。

2、什么是倒排索引

倒排索引是类似于哈希表一样的数据结构，完成由词条（也称为term/token）到文档id的映射。可以根据搜索词条快速找到该词条对应的所有文档id，然后根据id直接获取文档。

3、分析器

主要功能是完成需要分析的字段值到词条标准化的过程。主要由字符过滤器(0个或多个)、分词器、词条过滤器构成。

字符过滤器：主要在分词之前整理字符串，比如去掉html字符，完成&到and的转换等；

分词器：主要功能是将字符串分割成词条；

过滤器：每个词条按顺序通过词条过滤器，完成词条的标准化，如：统一为小写，增加一些同义词词条，去除无用词条等。

4、详细描述下索引数据写入的过程

协调节点默认使用文档id计算分片所在位置，然后将请求转发到相应分片；默认计算方式：

`shard = hash(document_id) % (num_of_primary_shards)`

分片接收到请求之后首先将数据写入到index-buffer，同时记录translog；默认情况下这个过程在主分片与副本分片都完成之后，再给客户端返回写入成功的响应。可以配置，写入大于n个分片之后就返回到客户端写入成功。

每隔一定时间就会写入到filesystem-cache (refresh过程)。

每隔一段时间或者filesystem-cache数据量到一定程度之后，就会写入到磁盘 (flush)。

如果两次flush直接出现异常，可以通过translog恢复数据。

特别说明为了提高写数据性能，可以将写translog设置为异步写，但异常情况下可能会造成少量数据丢失。

5、详细描述一下 Elasticsearch 更新和删除文档的过程。

(1) 删除和更新也都是写操作，但是 Elasticsearch 中的文档是不可变的，因此不能被删除或者改动以展示其变更；

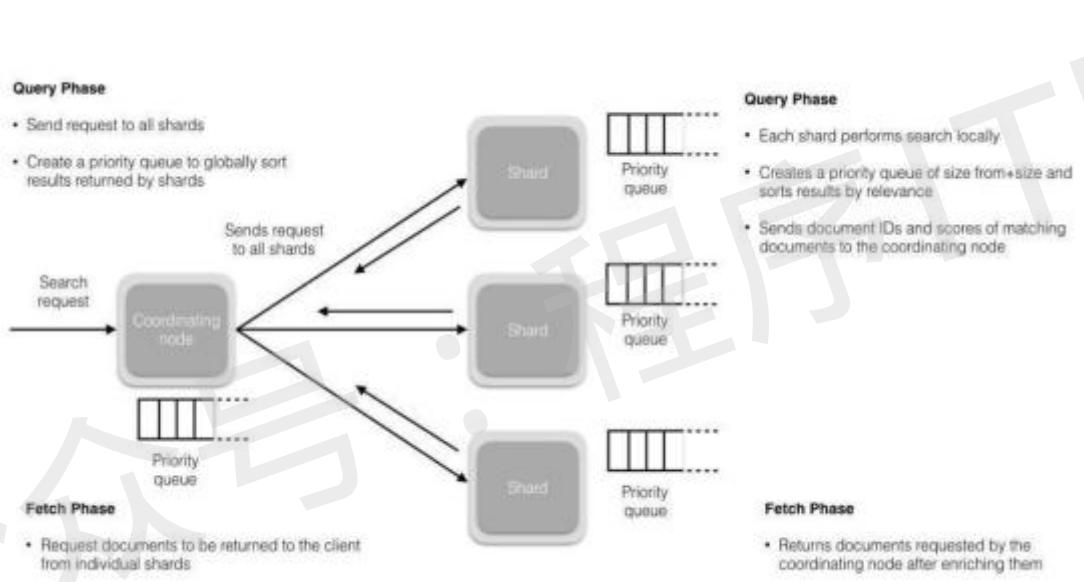
(2) 磁盘上的每个段都有一个相应的.del 文件。当删除请求发送后，文档并没有真的被删除，而是在.del 文件中被标记为删除。该文档依然能匹配查询，但是会在结果中被过滤掉。当段合并时，在.del 文件中被标记为删除的文档将不会被写入新段。

(3) 在新的文档被创建时，Elasticsearch 会为该文档指定一个版本号，当执行更新时，旧版本的文档在.del 文件中被标记为删除，新版本的文档被索引到一个新段。旧版本的文档依然能匹配查询，但是会在结果中被过滤掉。

6、详细描述一下 Elasticsearch 搜索的过程。

- (1) 搜索被执行成一个两阶段过程，我们称之为 Query Then Fetch；
- (2) 在初始查询阶段时，查询会广播到索引中每一个分片拷贝（主分片或者副本分片）。每个分片在本地执行搜索并构建一个匹配文档的大小为 from + size 的优先队列。

PS：在搜索的时候是会查询 Filesystem Cache 的，但是有部分数据还在 Memory Buffer，所以搜索是近实时的。
- (3) 每个分片返回各自优先队列中所有文档的 ID 和排序值给协调节点，它合并这些值到自己的优先队列中来产生一个全局排序后的结果列表。
- (4) 接下来就是 取回阶段，协调节点辨别出哪些文档需要被取回并向相关的分片提交多个 GET 请求。每个分片加载并丰富文档，如果有需要的话，接着返回文档给协调节点。一旦所有的文档都被取回了，协调节点返回结果给客户端。
- (5) 补充：Query Then Fetch 的搜索类型在文档相关性打分的时候参考的是本分片的数据，这样在文档数量较少的时候可能不够准确，DFS Query Then Fetch 增加了一个预查询的处理，询问 Term 和 Document frequency，这个评分更准确，但是性能会变差。*



7、在 Elasticsearch 中，是怎么根据一个词找到对应的倒排索引的？

- (1) Lucene的索引过程，就是按照全文检索的基本过程，将倒排表写成此文件格式的过程。
- (2) Lucene的搜索过程，就是按照此文件格式将索引进去的信息读出来，然后计算每篇文档打分(score)的过程。

8、Elasticsearch 在部署时，对 Linux 的设置有哪些优化方法？

- (1) 64 GB 内存的机器是非常理想的，但是 32 GB 和 16 GB 机器也是很常见的。少于 8 GB 会适得其反。
- (2) 如果你要在更快的 CPUs 和更多的核心之间选择，选择更多的核心更好。多个内核提供的额外并发远胜过稍微快一点点的时钟频率。

(3) 如果你负担得起 SSD，它将远远超出任何旋转介质。基于 SSD 的节点，查询和索引性能都有提升。如果你负担得起，SSD 是一个好的选择。

(4) 即使数据中心们近在咫尺，也要避免集群跨越多个数据中心。绝对要避免集群跨越大的地理距离。

(5) 请确保运行你应用程序的 JVM 和服务器的 JVM 是完全一样的。在Elasticsearch 的几个地方，使用 Java 的本地序列化。

(6) 通过设置 gateway.recover_after_nodes、gateway.expected_nodes、gateway.recover_after_time 可以在集群重启的时候避免过多的分片交换，这可能会让数据恢复从数个小时缩短为几秒钟。

(7) Elasticsearch 默认被配置为使用单播发现，以防止节点无意中加入集群。只有在同一台机器上运行的节点才会自动组成集群。最好使用单播代替组播。

(8) 不要随意修改垃圾回收器 (CMS) 和各个线程池的大小。

(9) 把你的内存的（少于）一半给 Lucene（但不要超过 32 GB！），通过ES_HEAP_SIZE 环境变量设置。

(10) 内存交换到磁盘对服务器性能来说是致命的。如果内存交换到磁盘上，一个100 微秒的操作可能变成 10 毫秒。再想想那么多 10 微秒的操作时延累加起来。不难看出 swapping 对于性能是多么可怕。

(11) Lucene 使用了大量的文件。同时，Elasticsearch 在节点和 HTTP 客户端之间进行通信也使用了大量的套接字。所有这一切都需要足够的文件描述符。你应该增加你的文件描述符，设置一个很大的值，如 64,000。

补充：索引阶段性能提升方法

(1) 使用批量请求并调整其大小：每次批量数据 5-15 MB 大是个不错的起始点。

(2) 存储：使用 SSD

(3) 段和合并：Elasticsearch 默认值是 20 MB/s，对机械磁盘应该是个不错的设置。如果你用的是 SSD，可以考虑提高到 100-200 MB/s。如果你在做批量导入，完全不在意搜索，你可以彻底关掉合并限流。另外还可以增加index.translog.flush_threshold_size 设置，从默认的 512 MB 到更大一些的值，比如 1 GB，这可以在一次清空触发的时候在事务日志里积累出更大的段。

(4) 如果你的搜索结果不需要近实时的准确度，考虑把每个索引的index.refresh_interval 改到 30s。

(5) 如果你在做大批量导入，考虑通过设置 index.number_of_replicas: 0 关闭副本。

9、对于 GC 方面，在使用 Elasticsearch 时要注意什么？

(1) 倒排词典的索引需要常驻内存，无法 GC，需要监控 data node 上 segmentmemory 增长趋势。

(2) 各类缓存，field cache, filter cache, indexing cache, bulk queue 等等，要设置合理的大小，并且要应该根据最坏的情况来看 heap 是否够用，也就是各类缓存全部占满的时候，还有 heap 空间可以分配给其他任务吗？避免采用 clear cache 等“自欺欺人”的方式来释放内存。

(3) 避免返回大量结果集的搜索与聚合。确实需要大量拉取数据的场景，可以采用scan & scroll api 来实现。

(4) cluster stats 驻留内存并无法水平扩展，超大规模集群可以考虑分拆成多个集群通过 tribe node 连接。

(5) 想知道 heap 够不够，必须结合实际应用场景，并对集群的 heap 使用情况做持续的监控。

(6) 根据监控数据理解内存需求，合理配置各类circuit breaker，将内存溢出风险降低到最低

10、Elasticsearch 对于大数据量（上亿量级）的聚合如何实现？

Elasticsearch 提供的首个近似聚合是 cardinality 度量。它提供一个字段的基数，即该字段的 distinct 或者 unique 值的数目。它是基于 HLL 算法的。HLL 会先对我们的输入作哈希运算，然后根据哈希运算的结果中的 bits 做概率估算从而得到基数。其特点是：可配置的精度，用来控制内存的使用（更精确 = 更多内存）；小的数据集精度是非常高的；我们可以通过配置参数，来设置去重需要的固定内存使用量。无论数千还是数十亿的唯一值，内存使用量只与你配置的精确度相关。

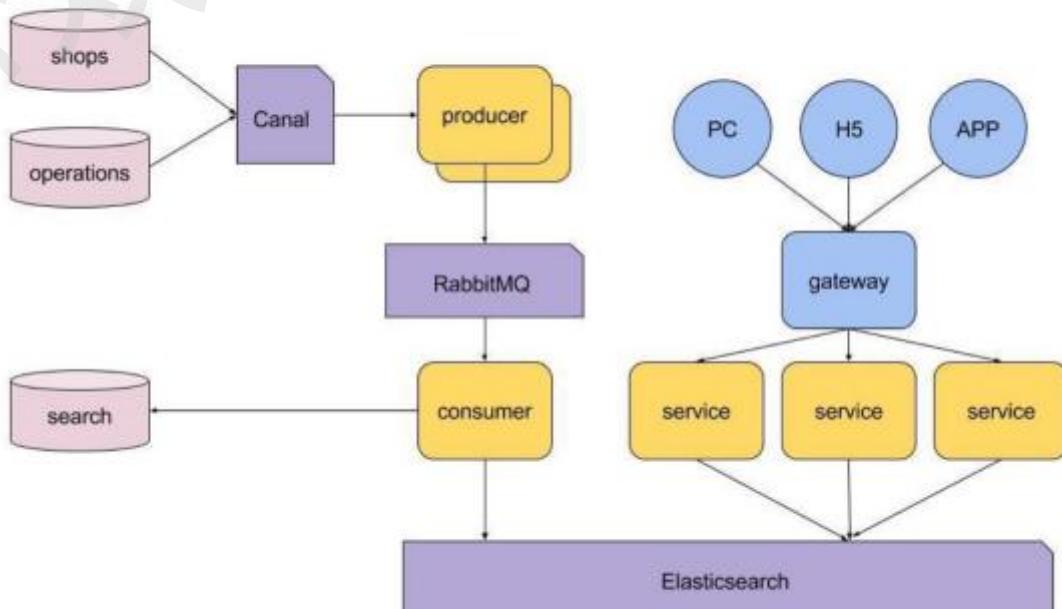
11、在并发情况下，Elasticsearch 如何保证读写一致？

- (1) 可以通过版本号使用乐观并发控制，以确保新版本不会被旧版本覆盖，由应用层来处理具体的冲突；
- (2) 另外对于写操作，一致性级别支持 quorum/one/all，默认为 quorum，即只有当大多数分片可用时才允许写操作。但即使大多数可用，也可能存在因为网络等原因导致写入副本失败，这样该副本被认为故障，分片将会在一个不同的节点上重建。
- (3) 对于读操作，可以设置 replication 为 sync(默认)，这使得操作在主分片和副本分片都完成后才会返回；如果设置 replication 为 async 时，也可以通过设置搜索请求参数 preference 为 primary 来查询主分片，确保文档是最新版本。

12、如何监控 Elasticsearch 集群状态？

Marvel 让你可以很简单的通过 Kibana 监控 Elasticsearch。你可以实时查看你的集群健康状态和性能，也可以分析过去的集群、索引和节点指标。

13、介绍下你们电商搜索的整体技术架构。



14、介绍一下你们的个性化搜索方案？

基于word2vec和Elasticsearch实现个性化搜索

- (1) 基于word2vec、Elasticsearch和自定义的脚本插件，我们就实现了一个个性化的搜索服务，相对于原有的实现，新版的点击率和转化率都有大幅的提升；
- (2) 基于word2vec的商品向量还有一个可用之处，就是可以用来实现相似商品的推荐；
- (3) 使用word2vec来实现个性化搜索或个性化推荐是有一定局限性的，因为它只能处理用户点击历史这样的时序数据，而无法全面的去考虑用户偏好，这个还是有很大的改进和提升的空间；

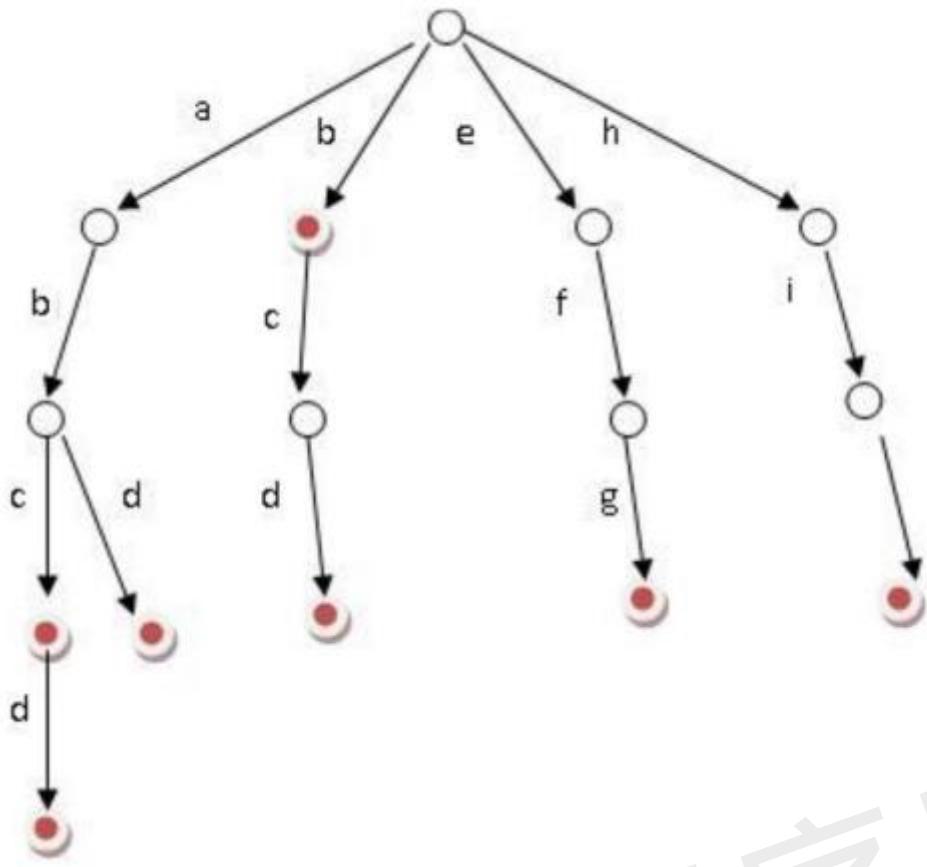
15、是否了解字典树？

常用字典数据结构如下所示：

数据结构	优缺点
排序列表Array/List	使用二分法查找，不平衡
HashMap/TreeMap	性能高，内存消耗大，几乎是原始数据的三倍
Skip List	跳跃表，可快速查找词语，在lucene、redis、Hbase等均有实现。相对于TreeMap等结构，特别适合高并发场景（ Skip List介绍 ）
Trie	适合英文词典，如果系统中存在大量字符串且这些字符串基本没有公共前缀，则相应的trie树将非常消耗内存（ 数据结构之trie树 ）
Double Array Trie	适合做中文词典，内存占用小，很多分词工具均采用此种算法（ 深入双数组Trie ）
Ternary Search Tree	三叉树，每一个node有3个节点，兼具省空间和查询快的优点（ Ternary Search Tree ）
Finite State Transducers (FST)	一种有限状态转移机，Lucene 4有开源实现，并大量使用

Trie 的核心思想是空间换时间，利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。它有 3 个基本性质：

- 1) 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
- 2) 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
- 3) 每个节点的所有子节点包含的字符都不相同。



(1) 可以看到，trie 树每一层的节点数是 26^i 级别的。所以为了节省空间，我们还可以用动态链表，或者用数组来模拟动态。而空间的花费，不会超过单词数×单词长度。

(2) 实现：对每个结点开一个字母集大小的数组，每个结点挂一个链表，使用左儿子右兄弟表示法记录这棵树；

(3) 对于中文的字典树，每个节点的子节点用一个哈希表存储，这样就不用浪费太大的空间，而且查询速度上可以保留哈希的复杂度 $O(1)$ 。

24、拼写纠错是如何实现的？

(1) 拼写纠错是基于编辑距离来实现；编辑距离是一种标准的方法，它用来表示经过插入、删除和替换操作从一个字符串转换到另外一个字符串的最小操作步数；

(2) 编辑距离的计算过程：比如要计算 batyu 和 beauty 的编辑距离，先创建一个 7×8 的表（batyu 长度为 5，coffee 长度为 6，各加 2），接着，在如下位置填入黑色数字。其他格的计算过程是取以下三个值的最小值：

如果最上方的字符等于最左方的字符，则为左上方的数字。否则为左上方的数字+1。（对于 3,3 来说为 0）

左方数字+1（对于 3,3 格来说为 2）

上方数字+1（对于 3,3 格来说为 2）

最终取右下角的值即为编辑距离的值 3。

		b	e	a	u	t	y	
	0	1	2	3	4	5	6	
b	1	0	1	2	3	4	5	
a	2	1	1	1	2	3	4	
t	3	2	2	2	2	2	3	
y	4	3	3	3	3	3	2	
u	5	4	4	4	3	4	3	

对于拼写纠错，我们考虑构造一个度量空间（Metric Space），该空间内任何关系满足以下三条基本条件：

$d(x,y) = 0$ -- 假如 x 与 y 的距离为 0，则 $x=y$

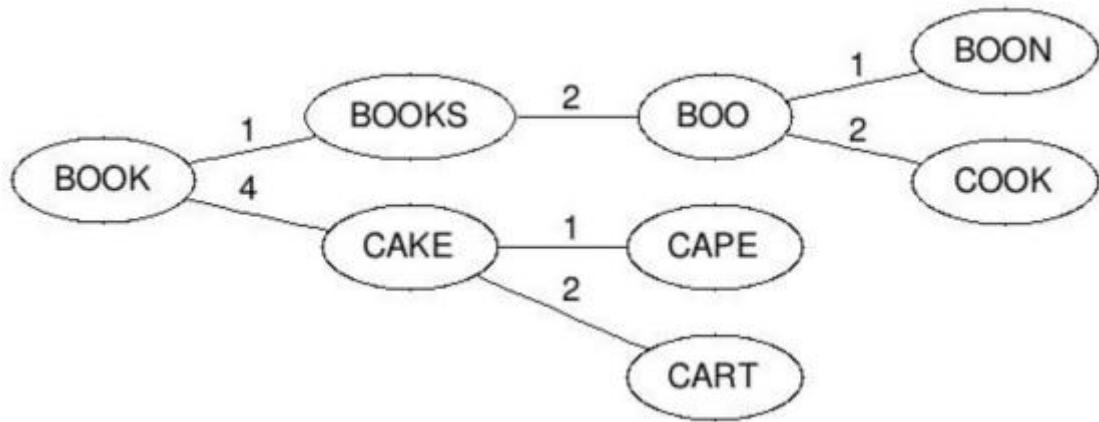
$d(x,y) = d(y,x)$ -- x 到 y 的距离等同于 y 到 x 的距离

$d(x,y) + d(y,z) \geq d(x,z)$ -- 三角不等式

(1) 根据三角不等式，则满足与 query 距离在 n 范围内的另一个字符转 B，其与 A 的距离最大为 $d+n$ ，最小为 $d-n$ 。

(2) BK 树的构造过程如下：每个节点有任意个子节点，每条边有个值表示编辑距离。所有子节点到父节点的边上标注 n 表示编辑距离恰好为 n 。比如，我们有棵树父节点是“book”和两个子节点“cake”和“books”，“book”到“books”的边标号 1，“book”到“cake”的边标号 4。从字典里构造好树后，无论何时你想插入新单词时，计算该单词与根节点的编辑距离，并且查找数值为 $d(\text{newword}, \text{root})$ 的边。递归得与各子节点进行比较，直到没有子节点，你就可以创建新的子节点并将新单词保存在那。比如，插入“boo”到刚才上述例子的树中，我们先检查根节点，查找 $d(\text{book}, \text{boo}) = 1$ 的边，然后检查标号为 1 的边的子节点，得到单词“books”。我们再计算距离 $d(\text{books}, \text{boo}) = 2$ ，则将新单词插在“books”之后，边标号为 2。

3、查询相似词如下：计算单词与根节点的编辑距离 d ，然后递归查找每个子节点标号为 $d-n$ 到 $d+n$ （包含）的边。假如被检查的节点与搜索单词的距离 d 小于 n ，则返回该节点并继续查询。比如输入“cape”且最大容忍距离为 1，则先计算和根的编辑距离 $d(\text{book}, \text{cape}) = 4$ ，然后接着找和根节点之间编辑距离为 3 到 5 的，这个就找到了“cake”这个节点，计算 $d(\text{cake}, \text{cape}) = 1$ ，满足条件所以返回“cake”，然后再找和“cake”节点编辑距离是 0 到 2 的，分别找到“cape”和“cart”节点，这样就得到“cape”这个满足条件的结果。

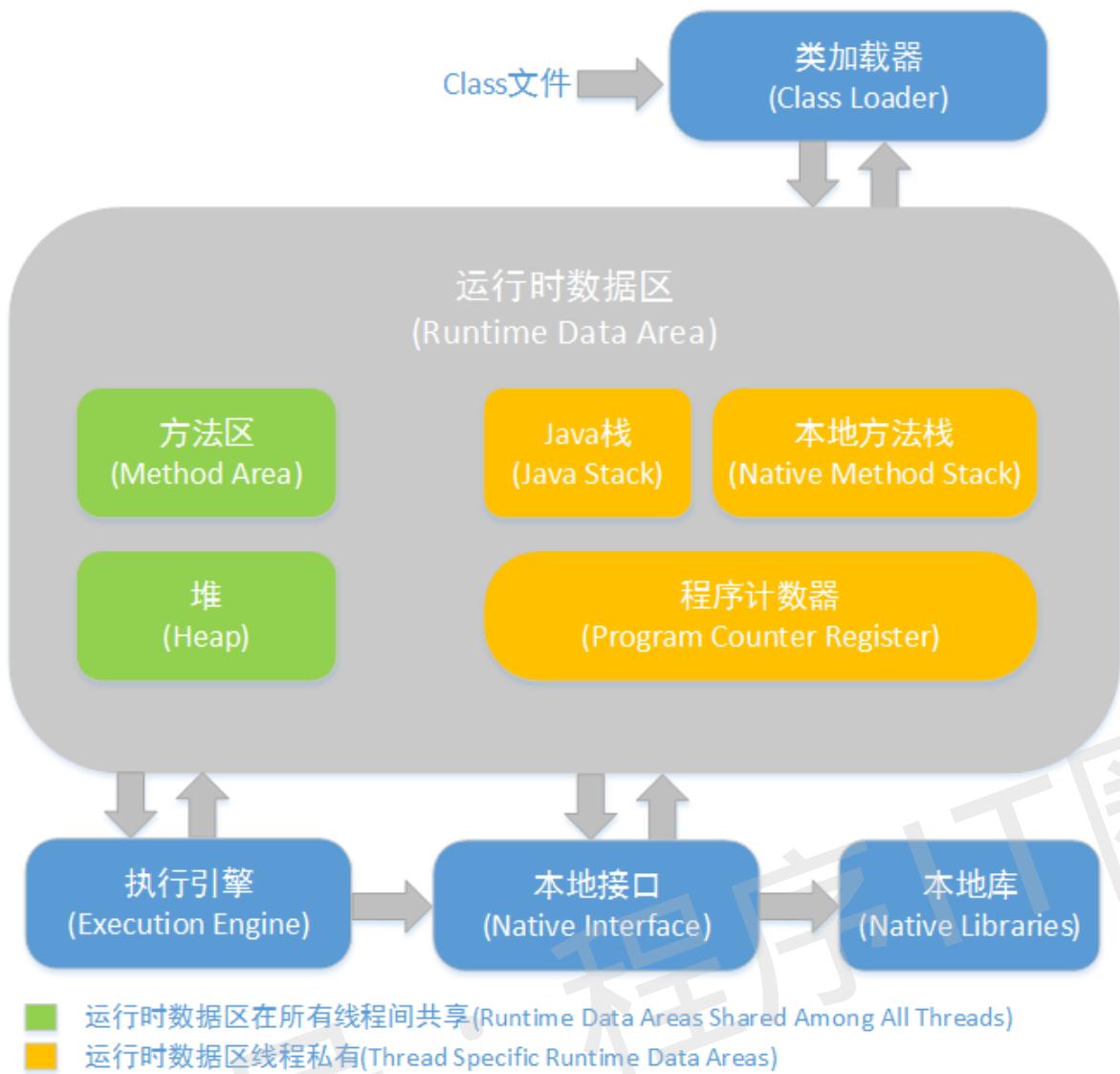


JVM常见24道面试题及答案

1.什么是Java虚拟机？为什么Java被称作是“平台无关的编程语言”？

Java虚拟机是一个可以执行Java字节码的虚拟机进程。Java源文件被编译成能被Java虚拟机执行的字节码文件。Java被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台单独重写或者是重新编译。Java虚拟机让这个变为可能，因为它知道底层硬件平台的指令长度和其他特性。

2.Java内存结构？



- **运行时数据区在所有线程间共享 (Runtime Data Areas Shared Among All Threads)**
- **运行时数据区线程私有 (Thread Specific Runtime Data Areas)**

方法区和堆是所有线程共享的内存区域；而Java栈、本地方法栈和程序员计数器是运行时线程私有的内存区域。

- Java堆 (Heap), 是Java虚拟机所管理的内存中最大的一块。Java堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。
- 方法区 (Method Area), 方法区 (Method Area) 与Java堆一样，是各个线程共享的内存区域，它用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。
- 程序计数器 (Program Counter Register), 程序计数器 (Program Counter Register) 是一块较小的内存空间，它的作用可以看做是当前线程所执行的字节码的行号指示器。
- JVM栈 (JVM Stacks), 与程序计数器一样，Java虚拟机栈 (Java Virtual Machine Stacks) 也是线程私有的，它的生命周期与线程相同。虚拟机栈描述的是Java方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈帧 (Stack Frame) 用于存储局部变量表、操作栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程，就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。
- 本地方法栈 (Native Method Stacks), 本地方法栈 (Native Method Stacks) 与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行Java方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的Native方法服务。

3. 解释内存中的栈(stack)、堆(heap)和方法区(method area)的用法

通常我们定义一个基本数据类型的变量，一个对象的引用，还有就是函数调用的现场保存都使用JVM中的栈空间；而通过new关键字和构造器创建的对象则放在堆空间，堆是垃圾收集器管理的主要区域，由于现在的垃圾收集器都采用分代收集算法，所以堆空间还可以细分为新生代和老生代，再具体一点可以分为Eden、Survivor（又可分为From Survivor和To Survivor）、Tenured；方法区和堆都是各个线程共享的内存区域，用于存储已经被JVM加载的类信息、常量、静态变量、JIT编译器编译后的代码等数据；程序中的字面量(literal)如直接书写的100、“hello”和常量都是放在常量池中，常量池是方法区的一部分。栈空间操作起来最快但是栈很小，通常大量的对象都是放在堆空间，栈和堆的大小都可以通过JVM的启动参数来进行调整，栈空间用光了会引发StackOverflowError，而堆和常量池空间不足则会引发OutOfMemoryError。

```
String str = new String("hello");
```

上面的语句中变量str放在栈上，用new创建出来的字符串对象放在堆上，而“hello”这个字面量是放在方法区的。

补充1：较新版本的Java（从Java 6的某个更新开始）中，由于JIT编译器的发展和“逃逸分析”技术的逐渐成熟，栈上分配、标量替换等优化技术使得对象一定分配在堆上这件事情已经变得不那么绝对了。

补充2：运行时常量池相当于Class文件常量池具有动态性，Java语言并不要求常量一定只有编译期间才能产生，运行期间也可以将新的常量放入池中，String类的intern()方法就是这样的。看看下面代码的执行结果是什么并且比较一下Java 7以前和以后的运行结果是否一致。

```
String s1 = new StringBuilder("go")
    .append("od").toString();
System.out.println(s1.intern() == s1);
String s2 = new StringBuilder("ja")
    .append("va").toString();
System.out.println(s2.intern() == s2);
```

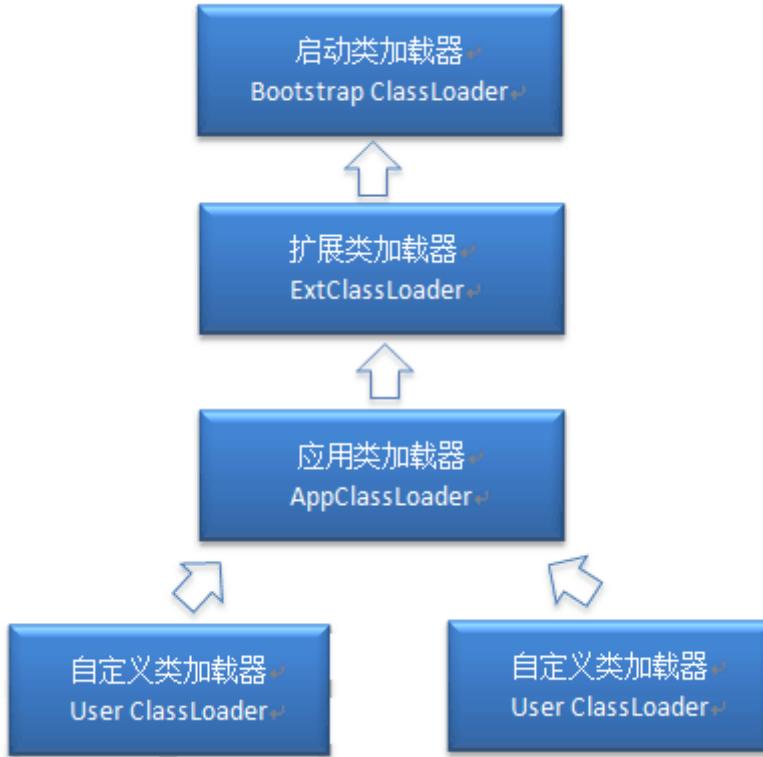
4. 对象分配规则

- 对象优先分配在Eden区，如果Eden区没有足够的空间时，虚拟机执行一次Minor GC。
- 大对象直接进入老年区（大对象是指需要大量连续内存空间的对象）。这样做的目的是避免在Eden区和两个Survivor区之间发生大量的内存拷贝（新生代采用复制算法收集内存）。
- 长期存活的对象进入老年区。虚拟机为每个对象定义了一个年龄计数器，如果对象经过了1次Minor GC那么对象会进入Survivor区，之后每经过一次Minor GC那么对象的年龄加1，知道达到阀值对象进入老年区。
- 动态判断对象的年龄。如果Survivor区中相同年龄的所有对象大小的总和大于Survivor空间的一半，年龄大于或等于该年龄的对象可以直接进入老年区。
- 空间分配担保。每次进行Minor GC时，JVM会计算Survivor区移至老年区的对象的平均大小，如果这个值大于老年区的剩余值大小则进行一次Full GC，如果小于检查HandlePromotionFailure设置，如果true则只进行Monitor GC,如果false则进行Full GC。

5. 什么是类的加载

类的加载指的是将类的.class文件中的二进制数据读入到内存中，将其放在运行时数据区的方法区内，然后在堆区创建一个java.lang.Class对象，用来封装类在方法区内的数据结构。类的加载的最终产品是位于堆区中的Class对象，Class对象封装了类在方法区内的数据结构，并且向Java程序员提供了访问方法区内的数据结构的接口。

6. 类加载器



- 启动类加载器：Bootstrap ClassLoader，负责加载存放在JDK\jre\lib(JDK代表JDK的安装目录，下同)下，或被-Xbootclasspath参数指定的路径中的，并且能被虚拟机识别的类库
- 扩展类加载器：Extension ClassLoader，该加载器由sun.misc.Launcher\$ExtClassLoader实现，它负责加载DK\jre\lib\ext目录中，或者由java.ext.dirs系统变量指定的路径中的所有类库（如javax.*开头的类），开发者可以直接使用扩展类加载器。
- 应用程序类加载器：Application ClassLoader，该类加载器由sun.misc.Launcher\$AppClassLoader来实现，它负责加载用户类路径（ClassPath）所指定的类，开发者可以直接使用该类加载器

7.描述一下JVM加载class文件的原理机制？

答：JVM中类的装载是由类加载器（ClassLoader）和它的子类来实现的，Java中的类加载器是一个重要的Java运行时系统组件，它负责在运行时查找和装入类文件中的类。由于Java的跨平台性，经过编译的Java源程序并不是一个可执行程序，而是一个或多个类文件。当Java程序需要使用某个类时，JVM会确保这个类已经被加载、连接（验证、准备和解析）和初始化。类的加载是指把类的.class文件中的数据读入到内存中，通常是创建一个字节数组读入.class文件，然后产生与所加载类对应的Class对象。加载完成后，Class对象还不完整，所以此时的类还不可用。当类被加载后就进入连接阶段，这一阶段包括验证、准备（为静态变量分配内存并设置默认的初始值）和解析（将符号引用替换为直接引用）三个步骤。最后JVM对类进行初始化，包括：1)如果类存在直接的父类并且这个类还没有被初始化，那么就先初始化父类；2)如果类中存在初始化语句，就依次执行这些初始化语句。类的加载是由类加载器完成的，类加载器包括：根加载器（BootStrap）、扩展加载器（Extension）、系统加载器（System）和用户自定义类加载器（java.lang.ClassLoader的子类）。从Java 2 (JDK 1.2) 开始，类加载过程采取了父亲委托机制（PDM）。PDM更好的保证了Java平台的安全性，在该机制中，JVM自带的Bootstrap是根加载器，其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载，父类加载器无能为力时才由其子类加载器自行加载。JVM不会向Java程序提供对Bootstrap的引用。下面是关于几个类加载器的说明：

- Bootstrap：一般用本地代码实现，负责加载JVM基础核心类库（rt.jar）；
- Extension：从java.ext.dirs系统属性所指定的目录中加载类库，它的父加载器是Bootstrap；
- System：又叫应用类加载器，其父类是Extension。它是应用最广泛的类加载器。它从环境变量classpath或者系统属性java.class.path所指定的目录中记载类，是用户自定义加载器的默认父加载器。

8.描述一下JVM加载class文件的原理机制?

JVM中类的装载是由类加载器 (ClassLoader) 和它的子类来实现的, Java中的类加载器是一个重要的Java运行时系统组件, 它负责在运行时查找和装入类文件中的类。

由于Java的跨平台性, 经过编译的Java源程序并不是一个可执行程序, 而是一个或多个类文件。当Java程序需要使用某个类时, JVM会确保这个类已经被加载、连接 (验证、准备和解析) 和初始化。类的加载是指把类的.class文件中的数据读入到内存中, 通常是创建一个字节数组读入.class文件, 然后产生与所加载类对应的Class对象。加载完成后, Class对象还不完整, 所以此时的类还不可用。当类被加载后就进入连接阶段, 这一阶段包括验证、准备 (为静态变量分配内存并设置默认的初始值) 和解析 (将符号引用替换为直接引用) 三个步骤。最后JVM对类进行初始化, 包括:

- 1)如果类存在直接的父类并且这个类还没有被初始化, 那么就先初始化父类;
- 2)如果类中存在初始化语句, 就依次执行这些初始化语句。

类的加载是由类加载器完成的, 类加载器包括: 根加载器 (BootStrap) 、扩展加载器 (Extension) 、系统加载器 (System) 和用户自定义类加载器 (java.lang.ClassLoader的子类) 。

从Java 2 (JDK 1.2) 开始, 类加载过程采取了父亲委托机制 (PDM) 。PDM更好的保证了Java平台的安全性, 在该机制中, JVM自带的Bootstrap是根加载器, 其他的加载器都有且仅有一个父类加载器。类的加载首先请求父类加载器加载, 父类加载器无能为力时才由其子类加载器自行加载。JVM不会向Java程序提供对Bootstrap的引用。下面是关于几个类加载器的说明:

- Bootstrap: 一般用本地代码实现, 负责加载JVM基础核心类库 (rt.jar) ;
- Extension: 从java.ext.dirs系统属性所指定的目录中加载类库, 它的父加载器是Bootstrap;
- System: 又叫应用类加载器, 其父类是Extension。它是应用最广泛的类加载器。它从环境变量 classpath或者系统属性java.class.path所指定的目录中加载类, 是用户自定义加载器的默认父加载器。

9.Java对象创建过程

1.JVM遇到一条新建对象的指令时首先去检查这个指令的参数是否能在常量池中定义到一个类的符号引用。然后加载这个类 (类加载过程在后边讲)

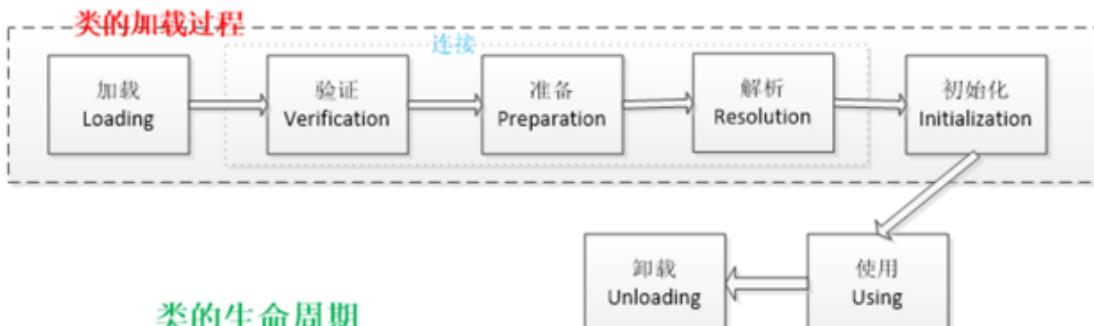
2.为对象分配内存。一种办法“指针碰撞”、一种办法“空闲列表”, 最终常用的办法“本地线程缓冲分配 (TLAB)”

3.将除对象头外的对象内存空间初始化为0

4.对对象头进行必要设置

10.类的生命周期

类的生命周期包括这几个部分, 加载、连接、初始化、使用和卸载, 其中前三部是类的加载的过程, 如下图:



- 加载, 查找并加载类的二进制数据, 在Java堆中也创建一个java.lang.Class类的对象

- 连接，连接又包含三块内容：验证、准备、初始化。1) 验证，文件格式、元数据、字节码、符号引用验证；2) 准备，为类的静态变量分配内存，并将其初始化为默认值；3) 解析，把类中的符号引用转换为直接引用
- 初始化，为类的静态变量赋予正确的初始值
- 使用，new出对象程序中使用
- 卸载，执行垃圾回收

11. Java对象结构

Java对象由三个部分组成：对象头、实例数据、对齐填充。

对象头由两部分组成，第一部分存储对象自身的运行时数据：哈希码、GC分代年龄、锁标识状态、线程持有的锁、偏向线程ID（一般占32/64 bit）。第二部分是指针类型，指向对象的类元数据类型（即对象代表哪个类）。如果是数组对象，则对象头中还有一部分用来记录数组长度。

实例数据用来存储对象真正的有效信息（包括父类继承下来的和自己定义的）

对齐填充：JVM要求对象起始地址必须是8字节的整数倍（8字节对齐）

12. Java对象的定位方式

句柄池、直接指针。

13. 如何判断对象可以被回收？

判断对象是否存活一般有两种方式：

- 引用计数：每个对象有一个引用计数属性，新增一个引用时计数加1，引用释放时计数减1，计数为0时可以回收。此方法简单，无法解决对象相互循环引用的问题。
- 可达性分析（Reachability Analysis）：从GC Roots开始向下搜索，搜索所走过的路径称为引用链。当一个对象到GC Roots没有任何引用链相连时，则证明此对象是不可用的，不可达对象。

14. JVM的永久代中会发生垃圾回收么？

垃圾回收不会发生在永久代，如果永久代满了或者是超过了临界值，会触发完全垃圾回收(Full GC)。如果你仔细查看垃圾收集器的输出信息，就会发现永久代也是被回收的。这就是为什么正确的永久代大小对避免Full GC是非常重要的原因。请参考下Java8：从永久代到元数据区(注：Java8中已经移除了永久代，新加了一个叫做元数据区的native内存区)

15. 引用的分类

- 强引用：GC时不会被回收
- 软引用：描述有用但不是必须的对象，在发生内存溢出异常之前被回收
- 弱引用：描述有用但不是必须的对象，在下一次GC时被回收
- 虚引用（幽灵引用/幻影引用）：无法通过虚引用获得对象，用PhantomReference实现虚引用，虚引用用来在GC时返回一个通知。

##GC是什么？为什么要GC？答：GC是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。Java程序员不用担心内存管理，因为垃圾收集器会自动进行管理。要请求垃圾收集，可以调用下面的方法之一：System.gc() 或Runtime.getRuntime().gc()，但JVM可以屏蔽掉显示的垃圾回收调用。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是一个单独的低优先级的线程运行，不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清除和回收，程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。在Java诞生初期，垃圾回收是Java最大的亮点之一，因为服务器端的编程需要有效的防止内存泄露问题，然而时过境迁，如今Java的

垃圾回收机制已经成为被诟病的东西。移动智能终端用户通常觉得iOS的系统比Android系统有更好的用户体验，其中一个深层次的原因就在于Android系统中垃圾回收的不可预知性。

补充：垃圾回收机制有很多种，包括：分代复制垃圾回收、标记垃圾回收、增量垃圾回收等方式。标准的Java进程既有栈又有堆。栈保存了原始型局部变量，堆保存了要创建的对象。Java平台对堆内存回收和再利用的基本算法被称为标记和清除，但是Java对其进行了改进，采用“分代式垃圾收集”。这种方法会跟Java对象的生命周期将堆内存划分为不同的区域，在垃圾收集过程中，可能会将对象移动到不同区域：

- 伊甸园（Eden）：这是对象最初诞生的区域，并且对大多数对象来说，这里是它们唯一存在过的区域。
- 幸存者乐园（Survivor）：从伊甸园幸存下来的对象会被挪到这里。
- 终身颐养园（Tenured）：这是足够老的幸存对象的归宿。年轻代收集（Minor-GC）过程是不会触及这个地方的。当年轻代收集不能把对象放进终身颐养园时，就会触发一次完全收集（Major-GC），这里可能还会牵扯到压缩，以便为大对象腾出足够的空间。与垃圾回收相关的JVM参数：

-Xms / -Xmx — 堆的初始大小 / 堆的最大大小 -Xmn — 堆中年轻代的大小 -XX:-DisableExplicitGC — 让System.gc()不产生任何作用 -XX:+PrintGCDetails — 打印GC的细节 -
XX:+PrintGCDetails — 打印GC操作的时间戳 -XX:NewSize / XX:MaxNewSize — 设置新生代大小/新生代最大大小 -XX:NewRatio — 可以设置老生代和新生代的比例 -
XX:PrintTenuringDistribution — 设置每次新生代GC后输出幸存者乐园中对象年龄的分布 -
XX:InitialTenuringThreshold / -XX:MaxTenuringThreshold：设置老年代阀值的初始值和最大值
-XX:TargetSurvivorRatio：设置幸存区的目标使用率

16. 判断一个对象应该被回收

1. 该对象没有与GC Roots相连

2. 该对象没有重写finalize()方法或finalize()已经被执行过则直接回收（第一次标记）、否则将对象加入到F-Queue队列中（优先级很低的队列）在这里finalize()方法被执行，之后进行第二次标记，如果对象仍然应该被GC则GC，否则移除队列。（在finalize方法中，对象很可能和其他GC Roots中的某一个对象建立了关联，finalize方法只会被调用一次，且不推荐使用finalize方法）

17. 回收方法区

方法区回收价值很低，主要回收废弃的常量和无用的类。

如何判断无用的类：

1. 该类所有实例都被回收（Java堆中没有该类的对象）
2. 加载该类的ClassLoader已经被回收
3. 该类对应的java.lang.Class对象没有在任何地方被引用，无法在任何地方利用反射访问该类

18. 垃圾收集算法

GC最基础的算法有三种：标记-清除算法、复制算法、标记-压缩算法，我们常用的垃圾回收器一般都采用分代收集算法。

- 标记-清除算法，“标记-清除”（Mark-Sweep）算法，如它的名字一样，算法分为“标记”和“清除”两个阶段：首先标记出所有需要回收的对象，在标记完成后统一回收掉所有被标记的对象。
- 复制算法，“复制”（Copying）的收集算法，它将可用内存按容量划分为大小相等的两块，每次只使用其中的一块。当这一块的内存用完了，就将还存活着的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉。
- 标记-压缩算法，标记过程仍然与“标记-清除”算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存

- 分代收集算法，“分代收集”(Generational Collection) 算法，把Java堆分为新生代和老年代，这样就可以根据各个年代的特点采用最适当的收集算法。

19. 垃圾回收器

- Serial收集器，串行收集器是最古老，最稳定以及效率高的收集器，可能会产生较长的停顿，只使用一个线程去回收。
- ParNew收集器，ParNew收集器其实就是Serial收集器的多线程版本。
- Parallel收集器，Parallel Scavenge收集器类似ParNew收集器，Parallel收集器更关注系统的吞吐量。
- Parallel Old 收集器，Parallel Old是Parallel Scavenge收集器的老年代版本，使用多线程和“标记 - 整理”算法
- CMS收集器，CMS (Concurrent Mark Sweep) 收集器是一种以获取最短回收停顿时间为 目标的收集器。
- G1收集器，G1 (Garbage-First)是一款面向服务器的垃圾收集器，主要针对配备多颗处理器及大容量内存的机器。以极高概率满足GC停顿时间要求的同时，还具备高吞吐量性能特征

20. GC日志分析

摘录GC日志一部分（前部分为年轻代gc回收；后部分为full gc回收）：

```
2016-07-05T10:43:18.093+0800: 25.395: [GC [PSYoungGen: 274931K->10738K(274944K)]  
371093K->147186K(450048K), 0.0668480 secs] [Times: user=0.17 sys=0.08, real=0.07  
secs]  
2016-07-05T10:43:18.160+0800: 25.462: [Full GC [PSYoungGen: 10738K->0K(274944K)]  
[ParOldGen: 136447K->140379K(302592K)] 147186K->140379K(577536K) [PSPermGen:  
85411K->85376K(171008K)], 0.6763541 secs] [Times: user=1.75 sys=0.02, real=0.68  
secs]
```

通过上面日志分析得出，PSYoungGen、ParOldGen、PSPermGen属于Parallel收集器。其中 PSYoungGen表示gc回收前后年轻代的内存变化；ParOldGen表示gc回收前后老年代的内存变化；PSPermGen表示gc回收前后永久区的内存变化。young gc主要是针对年轻代进行内存回收比较频繁，耗时短；full gc会对整个堆内存进行回收，耗时长，因此一般尽量减少full gc的次数

21. 调优命令

Sun JDK监控和故障处理命令有jps jstat jmap jhat jstack jinfo

- jps，JVM Process Status Tool,显示指定系统内所有的HotSpot虚拟机进程。
- jstat，JVM statistics Monitoring是用于监视虚拟机运行时状态信息的命令，它可以显示出虚拟机进程中的类装载、内存、垃圾收集、JIT编译等运行数据。
- jmap，JVM Memory Map命令用于生成heap dump文件
- jhat，JVM Heap Analysis Tool命令是与jmap搭配使用，用来分析jmap生成的dump，jhat内置了一个微型的HTTP/HTML服务器，生成dump的分析结果后，可以在浏览器中查看
- jstack，用于生成java虚拟机当前时刻的线程快照。
- jinfo，JVM Configuration info 这个命令作用是实时查看和调整虚拟机运行参数。

22. 调优工具

常用调优工具分为两类，jdk自带监控工具：jconsole和jvisualvm，第三方有：MAT(Memory Analyzer Tool)、GChisto。

- jconsole，Java Monitoring and Management Console是从java5开始，在JDK中自带的java监控和管理控制台，用于对JVM中内存，线程和类等的监控
- jvisualvm，jdk自带全能工具，可以分析内存快照、线程快照；监控内存变化、GC变化等。

- MAT, Memory Analyzer Tool, 一个基于Eclipse的内存分析工具, 是一个快速、功能丰富的Java heap分析工具, 它可以帮助我们查找内存泄漏和减少内存消耗
- GChisto, 一款专业分析gc日志的工具

23Minor GC与Full GC分别在什么时候发生?

新生代内存不够用时候发生MGC也叫YGC, JVM内存不够的时候发生FGC

24.你知道哪些JVM性能调优

- 设定堆内存大小

-Xmx: 堆内存最大限制。

- 设定新生代大小。新生代不宜太小, 否则会有大量对象涌入老年带

-XX:NewSize: 新生代大小

-XX:NewRatio 新生代和老生代占比

-XX:SurvivorRatio: 伊甸园空间和幸存者空间的占比

- 设定垃圾回收器 年轻代用 -XX:+UseParNewGC 老年代用-XX:+UseConcMarkSweepGC

这份Java面试题, 会持续完善, 更多面试题

请关注这个公众号, 每天只发Java推文

