Java

**多态是什么，栈和堆的区别**

1、栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。其操作方式类似于数据结构中的栈。

一种是基本类型(primitive types), 共有8种，即int, short, long, byte, float, double, boolean, char(注意，并没有string的基本类型)。这种类型的定义是通过诸如int a = 3; long b = 255L;的形式来定义的

2、堆区（heap） — 一般由程序员分配释放， 若程序员不释放，程序结束时可能由OS回收 。注意它与数据结构中的堆是两回事，分配方式倒是类似于链表。

Integer, String, Double等将相应的基本数据类型包装起来的类。这些类数据全部存在于堆中，Java用new()语句来显示地告诉编译器，在运行时才根据需要动态创建，因此比较灵活，但缺点是要占用更多的时间

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。

**垃圾怎么回收**

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。垃圾回收器通常是作为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”。这种方法会跟据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：

1.1:算法分析：

引用计数算法是垃圾回收器中的早起策略，在这种方法中，堆中的每个对象实例都有一个引用计数器，点一个对象被创建时，且该对象实例分配给一个变量，该变量计数设置为1 ，当任何其他变量赋值为这个对象的引用时，计数加1 ，（a=b ，则b引用的对象实例计数器+1）但当一个对象实例的某个引用超过了生命周期或者被设置为一个新值时，对象实例的引用计数器减1，任何引用计数器为0 的对象实例可以当做垃圾收集。 当一个对象的实例被垃圾收集是，它引用的任何对象实例的引用计数器减1.

**怎么实现多线程，线程怎么运行**

一、继承Thread类创建线程类

（1）定义Thread类的子类，并重写该类的run方法，该run方法的方法体就代表了线程要完成的任务。因此把run()方法称为执行体。

（2）创建Thread子类的实例，即创建了线程对象。

（3）调用线程对象的start()方法来启动该线程。

public class MyThread extends Thread {

　　public void run() {

　　 System.out.println("MyThread.run()");

　　}

}

MyThread myThread1 = new MyThread();

MyThread myThread2 = new MyThread();

myThread1.start();

myThread2.start();

二、通过Runnable接口创建线程类

（1）定义runnable接口的实现类，并重写该接口的run()方法，该run()方法的方法体同样是该线程的线程执行体。

（2）创建 Runnable实现类的实例，并依此实例作为Thread的target来创建Thread对象，该Thread对象才是真正的线程对象。

（3）调用线程对象的start()方法来启动该线程。

public class MyThread extends OtherClass implements Runnable {

　　public void run() {

　　 System.out.println("MyThread.run()");

　　}

}

MyThread myThread = new MyThread();

Thread thread = new Thread(myThread);

thread.start();

三、通过Callable和Future创建线程

（1）创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。

（2）创建Callable实现类的实例，使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。

（3）使用FutureTask对象作为Thread对象的target创建并启动新线程。

（4）调用FutureTask对象的get()方法来获得子线程执行结束后的返回值

public class TestCallable {

    public static void main(String[] args) {

        ThreadDemo td = new ThreadDemo();

        //1.执行 Callable 方式，需要 FutureTask 实现类的支持，用于接收运算结果。

        FutureTask<Integer> result = new FutureTask<>(td);

        new Thread(result).start();

        //2.接收线程运算后的结果

        try {

            Integer sum = result.get();

            System.out.println(sum);

            System.out.println("------------------------------------");

        } catch (InterruptedException | ExecutionException e) {

            e.printStackTrace();

        }

    }

}

class ThreadDemo implements Callable<Integer> {

    @Override

    public Integer call() throws Exception {

        int sum = 0;

        for (int i = 0; i <= 100000; i++) {

            sum += i;

        }

        return sum;

    }

}

//至此，一个线程就创建完成了。

**怎么保证线程安全**

1. 最简单的方式是加入synchronized关键字，只要将操作共享数据的语句加入synchronized关键字，在某一时段

只会让一个线程执行完，在执行过程中，其他线程不能进来执行

synchronized是对类的当前实例进行加锁，那么static synchronized恰好就是要控制类的所有实例的访问了(类锁）

2.使用锁

ReentrantLock

Lock不是Java语言内置的，synchronized是Java语言的关键字，因此是内置特性。Lock是一个类，通过这个类可以实现同步访问

synchronized不需要手动释放锁，当synchronized方法或者synchronized代码块执行完之后，系统会自动让线程释放对锁的占用；而Lock则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致出现死锁现象。

lock()：unlock.若lock被thread A取得，thread B会进入block狀態，直到取得lock；

tryLock()：若当下不能取得lock，thread就会放弃，可以设置一个超时时间参数，等待多久获取不到锁就放弃；

Lock lock = new ReentrantLock();

if(lock.tryLock()) {

try{

//处理任务

}catch(Exception ex){

}finally{

lock.unlock(); //释放锁

}

}else {

//如果不能获取锁，则直接做其他事情

}

**sleep() 、join（）、yield（）有什么区别**

1、sleep()方法

在指定的毫秒数内让当前正在执行的线程休眠（暂停执行）,但它并不释放对象锁

2.yield方法

暂停当前正在执行的线程对象，区别在于，它没有参数

3.join方法

等待该线程终止。

等待调用join方法的线程结束，再继续执行。如：t.join();//主要用于等待t线程运行结束，若无此句，main则会执行完毕，导致结果不可预测。

wait()、notify/notifyAll() 方法是Object的本地final方法，无法被重写

void wait( )

译：导致当前的线程等待，直到其他线程调用此对象的notify( ) 方法或 notifyAll( ) 方法

void notify()

译：唤醒在此对象监视器上等待的单个线程

<https://blog.csdn.net/coding_1994/article/details/80634792>

**代码覆盖率怎么看，你们项目测试验收的要求**

intellij - >Show Code Coverage Data

**Classloader工作原理**

程序在启动的时候，并不会一次性加载程序所要用的所有class文件，而是根据程序的需要，通过Java的类加载机制（ClassLoader）来动态加载某个class文件到内存当中的，从而只有class文件被载入到了内存之后，才能被其它class所引用。所以ClassLoader就是用来动态加载class文件到内存当中用的。

BootStrap ClassLoader：称为启动类加载器，是Java类加载层次中最顶层的类加载器，负责加载JDK中的核心类库，如：rt.jar、resources.jar、

Extension ClassLoader：称为扩展类加载器，负责加载Java的扩展类库，默认加载JAVA\_HOME/jre/lib/ext/目下的所有jar。

App ClassLoader：称为系统类加载器，负责加载应用程序classpath目录下的所有jar和class文件。

创建对象时构造器的调用顺序是：先初始化静态成员，然后调用父类构造器，再初始化非静态成员，最后调用自身构造器。

class A {

    static {

        System.out.print("1");

    }

    public A() {

        System.out.print("2");

    }

}

class B extends A {

    static {

        System.out.print("a");

    }

    public B() {

        System.out.print("b");

    }

}

public class Hello {

    public static void main(String[] args) {

        A ab = new B();

        ab = new B();

    }

执行结果：1a2b2b

父类静态代变量、   
父类静态代码块、   
子类静态变量、   
子类静态代码块、   
父类非静态变量（父类实例成员变量）、   
父类构造函数、   
子类非静态变量（子类实例成员变量）、   
子类构造函数。

**数据库**

**两张表合并后求总成绩，怎么做**

UNION 列出不同的值，不过 UNION ALL 命令会列出所有的值

SELECT E\_Name FROM Employees\_China

UNION

SELECT E\_Name FROM Employees\_USA

**什么是事务，什么是悲观锁，乐观锁**

1.数据库事务: 数据库事务通常指对数据库进行读或写的一个操作序列。

BEGIN TRANSACTION //事务开始

SQL1

SQL2

COMMIT/ROLLBACK //事务提交或回滚

原子性(Atomic)：事务中各项操作，要么全做要么全不做，任何一项操作的失败都会导致整个事务的失败；

一致性(Consistent)：事务结束后系统状态是一致的；

隔离性(Isolated)：并发执行的事务彼此无法看到对方的中间状态；

持久性(Durable)：事务完成后所做的改动都会被持久化，即使发生灾难性的失败。通过日志和同步备份可以在故障发生后重建数据。

---------------------

一、悲观锁

    1、排它锁，当事务在操作数据时把这部分数据进行锁定，直到操作完毕后再解锁，其他事务操作才可操作该部分数据。这将防止其他进程读取或修改表中的数据。

    2、实现：大多数情况下依靠数据库的锁机制实现

     一般使用 select ...for update 对所选择的数据进行加锁处理，例如select \* from account where name=”Max” for update， 这条sql 语句锁定了account 表中所有符合检索条件（name=”Max”）的记录。本次事务提交之前（事务提交时会释放事务过程中的锁），外界无法修改这些记录。

select status from t\_items where id=1 for update;

悲观锁就是for update（锁定查询的行）

乐观锁就是 version字段或者timestamp（比较跟上一次的版本号，如果一样则更新，如果失败则要重复读-比较-写的操作。）

1.查询出商品信息

select (status,status,version) from t\_goods where id=#{id}

2.根据商品信息生成订单

3.修改商品status为2

update t\_goods

set status=2,version=version+1

where id=#{id} and version=#{version};

二、乐观锁

    1、如果有人在你之前更新了，你的更新应当是被拒绝的，可以让用户重新操作。

    2、实现：大多数基于数据版本（Version）记录机制实现

     具体可通过给表加一个版本号或时间戳字段实现，当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加一。当我们提交更新的时候，判断当前版本信息与第一次取出来的版本值大小，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据，拒绝更新，让用户重新操作。

---------------------

**什么是索引**

为什么需要索引？数据在磁盘上是以块的形式存储的。为确保对磁盘操作的原子性，访问数据的时候会一并访问所有数据块。磁盘上的这些数据块与链表类似，即它们都包含一个数据段和一个指针，指针指向下一个节点（数据块）的内存地址，而且它们都不需要连续存储（即逻辑上相邻的数据块在物理上可以相隔很远）。

举个例子来讲，我们有一个数据表User.为了简便，这个表没有主键。

虽然这些数据都存在于一个User表中，但是物理上，这些数据可能存储在分散的数据块中。

索引是对某个字段进行排序的一种方式。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值又指向与它相关的记录。这种索引的数据结构是经过排序的，因而可以对其执行二分查找。

对上个表的Identity字段进行索引，就是在数据库存储空间上创建一块专用的控件，把User表的所有的Identity字段的值拿出来放到这里，并且对这些值进行排序，并且每个值都携带着这个Identity对应的行所在数据块的地址。因为Identity是进过排序的，按照一定的数据结构存储的，所以数据库引擎在查找的时候，比如说查找identity为5，引擎就会计算，5大概在整个排序结构的大致地方，然后到那里去拿出这个值看看是不是，不是的话就再次相应的向左或者向右移动去寻找。（二分法）

比如 select \* from User where Identity= 2 这条语句，是走索引查询的。因为是否走索引取决于这条查询语句的where子句、

索引的缺点

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

唯一索引：与"普通索引"类似，不同的就是：索引列的值必须唯一，但允许有空值。

主键索引：它 是一种特殊的唯一索引，不允许有空值。

组合索引 ALTER TABLE `table\_name` ADD INDEX index\_name ( `column1`, `column2`, `column3` )

唯一索引： Create unique Index <Index-Name> On <Table\_Name>(Column\_Name);

表明此索引的每一个索引值只对应唯一的数据记录，对于单列惟一性索引，这保证单列不包含重复的值。对于多列惟一性索引，保证多个值的组合不重复

聚集索引（也叫聚簇索引）：create clustered index clustered\_index on table\_name(colum\_name)

在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。 如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

**Exist和in**

in是确定给定的值是否与子查询或列表中的值相匹配。in在查询的时候，首先查询子查询的表，然后将内表和外表做一个笛卡尔积，然后按照条件进行筛选, in()适合B表比A表数据小的情况

exist 指定一个子查询，检测行的存在。遍历循环外表，然后看外表中的记录有没有和内表的数据一样的。匹配上就将结果放入结果集中。exists()适合B表比A表数据大的情况

select \* from A where id in(select id from B)

select a.\* from A a where exists(select 1 from B b where a.id=b.id)

**not in 和not exists**

如果查询语句使用了not in 那么内外表都进行全表扫描，没有用到索引；而not extsts 的子查询依然能用到表上的索引。所以无论那个表大，用not exists都比not in要快。

select user\_id,count(\*) from user where user\_name is not null group by user\_id having count(\*)>1;

**反射3种实现方式**

1. 通过Object类的getClass方法来获取

Date date1 = new Date();

Date date2 = new Date();

Class c1 = date1.getClass();

Class c2 = date2.getClass();

System.out.println(c1.getName());

2. 使用.class的方式

Class clazz = String.class;

System.out.println(clazz.getName());

3. 使用Class.forName方法

Class clazz = Class.forName("org.whatisjava.reflect.Foo");

**session 与 cookie 区别**

cookie 是 Web 服务器发送给浏览器的一块信息。浏览器会在本地文件中给每一个 Web 服务

器存储 cookie。以后浏览器在给特定的 Web 服务器发请求的时候，同时会发送所有为该服

务器存储的 cookie。下面列出了 session 和 cookie 的区别：

无论客户端浏览器做怎么样的设置，session都应该能正常工作。客户端可以选择禁用 cookie，

但是， session 仍然是能够工作的，因为客户端无法禁用服务端的 session。

具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

**ArrayList 和 LinkedList 有什么区别。**

ArrayList和LinkedList都实现了List接口，有以下的不同点：

1、ArrayList是基于索引的数据接口，它的底层是数组。它可以以O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList是以元素列表的形式存储它的数据，每一个元素都和它的前一个和后一个元素链接在一起，在这种情况下，查找某个元素的时间复杂度是O(n)。 时间复杂度O(n)表示程序运行时间跟n有关，并且是线性关系。O(1)，表示所需时间为常量，并且与n无关。

2、相对于ArrayList，LinkedList的插入，添加，删除操作速度更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。

3、LinkedList比ArrayList更占内存，因为LinkedList为每一个节点存储了两个引用，一个指向前一个元素，一个指向下一个元素。

**单例模式的实现 （构造函数private,不能new对象）**

饿汉式，线程安全

public class Singleton {

private static Singleton instance=new Singleton();

private Singleton(){

}

public static Singleton getInstance(){

return instance;

}

}

静态内部类

public class Singleton {

private Singleton(){

}

private static class SingletonHolder{

private final static Singleton instance=new Singleton();

}

public static Singleton getInstance(){

return SingletonHolder.instance;

}

}

**深拷贝和浅拷贝**

**1.浅拷贝**

浅拷贝是按位拷贝对象，它会创建一个新对象，这个对象有着原始对象属性值的一份精确拷贝。如果属性是基本类型，拷贝的就是基本类型的值；如果属性是内存地址（引用类型），拷贝的就是内存地址 ，因此如果其中一个对象改变了这个地址，就会影响到另一个对象。（string是引用类型，会影响原来的对象）

public class Person implements Cloneable{

public Person(Integer age, String name) {

super();

this.age = age;

this.name = name;

}

@Override

protected Object clone() throws CloneNotSupportedException {

return super.clone();

}

}

Person p = new Person(23, "zhang");

Person p1 = (Person) p.clone();

**深拷贝**

static class Body implements Cloneable{

public Head head;

public Body() {}

public Body(Head head) {this.head = head;}

@Override

protected Object clone() throws CloneNotSupportedException {

Body newBody = (Body) super.clone();

newBody.head = (Head) head.clone();

return newBody;

}

}

static class Head implements Cloneable{

public Face face;

public Head() {}

public Head(Face face){this.face = face;}

@Override

protected Object clone() throws CloneNotSupportedException {

return super.clone();

}

}

public static void main(String[] args) throws CloneNotSupportedException {

Body body = new Body(new Head());

Body body1 = (Body) body.clone();

System.out.println("body == body1 : " + (body == body1) );

System.out.println("body.head == body1.head : " + (body.head == body1.head));

}

**jav 斐波那契数列**

public static void testFibonacci1(int n) {

int a = 1, b=1, c = 0;

for (int i = 1; i <= n; i++) {

if (i == 1) {

System.out.println("第" + i + "等于" + a);

}else if (i == 2) {

System.out.println("第" + i + "等于" + b);

}else {

c = a+b;

a = b;

b = c;

System.out.println("第" + i + "等于" + c);

}

}

}

public static int testFibonacci3(int n) {

if (n == 1 || n == 2) {

return 1;

}else {

return test3(n-1) + test3(n-2);

}

}

**HTTP与TCP的区别和联系**

TCP对应于传输层，HTTP对应于应用层，从本质上来说，二者没有可比性。

Http协议是建立在TCP协议基础之上的，当浏览器需要从服务器获取网页数据的时候，会发出一次Http请求。Http会通过TCP建立起一个到服务器的连接通道，当本次请求需要的数据完毕后，Http会立即将TCP连接断开，这个过程是很短的。所以Http连接是一种短连接，是一种无状态的连接。

怎么测试纸杯，怎么测试电梯、想要面试软件测试工程师的朋友们，这三个问题可一定要要记得关注。

经常会有一些面试官会考验你测试思维是怎样的，所以面试官便会通过身边的一些物体来判断你的思维逻辑能力

对于以上的这些题目一般我们这样应对：主要考虑几个大的方面（界面、功能、性能，安全性，可用性）

测试纸杯：

首先拿到一个杯子我们先看是否有1.（说明书）

                                          2.界面（界面测试）看外观是否设计合理，是否带有log，标语以及是否有错别字，杯子的形状是怎么样的。

                                          3.功能测试是否能装水，杯子能装多少毫升的液体，能否泡茶，能否进行当容器进行装其他东西（）

                                          4.是否能装其他液体，能否装高温液体，是否可以放冰箱（性能）

                                          5.制作杯子的材料，是否有毒，放微波炉里是否会爆炸, 是否杯子是会熔化，杯子是否有缺口，是否会摔碎。（安全性）

                                          6.杯子是否烫手、是否有防滑措施、是否方便饮用（可用性）

怎么测试电梯：（界面、功能，性能、可用性）

                1.（说明书）

                2.电梯内的每个按键是否都能够使用，电梯是否能载人载物（功能测试），当超重时是不是否有警报

                3.电梯的等待时间，每次停留的时间，各个按键是否设计的合理（性能测试）

电梯梯测试可以从几个方面来进行,功能测试,性能测试,压力测试,可用性测试(Usability),兼容性测试,本地化/国际化测试,可维护性测试;

1. 功能测试,最基本的上下功能,开关功能,还有里面的各个按键。
2. 性能测试(很多人忽略的),比如电梯的调度算法,用户的等待时间,平均等待时间,上下的速度,耗电量等等。
3. 压力测试,比如承重量(你实际承受力是20,那么当进入19个人的时候就应该报警,或者是实际上用户有可能一股脑的全部冲进电梯,所以在静止的时候电梯需要考虑到这种情况),突然断电,门打不开等等。
4. 可用性测试,按钮是否方便,按键的感觉是否好,视觉效果,现在很多人诟病的事情是,开和关两个按钮的图示很不友好,在紧急的时候很容易搞错。
5. 兼容性测试,比如每个国家的电压不一样,是否考虑到这个情况。
6. 本地化/国际化测试,曾经看到一部电梯的使用手册翻译成英文,翻译得很差。
7. 可维护性测试,电梯如果坏了怎么去维修。

怎么测试一支笔：

               1.测试界面，颜色是否符合要求，所占用的空间有多大，是否环保

               2.测试质量以及功能：是否能写字流畅，写出的字的颜色是否符合要求，能使用多长时间

Restful

在UserApi中，有这样三个接口

1. PUT /users/{id} 更新用户信息

2. PATCH /users/role/{id} 更新用户角色

3. PATCH /users/status/{id} 更新用户状态

在这三个接口便可以看出PUT和PATCH的区别

PUT方法主要是用来更新整个资源的，而PATCH方法主要是用来执行某项操作并更新资源的某些字段

-语法为：chmod abc file

其中a,b,c各为一个数字，分别表示User、Group、及Other的权限。

若要rwx属性则4+2+1=7；   
若要rw-属性则4+2=6；   
若要r-x属性则4+1=7。

chmod -R lamport:users \*    
-rw------- (600) -- 只有属主有读写权限。    
-rw-r--r-- (644) -- 只有属主有读写权限；而属组用户和其他用户只有读权限。    
-rwx------ (700) -- 只有属主有读、写、执行权限。    
-rwxr-xr-x (755) -- 属主有读、写、执行权限；而属组用户和其他用户只有读、执行权限。    
-rwx--x--x (711) -- 属主有读、写、执行权限；而属组用户和其他用户只有执行权限。    
-rw-rw-rw- (666) -- 所有用户都有文件读、写权限。这种做法不可取。    
-rwxrwxrwx (777) -- 所有用户都有读、写、执行权限。更不可取的做法。

Selenium IE，关闭保护模式 ，缩放为100%

Big data technology

Hadoop

1. 分布式文件系统 HDFS (数据默认保存3份 master, slave1,slave2)

Datanodes：存储文件快的数据

Namenode：文件块的metadata，定位数据块

1. 海量数据计算框架MapReduce

Map阶段：并行处理输入数据块

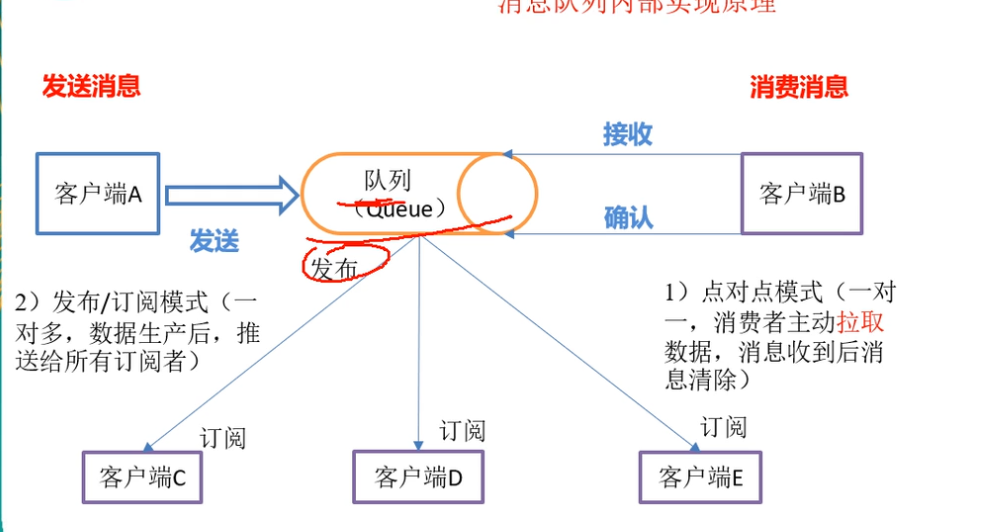
Reduce：对Map阶段一个汇总

其中Map对数据集上的独立元素进行指定的操作，生成键-值对形式中间结果。Reduce则对中间结果中相同“键”的所有“值”进行规约，以得到最终结果。

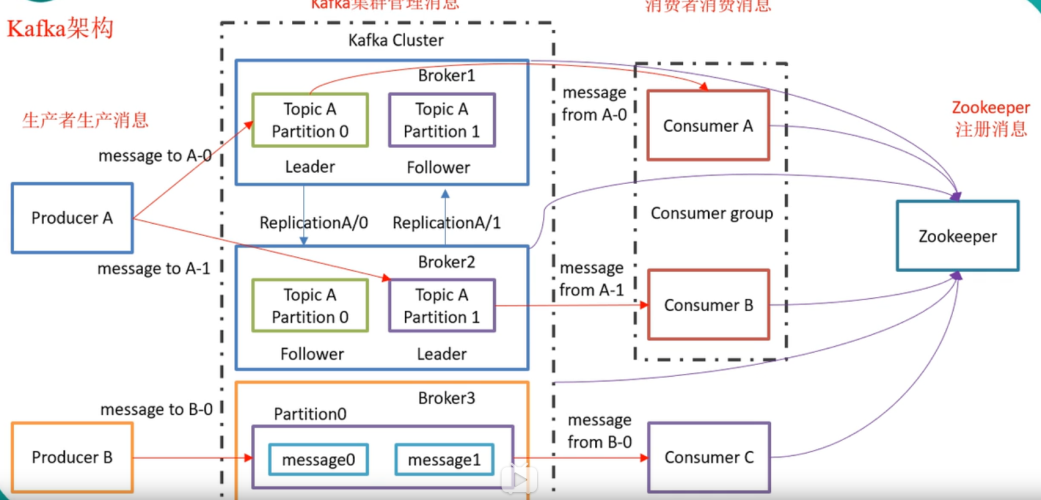
1. Yarn：集群资源管理系统 resource manager，node manager，调度资源

Kafka

1. 高吞吐量的分布式订阅消息系统



Kafka依赖于zookeeper，数据存在topic



Spark

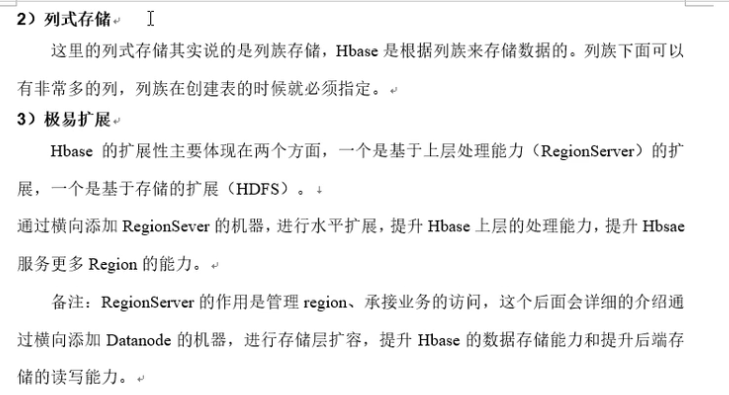
1. 基于内存的计算框架，比MapReduce快很多，通过yarn计算hadoop海量数据

HBASE

1. HBase是一个建立在HDFS之上，面向列的针对结构化数据的可伸缩、高可靠、高性能、分布式和面向列的动态模式数据库。

存储非结构化数据

这里的列指的是HBASE中的数据



HIVE

1. 基于Hadoop的数据仓库，可以将结构化数据映射成数据库表 （mysql作为metadata存储）

