1. **数据类型**

类型 位数 字节数 取值范围

byte： 8 1 -2^7 - 2^7-1

short: 16 2 -2^15 - 2^15-1

int: 32 4 -2^31 - 2^31-1

long: 64 8 -2^63 - 2^63-1

float: 32 4 2^-149 - 2^128-1

double: 64 8 2^-1074 - 2^1024-1

char: 16 2 0 - 2^16-1

boolean: 没有固定大小根据虚拟机定义

Bit(位)、byte、字节、汉字的关系

        1 bit     = 1  二进制数据

        1 byte  = 8  bit

        1 字母 = 1  byte = 8 bit

        1 汉字 = 2  byte = 16 bit

1. bit：位

    一个二进制数据0或1，是1bit；

2. byte：字节

   存储空间的基本计量单位，如：MySQL中定义 VARCHAR(45)  即是指 45个字节；

   1 byte = 8 bit

3. 一个英文字符占一个字节；

    1 字母 = 1 byte = 8 bit

4. 一个汉字占2个字节；

    1 汉字 = 2 byte = 16 bit

5. 标点符号

    A>.  汉字输入状态下，默认为全角输入方式；

    B>.  英文输入状态下，默认为半角输入方式；

    C>.  全角输入方式下，标点符号占2字节；

    D>.  半角输入方式下，标点符号占1字节；

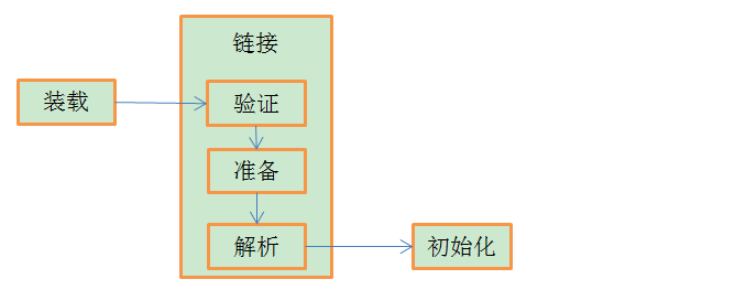
    故：汉字输入状态下的字符，占2个字节 (但不排除，自己更改了默认设置)；

            英文输入状态下的字符，占1个字节 (但不排除，自己更改了默认设置)；

1. **类加载机制**

JVM将类加载过程分为三个步骤：装载（Load），链接（Link）和初始化(Initialize)。

链接又分为三个步骤，如下图所示



1) 装载：通过类加载器，查找并加载类的二进制数据

2) 链接：

验证：确保被加载类的正确性；检查class文件是否符合JVM字节码格式的，正常情况由编译器生成的class文件肯定符合，如果不是，那需要验证通过才能下一步，安全考虑

准备：为类的静态变量分配内存，并将其初始化为默认值；如private static int a = 10

在准备阶段即a=0，然后解析，再到初始化的时候才a=10。

解析：把类中的符号引用转换为直接引用；

3) 初始化：为类的静态变量赋予正确的初始值，及创建对象

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这个类的java.lang.Class对象，用来封装类在方法区类的对象

4) 使用对象

5) 卸载

类加载器

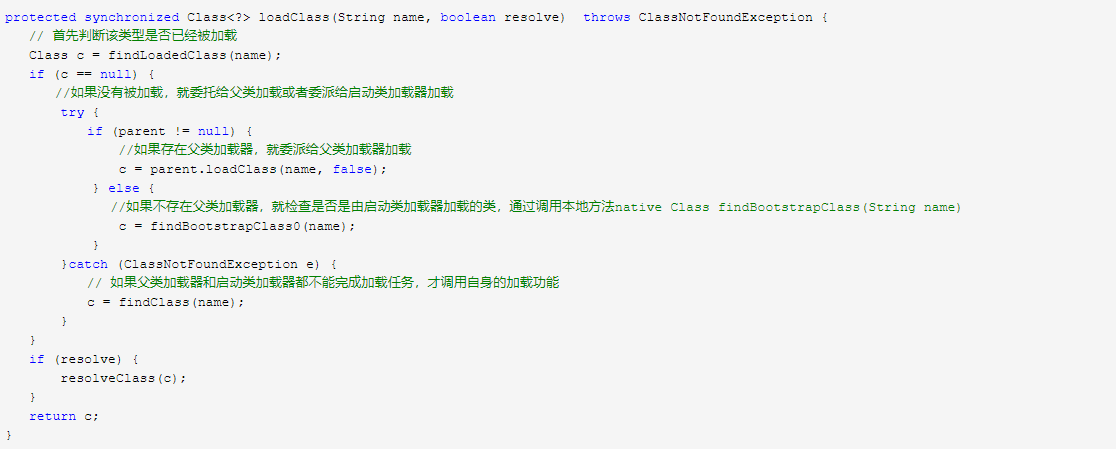
1、启动类加载器(Bootstrap ClassLoader)：负责加载 JAVA\_HOME\lib 目录中的，或通过-Xbootclasspath参数指定路径中的，且被虚拟机认可（按文件名识别，如rt.jar）的类

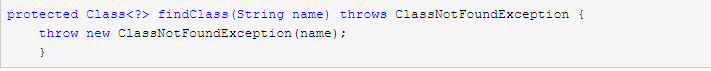
2、扩展类加载器(Extension ClassLoader)：负责加载 JAVA\_HOME\lib\ext 目录中的，或通过java.ext.dirs系统变量指定路径中的类库

3、应用程序类加载器(Application ClassLoader)：负责加载用户路径（classpath）上的类库

4、自定义类加载器

JDK中的ClassLoader





通过上面可以看出findClass()的实现，直接抛出一个异常，并且方法是protected，很明显这是留给我们开发者自己去实现的，我们可以通过重写findClass方法来实现我们自己的类加载器。

Springboot通过@EnableAutoConfiguration注解，会根据META-INF/spring.

factories文件自动配置，通过@Configuration代理原来的xml配置类，通过@ComponentScan配置类扫描路径，反射生成ioc容器

1. **gc垃圾回收机制**
2. **Jvm原理、调优**
3. **反射**
4. **代理**
5. **IO/NIO**
6. **集合原理**

在有序数组中，可以快速找到特定的值，但是想在有序数组中插入一个新的数据项，就必须首先找出新数据项插入的位置，然后将比新数据项大的数据项向后移动一位，来给新的数据项腾出空间，删除同理，这样移动很费时。显而易见，如果要做很多的插入和删除操作和删除操作，就不该选用有序数组。

另一方面，链表中可以快速添加和删除某个数据项，但是在链表中查找数据项可不容易，必须从头开始访问链表的每一个数据项，直到找到该数据项为止，这个过程很慢。

1. **多线程安全**
2. **Tcp、Socket、netty网络编程**
3. **Springboot类加载原理**
4. **Dubbo**
5. **Zookeeper**
6. **Redis**
7. **Rabbitmq（websocket、mqtt）**
8. **Mysql索引原理**

为了快速查询，很多时候，我们会建立不同的索引，满足需求，mysql主要索引引擎有MyISAM和InnoDB，两者都是使用B+Tree作为索引结构，两者的区别：一是主索引的区别，InnoDB的数据文件本身就是索引文件。而MyISAM的索引和数据是分开的。二是辅助索引的区别：InnoDB的辅助索引data域存储相应记录主键的值而不是地址。而MyISAM的辅助索引和主索引没有多大区别。

B+Tree参考 27、常用查找算法

性能优化点：

1、索引建立

2、sql中使用函数会导致索引失效

3、sql中like “xx%”索引生效，like “%xx”和like “%xx%”，索引不生效

4、in (1，2，3) 常量索引生效，in (子查询)，索引不生效

5、缩小结果集查询

1. **Hadoop**
2. **Hbase+Phoenix**
3. **ElasticSearch**
4. **Flink**
5. **Quartz**
6. **Docker**
7. **Greenplum(mpp)**
8. **分布式事务**
9. **分布式锁**
10. **数据一致性、高可用性、扩展性**
11. **常用查找算法**

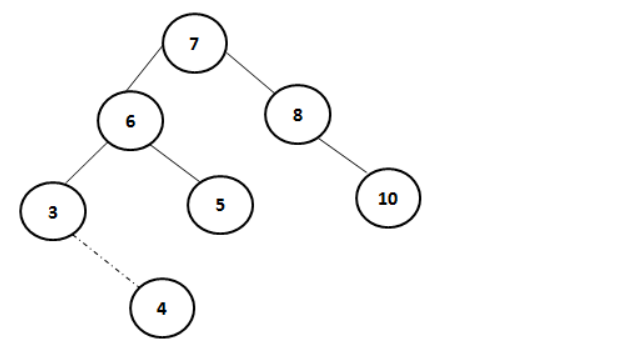
二叉树：

1、左子树上的所有节点值均小于根节点值，

2、右子树上的所有节点值均不小于根节点值，

3、左右子树也满足上述两个条件。

插入过程如下图：比较4与7，4<7，再比较4与7的左子树6，4<6，比较4与6的左子树3，4>3，比较4与3的右子树，为空，插入4。（字符串作为key比较可以用compareTo）

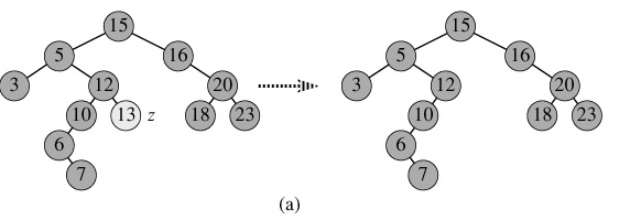


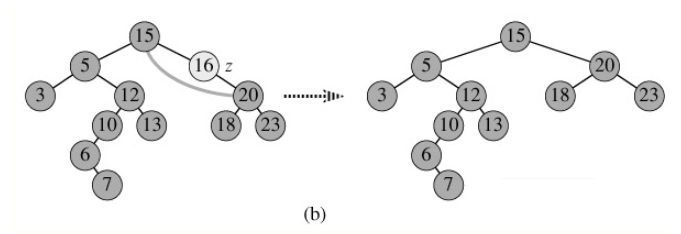
删除分为三种情况：

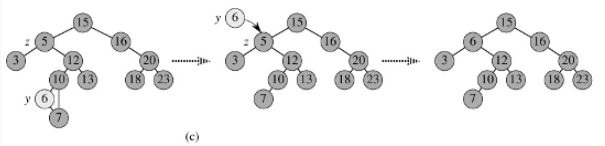
1. p为叶子节点，直接删除该节点，再修改其父节点的指针（注意是父节点和不是根节点），如图a。

2. p为单支节点（即只有左子树或右子树）。让p的子树与p的父亲节点相连，删除p即可；（注意分是根节点和不是根节点）；如图b。

3. p的左子树和右子树均不空。找到p的后继y，因为y一定没有左子树，所以可以删除y，并让y的父亲节点成为y的右子树的父亲节点，并用y的值代替p的值；或者方法二是找到p的前驱x，x一定没有右子树，所以可以删除x，并让x的父亲节点成为y的左子树的父亲节点。如图c。







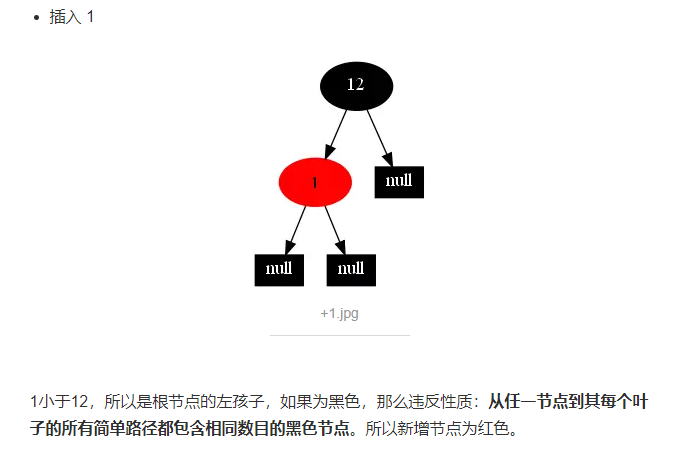
红黑树：

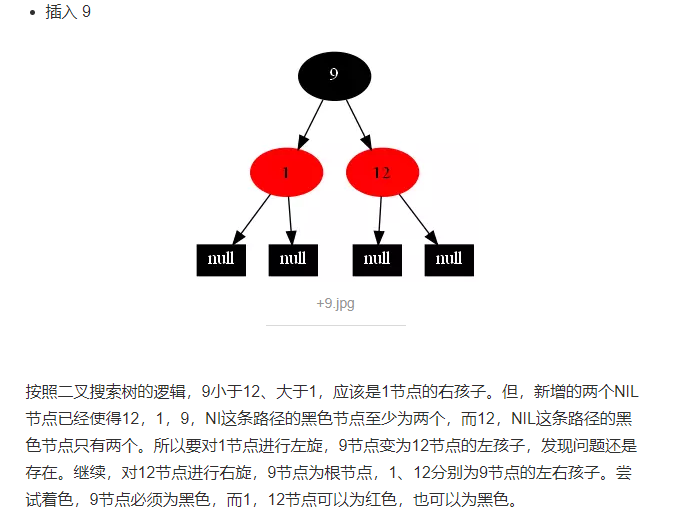
1、每个节点颜色不是黑色，就是红色2、根节点是黑色的3、如果节点是红色的，则它的子节点必须是黑色的（反之不一定，没有连续的红节点）4、从根节点到叶子节点的每条路径，必须包含相同数目的黑色节点

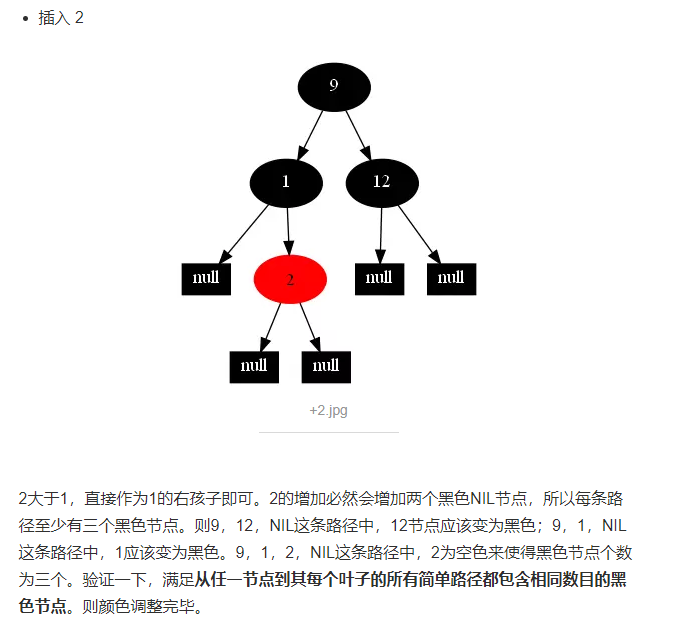
5、红黑树继承二叉树的性质

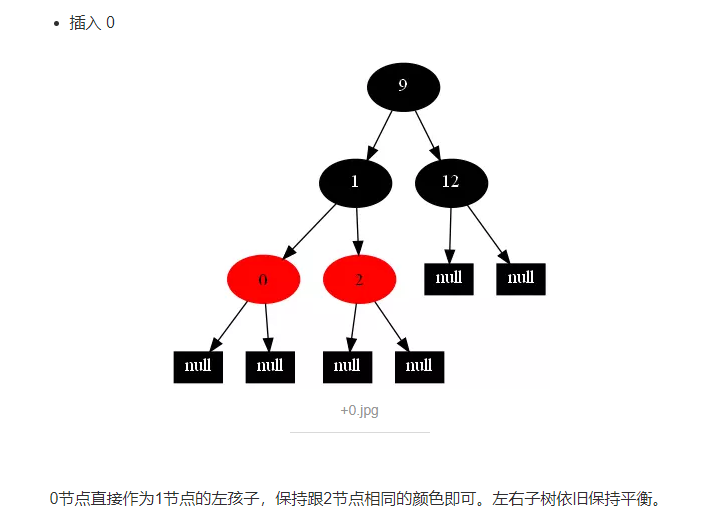
插入节点：

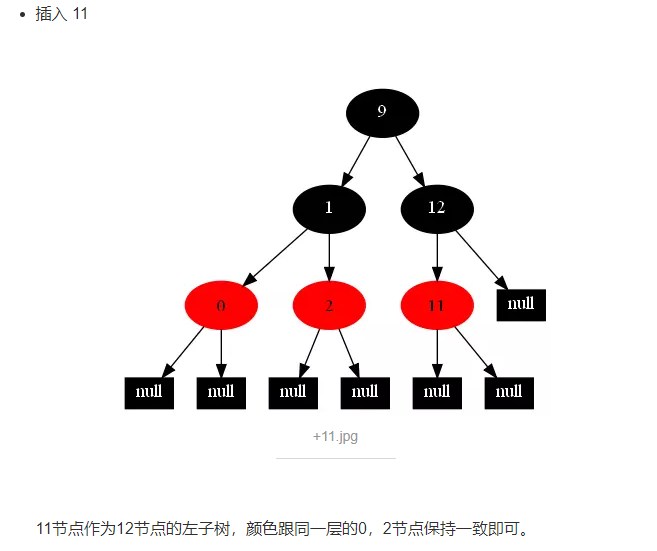
往一个空的红黑树中，依次插入数据：12 1 9 2 0 11 7 19 4

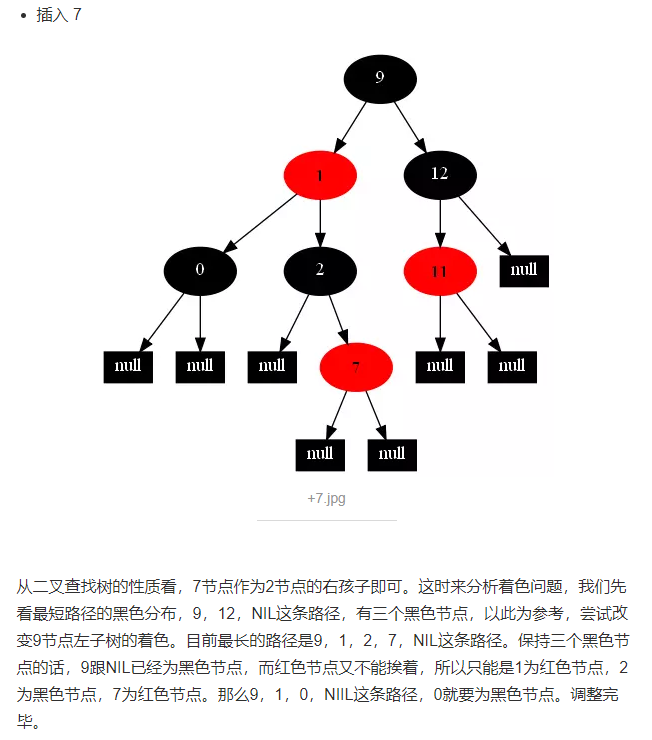


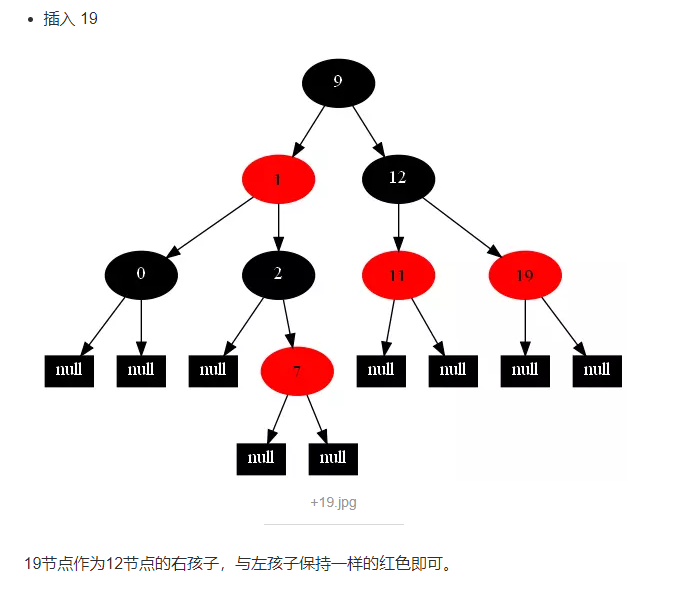


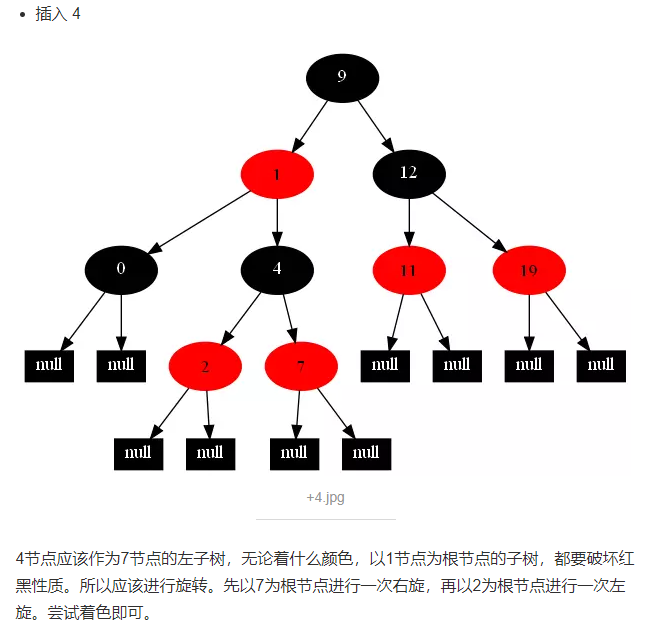












删除节点同插入分析方式。

通过任意一条从根到叶子简单路径上颜色的约束，红黑树保证最长路径不超过最短路径的二倍，因而近似平衡。

b树：

b+树：

hash：

倒排：

总结：

1、二叉树不是平衡树，让插入的值分别为1，2，3，4，5时，他会在右子树一条线上，这样跟链表是类似了，达不到高效查找的效果了

2、B树是一种多叉平衡查找树，红黑树是二叉平衡查找树结构，B树由于是多叉结构，对于元素数量非常多的情况下，树的深度不会像二叉结构那么大，可以保证查询效率。

**参考文献：**

**类加载机制**

[**https://www.cnblogs.com/xdouby/p/5829423.html**](https://www.cnblogs.com/xdouby/p/5829423.html)

[**https://blog.csdn.net/htf2620032/article/details/79298661**](https://blog.csdn.net/htf2620032/article/details/79298661) **springlboot类加载**

[**http://www.cnblogs.com/zheting/p/6707035.html**](http://www.cnblogs.com/zheting/p/6707035.html) **springboot**

[**https://blog.csdn.net/leileibest\_437147623/article/details/80898878**](https://blog.csdn.net/leileibest_437147623/article/details/80898878) **springboot**

**Mysql索引原理**

[**https://blog.csdn.net/u010842515/article/details/68929687**](https://blog.csdn.net/u010842515/article/details/68929687)MyISAM和InnoDB**引擎区别**

**常用算法：**

[**https://blog.csdn.net/eson\_15/article/details/51138663**](https://blog.csdn.net/eson_15/article/details/51138663) **二叉树**

[**https://blog.csdn.net/u014182411/article/details/69831492**](https://blog.csdn.net/u014182411/article/details/69831492) **二叉树**

[**https://www.cnblogs.com/aiyelinglong/archive/2012/03/27/2419972.html**](https://www.cnblogs.com/aiyelinglong/archive/2012/03/27/2419972.html) **二叉树**

[**https://blog.csdn.net/eson\_15/article/details/51144079**](https://blog.csdn.net/eson_15/article/details/51144079) **红黑树**

[**https://www.jianshu.com/p/d25e490e2441**](https://www.jianshu.com/p/d25e490e2441) **红黑树 +1**

[**https://blog.csdn.net/tanrui519521/article/details/80980135**](https://blog.csdn.net/tanrui519521/article/details/80980135) **红黑树**

[**https://www.jianshu.com/p/7dedb7ebe033**](https://www.jianshu.com/p/7dedb7ebe033) **B树**

[**https://www.jianshu.com/p/6f68d3c118d6**](https://www.jianshu.com/p/6f68d3c118d6) **b+树**

[**https://www.jianshu.com/p/db226e0196b4**](https://www.jianshu.com/p/db226e0196b4) **b树、B+树**

[**https://www.cnblogs.com/boothsun/p/8970952.html**](https://www.cnblogs.com/boothsun/p/8970952.html) **二叉、b、b+树**

[**https://www.cnblogs.com/dreamroute/p/8484457.html**](https://www.cnblogs.com/dreamroute/p/8484457.html) **倒排索引**

[**https://blog.csdn.net/CrankZ/article/details/80615789**](https://blog.csdn.net/CrankZ/article/details/80615789) **倒排索引**