```
public void writeObject(ObjectOutputStream out) {
    try {
        //将this对象的成员序列化
        out.writeObject(this.getDocId());
        out.writeObject(this.getFreq());
        out.writeObject(this.getPositions());
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void readObject(ObjectInputStream in) {
    try{
        this.docId = (int)in.readObject();
        this.freq = (int)in.readObject();
        this.positions = (List<Integer>) in.readObject();
    } catch (ClassNotFoundException | IOException e) {
        e.printStackTrace();
    }
}

public void sort() {
    Collections.sort(list);
}

public AbstractDocument build(int docId, String docPath, AbstractTermTupleStream termTupleStream) {
    AbstractDocument document = new Document(docId,docPath);
    //依次从三元组流中读取加入文档Document对象中
    AbstractTermTuple termTuple = new TermTuple();
    while ((termTuple = termTupleStream.next())!=null){
        document.addTuple(termTuple);
    }
    return document;
}

public AbstractDocument build(int docId, String docPath, File file) {
    try {
        //将文件抽象成BufferedReader对象进行读取
        BufferedReader reader = new BufferedReader(new InputStreamReader(new FileInputStream(file)));
        //传入TermTupleScanner，将文档转换成词汇三元组流
        AbstractTermTupleStream termTupleStream = new TermTupleScanner(reader);
        //装饰者模式：进行单词长度过滤
        AbstractTermTupleStream lengthFilter = new LengthTermTupleFilter(termTupleStream);
        //装饰者模式：进行模式过滤
        AbstractTermTupleStream patternFilter = new PatternTermTupleFilter(lengthFilter);
        //装饰者模式：进行停用词过滤
        AbstractTermTupleStream stopWordsFilter = new StopWordTermTupleFilter(patternFilter);
        //调用（复用）上一个重载build函数得到文件对象
        return build(docId,docPath,stopWordsFilter);
    }catch (java.io.FileNotFoundException e){
        e.printStackTrace();
    }
    return null;
}

public void addDocument(AbstractDocument document) {
    //把该文档信息加入docIdToDocPathMapping
    docIdToDocPathMapping.put(document.getDocId(),document.getDocPath());
    //对文档中的三元组进行遍历加入termToPostingListMapping
    for (AbstractTermTuple termTuple:document.getTuples()){
        //判断当前termToPostingListMapping是否包含此单词
        if(termToPostingListMapping.containsKey(termTuple.term)){
            AbstractPostingList curPostingList = termToPostingListMapping.get(termTuple.term);
            //判断该词汇对应的PostingList是否包含该文件的Posting
            if(curPostingList.indexOf(document.getDocId())==-1){
                //不包含该文件的Posting，创造一个Posting加入当前PostingList中
                AbstractPosting curPosting = new Posting();
                curPosting.setFreq(termTuple.freq);
                curPosting.setDocId(document.getDocId());
                curPosting.getPositions().add(termTuple.curPos);
                curPostingList.add(curPosting);
            }else {
                //包含该文件的Posting，在该Posting中加入当前三元组对应的位置
                AbstractPosting curPosting = curPostingList.get(curPostingList.indexOf(document.getDocId()));
                curPosting.getPositions().add(termTuple.curPos);
                int freq = curPosting.getFreq();
                curPosting.setFreq(freq+1);
            }
        }else {
            //当前termToPostingListMapping不包含此单词，构建当前词汇和PostingList的映射并加入Map中
            AbstractPosting posting = new Posting();
            posting.setDocId(document.getDocId());
            posting.setFreq(termTuple.freq);
            posting.getPositions().add(termTuple.curPos);
            AbstractPostingList postingList = new PostingList();
            postingList.add(posting);
            termToPostingListMapping.put(termTuple.term,postingList);
        }
    }
}

public void load(File file) {
    try(ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(file))){
        this.docIdToDocPathMapping = (Map<Integer, String>) (inputStream.readObject());
        this.termToPostingListMapping = (Map<AbstractTerm,AbstractPostingList>)(inputStream.readObject());
    }catch (ClassNotFoundException e){
        e.printStackTrace();
    }catch (IOException e){
        e.printStackTrace();
    }
}

public void save(File file) {
    try(ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(file))){
        outputStream.writeObject(this.docIdToDocPathMapping);
        outputStream.writeObject(this.termToPostingListMapping);
    }catch (FileNotFoundException e){
        e.printStackTrace();
    }catch (IOException e){
        e.printStackTrace();
    }
}

public Set<AbstractTerm> getDictionary() {
    return termToPostingListMapping.keySet();
}

public AbstractIndex buildIndex(String rootDirectory) {
    //得到该目录下的文件名
    List<String> fileNames = FileUtil.list(rootDirectory,".txt");
    Collections.sort(fileNames);//对文件名从小到大排序
    AbstractIndex index = new Index();
    //遍历所有文件名
    for (String filename:fileNames){
        //把文件构造成Document对象
        AbstractDocument document = docBuilder.build(docId++,filename,new File(filename));
        if(document!=null){
            //把当前文件加入倒排索引中
            index.addDocument(document);
        }
    }
    //倒排索引序列化到对应目录的对应文件中
    index.save(new File(Config.INDEX_DIR + "index.dat"));
    return index;
}

public TermTupleScanner(BufferedReader input){
    this.input = input;
    //从input中读取所有内容
    String s = null;
    try{
        StringBuffer buf = new StringBuffer();
        while( (s = input.readLine()) != null){
            buf.append(s).append("\n"); //reader.readLine())返回的字符串会去掉换行符，因此这里要加上
        }
        s = buf.toString().trim(); //去掉最后一个多的换行符
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
    //判断是否要忽略大小写，若要忽略则转化成小写
    if(Config.IGNORE_CASE)
        s = s.toLowerCase();
    //进行分词
    StringSplitter splitter = new StringSplitter();
    splitter.setSplitRegex(Config.STRING_SPLITTER_REGEX);
    List<String> strings = splitter.splitByRegex(s);
    int pos = 0;
    //创建所有词汇的三元组list并保存为该类的实例数据成员，便于后续迭代读取
    for (String cur:strings){
        AbstractTermTuple termTuple = new TermTuple(cur,pos++);
        lists.add(termTuple);
    }
}

cur.term.getContent().matches(Config.TERM_FILTER_PATTERN)

public void open(String indexFile) {
    this.index.load(new File(indexFile));
}

public AbstractHit[] search(AbstractTerm queryTerm, Sort sorter) {
    List<AbstractHit> hitList = new ArrayList<AbstractHit>();
    for (Map.Entry<AbstractTerm, AbstractPostingList> entry:index.termToPostingListMapping.entrySet()){
        //遍历倒排索引，判断当前词汇是否为查询词
        if(entry.getKey().equals(queryTerm)){
            //找到查询词的PostingList，对应建立一个个Hit对象的并加入List中保存
            for (int i=0;i<entry.getValue().size();i++){
                AbstractPosting curPosting = entry.getValue().get(i);
                String path = index.docIdToDocPathMapping.get(curPosting.getDocId());
                AbstractHit hit = new Hit(curPosting.getDocId(),path);
                hit.getTermPostingMapping().put(entry.getKey(),curPosting);
                hitList.add(hit);
            }
        }
    }
    //计算Hit对象的分数
    for (AbstractHit hit:hitList){
        sorter.score(hit);
    }
    //排序
    sorter.sort(hitList);
    //将List转化为数组返回
    AbstractHit[] hits = new Hit[hitList.size()];
    for (int i=0;i<hitList.size();i++){
        hits[i]=hitList.get(i);
    }
    return hits;
}

public AbstractHit[] search(AbstractTerm queryTerm1, AbstractTerm queryTerm2, Sort sorter, LogicalCombination
combine) {
    List<AbstractHit> hitList = new ArrayList<AbstractHit>();
    //对两个检索词分别进行搜索得到命中数组
    AbstractHit[] hits1 = this.search(queryTerm1,sorter);
    AbstractHit[] hits2 = this.search(queryTerm2,sorter);
    if(combine == LogicalCombination.OR){
        //对其中一个数组直接加入全部命中
        for (AbstractHit hit1:hits1){
            hitList.add(hit1);
        }
        //遍历另一个数组
        for (AbstractHit hit2:hits2){
            AbstractHit thisHit = null;
            //对两个命中数组做一个合并，遍历查找hitList中是否包含该命中的相应文件id，因为Hit是通过docId来进行唯一性区
分的
            for (AbstractHit curHit:hitList){
                if(curHit.getDocId()==hit2.getDocId()){
                    thisHit=curHit;
                    break;
                }
            }
            if(thisHit!=null){
                //找到对应的hit，则直接在该hit中加入另一个词汇的信息即可
                thisHit.getTermPostingMapping().putAll(hit2.getTermPostingMapping());
            }else {
                //没有找到对应的hit，则直接加入
                hitList.add(hit2);
            }
        }
    }else if(combine == LogicalCombination.AND){
        //若为与，则双重循环查找两个命中数组中都有的文件，建立新的命中对象加入命中数组
        for (AbstractHit hit1:hits1){
            for (AbstractHit hit2:hits2){
                if(hit1.getDocId()==hit2.getDocId()){
                    AbstractHit hit = new Hit(hit1.getDocId(),hit1.getDocPath());
                    hit.getTermPostingMapping().putAll(hit1.getTermPostingMapping());
                    hit.getTermPostingMapping().putAll(hit2.getTermPostingMapping());
                    hitList.add(hit);
                }
            }
        }
    }
    //计算Hit对象的分数
    for (AbstractHit hit:hitList){
        sorter.score(hit);
    }
    //排序
    sorter.sort(hitList);
    //将List转化为数组返回
    AbstractHit[] hits = new Hit[hitList.size()];
    for (int i=0;i<hitList.size();i++){
        hits[i]=hitList.get(i);
    }
    return hits;
}
```

测试构建倒排索引：

```
AbstractDocumentBuilder documentBuilder = new DocumentBuilder();
AbstractIndexBuilder indexBuilder = new IndexBuilder(documentBuilder);
String rootDir = Config.DOC_DIR;
System.out.println("Start build index ...");
AbstractIndex index = indexBuilder.buildIndex(rootDir);
index.optimize();
System.out.println(index);  //控制台打印index的内容

//测试保存到文件
String indexFile = Config.INDEX_DIR + "index.dat";
index.save(new File(indexFile));    //索引保存到文件

//测试从文件读取
AbstractIndex index2 = new Index();  //创建一个空的index
index2.load(new File(indexFile));       //从文件加载对象的内容
System.out.println("\n-------------------\n");
System.out.println(index2);  //控制台打印index2的内容
```

测试搜索：

```
Sort simpleSorter = new SimpleSorter();
String indexFile = Config.INDEX_DIR + "index.dat";
AbstractIndexSearcher searcher = new IndexSearcher();
searcher.open(indexFile);

AbstractHit[] hits = searcher.search(new Term("coronavirus"),simpleSorter);
for(AbstractHit hit : hits){
    System.out.println(hit);
}
```