## 使用Java内存映射(Memory-Mapped Files)处理大文件

### NIO中的内存映射

**（1）什么是内存映射文件**

内存映射文件，是由一个文件到一块内存的映射，可以理解为将一个文件映射到进程地址，然后可以通过操作内存来访问文件数据。说白了就是使用虚拟内存将磁盘的文件数据加载到虚拟内存的内存页，然后就可以直接操作内存页数据。

我们读写一个文件使用read()和write()方法，这两个方法是调用系统底层接口来传输数据，因为内核空间的文件页和用户空间的缓冲区没有一一对应，所以读写数据时会在内核空间和用户空间之间进行数据拷贝，在操作大量文件数据时会导致性能很低，使用内存映射文件可以非常高效的操作大量文件数据。

通过内存映射机制操作文件比使用常规方法和使用FileChannel读写高效的多。

内存映射文件使用文件系统建立从用户空间到可用文件系统页的虚拟内存映射，这样做有以下好处：

- 用户进程把文件数据当内存数据，无需调用read()或write()
- 当用户进程接触到映射内存空间，会自动产生页错误，从而将文件数据从磁盘读到内存；若用户空间进程修改了内存页数据，相关页会自动标记并刷新到磁盘，文件被更新
- 操作系统的虚拟内存对内存页进行高速缓存，自动根据系统负载进行内存管理
- 用户空间和内核空间的数据总是一一对应，无需执行缓冲区拷贝
- 大数据的文件使用映射，无需消耗大量内存即可进行数据拷贝

**（2）如何创建内存映射文件**

```
RandomAccessFile raf = new RandomAccessFile("test.txt", "rw");
FileChannel fc = raf.getChannel();
//将test.txt文件所有数据映射到虚拟内存，并只读
MappedByteBuffer mbuff = fc.map(MapMode.READ_ONLY, 0, fc.size());
```

**（3）MappedByteBuffer API**

MappedByteBuffer是ByteBuffer的子类，所以可被通道读写。MappedByteBuffer提供的方法：

load()：加载整个文件到内存

isLoaded()：判断文件数据是否全部加载到了内存

force()：将缓冲区的更改刷新到磁盘

### 读取大文件

### 下面的测试转自 Java中用内存映射处理大文件

在处理大文件时，如果利用普通的FileInputStream 或者FileOutputStream 抑或RandomAccessFile 来进行频繁的读写操作，都将导致进程因频繁读写外存而降低速度.

如下为一个对比实验：

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class Test {


public static void main(String[] args) {
try {
FileInputStream fis=new FileInputStream("/home/tobacco/test/res.txt");
int sum=0;
int n;
long t1=System.currentTimeMillis();
try {
while((n=fis.read())>=0){
sum+=n;
}
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

try {
FileInputStream fis=new FileInputStream("/home/tobacco/test/res.txt");
BufferedInputStream bis=new BufferedInputStream(fis);
int sum=0;
int n;
long t1=System.currentTimeMillis();
try {
while((n=bis.read())>=0){
sum+=n;
}
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

MappedByteBuffer buffer=null;
try {
buffer=new RandomAccessFile("/home/tobacco/test/res.txt","rw").getChannel().map(FileChannel.MapMode.READ_WRITE, 0, 1253244);
```

```
int sum=0;
int n;
long t1=System.currentTimeMillis();
for(int i=0;i<1253244;i++){
n=0x000000ff&buffer.get(i);
sum+=n;
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}

}
```

测试文件为一个大小为1253244字节的文件。测试结果：

```
sum:220152087 time:1464
sum:220152087 time:72
sum:220152087 time:25
```

说明读数据无误。删去其中的数据处理部分：

```
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class Test {


public static void main(String[] args) {
try {
FileInputStream fis=new FileInputStream("/home/tobacco/test/res.txt");
int sum=0;
int n;
long t1=System.currentTimeMillis();
try {
while((n=fis.read())>=0){
//sum+=n;
}
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

try {
FileInputStream fis=new FileInputStream("/home/tobacco/test/res.txt");
BufferedInputStream bis=new BufferedInputStream(fis);
int sum=0;
int n;
long t1=System.currentTimeMillis();
try {
while((n=bis.read())>=0){
//sum+=n;
}
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

MappedByteBuffer buffer=null;
try {
buffer=new RandomAccessFile("/home/tobacco/test/res.txt","rw").getChannel().map(FileChannel.MapMode.READ_WRITE, 0, 1253244);
int sum=0;
int n;
long t1=System.currentTimeMillis();
for(int i=0;i<1253244;i++){
//n=0x000000ff&buffer.get(i);
//sum+=n;
}
long t=System.currentTimeMillis()-t1;
System.out.println("sum:"+sum+" time:"+t);
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}

}
```



## 相册



## 积分与排名

积分 - 285749
排名 - 766

## 阅读排行榜



## 评论排行榜

```
75
76  }
```

测试结果：

```
sum:0 time:1458
sum:0 time:67
sum:0 time:8
```

由此可见，将文件部分或者全部映射到内存后进行读写，速度将提高很多。
这是因为内存映射文件首先将外存上的文件映射到内存中的一块连续区域，被当成一个字节数组进行处理，读写操作直接对内存进行操作，而后再将内存区域重新映射到外存文件，这就节省了中间频繁的对外存进行读写的时间，大大降低了读写时间。



分类：Network



邴越
关注 - 39
粉丝 - 374
+加关注



« 上一篇： Spring中的设计模式学习
» 下一篇： 《Java程序性能优化》之设计优化

posted @ 2014-05-14 13:21 邴越 阅读(547) 评论(0) 编辑 收藏