链滴

# JavaIO 与 NIO 下载网络文件详解

作者：liumapp

原文链接：https://ld246.com/article/1543914223766

来源网站：链滴

给定一串url，如何用最高效的方式下载到本地，这篇博文记录了Java IO与NIO两种实现方式，并对现原理进行了一定梳理。

案例代码：github/qtools/UrlDownloadTool.java

# 1.1 使用JavaIO

就下载文件而言，一般最常使用的就是Java IO。直接用URL类就可以跟网络资源建立连接再下载，并过openStream()方法来获得一个输入流。

```
BufferedInputStream in = new BufferedInputStream(new URL(FILE_URL).openStream());
```

上面的代码，我用到了BufferedInputStream，通过缓存的形式来提升性能：

每一次用read()方法读取一个字节时，都会调用一次底层文件系统，所以每当JVM调用read()的时候程序执行上下文都会从用户模式切换到内核模式，执行结束后再切换回来。

从性能角度来看，这种上下文切换的成本是高昂的：比如我们在读取一个字节数很高的文件时，大量上下文切换将会很影响程序性能。

所以这里我们最好使用BufferedInputStream来规避这种情况（具体原理请见下文）

而要把读取到的URL文件字节写入到本地文件，一般直接用FileOutputSream类的write()方法就可以：

```
try (BufferedInputStream in = new BufferedInputStream(new URL(FILE_URL).openStream());
  FileOutputStream fileOutputStream = new FileOutputStream(FILE_NAME)) {
    byte dataBuffer[] = new byte[1024];
    int bytesRead;
    while ((bytesRead = in.read(dataBuffer, 0, 1024)) != -1) {
        fileOutputStream.write(dataBuffer, 0, bytesRead);
    }
} catch (IOException e) {
    // handle exception
}
```

在使用BufferedInputStream的时候，read()方法会根据我们设置的buffer size一次性读取等量的字（不设置的话，jkd1.8里是默认8192个字节）

上面的示例代码里，dataBuffer已经规定了一次性读取1024个字节，所以第二次读取的时候就不需要使用BufferedInputStream了

上面那个示范代码其实是针对jdk1.6等版本的，jkd1.7以后，实现同样的功能不需要这么啰嗦了

一个Files.copy()方法就可以搞定

```
InputStream in = new URL(FILE_URL).openStream();
Files.copy(in, Paths.get(FILE_NAME), StandardCopyOption.REPLACE_EXISTING);
```

使用Java IO实现网络资源的下载就是这么简单，不过它也有缺点：所有的缓存字节都会直接存储在存中

而使用NIO的话，我们就不需要用到缓存，而是直接从两个通道进行字节的流动

## 1.2 使用 NIO

首先从URL stream中创建一个ReadableByteChannel来读取网络文件:

```
ReadableByteChannel readableByteChannel = Channels.newChannel(url.openStream());
```

通过ReadableByteChannel读取到的字节会流动到一个FileChannel中，然后再关联一个本地文件进下载操作:

```
FileOutputStream fileOutputStream = new FileOutputStream(FILE_NAME);
FileChannel fileChannel = fileOutputStream.getChannel();
```

最后用transferFrom()方法就可以把ReadableByteChannel获取到的字节写入本地文件:

```
fileOutputStream.getChannel()
    .transferFrom(readableByteChannel, 0, Long.MAX_VALUE);
```

transferTo()或者transferFrom()方法明显比之前的创建缓存区保存字节要有效率的多，因为数据可直接移动到文件系统而不需要复制任何字节到程序的内存栈中

尤其是在Linux或者Unix操作系统中，这种方式使用了一种称之为zero-copy的技术，来减少上下文内核模式和用户模式之间的切换次数

## 1.3 恢复下载

考虑到网络连接偶尔会有中断的情况，而网络中断后恢复下载明显要比重新下载要有效率的多

所以接下来记录如何实现恢复下载的功能

首先要做的就是先记录下要下载文件的大小，这一步可以直接通过HTTP HEAD来获取:

```
URL url = new URL(FILE_URL);
HttpURLConnection httpConnection = (HttpURLConnection) url.openConnection();
httpConnection.setRequestMethod("HEAD");
long remoteFileSize = httpConnection.getContentLengthLong();
```

拿到要下载的文件大小后，就可以对文件是否下载完成进行判断

如何没有下载完成，那么第二次下载直接从最新的一个字节开始下载即可:

```
long existingFileSize = outputFile.length();
if (existingFileSize < fileLength) {
    httpFileConnection.setRequestProperty(
      "Range",
      "bytes=" + existingFileSize + "-" + fileLength
    );
}
```

剩下的事情跟前面介绍的方法一样，唯一要改动的代码就是:

```
OutputStream os = new FileOutputStream(FILE_NAME, true);
```