

---

# Compression Zip et GZip avec l'API Zip

Hugo Etiévant

*Dernière mise à jour : 20 juin 2004*

# Introduction

---

Un programme informatique manipule souvent des fichiers dont certains doivent être compressés du fait de leur volume ou par soucis de créer une archive unique facilement distribuable.

Le format Zip est largement répandu, son algorithme est simple et libre.

# Présentation de l'API Zip

---

L'API Zip de Java est présente dans le JDK depuis la version 1.1

Elle permet de manipuler des fichiers au format **Zip** et **GZip**. L'algorithme utilisé est la méthode **DEFLATE**. Cette API fournit également des méthodes utilitaires pour le contrôle de l'intégrité des fichiers via les méthodes **CRC-32** et **Adler-32**.

Cette API s'appelle de cette façon :

```
import java.util.zip.*;
```

# Compression (1)

---

La compression nécessite de connaître :

- le ou les fichiers à compresser
- le nom de l'archive à créer
- la méthode de compression (optionnelle)
- le taux de compression (optionnelle)

# Compression (2) – fichier de sortie

---

0. Une valeur de taille de tampon pour les buffers d'entrée et de sortie utilisés par la suite, et un buffer de données

```
static final int BUFFER = 2048;  
byte data[] = new byte[BUFFER];
```

1. Création d'un flux d'écriture vers un fichier, ce fichier sera l'archive Zip finale

```
FileOutputStream dest = new FileOutputStream("archive.zip");
```

2. Création d'un buffer de sortie afin d'améliorer les performances d'écriture

```
BufferedOutputStream buff = new BufferedOutputStream(dest);
```

3. Création d'un flux d'écriture Zip vers ce fichier à travers le buffer

```
ZipOutputStream out = new ZipOutputStream(buff);
```

# Compression (3) – paramètres

---

4. Spécifier la méthode de compression désirée

```
out.setMethod(ZipOutputStream.DEFLATED);
```

5. Spécifier le taux de compression (entier positif entre 0 et 9)

```
out.setLevel(9);
```

# Compression (4) – entrées de l'archive

---

6. Lister les fichiers à compresser. Dans notre exemple, ils sont supposés être dans le tableau files[]

```
for(int i=0; i<files.length; i++) {  
    FileInputStream fi = new FileInputStream(files[i]);  
    ...
```

7. Leur créer un buffer d'entrée

```
BufferedInputStream buffi = new BufferedInputStream(fi, BUFFER);
```

8. Pour chacun d'eux, créer une entrée Zip

```
ZipEntry entry = new ZipEntry(files[i]);
```

9. Et affecter cette entrée au flux de sortie

```
out.putNextEntry(entry);
```

# Compression (5) – écriture de l'archive

---

10. Écriture des entrées dans le flux de sortie par paquet de taille égale aux tampons d'entrée et de sortie

```
int count;
while((count = buffi.read(data, 0, BUFFER)) != -1) {
    out.write(data, 0, count);
}
```

11. Fermeture de l'entrée en cours

```
out.closeEntry();
```

12. Fermeture des flux

```
buffi.close();
}
out.close();
```

Et voilà, votre archive Zip est créée. Elle pourra être décompressée avec WinZip (Windows) ou unzip (Unix).

# Listage des entrées d'une archive (1)

---

Pour lister le contenu d'une archive Zip et en afficher les propriétés il faut procéder ainsi :

## 1. Ouverture du fichier Zip

```
ZipFile zf = new ZipFile("archive.zip");
```

## 2. Extraction de ses entrées

```
Enumeration entries = zf.entries()
```

## 3. Parcours de chacune des entrées

```
while(entries.hasMoreElements()) {  
    ZipEntry e = (ZipEntry)entries.nextElement();  
    ...  
}
```

# Listage des entrées d'une archive (2)

---

## 4. Extraction des informations désirées

```
System.out.println(e.getName());  
}
```

Et voilà !

# Propriétés d'une entrée (1)

Méthode	Description
String <code>getComment()</code>	Retourne le commentaire associé à l'entrée, null si aucun
long <code>getCompressedSize()</code>	Retourne la taille de l'entrée après compression, -1 si inconnue
long <code>getCrc()</code>	Retourne le contrôle d'erreur cyclique CRC ou -1 si inconnu
byte[] <code>getExtra()</code>	Retourne le champs optionnel ou -1 si inconnu
int <code>getMethod()</code>	Retourne la méthode de compression employée, -1 si inconnue
String <code>getName()</code>	Retourne le nom de l'entrée
long <code>getSize()</code>	Retourne la taille de l'entrée avant compression, -1 si inconnue
long <code>getTime()</code>	Retourne la date et heure de dernière modification, -1 si inconnue
int <code>hashCode()</code>	Retourne le hashcode (condensat) de cette entrée
boolean <code>isDirectory()</code>	Retourne vrai (TRUE) si l'entrée est un répertoire
String <code>toString()</code>	Retourne une représentation de l'entrée sous forme de chaîne de caractères

# Propriétés d'une entrée (2)

---

Les getters précédents ont leurs setters associés ci-bas :

Méthode	Description
void <b>setComment</b> (String c)	Affectation du commentaire associé à l'entrée
void <b>setCompressedSize</b> (long csize)	Affectation de la taille de l'entrée après compression
void <b>setCrc</b> (long crc)	Affectation du CRC de l'entrée
void <b>setExtra</b> (byte[] extra)	Affectation d'un champs optionnel
void <b>setMethod</b> (int method)	Affectation d'une méthode de compression à l'entrée
void <b>setSize</b> (long size)	Affectation de la taille de l'entrée avant compression
void <b>setTime</b> (long time)	Affectation de la date et heure de dernière modification

# Ratio

---

Le calcul du ratio s'effectue ainsi :

$$\frac{\left( \begin{array}{c} \text{taille sans} \\ \text{compression} \end{array} - \begin{array}{c} \text{taille après} \\ \text{compression} \end{array} \right) \times 100}{\text{taille sans compression}}$$

Exemple :

```
System.out.println(e.getName() + " ratio: " + ((e.getSize() -  
e.getCompressedSize()) * 100) / e.getSize() + "%");
```

# Date

---

La date est au format TimeStamp (nombre de seconde écoulées entre la date et le 1er janvier 1970), il est alors nécessaire de la formater au format voulu (ici, francophone).

Exemple :

```
DateFormat df = new SimpleDateFormat ("dd/mm/yyyy  
hh:mm:ss");  
Date d = null;  
d = new Date(e.getTime());  
System.out.println(" date: "+df.format(d));
```

# Méthode

---

Il n'existe que deux méthodes : **DEFLATED** (avec compression) et **STORED** (sans compression).

Exemple :

```
String method = null;
if (e.getMethod() == ZipEntry.DEFLATED) {
    method = new String("DEFLATED");
} elseif (e.getMethod() == ZipEntry.STORED) {
    method = new String("STORED");
}
```

# Qualité de la compression

---

Il existe 10 niveaux numérotés de 0 à 9.

Il existe aussi des constantes :

Constante	Description	Niveau
<code>Deflater.BEST_COMPRESSION</code>	Meilleur compression	9
<code>Deflater.DEFAULT_COMPRESSION</code>	Compression par défaut	5
<code>Deflater.BEST_SPEED</code>	Meilleur rapidité	1
<code>Deflater.NO_COMPRESSION</code>	Pas de compression	0

Exemple :

```
out.setLevel(Deflater.BEST_COMPRESSION);
```

# Décompression (1) – fichiers d'entrée

---

0. Une valeur de taille de tampon pour les buffers d'entrée et de sortie utilisés par la suite, et un buffer de données

```
static final int BUFFER = 2048;  
byte data[] = new byte[BUFFER];
```

1. Déclaration d'un fichier destination

```
BufferedOutputStream dest = null;
```

2. Ouverture du fichier à décompresser

```
FileInputStream fis = new FileInputStream("archive.zip");
```

3. Ouverture buffer sur ce fichier

```
BufferedInputStream buffi = new BufferedInputStream(fis);
```

4. Ouverture de l'archive Zip via ce buffer

```
ZipInputStream zis = new ZipInputStream(buffi);
```

# Décompression (2) – écriture

---

5. Parcours des entrées de l'archive

```
ZipEntry entry;  
while((entry = zis.getNextEntry()) != null) {  
    ...  
}
```

6. Création du fichier de sortie à partir du nom de l'entrée

```
FileOutputStream fos = new FileOutputStream(entry.getName());
```

7. Affectation au buffer de sortie de ce flux vers fichier

```
dest = new BufferedOutputStream(fos, BUFFER);
```

8. Écriture sur disque

```
while ((count = zis.read(data, 0, BUFFER)) != -1) {  
    dest.write(data, 0, count);  
}
```

# Décompression (3) – fermeture

---

9. Vidage du tampon en écriture

```
dest.flush();
```

10. Fermeture du flux de sortie

```
dest.close();
```

```
}
```

11. Fermeture de l'archive

```
zis.close();
```

# Checksums

---

Pour inclure un contrôle d'intégrité dans vos applications, il faut ajouter un flux de contrôle entre le flux d'écriture sur fichier et le flux de buffer de sortie. Il existe deux méthodes de contrôle : **Adler32** et **CRC32**. Le premier est le plus rapide.

Exemple :

```
// création d'un flux d'écriture sur fichier
FileOutputStream dest = new FileOutputStream("archive.zip");
// ajout du checksum : Adler32 (plus rapide) ou CRC32
CheckedOutputStream checksum =
    new CheckedOutputStream(dest, new Adler32());
// création d'un buffer d'écriture
BufferedOutputStream buff = new BufferedOutputStream(checksum);
// création d'un flux d'écriture Zip
ZipOutputStream out = new ZipOutputStream(buff);
```

# Accents

---

Malheureusement, les accents et caractères spéciaux sont mal interprétés par l'API `java.util.zip`, il faut donc transformer les noms qui en contiennent.

Exemple d'une fonction effectuant la conversion :

```
import sun.text.Normalizer;
public static String unAccent(String s) {
    String temp = Normalizer.normalize(s, Normalizer.DECOMP, 0);
    return temp.replaceAll("[^\\p{ASCII}]", "");
}
```

Que l'on ajoute ici :

```
ZipEntry entry = new ZipEntry(unAccent(files[i]));
```

# Méthode rapide : ZipFile

---

(1)

Il existe un moyen d'accéder rapidement aux entrées d'une archive :

```
ZipFile zipfile = new ZipFile("archive.zip");
Enumeration entries = zipfile.entries();
while (entries.hasMoreElements()) {
    ZipEntry e = ((ZipEntry)entries.nextElement());
    System.out.println(e.getName());
}
zipfile.close();
```

# Méthode rapide : **ZipFile**

(2)

Voici la liste des méthodes de la classe ZipFile :

Méthode	Description
<code>ZipFile(String name)</code>	Ouvre l'archive Zip
<code>void close()</code>	Ferme l'archive Zip
<code>Enumeration entries()</code>	Retourne l'ensemble des entrées
<code>void finalize()</code>	Fermeture de l'archive lorsqu'inactive
<code>ZipEntry getEntry(String name)</code>	Retourne l'entrée de nom spécifié
<code>InputStream getInputStream(ZipEntry entry)</code>	Retourne un flux d'entrée vers le fichier associé à l'entrée spécifiée
<code>String getName()</code>	Retourne le chemin de l'archive
<code>int size()</code>	Retourne le nombre d'entrées

# Historique

---

- ▶ **20 juin 2004** : création du document (24 diapos)

Agissez sur la qualité de ce document en envoyant vos critiques et suggestions à l'auteur.

Pour toute question technique, se reporter au forum Java de [Developpez.com](http://Developpez.com)

Reproduction autorisée uniquement pour un usage non commercial.

Hugo Etiévant  
[cyberzoide@yahoo.fr](mailto:cyberzoide@yahoo.fr)

<http://cyberzoide.developpez.com/>

