Attendre le chargement d'une image

Michel CASABIANCA
casa@sweetohm.net

Java est un langage qui a été créé avec internet en tête. Cette simple constatation peut se faire sentir à chaque détour de ce langage. C'est le cas pour les images : dans un langage "classique", lorsqu'on demande le chargement d'une image en mémoire, le déroulement du programme est arrêté pendant l'opération qui est supposée être très brève puisque l'image est sur disque. Si cette image doit être rapatriée à partir du réseau, cette opération sera beaucoup plus longue, et le programme peut être bloqué inutilement. C'est pourquoi les concepteurs de Java ont adopté un système de chargement en parallèle : lorsqu'on demande le chargement d'une image, celle ci est chargée en tâche de fond, et le programme continue.

Cette méthode de chargement différé des images est souvent très utile, mais il est des cas où il est préférable de s'en passer et de suspendre l'exécution du programme pendant le chargement. C'est le cas lorsqu'on veut faire certaines opérations graphiques avec cette image, ou bien encore si l'on travaille en local. D'autre part, on peut préférer cette méthode pour des motifs d'ordre esthétique (on peut ne pas aimer voir s'afficher les images progressivement). Nous allons donc étudier dans la suite de cet article comment procéder pour attendre le chargement d'une image.

Première méthode : imageUpdate()

Cette méthode fait appel à la méthode imageUpdate() qu'implémente tout ImageObserver (et donc en particulier une Applet). Un ImageObserver est un objet qui est intéressé par le chargement d'une image, et qui veut être informé de son déroulement. En pratique, ImageObserver est une interface qui ne comporte qu'une seule méthode :

```
public boolean imageUpdate(Image img,int infoFlags,
  int x,int y,int width,int height)
```

Cette méthode est appelée régulièrement au cours du chargement de cette image. Comment le système sait-il que cet objet est intéressé par le déroulement du chargement de cette image ? Tout simplement parceque vous lui avez dit ! En effet, la méthode pour afficher une image (drawImage(Image img,int x,int y,ImageObserver observer)) demande en dernier argument un ImageObserver. Souvent cet ImageObserver est l'applet elle même, donc on passe en dernier argument this. Il existe d'autres méthodes demandant en argument un ImageObserver : checkImage, prepareImage, getHeight, getWidth et getProperty. Toutes ces méthodes demandent au préalable le chargement de l'image. Contrairement à ce qu'on pourrait croire, la méthode getImage ne charge pas réellement l'image, mais indique où l'on peut trouver l'image. L'image n'est chargée que lorsqu'on en a besoin, donc lorsqu'on appelle l'une des fonctions qui demandent un ImageObserver en argument.

La question que l'on peut se poser concernant imageUpdate() est : "A quoi ça sert ?". Cette méthode renseigne sur le chargement de l'image. Le champ le plus important est infoFlags : il contient, sous forme de flags (donc de bits) les renseignements suivants :

- WIDTH : la largeur de l'image est maintenant connue. Donc un appel à getWidth() renverra la largeur de l'image (et non pas -1 qui est la valeur renvoyée lorsqu'elle n'est pas encore connue).
- HEIGHT : idem pour la hauteur de l'image.
- PROPERTIES : les propriétés de l'image sont connues, donc un appel à getProperties() sera renseigné.
- SOMEBITS : indique que quelques pixels de l'image ont été chargés. Lorsque ce flag est activé, x, y, width et heigth donnent la portion de l'image qui est chargée (et donc que l'on peut maintenant dessiner).
- FRAMEBITS : indique qu'une frame au moins d'une image multiframe a été chargée. Les champs x, y, width et heigth n'ont alors aucune signification.
- ALLBITS : indique que toute l'image a été chargée et qu'on peut maintenant l'afficher.
- ERROR : indique qu'une erreur s'est produite lors du chargement. ABORT peut être activé ou non. Une deuxième tentative de chargement est inutile car elle échoura. Une telle erreur peut se produire si l'URL de l'image n'est pas valide ou si le fichier image lui même n'est pas valide.
- ABORT : indique une erreur récupérable (si ERROR n'est pas activé). On peut donc retenter un chargement de l'image. Cette erreur peut survenir s'il s'est produit une erreur sur le réseau lui même.

Pour savoir si un de ces drapeaux est activé, on fera un test avec un ET logique. Par exemple pour savoir si l'image est chargée (le test le plus courant dans une méthode imageUpdate), on fera le test suivant :

```
if((infoFlags & ImageObserver.ALLBITS)==ALLBITS) {
  ...
}
```

Dans le corps de cette méthode imageUpdate(), on fera les tests qui s'imposent et on exécutera le code correspondant. Donc pour attendre l'affichage de l'image, il ne faut rien faire tant que toute l'image n'est pas chargée, et l'afficher lorsque le flag ALLBITS est activé, ce qui donnera le code suivant pour la méthode :

```
public boolean imageUpdate(Image img,int infoFlags,
  int x,int y,int width,int height) {
  if((infoFlags & ImageObserver.ALLBITS)==ALLBITS) {
    repaint();
    return false;
  }
  else return true;
}
```

Il reste encore à éclaicir une chose qui peut sembler étrange : la valeur de retour booléenne de cette méthode imageUpdate(). Cette valeur indique si l'on doit continuer le chargement. Donc, tant que l'image n'est pas totalement chargée, on doit retourner true (pour continuer le chargement), et retourner false lorsqu'elle est entièrement chargée.

Nous pouvons, maintenant que nous avons vu la théorie, passer aux travaux pratiques avec un petit

exemple tout simple : une applet qui charge une image, mais qui ne l'affiche que lorsqu'elle est complètement chargée :

```
import java.awt.*;
import java.awt.image.*;

public class AttenteImages extends java.applet.Applet {
  /** image à charger */
  Image image;

  /** on renseigne sur la localisation de l'image */
  public void init() {
    image=getImage(getDocumentBase(),"image.jpg");
  }

  /** routine d'affiche de l'applet : affiche l'image */
  public void paint(Graphics g) {
    /* si l'image ne peut encore être affichée, on affiche une message
    indiquant que l'image est en cours de chargement */
    if(!g.drawImage(image,0,0,this)) {
      g.drawString("Image en cours de chargement...",20,128);
    }
  }

  /** bloque l'affichage tant que l'image n'est pas chargée */
  public boolean imageUpdate(Image img,int infoFlags,
    int x,int y,int width,int height) {
    /* on teste le flag de fin de chargement de l'image */
    if((infoFlags & ImageObserver.ALLBITS)==ALLBITS) {
      /* si c'est OK, on affiche l'image */
      repaint();
      return false;
    }
    else return true;
  }
}
```

Deuxième méthode : MediaTracker

Cette deuxième méthode utilise la classe MediaTracker qui permet (comme son nom l'indique) de suivre le chargement d'un document. Il faut alors procéder en 4 temps :

Donner l'URL de l'image à l'objet Image

```
image=getImage(getCodeBase(),"image.jpg");
```

Créer une instance de MediaTracker

```
MediaTracker tracker=new MediaTracker(this);
```

Le lecteur attentif aura remarqué qu'en argument du constructeur du MediaTracker, on envoie this, donc une référence sur l'applet. Cette référence n'est autre qu'un moyen pour indiquer l'ImageObserver concerné par le chargement de l'image.

Ajouter l'image dans un groupe du MediaTracker

```
tracker.addImage(image,0);
```

En argument à la méthode addImage(), le MediaTracker demande un numéro de groupe. Ce numéro permet de créer des "lots" d'images que l'on pourra traiter différemment. Ceci dit, dans la plupart des cas, on placera toutes les iamges à charger dans un même groupe.

Demander le chargement de l'image

```
try {tracker.waitForID(0);}
catch(InterruptedException e) {}
```

Pour finir, la méthode waitForID() charge les images du groupe en bloquant le cours du programme. Cette méthode peut éjecter une InterruptedException qu'il faut donc intercepter. Dans ce court exemple, je ne prends pas de mesures si le chargement de l'image a été interrompu, mais il est évident qu'on a intérêt à traiter cette exception (en affichant un message d'erreur et en arrêtant le programme par exemple). On peut savoir s'il s'est produit une erreur lors du chargement avec les méthodes isErrorAny() ou isErrorID(),et récupérer les messages d'erreur avec getErrorAny() et getErrorID().

Cette méthode étant bloquante, on pourra la placer dans un thread pour ne pas interrompre l'exécution du programme.

Voici maintenant le source d'un exemple complet qui fait exactement la même chose que le précédent :

```
import java.awt.*;

public class AttenteImages2 extends java.applet.Applet
{
  /** image à afficher */
  Image image;

  public void init() {
    /* donne l'URL de l'image */
    image=getImage(getDocumentBase(),"image.jpg");

    /* on crée le média tracker, on y inscrit l'image et on la charge */
    MediaTracker tracker=new MediaTracker(this);
    tracker.addImage(image,0);
    try {tracker.waitForID(0);}
    catch(InterruptedException e) {}
  }
```

```
        public void paint(Graphics g) {
          g.drawImage(image,0,0,this);
        }
      }
```

Conlusion

Maintenant que nous avons vu ces deux méthodes, nous pouvons nous demander laquelle est la meilleure. A mon humble avis, c'est celle du MediaTracker : elle est plus simple à implémenter et elle parait plus sure car le code est bien localisé (on s'occupe du chargement de l'image une fois pour toute, et après on n'a plus à s'en préocuper). Personnellement, je n'utilise que le MediaTracker.

Vous pouvez charger les sources des exemples en cliquant ici.