4

Gestion des notifications et des événements

Dans ce chapitre, nous abordons le système de notifications utilisé en Objective-C. Dans le jargon Cocoa, les termes *événement* (*event*) et *notification* (*notification*) désignent deux concepts très proches, mais distincts, qui sont en général confondus dans par les autres langages de programmation sous le terme *événement* (par exemple, par Java ou C#).

Pour nous, le terme événement désigne une action provenant du matériel (souris, clavier, etc.), tandis que notification désigne le moyen par lequel un objet notifie les autres objets de l'occurrence d'un

événement logiciel. Ce système ce base sur le *modèle de conception* (*design pattern*) appelé *publication/abonnement* (*publish/subscribe*) qui fera l'objet de la première section.

Principe du modèle de conception publication/abonnement

Les développeurs C# et Java sont habitués à l'utilisation d'objets de type *événement* pour la gestion des événements : C# possède le mot-clé `event` permettant d'introduire un membre de type événement parmi les différentes variables d'instance de votre classe. Java permet d'obtenir le même résultat, mais en dérivant une nouvelle classe depuis `java.util.EventObject`. Le modèle de conception implémenté ici au niveau langage est le modèle *observateur/observable*.

Ce modèle consiste donc à mettre en relation directe les objets intéressés par les changements d'états d'un autre objet. On dit alors que les objets observateurs s'abonnent à certains événements de l'objet tiers (*observable*) qui prend alors à sa charge le rôle de les notifier lorsqu'une certaine propriété est modifiée ou lorsqu'un certain événement se produit.

Objective-C propose une généralisation de cette approche, appelée le modèle de conception *publication/abonnement* : ce ne sont pas les ici les observables

(objets observés) qui prennent à leur charge d'informer les observateurs, mais un intermédiaire appelé centre de notifications qui, dans le cas de Cocoa est une instance de la classe `NSNotificationCenter`.

Un centre de notifications est donc tout simplement un objet qui prend à sa charge la transmission des notifications depuis l'objet émetteur vers l'objet observateur. Une analogie serait par exemple Tweeter : vous (objet observé) envoyez un *tweet* (notification) à Tweeter.com qui occupe le rôle du centre de notifications qui va l'acheminer vers le destinataire (observateur) en leur envoyant, par exemple, un SMS.

Cette implémentation est plus générique et découple totalement les différents objets : un objet éditeur envoie ses événements au centre de notifications et n'a donc besoin de rien savoir a propos de ses abonnés, et les abonnés ne communiquent qu'avec le centre de notifications.

Les principaux avantages résultant de ce découplage sont qu'un objet envoyant une notification n'est pas en charge de savoir si des observateurs sont abonnés, et de plus le centre de notifications faisant office d'intermédiaire, l'objet n'a pas non plus besoin de savoir si les objets sont locaux ou pas. Il est ainsi possible d'abstraire les communications distantes et de créer des systèmes distribués très facilement.

Obtenir le centre de notifications par défaut

```
[NSNotificationCenter defaultCenter]
```

Chaque processus dispose d'un centre de notifications par défaut, et vous n'avez pas à créer d'instance de cette classe. Vous obtenez l'instance par défaut en envoyant le message `defaultCenter` à l'objet-classe `NSNotificationCenter`.

Comme nous l'avons vu dans la section précédente, il est possible de créer de manière transparente un centre de notifications distribué. `NSNotificationCenter` n'envoie les notifications que dans le processus en cours d'exécution. Mais il existe également `NSDistributedNotificationCenter` qui permet d'envoyer les notifications vers différents processus (uniquement en local dans l'implémentation actuelle). Les notifications distribuées sont beaucoup plus lourdes, il ne faut donc pas de les utiliser par défaut.

Les notifications distribuées sont très importantes pour le développement d'applications distribuées. Toutefois, `NSDistributedNotificationCenter` et les sujets relatifs sortent du cadre de cet ouvrage et nous vous recommandons de vous reporter à la documentation d'Apple pour les découvrir.

Poster une notification synchrone

```
    [[NSNotificationCenter defaultCenter]
    ➥ postNotification:[NSNotification
    ➥ notificationWithName:@"maNotification"
    ➥ object:instance]];
//équivalent à :
NSNotification * notif = [NSNotification
➥ notificationWithName:@"maNotification"
➥ object:instance]
[[NSNotificationCenter defaultCenter]
➥ postNotification:notif];
//équivalent à :
[[NSNotificationCenter defaultCenter]
➥ postNotification:@"maNotification"
➥ object:instance]];
```

Comme vous pouvez le voir, il est assez simple d'envoyer une notification synchrone : il suffit de créer une nouvelle instance de `NSNotification` et de l'envoyer au centre de notifications par défaut.

Il est également possible d'envoyer directement le message `postNotification` au centre par défaut avec les arguments nécessaires à la création de l'instance de `NSNotification`, et le centre de notifications gérera le reste.

Il existe trois versions de la méthode `postNotification`, deux d'entre-elles étant des versions simplifiées de la principale :

```
- postNotification:  //version principale
                  //NSNotification comme argument
- postNotificationName:object: //userInfo est null
- postNotificationName:object:userInfo:
```

Voici un autre exemple montrant l'utilisation de `postNotificationName:object:userInfo:` et donc également comment il est possible de passer des informations supplémentaires sous la forme du dictionnaire `userInfo` :

```
NSDictionary* dic = [NSDictionary
➥ dictionaryWithObjectsAndKeys: @"clef1",
➥ [NSNumber numberWithInt:1], @"clef2",
➥ [NSNumber numberWithInt:2], @"clef3",
➥ [NSNumber numberWithInt:3], nil];
[[NSNotificationCenter defaultCenter]
➥ addObserver:instance
➥ selector:@selector(afficherNotification:)
➥ name:@"maNotification" object:instance];
[[NSNotificationCenter defaultCenter]
➥ postNotificationName:@"maNotification"
➥ object:instance userInfo:dic];
```

Les notifications envoyées ainsi au centre de notifications sont synchrones : la méthode `postNotification:` ne rend la main qu'une fois toutes

les notifications envoyées et que les observateurs ont traité la notification. C'est donc une manière de procéder peu performante.

Poster une notification asynchrone

```
NSNotification * notification = [NSNotification
➥ notificationWithName:@"maNotification"
➥ object:instance userInfo:dic];
[[NSNotificationQueue defaultQueue]
➥ enqueueNotification:notification
➥ postingStyle:NSPostASAP];
```

Comme nous l'avons vu à la section précédente, les envois de notifications au centre par défaut *via* les différentes méthodes `postNotification:` sont synchrones. Il est possible d'envoyer les notifications de manière asynchrone, *via* l'utilisation des files d'attente de notifications, instances de la classe `NSNotificationQueue`. Nous verrons dans la section suivante que ces files fournissent une seconde fonctionnalité très importante.

La méthode de classe `defaultQueue` retourne la file d'attente par défaut du centre de notifications par défaut.

Info

Dans l'exemple précédent, nous avons créé une instance de `NSNotification` que nous passons à la file d'attente par défaut *via* la fonction `enqueueNotification:postingStyle:`. Nous avons dissocié l'appel en deux étapes afin d'améliorer la lisibilité du code dans l'ouvrage. En général, les développeurs Objective-C préfèrent les appels imbriqués.

Il existe trois différentes manières d'envoyer la notification, qui sont définies dans le fichier `NSNotification.h` sous la forme d'une énumération :

```
//NSNotificationQueue.h
enum {
    NSPostWhenIdle = 1,
    NSPostASAP = 2,
    NSPostNow = 3
};
```

- Comme son nom l'indique `NSPostNow` signifie que la notification doit être envoyée immédiatement. La notification ne sera donc *pas* asynchrone, mais bénéficiera toutefois de la fonction de *regroupement* (*coalescing*) des notifications de la file d'attente.Vous la découvrirez à la section suivante.
- Lorsque vous avez besoin d'accéder à une ressource rare ou chère, et que vous souhaitez toutefois que votre notification soit envoyée dès que

possible, vous utiliserez le style `NSPostASAP` (*As Soon As Possible*, dès que possible). Cette méthode est asynchrone et retourne immédiatement.

- Enfin, `NSPostWhenIdle` propose d'envoyer la notification lorsque la file d'attente est en attente de nouvelles notifications. Cela signifie que `NSPostWhenIdle` est une manière d'envoyer les notifications avec une priorité minimale. Cette méthode est asynchrone et retourne immédiatement.

Info

Chaque thread dispose d'une file d'attente par défaut, mais il est possible de créer vos propres files d'attente et les associer au centre de notifications et à vos thread. Ceci est une utilisation avancée qui ne sera pas traitée dans ici, et nous vous renvoyons donc à la documentation d'Apple.

Il est possible de retirer une notification d'une file d'attente grâce à la méthode `dequeueNotifications-Matching:coalesceMask:` de `NSNotificationQueue`. Son fonctionnement est similaire à celui de la méthode `removeObserver:name:object:` de `NSNotificationCenter`, mais contrairement à celle-ci, son utilisation reste très rare.