31

Appel d'un service

Les services peuvent être utilisés par n'importe quel composant de l'application capable d'attendre pendant un certain temps. Ceci inclut les activités, les fournisseurs de contenu et les autres services. Par contre, ceci ne comprend pas les récepteurs d'intention purs (les écouteurs qui ne font pas partie d'une activité) car ils sont automatiquement supprimés après le traitement d'une intention.

Pour utiliser un service local, vous devez le lancer, obtenir un accès à l'objet service, puis appeler ses méthodes. Vous pouvez ensuite arrêter le service lorsque vous n'en avez plus besoin ou, éventuellement, le laisser s'arrêter de lui-même. L'utilisation des services distants est un peu plus complexe et ne sera pas présentée dans ce livre.

Dans ce chapitre, nous étudierons la partie cliente de l'application `Service/WeatherPlus`. L'activité `WeatherPlus` ressemble énormément à l'application `Weather` originale – comme le montre la Figure 31.1, il s'agit simplement d'une page web qui affiche les prévisions météorologiques.

La différence est qu'ici ces prévisions changent lorsque le terminal "se déplace", en reflétant les modifications fournies par le service.

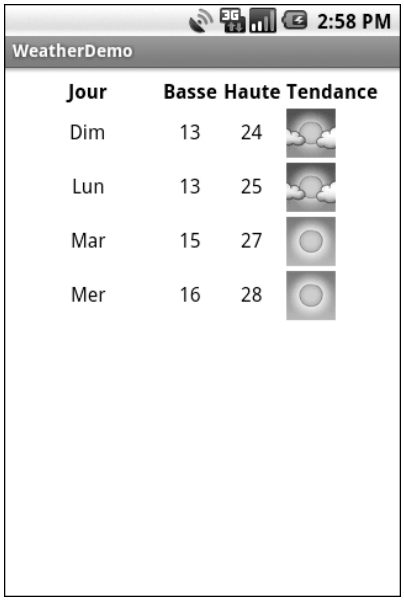

Figure 31.1
Le client du service `WeatherPlus`.

Transmission manuelle

Pour démarrer un service, il suffit d'appeler `startService()` en lui passant l'intention qui indique le service à lancer (là encore, le plus simple consiste à préciser la classe du service s'il s'agit de votre propre service).

Inversement, on l'arrête par un appel à la méthode `stopService()`, à laquelle on passe l'intention fournie à l'appel `startService()` correspondant.

Lorsque le service est lancé, on doit communiquer avec lui. Cette communication peut être exclusivement réalisée *via* les "extras" que l'on a fournis dans l'intention ou, s'il s'agit d'un service local qui offre un singleton, vous pouvez utiliser ce dernier.

Dans le cas de `WeatherPlus` et de `WeatherPlusService`, le client utilise le singleton du service à chaque fois qu'il veut obtenir de nouvelles prévisions :

```
private void updateForecast() {
  try {
    String page=WeatherPlusService
                  .singleton
                  .getForecastPage();
    browser.loadDataWithBaseURL(null, page, "text/html",
                                "UTF-8", null);
  }
  catch (final Throwable t) {
    goBlooey(t);
  }
}
```

Une partie épineuse de ce code consiste à s'assurer que le singleton est bien là quand on a besoin de lui. L'appel à `startService()` étant asynchrone, vous reprenez le contrôle tout de suite, or le service démarrera bientôt, mais pas immédiatement. Dans le cas de `WeatherPlus`, on peut s'en contenter car on n'essaie pas d'utiliser le singleton tant que le service ne nous a pas prévenus de la disponibilité d'une prévision, *via* une intention de diffusion. Cependant, dans d'autres situations, il faudra peut-être appeler la méthode `postDelayed()` d'un `Handler` afin de reporter d'une seconde ou deux l'utilisation du service, en espérant que le singleton sera devenu disponible entre-temps.

Capture de l'intention

Au chapitre précédent, nous avons vu comment le service diffusait une intention pour signaler à l'activité `WeatherPlus` qu'un mouvement du terminal avait provoqué une modification des prévisions. Nous pouvons maintenant étudier la façon dont l'activité reçoit et utilise cette intention.

Voici les implémentations d'`onResume()` et d'`onPause()` de `WeatherPlus` :

```
@Override
public void onResume() {
  super.onResume();
  registerReceiver(receiver,
        new IntentFilter(WeatherPlusService.BROADCAST_ACTION));
}
@Override
public void onPause() {
  super.onPause();
  unregisterReceiver(receiver);
}
```

Dans `onResume()`, nous enregistrons un `BroadcastReceiver` statique pour recevoir les intentions qui correspondent à l'action déclarée par le service. Dans `onPause()`, nous désactivons ce récepteur car nous ne recevrons plus ces intentions pendant que nous sommes en pause.

Le `BroadcastReceiver`, de son côté, met simplement les prévisions à jour :

```
private BroadcastReceiver receiver=new BroadcastReceiver() {
  public void onReceive(Context context, Intent intent) {
    updateForecast();
  }
};
```

32

Alerter les utilisateurs avec des notifications

Les messages qui surgissent, les icônes et les "bulles" qui leur sont associées, les icônes qui bondissent dans la barre d'état, etc. sont utilisés par les programmes pour attirer votre attention, et parfois pour de bonnes raisons.

Votre téléphone vous alerte aussi probablement pour d'autres motifs que la réception d'un appel : batterie faible, alarme programmée, notifications de rendez-vous, réception de SMS ou de courrier électronique, etc.

Il n'est donc pas étonnant qu'Android dispose d'un framework complet pour gérer ce genre d'événements, désignés sous le terme de notifications.

Types d'avertissements

Un service qui s'exécute en arrière-plan doit pouvoir attirer l'attention des utilisateurs lorsqu'un événement survient – la réception d'un courrier, par exemple. En outre, le service doit également diriger l'utilisateur vers une activité lui permettant d'agir en réponse à cet événement – lire le message reçu, par exemple. Pour ce type d'action,

Android fournit des icônes dans la barre d'état, des avertissements lumineux et d'autres indicateurs que l'on désigne globalement par le terme de notifications.

Votre téléphone actuel peut posséder ce type d'icône pour indiquer le niveau de charge de la batterie, la force du signal, l'activation de Bluetooth, etc. Avec Android, les applications peuvent ajouter leurs propres icônes dans la barre d'état et faire en sorte qu'elles n'apparaissent que lorsque cela est nécessaire (lorsqu'un message est arrivé, par exemple).

Vous pouvez lancer des notifications *via* le `NotificationManager`, qui est un service du système. Pour l'utiliser, vous devez obtenir l'objet service *via* un appel à la méthode `getSystemService` (`NOTIFICATION_SERVICE`) de votre activité. Le `NotificationManager` vous offre trois méthodes : une pour avertir (`notify()`) et deux pour arrêter d'avertir (`cancel()` et `cancelAll()`).

La méthode `notify()` prend en paramètre une `Notification`, qui est une structure décrivant la forme que doit prendre l'avertissement. Les sections qui suivent décrivent tous les champs publics mis à votre disposition (mais souvenez-vous que tous les terminaux ne les supportent pas nécessairement tous).

Notifications matérielles

Vous pouvez faire clignoter les LED du terminal en mettant `lights` à `true` et en précisant leur couleur (sous la forme d'une valeur `#ARGB` dans `ledARGB`). Vous pouvez également préciser le type de clignotement (en indiquant les durées d'allumage et d'extinction en millisecondes dans `ledOnMS` et `ledOffMS`).

Vous pouvez faire retentir un son en indiquant une `Uri` vers un contenu situé, par exemple, dans un `ContentManager` (`sound`). Considérez ce son comme une sonnerie pour votre application.

Enfin, vous pouvez faire vibrer le terminal en indiquant les alternances de la vibration (`vibrate`) en millisecondes dans un tableau de valeurs `long`. Cette vibration peut être le comportement par défaut ou vous pouvez laisser le choix à l'utilisateur, au cas où il aurait besoin d'un avertissement plus discret qu'une sonnerie.

Icônes

Alors que les lumières, les sons et les vibrations sont destinés à faire en sorte que l'utilisateur regarde son terminal, les icônes constituent l'étape suivante consistant à signaler ce qui est si important.

Pour configurer une icône pour une `Notification`, vous devez initialiser deux champs publics : `icon`, qui doit fournir l'identifiant d'une ressource `Drawable` représentant l'icône voulue, et `contentIntent`, qui indique le `PendingIntent` qui devra être déclenché lorsqu'on clique sur l'icône. Vous devez vous assurer que le `PendingIntent` sera capturé

– éventuellement par le code de votre application – pour prendre les mesures nécessaires afin que l'utilisateur puisse gérer l'événement qui a déclenché la notification.

Vous pouvez également fournir un texte qui apparaîtra lorsque l'icône est placée sur la barre d'état (`tickerText`).

Pour configurer ces trois composantes, l'approche la plus simple consiste à appeler la méthode `setLatestEventInfo()`, qui enveloppe leurs initialisations dans un seul appel.

Les avertissements en action

Examinons le projet `Notifications/Notify1`, notamment sa classe `NotifyDemo` :

```
public class NotifyDemo extends Activity {
  private static final int NOTIFY_ME_ID=1337;
  private Timer timer=new Timer();
  private int count=0;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Button btn=(Button)findViewById(R.id.notify);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        TimerTask task=new TimerTask() {
          public void run() {
            notifyMe();
          }
        };

        timer.schedule(task, 5000);
      }
    });

    btn=(Button)findViewById(R.id.cancel);

    btn.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        NotificationManager mgr=
          (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

        mgr.cancel(NOTIFY_ME_ID);
      }
```

```
    });
  }

  private void notifyMe() {
    final NotificationManager mgr=
      (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
    Notification note=new Notification(R.drawable.red_ball,
                                       "Message d'etat !",
                                       System.currentTimeMillis());
    PendingIntent i=PendingIntent.getActivity(this, 0,
                          new Intent(this, NotifyMessage.class),
                                          0);

    note.setLatestEventInfo(this, "Titre de la notification",
                          "Message de notification", i);
    note.number=++count;

    mgr.notify(NOTIFY_ME_ID, note);
  }
}
```

Cette activité fournit deux gros boutons, un pour lancer une notification après un délai de 5 secondes, l'autre pour annuler cette notification si elle est active. La Figure 32.1 montre l'aspect de son interface lorsqu'elle vient d'être lancée.

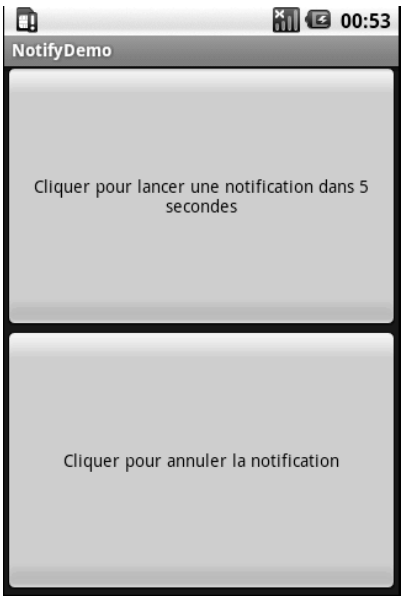

Figure 32.1
La vue principale de `NotifyDemo`.

La création de la notification dans `notifyMe()` se déroule en cinq étapes :

1. Obtenir l'accès à l'instance de `NotificationManager`.
2. Créer un objet `Notification` avec une icône (une boule rouge), un message qui clignote sur la barre d'état lorsque la notification est lancée et le temps associé à cet événement.
3. Créer un `PendingIntent` qui déclenchera l'affichage d'une autre activité (`NotifyMessage`).
4. Utiliser `setLatestEventInfo()` pour préciser que, lorsqu'on clique sur la notification, on affiche un titre et un message et que, lorsqu'on clique sur ce dernier, on lance le `PendingIntent`.
5. Demander au `NotificationManager` d'afficher la notification.

Par conséquent, si l'on clique sur le bouton du haut, la boule rouge apparaîtra dans la barre de menu après un délai de 5 secondes, accompagnée brièvement du message d'état, comme le montre la Figure 32.2. Après la disparition du message, un chiffre s'affichera sur la boule rouge (initialement 1) – qui pourrait par exemple indiquer le nombre de messages non lus.

En cliquant sur la boule rouge, un tiroir apparaîtra sous la barre d'état. L'ouverture de ce tiroir montrera les notifications en suspens, dont la nôtre, comme le montre la Figure 32.3.

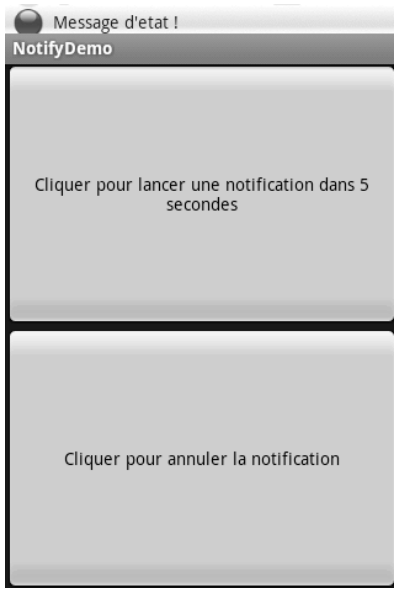

Figure 32.2
Notre notification apparaît dans la barre d'état, avec notre message d'état.

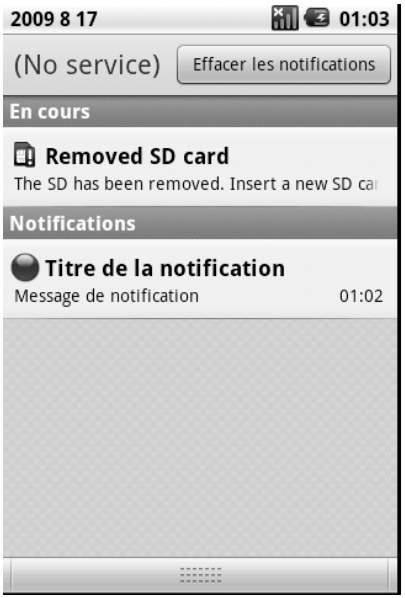

Figure 32.3
Le tiroir des notifications complètement ouvert, contenant notre notification.

En cliquant sur l'entrée de la notification dans la barre de notification, on déclenche une activité très simple se bornant à afficher un message – dans une vraie application, cette activité réaliserait un traitement tenant compte de l'événement qui est survenu (présenter à l'utilisateur les nouveaux messages de courrier, par exemple).

Si l'on clique sur le bouton d'annulation ou sur le bouton "Effacer les notifications" du tiroir, la balle rouge disparaît de la barre d'état.

Partie VI

Autres fonctionnalités d'Android