

Les services

Présentation

- Un service est un composant qui peut effectuer des opérations en arrière plan et ne fournit pas d'interface utilisateur
 - Par exemple: gérer les transactions réseau, lire de la musique
 - Déclaration

```
<manifest ... >
...
<application ... >
    <service android:name=".ExampleService" />
    ...
</application>
</manifest>
```

Modes d'activation d'un service

- Deux modes pour lancer un service :
 - Démarré par un autre composant (mode Started)
 - Quand un composant d'application (par exemple, une activité) démarre ce service en appelant **startService()**
 - Lié à d'autres composants (mode Bound)
 - Quand un composant d'application se lie à ce service en appelant **bindService()**

Modes d'activation d'un service

- Démarré (Started)
 - Une fois démarré, un service peut s'exécuter en arrière-plan indéfiniment, même si le composant qui a commencé est détruit
 - Un service démarré effectue une opération unique et ne retourne pas de résultat à l'appelant
 - Par exemple,
 - télécharger un fichier sur le réseau
 - Lorsque l'opération est terminée, le service doit s'arrêter

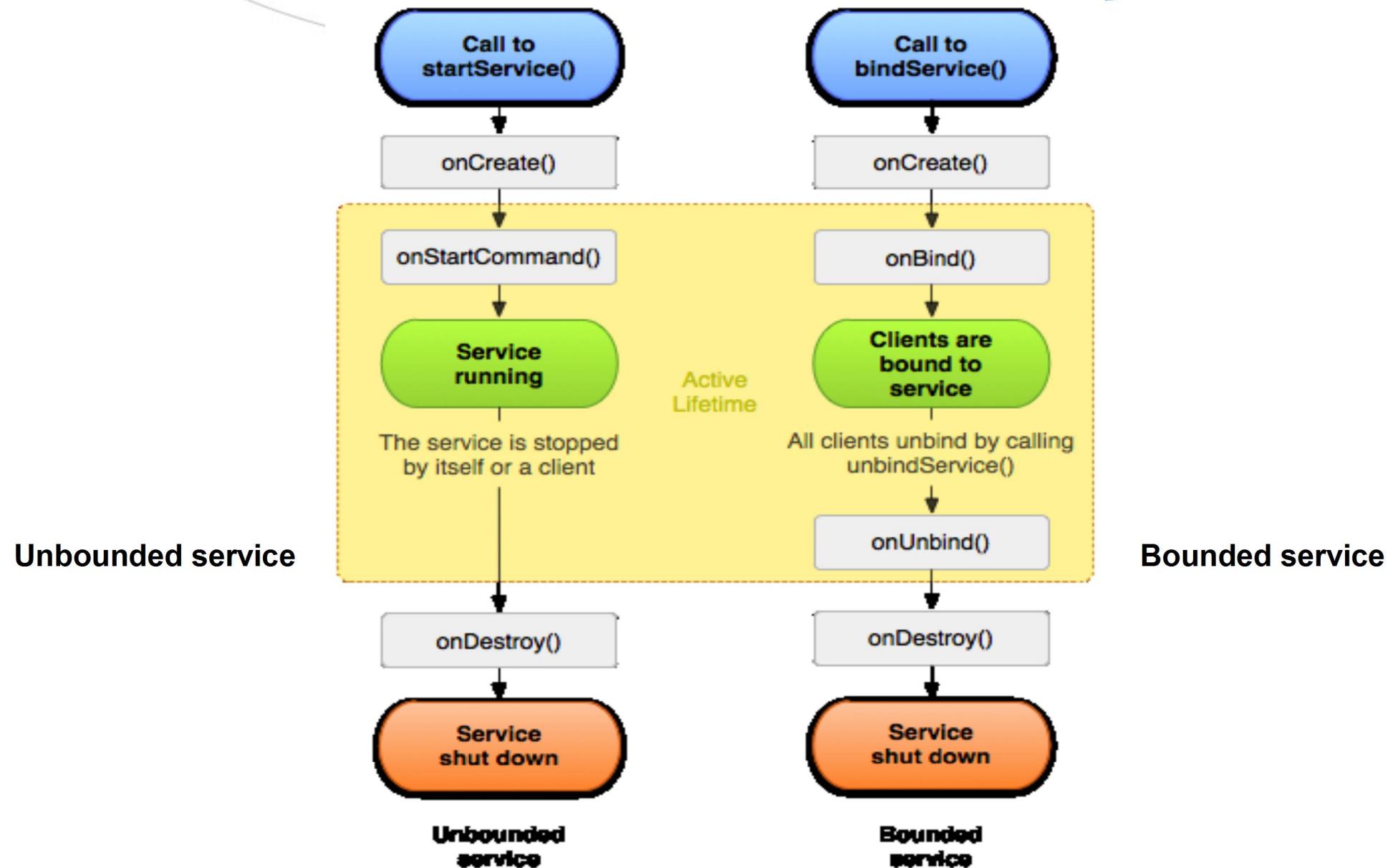
Lancer un service

- Démarrer un service en passant un Intent à startService ()
 - Android appelle la méthode **onStartCommand ()** du service et lui transmet l'intention
- ```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```
- Par exemple, une activité doit enregistrer des données dans une base de données en ligne. L'activité peut démarrer un service et lui transmettre les données à sauvegarder en lui passant une intention via **startService ()**
  - Un service peut définir des filtres qui permettent de capter les appels avec intentions (intents) implicites

# Modes d'activation d'un service

- Lié (Bound)
  - Un service lié propose une interface client-serveur qui permet aux composants d'interagir avec le service
    - Envoyer des demandes
    - Obtenir des résultats
    - Faire à travers des processus de communication inter-processus (IPC).
  - Un service lié ne fonctionne que tant qu'un ou plusieurs composants d'autres applications sont liés à celui-ci
    - Quand tous les liaisons sont détruites, le service est détruit

# Cycle de vie d'un service



# Création de service

- Deux classes peuvent être étendues pour créer un service démarré:
  - Service
    - Il s'agit de la classe de base pour tous les services
  - IntentService
    - Il s'agit d'une sous-classe de Service qui utilise un même thread pour traiter toutes les demandes, une à la fois

# Création de service

- **onStartCommand ()**
  - Le système appelle cette méthode lorsqu'un autre composant tel une activité, active un service en appelant **startService()**
  - Une fois que cette méthode lancée, le service est démarré et peut fonctionner en tâche de fond indéfiniment
    - C'est au développeur d'arrêter le service lorsque son travail est fait, en appelant **stopSelf ()** ou **StopService ()**

# Création de service

- onBind ()
  - Le système appelle cette méthode quand un autre composant veut se lier avec le service
    - Par exemple, pour effectuer RPC en appelant **bindService()**
  - Utilisation d'une interface par les clients pour communiquer avec le service, en renvoyant un IBinder.
  - Si un service est appelé par **bindService ()**, une fois dissocié de tous ses clients, le système le détruit

# Création de service

- Étendre la classe IntentService
  - Permet de traiter les demandes du service dans l'ordre de leur arrivée
  - Le IntentService effectue les opérations suivantes:
    - Crée un thread par défaut qui exécute toutes les intentions passées à **onStartCommand()**
    - Crée une file d'attente des demandes vers le service
    - Arrête le service après que toutes les demandes ont été traitées
    - Fournit une implémentation par défaut de **onBind ()** qui renvoie la valeur null.
    - Fournit une implémentation par défaut de onStartCommand ()
  - Le développeur doit implémenter **onHandleIntent()**

```
public class HelloIntentService extends IntentService {

 /**
 * A constructor is required, and must call the super IntentService(String)
 * constructor with a name for the worker thread.
 */
 public HelloIntentService() {
 super("HelloIntentService");
 }
 /**
 * The IntentService calls this method from the default worker thread with the intent that *started
 * the service. When this method returns, IntentService stops the service, as appropriate.
 */
 @Override
 protected void onHandleIntent(Intent intent) {
 // Normally we would do some work here, like download a file.
 // For our sample, we just sleep for 5 seconds.

 long endTime = System.currentTimeMillis() + 5*1000;
 while (System.currentTimeMillis() < endTime) {
 synchronized (this) {
 try {
 wait(endTime - System.currentTimeMillis());
 } catch (Exception e) {
 }
 }
 }
 }
}
```

# Création de service

- Étendre la classe de service
  - Permet au service d'effectuer un multi-threading des tâches
  - Possibilité de traiter plusieurs demandes en même temps
    - Créer un nouveau thread pour chaque requête et l'exécuter immédiatement, au lieu d'attendre la fin de la requête précédente
      - Chaque appel à **onStartCommand()** peut être traité à part

```
public class HelloService extends Service {
 private Looper mServiceLooper;
 private ServiceHandler mServiceHandler;

 // Handler that receives messages from the thread
 private final class ServiceHandler extends Handler {
 public ServiceHandler(Looper looper) {
 super(looper);
 }
 @Override
 public void handleMessage(Message msg) { //... }
 }

 @Override
 public void onCreate() { //... }

 @Override
 public int onStartCommand(Intent intent, int flags, int startId)
 { //... }

 @Override
 public IBinder onBind(Intent intent) { //... }

 @Override
 public void onDestroy() { //... }
}
```

# Création d'un service

```
@Override
 public void handleMessage(Message msg) {
 // Normally we would do some work here, like download a file.
 // For our sample, we just sleep for 5 seconds.

 long endTime = System.currentTimeMillis() + 5*1000;
 while (System.currentTimeMillis() < endTime) {
 synchronized (this) {
 try {
 wait(endTime - System.currentTimeMillis());
 } catch (Exception e) {
 }
 }
 }

 // Stop the service using the startId, so that we don't stop
 // the service in the middle of handling another job

 stopSelf(msg.arg1);
 }
}
```

# Création d'un service

```
@Override
 public void onCreate() {
 // Start up the thread running the service. Note that we create a
 // separate thread because the service normally runs in the
 // process's main thread, which we don't want to block.
 // We also make it background priority so CPU-intensive
 // work will not disrupt our UI.

 HandlerThread thread = new
 HandlerThread("ServiceStartArguments",
 Process.THREAD_PRIORITY_BACKGROUND);
 thread.start();

 // Get the HandlerThread's Looper and use it for our Handler

 mServiceLooper = thread.getLooper();
 mServiceHandler = new ServiceHandler(mServiceLooper);
 }
```

# Création d'un service

```
@Override

public int onStartCommand(Intent intent, int flags, int startId) {
 Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();

 // For each start request, send a message to start a job and
 // deliver the start ID so we know which request we're stopping
 // when we finish the job

 Message msg = mServiceHandler.obtainMessage();
 msg.arg1 = startId;
 mServiceHandler.sendMessage(msg);

 // If we get killed, after returning from here, restart
 return START_STICKY;
}
```

# Création d'un service

```
@Override
public IBinder onBind(Intent intent) {
 // We don't provide binding, so return null
 return null;
}

@Override
public void onDestroy() {
 Toast.makeText(this, "service done",
 Toast.LENGTH_SHORT).show();
}
```