Android(Java6)における
非同期モナドの実用

わかめモナ化 2012/11/17 @名大多元109

@keigoi

available at:
https://github.com/keigoi/async-android

(0) Androidアプリと非同期処理

- Androidアプリ＝Activity, Service 等のコンポーネントの集まり
- Activity = ユーザインタフェースをもつ、アプリの「顔」
- **Activityの応答性：**

  ただ1つの**UIスレッド**のみがUIを操作できる

  UIスレッドが占有されると**アプリが「固まる」**

- 時間のかかる処理→他のスレッドに任せる（非同期処理）

(1)AndroidのAsyncTask

- AsyncTaskクラス: 非同期処理のための便利クラス

```
private class DownloadFilesTask
    extends AsyncTask<URL, Integer, Long> {

        protected Long doInBackground(URL... urls) {
         (ダウンロード：バックグラウンドスレッドで実行される)

        }

        protected void onProgressUpdate(Integer... progress) {
         (進捗の更新：メインスレッドで実行される)

        }

        protected void onPostExecute(Long result) {
          (ダウンロード完了：メインスレッドで実行される)

        }
}
```

AsyncTaskの問題点

- **合成性**がない
  - 複数のAsyncTaskを「つなげる」方法の不在
  - 再利用性が低い
- 型引数が多い
  - AsyncTask<Param,Progress,Result>
  - 結果を表すResultがあれば多くの場合OK

(2)非同期処理=継続渡しスタイル

- 例：JavaScript

- getUrl(“http://...”, function(data) {
  text.value = data;
  });

通信の後にやりたいこと＝

継続を関数にして渡す

(3)継続渡しを合成しよう:

継続モナドをJavaで

- まずは継続の定義

```
interface Cont<A> {
  void apply(A a); /* 継続 */
}
```

- ついでに関数も

```
interface Lambda<A, B> {
  A f(B b);
}
```

継続モナドって何だ

- class ContMonad<A> { ... }　（次頁
- 型Aの**継続**を受け取る計算

継続モナド

```
abstract class ContMonad<A> {
   abstract void exec(Cont<A> cont);

   static <A> ContMonad<A> ret(final A a) {
      return new ContMonad<A>(){
         void exec(Cont<A> cont) {
            cont.apply(a);
         }
      };
   }

   final <B> ContMonad<B> bind(final Lambda<A, ContMonad<B>> f) {
      final ContMonad<A> self = this;

      return new ContMonad<B>() {

         void exec(final Cont<B> cont) {

            self.exec(new Cont<A>() {
               public void apply(A a) {
                  f.f(a).exec(cont);
               }
            });
      }
```

こんなクラスをつくった

- Background<A>
  Aを返す計算をバックグラウンド実行

- InUiThread<A>
  Aを返す計算をUIスレッドで実行

- Pure<A>
  ただちに継続にAを渡す

非同期タスクの再利用性が高まった

こんなこともできる

- 通信エラー時の再実行
- 継続がオブジェクトになっているので
  任意の継続モナドを何度でもやりなおせる

ほかの話題：

Java6でbindはつらい

- new Download().bind(new Lambda<Data,Async<Unit>>(){
  public Async<Unit> f(Data d) {
  ....
  }
  }).bind()....

そこで：インナークラスを使って bindを書けるようにした

```
new Download().bind(new Lambda<Data,Async<Unit>>(){
    public Async<Unit> f(Data d) {
        ....
    }
});
```

↓

```
new Download().new Bind<Unit>(){
    public Async<Unit> bind(Data d) {
        ....
    }
});
```

（カッコと型引数がちょっと不要に）

Bindをどこで実行するか

- new Download().new BindInUiThread<Unit>{
  Async<Unit> bind(Data data) {
  /* UIスレッドでここを実行 */
  }
  }

- new Download().new BindInBackground<Unit>{
  Async<Unit> bind(Data data) {
  /* バックグラウンドでここを実行 */
  }
  }

他の話：
なぜかeclipseのコンパイラのバグを見つけた

インナークラスのコンストラクタを外から無名クラスとして継承すると型エラー

- http://qiita.com/items/06ff1ca1ec7b01c276cc

まとめ

- Androidで非同期処理は重要
  - だがAsyncTaskには**合成性がない**
- 非同期処理＝継続渡し
- 継続渡しの**合成**＝継続**モナド**
- 非同期モナド＝実行するスレッドを指定できる継続モナド
- 仕事でバリバリ使ってます

available at:

https://github.com/keigoi/async-android

以上