

## 第3章 アクターとメッセージによるプログラミング

この章では、ある応用分野の例を見てゆきます。この分野は Scala がとてもよく似合っています。電子オークションサービスを実装する仕事を考えてみてください。Erlang 風のアクタープロセスモデルを使ってオークション参加者の実装をしましょう。アクターとはメッセージを受信するオブジェクトです。各アクターはメッセージ受信用の「メールボックス」を持ち、それはキューとして表現されます。メールボックス中のメッセージを順番に処理することも、特定のパターンにマッチするメッセージを検索することもできます。

取引される商品ごとに一つの競売人アクターがあり、そのアクターが、商品についての情報公開、クライアントからの申し込み受け付け、取引終了時の売り手および落札者への通知します。ここでは簡単な実装の概要を示します。

最初のステップとして、オークションでやり取りされるメッセージを定義します。クライアントからオークションサービスへのメッセージである AuctionMessage と、サービスからクライアントへの返答である AuctionReply の、2つの抽象基底クラス (abstract base class) があります。両基底クラスには複数のケース (case) があり、図 3.1 で定義されています。

```
---
import scala.actors.Actor

abstract class AuctionMessage
case class Offer(bid: Int, client: Actor) extends AuctionMessage
case class Inquire(client: Actor) extends AuctionMessage

abstract class AuctionReply
case class Status(asked: Int, expire: Date) extends AuctionReply
case object BestOffer extends AuctionReply
case class BeatenOffer(maxBid: Int) extends AuctionReply
case class AuctionConcluded(seller: Actor, client: Actor)
extends AuctionReply
case object AuctionFailed extends AuctionReply
case object AuctionOver extends AuctionReply
```

--- Listing 3.1: オークションサービスのメッセージクラス ---

各基底クラスごとに、クラス内の特定のメッセージ形式を定義する **ケースクラス** (case class) が複数あります。それらのメッセージが、小さな XML 文書に最終的にうまく対応できればよいのですが・・・ここでは自動化ツールが存在すると仮定して、XML 文書と、先のように定義された内部データ構造とを変換するとしましょう。

図 3.2 で示す Scala 実装、Auction クラスは、ある商品のオークションを調整する競売人アクターのためのものです。このクラスのオブジェクトは、次を指定して生成されます。

- オークション終了時に通知する必要がある売り手アクター
- 最低オークション価格
- オークションの終了予定時刻

アクターの振る舞いは act メソッドで定義されています。このメソッドでは、TIMEOUT メッセージによってオークション終了が通知されるまで (receiveWithin を用いて) メッセージを選択し、それに対応することを繰り返します。最終的に停止する前まで、定数 timeToShutdown で定められた期間はアクティブであり続け、さらなる申し出に対してはオークションを締切った旨を返答します。

このプログラムで使われている構文について、さらにいくつか解説します。

- クラス Actor の receiveWithin メソッドはパラメータとして、期間 (ミリ秒単位) とメールボックスのメッセージを処理する関数をとります。関数は一連のケースとして与えられ、各ケースはパターンと、そのパターンに対応したメッセージを処理するアクションを指定します。receiveWithin メソッドは、メールボックスからパターンにマッチする最初のメッセージを選び、対応するアクションを適用します。
- receiveWithin の最後のケースは TIMEOUT パターンで守られています。もし有効時間内にメッセージを受け取らなければ、receiveWithin メソッドの引数として渡された期間経過後に、このパターンが起動されます。TIMEOUT は特別なメッセージで、Actor の実装そのものによって起動されます。
- 応答メッセージを送信する構文として、destination ! SomeMessage (宛先 ! メッセージ) を用います。ここで ! は、アクターとメッセージを引数とする二項演算子のように使われています。これは Scala では、メソッド呼び出し destination.!(SomeMessage)、すなわち、メッセージをパラメータとした destination アクターの ! メソッド呼び出しと同じです。

```
---
class Auction(seller: Actor, minBid: Int, closing: Date) extends Actor {
  val timeToShutdown = 36000000 // msec
```

