第2章

日付と時間の取り扱い

プログラム開発に不可欠な処理として、日付や時間の取り扱いが挙げられます。たとえば、ゲーム、金融システム、ネットショップ、どれも何らかの形で日付や時間を用いていることは、容易に想像できるでしょう。

この章では、Java API のなかでも特に利用頻度が高い日付や時間に関するクラスの使い方を紹介します。

2.1 日付取り扱いの基本

2.1.1 Date 型のおさらい

Java で日付情報を扱う場合、確か Date 型を使うのよね。

そうだよ。念のため、おさらいしておこう。

Java における日付の取り扱いについては、『スッキリわかる Java 入門』でも学習しましたが、本書でも簡単におさらいしておきましょう。日付情報を扱うには、java.util.Date クラスを利用するのでしたね。

Date クラス

- Java で時刻情報を取り扱う場合に標準的に利用される。
- java.util パッケージに属している。
- 内部に**エポック**（基準時刻である 1970 年元日 0 時ちょうど）からの経過ミリ秒数を long 値で保持している。
- new にて引数なしでインスタンス化すると、現在の日時情報が格納される。
- new にて long 値を引数として渡しインスタンス化すると、指定時刻の情報が格納される。
- getTime() や setTime() を用いて、インスタンス内に保持する long 値を取得・設定できる。

Date クラスを用いたプログラムの例を次のリスト 2-1 に掲げます。

リスト 2-1

```
1  import java.util.Date;
2  public class Main {
3    public static void main(String[] args) {
4      Date now = new Date();
5      System.out.println(now);
6      System.out.println(now.getTime());
7      Date past = new Date(1316622225935L);
8      System.out.println(past);
9    }
10 }
```

import しておくと便利

現在の日時を取得

実行結果

```
Fri Aug 12 16:05:55 GMT+09:00 2011
1313132755277
Thu Sep 22 01:23:45 GMT+09:00 2011
```

（※実行の日時により以上の日付と数値は変わります）

ただし、Date クラスには、人間が取り扱いづらいという課題がありました。そのため、「年・月・日・時・分・秒」などのそれぞれを画面から指定する場合には java.util.Calendar クラスを利用したり、「西暦 2014 年 8 月 10 日」のような人が読みやすい文字列に変形するために java.text.SimpleDateFormat クラスを利用したりする必要がありましたね。

Calendar クラスおよび SimpleDateFormat クラス

- 「年・月・日・時・分・秒」の各情報を取り扱うには Calendar クラスを用いる。
- Date クラスが保持する日時情報を読みやすく変換するには SimpleDateFormat クラスを用いる。

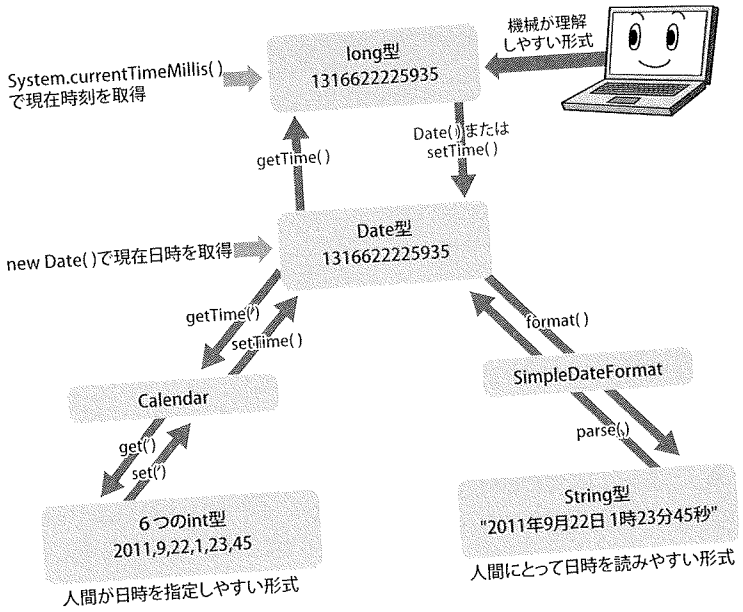

図 2-1　Date 型と関連する各種クラスの関係性

API の改訂と普及の速度

次節からは Date クラスや Calendar クラスに関する問題点と、その問題点を克服するために Java8 で加わった新しい日付 API について紹介していきます。

しかし一般的に、文字列や日時の取り扱いのようなプリミティブな API（さまざまなプログラムのあらゆる場所で利用される可能性がある原始的な API）について、新しいものが十分に普及し、旧来のものを置き換えるまでには長い時間を要します。

これからもしばらくは、Date クラスや Calendar クラスが広く利用され続けるでしょう。

よって、特に入門者の方は、日付の取り扱いに関する学習をここでひとまず終え、第 2 章の残りの部分を読み飛ばし、第 3 章に進んでいただいても構いません。2.2 節と 2.3 節は、将来業務などの必要に応じて読んでください。

2.2 従来型 API が抱える課題

2.2.1 Date や Calendar の問題点

あぁ、そっかぁ！ 3 月を指定するときは 3 じゃなくて 2 だった！
あぁ、5 時間も悩み続けたのに…。

前節で振り返った Date クラスや Calendar クラスは、Java が誕生したばかりの頃から利用可能だった古参の API です。Java で日時情報を扱う場合の標準的な手法として長く利用されてきた一方、設計上の問題も多く抱えており、「使いにくい」「不具合につながるような使い方をしてしまいやすい」等の課題も指摘されてきました。

Date や Calendar が抱えている代表的な問題

①使い方がまぎらわしい API が存在する。
②並列処理で用いるとインスタンスの内容が壊れることがある。

冒頭の湊くんのミスは、①の問題に起因します。やはり、Calendar クラスのコンストラクタの引数に、私たちの直感と反した値を指定しなければならないのは、やや不親切な仕様といえるでしょう（Calendar を用いて「月」の情報を取得・設定する場合、1 ～ 12 ではなく 0 ～ 11 で指定することになっています）。

また、本書の第 19 章で紹介するスレッドは、複数の処理を同時に実行する非常に強力な機能である一方、Date や Calendar のようなクラスを併用すると、まれに変数の中身が異常な値に書き換わってしまうことがあります。

日付関連の処理は、さまざまな種類のプログラムの至るところで記述されるものですから、無意識にスレッドと組み合わせて使ってしまうリスクが比較的高いことが広く懸念されてきました。

Date や Calendar は一見簡単そうに見えるけど、本当に安全に使うことはとても難しいんだ。

2.2.2 Date や Calendar の限界

Java 開発者の間で Date クラスや Calendar クラスの評判があまり良くない背景には、前項で紹介した問題以外にもいくつかの理由があります。

これらのクラスは、日付や時刻を正確かつ便利に扱うために必要な機能を十分に備えておらず、いざ本格的に日時情報を取り扱おうとすると開発者に大きな負担を強いることがままあるのです。

Date や Calendar の機能的な限界

- 最小でも「ミリ秒」単位でしか時間を扱えない。
- 私たちが日常利用する「曖昧な日時」を表せない。
- 私たちが日常利用する「時間の幅」を表せない。

ミリ秒単位で扱えれば十分じゃないですか。

人間にとってはね。でも、「ナノ秒単位」で動くコンピュータにとって、1ミリ秒は長すぎることもあるんだ。

こうした限界の詳細やその克服方法については、次節以降で説明していきましょう。

2.3 Time API

2.3.1 Java8 で加わった新しい API

前節で挙げた課題を克服するために、Java の最新版 (Java8) から、新しい日時関連のクラス群が java.time パッケージとして加わりました（表 2-1）。これら新しい API のクラスたちは、従来用いられてきた Date や Calendar と比較して直感的にわかりやすい API 構造となっているほか、先述のスレッドと併用してもインスタンスの中身が決して壊れないような設計上の工夫がなされています。

表 2-1　Java8 で java.time パッケージに加わった代表的なクラス

クラス名	機能と役割
Instant	世界における、ある「瞬間」の時刻を、ナノ秒単位で厳密に指し示し、保持する
ZonedDateTime	
LocalDateTime	日常的に使われる「曖昧な日時」を保持する
Duration	2 つの異なる時刻や日付の期間を保持する
Period	

java.time パッケージは、表 2-1 に挙げた 5 つ以外にもたくさんのクラスとメソッドを含んでいますが、以降では、この 5 つのクラスを中心に基本的な概念や代表的なクラスの利用法について紹介していきます。

通常の用途なら、この 5 つをしっかりマスターしておけば大丈夫。詳細な仕様は API リファレンスで都度調べてみてほしい。

Time API は Java8 以降でしか使えない

以降に掲載するコードは Java8 以降の環境でしかコンパイル・実行できません。

2.3.2 より正確な「瞬間」を表すクラス

Instant クラスは、新しい日時 API のなかで最も基礎となるクラスです。Instant は英語で「瞬間」を意味しますが、その名のとおり、エポックからの経過時間をナノ秒数で格納することで、この世界における「ある瞬間」を指し示すことができます。旧来の API における Date とほぼ同じ役割ですが、ナノ秒単位で正確に瞬間を表せるところがポイントです。

ZonedDateTime クラスも、Instant 同様、ある瞬間を格納することができるクラスです。ただし、このクラスはエポックからの経過時間ではなく、たとえば「東京における　西暦 2014 年 8 月 10 日 7 時 11 分 9 秒 392881 ナノ秒」という形式でその瞬間を格納、管理します。Calendar クラスの後継のようなクラスです。

なるほど。「ある瞬間」を指し示すために、2 つの方法があるのね。

でも、「東京における」っていう場所情報が、なんで必要なのかな？

「年・月・日・時・分・秒・ナノ秒」の情報だけでは、この世界のある瞬間を正確に指し示すことはできません。なぜなら、同じ「2014 年 1 月 1 日 12 時 0 分 0 秒 0 ナノ秒」であっても、東京に住む人とロンドンに住む人では違う瞬間を指すからです（図 2-2）。

和暦を扱う API

Java8 から加わったクラスのなかに、java.time.chrono.JapaneseDate クラスがあります。このクラスを用いれば、「平成 26 年 1 月 23 日」のような和暦の情報を取り扱うことができます。

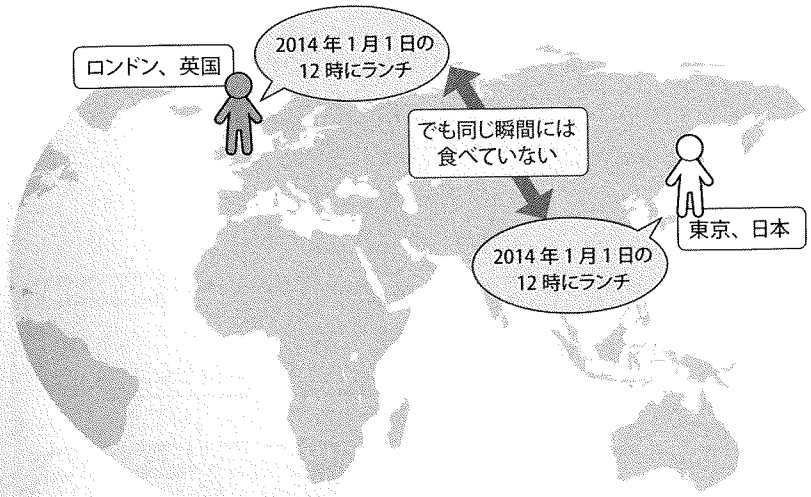

図 2-2　タイムゾーンがないと、正確な瞬間を指し示せない

そこで、ZonedDateTime は、どの都市の人を基準にするかを明確にするために、**タイムゾーン** (time zone) と呼ばれる情報を含んでいるのです。タイムゾーンは、「Asia/Tokyo」や「Europe/London」などの文字列で表現される情報で、その一覧は IANA という国際標準化団体で管理されています。Java の世界では、タイムゾーン情報は ZoneId クラスのインスタンスとして扱いますが、「new ZoneId()」としてインスタンスを生成することはできないため、静的メソッド of() を利用します。Instant と ZonedDateTime の利用例を示したリスト 2-2 で、ZoneId クラスの利用法も確認しましょう。

リスト 2-2　Instant および ZonedDateTime の利用例

Main.java

```
1  import java.time.*;
2
3  public class Main {
4    public static void main(String[] args) {
5      // Instantの生成
```

```
6       Instant i1 = Instant.now();
7
8       // Instantとlong値との相互変換
9       Instant i2 = Instant.ofEpochMilli(31920291332L);
10      long l = i2.toEpochMilli();
11
12      // ZonedDateTimeの生成
13      ZonedDateTime z1 = ZonedDateTime.now();
14      ZonedDateTime z2 = ZonedDateTime
15          .of(2014, 1, 2, 3, 4, 5, 6, ZoneId.of("Asia/Tokyo"));
16
17      // InstantとZonedDateTimeの相互変換
18      Instant i3 = z2.toInstant();
19      ZonedDateTime z3 = i3.atZone(ZoneId.of("Europe/London"));
20
21      // ZonedDateTimeの利用
22      System.out.println("東京:" + z2.getYear() + z2.getMonth()
23          + z2.getDayOfMonth());
24      System.out.println("ロンドン:" + z3.getYear() + z3.getMonth()
25          + z3.getDayOfMonth());
26      if (z2.isEqual(z3)) {
27          System.out.println("これらは同じ瞬間を指しています");
28      }
29  }
30  }
```

現在日時を取得

現在日時を取得

「東京時間 2014 年 1 月 2 日 3 時 4 分 5 秒 6 ナノ秒」を指定して取得

同じ瞬間の判定には、equals() ではなく isEqual() を使う

2.3.3　曖昧な日時を表すクラス

そして Java8 では、やっと「曖昧な日時」を表せるようになったんだ！

えっ…曖昧な情報なんて、使えて何がうれしいんですか？

ここでふたたび、図 2-2 を見てみましょう。タイムゾーン情報が付加されていない「12 時」という表現は、正確に世界のある瞬間を指し示すことはできません。だからこそ、Calendar や ZonedDateTime は正確に瞬間を指し示すために内部でタイムゾーンをきちんと管理するしくみになっています。

一方、私たちの日常生活を考えてみましょう。待ち合わせの時間、生年月日などを私たちが現実世界で扱う際、タイムゾーンまで考慮することはほとんどありません。**私たちは、通常、タイムゾーンの情報が欠落した「曖昧な日時」を使っている**のです。にもかかわらず、これまで Java には、私たちが日常的に使う「曖昧な日時」を格納することができるクラスが存在しませんでした。

曖昧な日時の必要性

私たちは、普段、タイムゾーン情報が欠落した「曖昧な日時」を使って生活している。しかし、そのような日時情報を正しく Java の世界で再現する手段がなかった。

そんなの、Calendar や ZonedDateTime クラスで、タイムゾーン部分にゼロを格納しておけばいいじゃないですか？

そうかな。オブジェクト指向の本質を思い出してごらん。

Java8 の登場以前は、仕方なく湊くんのように「曖昧な日時情報を厳密な Calendar クラスにムリヤリ格納する」ことも多く行われてきました。しかし、タイムゾーン部分がゼロであることは、本来、「世界標準時から時差ゼロ（ロンドン等のタイムゾーン）」を意味するものであって、「タイムゾーンを考慮しない」という意味ではありません。このように、現実世界を正しく再現できないことは、

数多くの混乱や不具合の原因となってきました。

そこでJava8から新たに加わったのが、LocalDateTimeクラスです。

このクラスはZonedDateTimeとよく似ていますが、タイムゾーン情報だけは格納しません。タイムゾーン情報がないため、LocalDateTimeインスタンス単体では「どの瞬間を指し示しているのか」を確定できなくなってしまいます。しかし、私たちが日常的に使う日時情報を格納するためには最適なクラスです。

ZonedDateTimeとLocalDateTime

私たちが普通のプログラミングで広く利用すべきはLocalDateTimeである。ZonedDateTimeを使う機会はあまり多くない。

LocalDateTimeの利用例を、リスト2-3に示しましょう。

リスト2-3　LocalDateTimeの利用例

Main.java

```
1  import java.time.*;
2
3  public class Main {
4    public static void main(String[] args) {
5      // LocalDateTimeの生成方法
6      LocalDateTime l1 = LocalDateTime.now();
7      LocalDateTime l2 = LocalDateTime.of(2014, 1, 1, 9, 5, 0, 0);
8
9      // LocalDateTimeとZonedDateTimeの相互変換
10     ZonedDateTime z1 = l2.atZone(ZoneId.of("Europe/London"));
11     LocalDateTime l3 = z1.toLocalDateTime();
12   }
13 }
```

現在日時を取得

2014年1月1日9時5分を指定して取得

2.3.4　その他の日時を表すクラス

ZonedDateTimeからタイムゾーン情報を削ったものがLocalDateTimeであることを学びました。Java8では、このLocalDateTimeからさらにいくつかの情報を削ったクラスも提供されています（表2-2）。

表2-2　時刻を表すその他のクラス　（○：格納する　×：格納しない）

クラス	年	月	日	時間	ゾーン	用途や例
ZonedDateTime	○	○	○	○	○	厳密な日時情報
LocalDateTime	○	○	○	○	×	日常使う日時情報
LocalDate	○	○	○	×	×	誕生日など
LocalTime	×	×	×	○	×	アラーム時刻など
Year	○	×	×	×	×	著作発表年など
YearMonth	○	○	×	×	×	カード有効期限など
Month	×	○	×	×	×	決算月など
MonthDay	×	○	○	×	×	日本の祝日など

確かに「こどもの日は5月5日だね」って言うとき、私たちは年の情報を意識してませんものね。

私たちが日常使うさまざまな種類の「曖昧な時間表現」を、そのままJavaの世界でも扱えるようになるんだ。

ところで、表2-2に挙げたクラスは、曖昧さに違いはあるものの、すべて**時間軸上のある時点を指し示す情報を格納するクラス**という点で共通しています。実際、これらのクラスはすべてjava.time.Temporalインタフェースを実装しており、同様の動作をするメソッドには共通のメソッド名を使うように設計されています（表2-3）。

表2-3　特定日時を指し示すクラスで共通に利用されるメソッド名

メソッド名	静的	解説
now()	○	現在日時からインスタンスを生成する
of() / of～()	○	他の種類から変換してインスタンスを生成する
parse()	○	"2010/4/12" 等の文字列からインスタンスを生成する。文字列書式はDateTimeFormatterで指定する