l'API Date and Time de Java 8

Temps de lecture : 5 minutes

Présentation de l'API Date and Time

Le support de la gestion des `Date` et `Calendar` présenté dans les chapitres précédents comporte les désavantages suivants :

- Leur utilisation n'est pas simple
- Ils ne sont pas immuables donc pas thread-safe
- Les objets encapsulent obligatoirement la date et l'heure
- Le mois est numéroté de 0 à 11 ce qui n'est pas intuitif
- SimpleDateFormat n'est pas thread-safe non plus et

Pour pallier à ces problèmes, il est préférable de se baser sur les classes de `java.time.*` disponibles depuis `Java 1.8` : *l'API Date and Time*. Celle-ci offre à l'inverse les avantages suivants :

- Utilisation simple et intuitive
- Supporte plus de fonctionnalités : date, heure, intervalle de temps, calendrier, fuseau et décalage horaire, ..
- Immutabilité donc thread-safe : la plupart des principaux objets sont immuables

Note : il faut considérer que du code existant dans certains projets se basent encore sur les classes `Date` et `Calendar`, il est donc important de connaître leur fonctionnement également.

Les classes de `java.time.*`

Ce cours n'est pas exhaustif sur *l'API Date and Time* mais il vise à présenter les classes les plus utiles.

La classe `Instant`

Elle permet l'encapsulation d'un `timestamp` :

```
var epoch = Instant.EPOCH; //Représente 1970-01-01T00:00:0
0Z
var instant = Instant.now();
```

De nombreuses méthodes sont disponibles pour manipuler ces timestamps comme des méthodes de comparaisons (`isBefore` , `isAfter` , ...).

La classe `LocalDate`

La classe `LocalDate` encapsule de manière immuable une date sous la forme d'une année, d'un mois et d'un jour dans le calendrier ISO sans fuseau horaire :

```
LocalDate dateNow = LocalDate.now();
LocalDate date = LocalDate.of(2014, Month.DECEMBER, 25);
```

Comme elle ne possède pas de fuseau horaire, elle ne représente pas un point dans le temps.

La classe `LocalTime`

La classe `LocalTime` encapsule de manière immuable une heure (heure, minute, seconde, nanoseconde) sans fuseau horaire :

```
LocalTime timeNow = LocalTime.now();
LocalTime time = LocalTime.of(12, 32, 22, 24);
```

La classe `LocalDateTime`

La classe `LocalDateTime` encapsule de manière immuable une date (année, mois, jour) et une heure (heure, minute, seconde, nanoseconde) sans fuseau horaire :

```
LocalDateTime dateTimeNow = LocalDateTime.now();
LocalDateTime dateTime = LocalDateTime.of(2014, Month.DECE
MBER, 25, 12, 32, 22, 23);
```

Les dates avec fuseaux et décalages horaires

L'API Date and Time propose deux classes pour encapsuler un fuseau horaire ou un décalage horaire :

- `ZoneId` : encapsule un fuseau horaire qui possède un décalage horaire et des règles pour déterminer l'heure utilisée dans ce fuseau (comme les heures été / hiver)
- `ZoneOffset` : encapsule un décalage horaire

Ces deux classes sont elles-même encapsulées dans les classes :

- `OffsetTime` : encapsule une heure avec un décalage horaire
- `ZonedDateTime` : encapsule une date-heure avec un fuseau horaire
- `OffsetDateTime` : encapsule une date-heure avec un décalage horaire

Voici des exemples d'utilisation :

```
var offsetTime = OffsetTime.of(12, 05, 33, 22, ZoneOffset.
ofHours(-2));

var zonedDateTime = ZonedDateTime.of(
        LocalDate.of(2022, 01, 12),
        LocalTime.of(12, 05, 33, 22),
        ZoneId.of("Europe/Paris")
);

var offsetDateTime = OffsetDateTime.of(
        LocalDateTime.of(2022, 01, 12, 12, 05, 22, 33),
        ZoneOffset.ofHours(4)
);
```

Note : Il est possible d'afficher les différentes zones standard dans le `JDK` avec le code : `System.out.println(ZoneId.getAvailableZoneIds());`

À l'affichage des ces dates / heures, l'information du décalage sera automatiquement pris en compte en fonction du décalage horaire ou du fuseau horaire. La conversion en `timestamp` au format `UTC` sera également géré automatiquement pour nous. Grâce à ces mécanismes, nous sommes sûr de ne pas avoir de dates ambigues.

Les calendriers

Par défaut, *l'API Date and Time* utilise le calendrier ISO-8601 qui repose sur le calendrier Grégorien. Cependant, le package `java.time.chrono` propose ces

autres calendriers standards : Hijrah, Japanese, Minguo, ThaiBuddhist.
Un calendrier permet de représenter de manière humaine un point dans le temps.

Le `JDK` fournit donc en standard plusieurs implémentations de calendriers :

Calendrier	Chronology	ChronoLocalDate	Era
ISO	IsoChronology	LocalDate	IsoEra
Hijrah	HijrahChronology	HijrahDate	HijrahEra
Japanese	JapaneseChronology	JapaneseDate	JapaneseEra
Minguo	MinguoChronology	MinguoDate	MinguoEra
ThaiBuddhist	ThaiBuddhistChronology	ThaiBuddhistDate	ThaiBuddhistEra

La première ligne représente la ligne des interfaces à implémenter pour avoir un calendrier et des dates respectant ces calendriers. Les autres lignes représentent les différentes implémentations présentes dans le `JDK` en standard.

Note : L'interface `Era` représente les ères (par exemple, les ères avant et après Jésus Christ chez nous). Les calendriers encapsulent donc la classe `Era` car celle-ci représente une information nécessaire à la bonne définition d'une date.

On remarque que `LocalDate`, que nous avons déjà utilisé, s'appuie donc par défaut sur le calendrier ISO. Quand aux autres implémentations, il est très facile de les obtenir en se basant sur `LocalDate` comme dans l'exemple suivant :

```
LocalDate date = LocalDate.of(2014, Month.DECEMBER, 25);

JapaneseDate jdate = JapaneseDate.from(date);
HijrahDate hdate = HijrahDate.from(date);
MinguoDate mdate = MinguoDate.from(date);
ThaiBuddhistDate tdate = ThaiBuddhistDate.from(date);
```

Aller plus loin

Pour ne pas rendre ce cours trop difficile à comprendre, nous n'allons pas plus dans les détails de *l'API Date and Time*. Pour autant, elle est très complète et propose beaucoup de fonctionnalités. Si vous avez besoin d'aller plus loin, n'hésitez pas à vous référer aux documentations suivantes :

`JSR (Java Specification Requests) 310`

Il est possible de visualiser les spécifications de *l'API Date and Time* grâce à la `JSR 310`.

Note : Les `JSR` sont un moyen pour la communauté `Java` de proposer de nouvelles évolutions dans le langage Java.

Documentation *Api Date and Time*

L'ensemble de l'API est documentée ici.