2015-07-08

Java8のOptionalの使い方について

Java

Java SE 8 から新たに追加された Optional クラスについて使い方をまとめました。

Optionalとは

Java SE 8 で新たに `java.util.Optional` クラスが導入されました。Optional クラスは値を持たない場合がある（nullである）ことを表すコンテナオブジェクトで、nullをより安全に扱うためのメソッドを提供しています。

今までは戻り値が null の可能性がある場合は NullPointerException を防ぐために if文で null チェックをしていました。

```
Person person = getPerson();
if (person != null) {
    person.getName();
}
```

Optionalを使用すると値が存在する（nullでない）場合だけ処理することを一文で表現できます。

```
Optional<Person> personOpt = Optional.ofNullable(getPerson());
personOpt.ifPresent(person -> person.getName());
```

Optional (Java Platform SE 8)

Optionalの生成

Optional のコンストラクタはprivateコンストラクタのみであるため、インスタンスを生成する際は static メソッドを使用します。

staticメソッド	戻り値
Optional.empty()	値が null の Optionalインスタンス
Optional.of(T value)	値が value の Optionalインスタンス
Optional.ofNullable(T value)	値が null の場合は `empty` 、nullでない場合は `of` メソッドの結果

Optional を使用するパターンというのは、値を持たない場合がある事を考慮する必要がある時になると思います。そのため `Optional.ofNullable` で全て賄う事も可能ではありますが、値が入る事が確実な場合には `Optional.of` を使用し、null を返したい処理の場合は `Optional.empty` を使用した方が明示的で可読性も上がります。 `Optional.of` で null を与えると NullPointerException が発生しますので、 `Optional.ofNullable` のみだと本当の異常データが見逃されてしまいますよね。

Optionalから値を取得する

ここからはインスタンスメソッドになります。Optionalから値を取得するメソッドは以下の4つです。基本的には保持している値を返すメソッドになりますが、値が無い場合の処理としてパターンが分けられています。

メソッド	値が無い場合の処理
get()	NoSuchElementException をスロー
orElse(T other)	引数で指定した値を返す
orElseGet(Supplier<? extends T> other)	Supplier の get メソッドを実行
orElseThrow(Supplier<? extends X> exceptionSupplier)	Supplier の get メソッドを実行

`get` メソッドは値が存在するのが確実な時に使用します。値が null の時は NoSuchElementException をスローします。そのため、Optional を使用する場合にはあまり出番の少ないメソッドです。

`orElse` メソッドは値が null の場合に、引数で指定したインスタンスを返してくれます。値を持ってない時のデフォルト値を指定できるということですね。また、当然 String の Optional に Integer のデフォルト値を設定するという事などはできません。

`orElseGet` メソッドは基本的には `orElse` メソッドの場合と同じですが、関数型インターフェースの Supplier を引数にとります。Supplier は引数なしで値を返す処理でしたね。使い所としては Supplier の getメソッドが呼ばれた時だけ渡したラムダ式が呼ばれるので、重たい処理を行う場合に有効です。遅延評価といいます。

`orElseThrow` メソッドも `ofElseGet` と同様に Supplier を引数にとりますが、こちらは値が null の場合に投げる例外を指定します。

Optionalの値を判定する

`isPresent` メソッドは値が存在すれば true、値が存在しない場合は false を返すだけのメソッドです。従来型の `if (result != null)` という判定と変わりませんね。 `get` メソッドと同じようにあまり使う機会がなく、使ったら負けとまで言われてるようですが、私はそこそこ使う事がありそうな気もします。

Optionalの値を処理する

`ifPresent` メソッドは関数型インターフェースの Consumer を引数にとり、値が null でなければ `Consumer#accept` を実行します。Consumer は値を返さずに副作用を起こすための処理です。

`isPresent` と `get` メソッドを使用すると従来型の null チェックと変わらない書き方になってしまいます。

```
if (personOpt.isPresent()) {
    personOpt.get().getName();
}
```

このような場合は `ifPresent` を使用した方がいいでしょう。

```
personOpt.ifPresent(person -> person.getName());
```

Optionalを処理してOptionalで返すメソッド

ここであげるメソッドは Stream でも同様のメソッドがありますので、既にご存知の場合は馴染みやすいかと思います。戻り値が Optional のメソッドは `map` `flatMap` `filter` の3種類です。

☑ filter

`fileter` メソッドは値が存在し、それが指定された条件に一致する場合はその値を記述するOptionalを返し、そうでない場合は空のOptionalを返します。判定を行うための関数型インターフェースである Predicate を引数にとります。基本的には Stream#filter と同じですが、根本的に違う部分もあるので気をつけましょう。処理は以下の通りです。

- Optionalに保持する値が null の場合 -> `Optional.empty()` を返す。
- Predicate#testメソッドの結果が false の場合 -> `Optional.empty()` を返す。
- Predicate#testメソッドの結果が true の場合 -> そのままOptionalを返す。

Stream の filter と違う部分とは結果が false の時も空の Optional を返すという点です。Stream#filterの場合は条件に一致するものだけで構成される Stream を返すので気を付けましょう。

☑ map

`map` メソッドは値が存在する場合は、指定された関数型インターフェースの Function をその値に適用し、結果が null でなければ結果を保持した Optional を返します。Function は値を変換するための関数型インターフェースです。基本的には Stream の map と同じような感じですが、内部では以下のような処理をしています。

- Optionalに保持する値が null の場合 -> `Optional.empty()` を返す。

- Function#applyメソッドの結果が null の場合 -> `Optional.empty()` を返す。
- Function#applyメソッドの結果が null 以外の場合 -> Optionalに値を保持して返す。

☑ flatMap

`flatMap` メソッドは値が存在する場合は、指定されたOptional生成マッピング関数をその値に適用し、その結果を返します。そうでない場合は空のOptionalを返します。Optional を返す Function を引数にとりますが Optional がネストされていくことはありません。同じ事を map でやろうとするとネストしてしまいます。

- Optionalに保持する値が null の場合 -> `Optional.empty()` を返す。
- Function#applyメソッドの結果が null の場合 -> NullPointerExceptionをスローする。
- Function#applyメソッドの結果が null 以外の場合 -> 結果の Optional をそのまま帰す。