# JSON Binding API 入門, 第 1 回: JSON Binding API の概要

## JSON-B のデフォルトの機能、カスタム・アノテーション、ランタイム構成、その他を導入する

Alex Theedom

2018年 8月 23日

長い間待った末に、ついに Java EE 8 が、コアとなる Java エンタープライズ・プラットフォームに強力な JSON バインディング機能を導入しました。この記事では、この新たな Java API for JSON Binding で用意されているデフォルトの機能、カスタム・アノテーション、ランタイム構成を紹介します。

**この連載について**: Java EE ではこれまで長い間 XML をサポートしてきましたが、JSON データの組み込みサポートは著しく欠けていました。Java EE 8 はこの事態を変えるべく、コアとなる Java エンタープライズ・プラットフォームに強力な JSON バインディング機能を導入しています。JSON-B を採用して、Java エンタープライズ・アプリケーション内で JSON ドキュメントを処理するための JSON Processing API やその他のテクノロジーと JSON-B がどのように結合するのかを学んでください。

Java EE 8 で呼び物となっている Java API for JSON Binding (JSON-B) 1.0 (JSR 367) は、JSON (JavaScript Object Notation) に対する Java のサポートを強化する機能です。

JSON は、接続されたシステム間でも、接続されていないシステム間でも情報を転送する際の標準となっています。クラウドや Web サービスのアプリケーションでよく使われている JSON は、マイクロサービス・ベースのシステム内で RESTful Web サービスによってデータを送受信する際のデータ交換フォーマットとして登場しました。

JSON-B は、JSON の管理を目的とした Jave EE の一連の API と機能を補完するものとして、Java オブジェクトから JSON ドキュメントへの変換、またはその逆の変換を行うための単純かつ統一された標準です。JSON-B の使いやすさは、形式関連の規則が最小限に抑えられていることと、デフォルトの構成が賢明かつ直感的であることから実現されています。

JSON Binding は、JSON Processing (JSON-P) とも密接に結合します。JSON-P も同じくエンタープライズ・システム内で JSON を管理するために使用する Java EE の API であり、下位レベルの API として、JSON を処理および変換する際の 2 つのモデル (ストリーミングおよびオブジェクト) を規定しています。これら 2 つの API がまとまって、Java エンタープライズ・システム内で JSON を使用するための堅牢なフレームワークを実現します。

## JSON-B を使用したデフォルト・バインディングとカスタム・バインディング

JSON-B には最初から、シリアライズとデシリアライズの際のデフォルト・マッピングが用意されています。これらのデフォルト・マッピングは、日常的なプログラミングのシナリオに適したものとなっています。単純なカスタマイズを行う場合に備えて、この API には、アダプターと、いくつかのカスタム・シリアライザーおよびデシリアライザーが用意されています。より高度なシナリオでは、クラス内でアノテーションを使用するか、ランタイム構成ビルダーを使用して、JSON-B のデフォルト設定をオーバーライドすることができます。

全体として、JSON-B はエンタープライズ開発者にとって一般的な業界の慣例と方法を体系化していると言えます。JSON-B では、アノテーションを使用してクラスやフィールドをマークし、マッピングのセマンティクスを設定します。また、複雑なデータ構造を扱うために必要とされる拡張性も提供します。

最も重要なことは、この API が Java クラスと JSON ドキュメントとの間のバインディングを直感的かつ容易な形でサポートしているという点です。したがって、JSON を扱った経験がない開発者でも、Java API for JSON Binding を簡単に理解して使用できます。GSON、Boon、Jackson などの JSON デシリアライズ/シリアライズ・ライブラリーを使い慣れている開発者にとっては、JSON-B は馴染み深く、使いやすいものに感じられるはずです。

## 2 つのインターフェース: Jsonb と JsonBuilder

機能の点から言うと、JSON-B API を構成するエントリー・ポイント・インターフェースには javax.json.bind.Jsonb と javax.json.bind.JsonBuilder の 2 つがあります。`Jsonb` インターフェースでは、`toJson()` メソッドと `fromJson()` メソッドによってシリアライズ、デシリアライズのそれぞれを可能にします。`JsonbBuilder` インターフェースは、一連のオプション構成に基づいて `Jsonb` インスタンスを構成し、クライアントがそのインスタンスにアクセスできるようにします。

### シリアライズして再びデシリアライズする例

新しい API を理解するには、実際に自分で使ってみることが何よりも効果的な方法です。この例では、単純なクラスを使用して基本的なサンプル・アプリケーションを作成します。まず、リスト 1 の `Book` クラスを JSON ドキュメントにシリアライズします。その後、この JSON ドキュメントを `Book` オブジェクトに再びデシリアライズします。

### リスト 1. 最小限の Book クラス

```
public class Book {
   private String id;
   private String title;
   private String author;
   // Plumbing code removed for brevity
}
```

`Book` オブジェクトをシリアライズするには、最初に `Jsonb` インスタンスを作成する必要があります。それには、`JsonbBuilder` API の static ファクトリー・メソッド `create()` を使用します。`Jsonb` インスタンスを作成した後は、多重定義された `toJson()` メソッドのうちの 1 つを呼び出して、そのメソッドにシリアライズ対象のオブジェクトを渡します (リスト 2 を参照)。

### リスト 2. Book インスタンスをシリアライズする

```
Book book = new Book("ABCD-1234", "Fun with Java", "Alex Theedom");
Jsonb jsonb = JsonbBuilder.create();
String json = jsonb.toJson(book);
```

`toJson()` メソッドから、リスト 3 に示すシリアライズ後の `Book` オブジェクトが返されます。

### リスト 3. シリアライズされて返された Book インスタンス

```
{
  "author": "Alex Theedom",
  "id": "ABCD-1234",
  "title": "Fun with Java"
}
```

リスト 3 に示されているプロパティーの辞書式順序に注目してください。この順序はデフォルトでのプロパティーの順序であり、カスタマイズすることもできます。JSON-B のカスタマイズについては、この記事の後のほうで詳しく説明します。

JSON-B でのデシリアライズも同じく単純です。`Jsonb` のインスタンスに基づき、多重定義された `fromJson()` メソッドのうちの 1 つを呼び出し、そのメソッドにシリアライズ対象の JSON ドキュメントと併せてオブジェクトの型を渡せばよいだけです (リスト 4 を参照)。

### リスト 4. JSON ストリングをデシリアライズする

```
Book book = jsonb.fromJson(json, Book.class);
```

**並列処理と JSON-B**: リスト 4 に示されているように、可能であれば常に `JsonbBuilder` と `Jsonb` のインスタンスを再利用することがベスト・プラクティスです。JSON-B API に関するこのリンク先の Java ドキュメントで示唆されているように、標準的な使用ケースでは、アプリケーションごとに必要となる `Jsonb` インスタンスは 1 つだけです。JSON-B のメソッドのすべては、同時に実行される複数のスレッドで安全に使用できます。

JSON-B 仕様では変換の前後で等価性を維持することを要件としていませんが、推奨はしています。保証はないものの、ほとんどの場合はシリアライズした JSON 出力をデシリアライズされた `fromJson()` メソッドにフィードすると、シリアライズされる前と同等のオブジェクトになります。

## JSON-B のデフォルト設定

便宜上、`Jsonb` インスタンスは、一般に認められた標準が反映されたデフォルト設定を使用して事前構成された形になっています。JSON-B のデフォルトのストラテジーとマッピングの概要は以下のとおりです。この連載の以降の記事ではこれらのデフォルト設定をいくつか抜粋し、さらに詳しく見ていきます。

- **プロパティーの命名ストラテジー**により、クラスのフィールドが、シリアライズされた JSON 内でどのように表現されるのかが決まります。デフォルトでは、JSON プロパティー名は、シリアライズ対象のクラスに含まれるフィールドと完全に同じ名前を引き継ぎます。
- **Pプロパティーの可視性ストラテジー**Pにより、フィールドへのアクセス方法と、フィールドおよび Bean メソッド上のアクセス修飾子に与えられる重要度が決まります。シリアライズ

の際も、デシリアライズの際も、JSON-B のデフォルトの動作は、public フィールド、public アクセサー・メソッド、および public ミューテーター・メソッドにだけアクセスするというものです。public メソッドまたは public フィールドが存在しなければ、そのプロパティーは無視されます。
- **プロパティーの順序ストラテジー**により、出力される JSON 内でのプロパティーの出現順序が指定されます。JSON-B のデフォルトの順序は、辞書式順序です。
- **バイナリー・データ・ストラテジー**により、デフォルトではバイト配列を使用して、バイナリー・データが符号化されます。
- **null 値ストラテジー**により、null 値を持つオブジェクト・フィールドは無視され、コレクション内の null 値は維持されるように指定されます。JSON プロパティーに null 値が含まれる場合、対応するオブジェクト・フィールドの値は null として設定されます。

理解しておかなければならないその他のデフォルトには、以下の 3 つがあります。

- JSON-B は I-JSON プロファイルに準拠します。ただし 3 つの例外として、JSON-B は最上位レベルの JSON のシリアライズを非オブジェクトまたは配列テキストに制限していないこと、`base64url` 符号化方式を使用したバイナリー・データをシリアライズしないこと、時間に基づくプロパティーに追加の制約事項を適用しないことが挙げられます。また、カスタマイズ手段を使用して I-JSON への厳格な準拠を有効にすることも可能です。
- JSON-B のデータ・フォーマットとロケール設定は、使用しているマシンの設定に依存します。
- JSON データは改行やインデントなしで表現され、UTF-8 を使用して符号化されます。

## JSON-B での型の処理

Java API for JSON Binding は RFC 7159 に準拠したシリアライズおよびデシリアライズを行い、Java データ型とプリミティブ型のすべてをサポートしています。この連載では、以下の原則および手法の多くを調査して使用します。

- **基本型**: 基本型には、すべてのプリミティブ型とプリミティブ型に関連するラッパー・クラス `Number` および `String` が含まれます。
  - シリアライズの際は、インスタンス・フィールドに対して `toString()` メソッドが呼び出され、それによって `String` 値が生成されます。
  - `Number` 型のフィールドは例外です。`Number` 型では `doubleValue()` メソッドを使用してプリミティブ型を生成するためです。
  - プリミティブ型の値は手付かずのまま残されます。
  - デシリアライズの際は、該当する `parseXXX()` メソッドが呼び出されて、必要な基本型への変換が行われます。
- **Java SE データ型**: これらの型は、一連の `Optional` クラス (`OptionalInt` など)、`BigInteger` および `BigDecimal` クラス、URI および `URL` クラスを指します。
  - Optional 型の値は `String` 型に変換されます。その方法は、内部のデータ型を取得し、そのオブジェクトを `String` 型またはプリミティブ型に変換する際の規則に従うというものです (一例として、`Integer` 型のインスタンスに対しては `intValue()` が呼び出されます)。
  - 空のオプションは null 値として扱われ、null 値の処理に対するカスタマイズ設定に応じて処理されます。

    - デシリアライズのプロセスでは、`String` 型パラメーターを受け入れる、該当する型のコンストラクション・メソッドが利用されます。その一例としては、新規の `BigDecimal("1234")` が挙げられます。
- **日付型**: JSON-B は、`Calendar`、`TimeZone`、および `java.time.*` パッケージを含め、標準的な Java 日付型のすべてをサポートしています。シリアライズの際に使用されるフォーマットは、`ISO_DATE`、`ISO_DATE_TIME`、および `ISO_INSTANT/LOCAL_*/ZONED_*/OFFSET_*` です。API 仕様 (セクション 3.5) で、各日付型で使用されるフォーマットの構成要素について詳しく説明しています。
- **配列とコレクション型**: サポートされているすべての Java データ型およびプリミティブ型の単一の配列でも多次元配列でも、シリアライズ/デシリアライズすることができます。null 値の要素は、生成される JSON または Java 配列の中で null 値として表現されます。
- **JSON Processing の型**: JSON Processing のすべての型は、シリアライズおよびデシリアライズでサポートされています。
- **Java クラス**: Java クラスをシリアライズ可能にするには、引数なしの `public` または `protected` コンストラクターを定義するか、カスタム・オブジェクトを作成するために使用するメソッドを明示的に指定する必要があります。以下の規則に注意してください。
    - シリアライズの場合、制約は必要ありません。JSON-B では、`public` クラス、`protected` クラス、および `protected static` クラスをデシリアライズできます。
    - 匿名クラスはサポートされていません。匿名クラスをシリアライズすると、JSON オブジェクトが生成されます。
    - Java 基本データ型、Java SE データ型、`Date/Time` 型、`Collection/Map` 型に関連するもの以外のインターフェースは、デシリアライズの際にサポートされません。ランタイム型はシリアライズに使用されます。

## デフォルトのマッピング

次は、これまでに紹介したデフォルト設定と型の多くが含まれる例を見ていきましょう。リスト 5 の `Magazine` クラスは、一連の Java データ型、プリミティブ型、コレクションを提示しています。`internalAuditCode` フィールドにはセッター・メソッドしか定義されていないことに注意してください。

### リスト 5. Magazine クラス

```
public class Magazine {
   private String id;
   private String title;
   private Author author;
   private Float price;
   private int pages;
   private boolean inPrint;
   private Binding binding;
   private List<String> languages;
   private URL website;
   private String internalAuditCode; // Only has setter method
   private LocalDate published;
   private String alternativeTitle;
   // Plumbing code removed for brevity
}
```

リスト 6 で作成されている `Magazine` オブジェクトの `language` 配列には null フィールドが含まれています。また、このオブジェクトの `alternativeTitle` フィールドは null に設定されています。

### リスト 6. Magazine オブジェクトの作成

```
Magazine magazine = new Magazine();
magazine.setId("ABCD-1234");
magazine.setTitle("Fun with Java");
magazine.setAuthor(new Author("Alex", "Theedom"));
magazine.setPrice(45.00f);
magazine.setPages(300);
magazine.setInPrint(true);
magazine.setBinding(Binding.SOFT_BACK);
magazine.setLanguages(
    Arrays.asList("French", "English", "Spanish", null));
magazine.setWebsite(new URL("https://www.readlearncode.com"));
magazine.setInternalAuditCode("IN-675X-NF09");
magazine.setPublished(LocalDate.parse("01/01/2018",
    DateTimeFormatter.ofPattern("MM/dd/yyyy")));
magazine.setAlternativeTitle("IN-675X-NF09");
String json = JsonbBuilder.create().toJson(magazine);
```

リスト 7 に、このオブジェクトをシリアライズした結果を示します。このオブジェクトをシリアライズすると、配列内の null 値は維持される一方、null 値が含まれている `alternativeTitle` フィールドは除外されます。

また、この JSON 出力からは `internalAuditCode` の値も除外されていることに注意してください。この値を含めることができないわけは、値を取得する際に使用できるゲッター・メソッドも public フィールドもないためです。

### リスト 7. Magazine の JSON 表現

```
{
  "author": {
    "firstName": "Alex",
    "lastName": "Theedom"
  },
  "binding": "SOFT_BACK",
  "id": "ABCD-1234",
  "inPrint": true,
  "languages": [
    "French",
    "English",
    "Spanish",
    null
  ],
  "pages": 300,
  "price": 45.0,
  "published": "2018-01-01",
  "title": "Fun with Java",
  "website": "https://www.readlearncode.com"
}
```

## カスタム・マッピング

JSON-B のデフォルトのマッピング動作は、単純なシナリオのほとんどで十分必要を満たすことができるはずです。その一方で、複雑なシナリオやニーズを満たせるよう、この API にはマッピングの動作をカスタマイズするさまざまな方法が用意されています。

カスタマイズ・モデルには、アノテーション・モデルとランタイム・モデルの 2 つがあります。この 2 つを同時に使用することも、それぞれ単独で使用することもできます。アノテーション・

モデルでは、動作をカスタマイズするフィールドをマークするために組み込みアノテーションを使用します。ランタイム・モデルでは、構成ストラテジーを作成して `Jsonb` インスタンス上に設定します。アノテーションとランタイム構成は、シリアライズとデシリアライズの両方に適用することができます。

### アノテーションを使用して JSON-B をカスタマイズする

カスタマイズするマッピング動作を関連付けるフィールド、JavaBean プロパティー、型、またはパッケージを、アノテーションを使ってマークします。以下は、その一例です。

### リスト 8. フィールドをアノテーションでマークする

```
@JsonbProperty("writer")
private Author author;
```

`author` フィールドに設定されたアノテーションによって、JSON 出力内のプロパティーの名前が `author` から `writer` に変更されます。

### ランタイム構成を使用して JSON-B をカスタマイズする

ランタイム・モデルでは、`JsonbConfig` のインスタンスを作成し、そのインスタンスを `JsonBuilder` の `create()` メソッドに渡します (リスト 9 を参照)。

### リスト 9. Jsonb 構成インスタンスを作成する

```
JsonbConfig jsonbConfig = new JsonbConfig()
       .withPropertyOrderStrategy(PropertyOrderStrategy.REVERSE)
       .withNullValues(true);

Jsonb jsonb = JsonbBuilder.create(jsonbConfig);
```

この例の構成では、プロパティーの順序を `reverse` に設定し、すべての null 値を維持します。

## 基本的なカスタマイズ

ほとんどの単純な使用ケースにはほとんどそのままの構成を使用して対応できますが、デフォルトのマッピングで対応できることは限られています。シリアライズやデシリアライズのプロセスをさらに細かく制御する必要がある場合に備え、JSON-B では一連のカスタマイズ手段を使用できるようになっています。

### プロパティーの名前と順序のカスタマイズ

JSON-B にはプロパティー関連のカスタマイズ手段がいくつか用意されているので、これらの手段について把握しておく必要があります。

フィールドに `@JsonbTransient` アノテーションを設定すると、そのフィールドのプロパティーは除外されます。また、プロパティーの名前を変更するには `@JsonProperty` アノテーションを設定します。命名ストラテジーを設定するには、`JsonbConfig` インスタンスの `withPropertyNamingStrategy()` メソッドに定数 `PropertyNamingStrategy` を渡します。

JSON 出力でのプロパティーの出現順序は、`@JsonbPropertyOrder()` アノテーションまたは `withPropertyOrderStrategy()` メソッドを使用してクラス・レベルで設定します。このアノテーションおよびメソッドに渡す定数 `PropertyOrderStrategy` によって、プロパティーの順序ストラテジーが決まります。

## 無視するプロパティーおよび可視性の構成

指定のプロパティーの可視性を指定するには、`@JsonbVisibility()` アノテーションまたは `withPropertyVisibilityStrategy()` メソッドを使用します。

## null の処理

null を処理する際のデフォルトの動作を変更するには 3 つの方法があります。

1. パッケージまたはクラスに `@JsonbNillable` アノテーションを設定する
2. 該当するフィールドまたは JavaBean プロパティーを `@JsonbProperty` アノテーションでマークし、`nillable` パラメーターを true に設定する
3. `withNullValues()` メソッドに true または false を渡す

リスト 10 では、2 番目と 3 番目の方法を使用する例として、クラスに `@JsonbNillable` アノテーションを設定し、`nillable` を true に設定しています。`@JsonbNillable` というクラス・レベルのアノテーションを使用して、すべてのフィールドに対する null の処理を構成することもできます。また、フィールド・レベルのアノテーションを使用すれば、フィールド・レベルでのよりきめ細かい制御が可能になります。

## リスト 10. null の処理を構成する

```
public class Magazine {
    @JsonbProperty(value = "writer", nillable = true)
    private Author author;
}

@JsonbNillable
    public class Magazine {
    private Author author;
}
```

## カスタム作成機能

前述のように、JSON-B では、すべてのクラスに引数なしの public コンストラクターがあることを期待し、そのコンストラクターを使用してクラスのインスタンスを作成することになっています。このようなコンストラクターによって必要を満たせない場合は、カスタム・コンストラクターを使用するか、static ファクトリー・メソッドを使用して対応することができます。いずれの方法でも、アノテーション `@JsonCreator` を使用します。

## カスタムの日付型式と数値のフォーマット

JSON-B では、`@JsonbDateFormat()` アノテーションを使用して日付形式を設定し、使用するロケールおよび日付パターンを渡すように指定しています。このアノテーションは、パッケージ

からフィールドに至るまで、どの場所でも使用されます。日付形式を設定する別の方法としては、`withDateFormat()` メソッドを使用することもできます (リスト 11 を参照)。

### リスト 11. 日付形式

```
new JsonbConfig()
     .withDateFormat("MM/dd/yyyy", Locale.ENGLISH);
```

### バイナリー・データの処理

JSON-B では、バイナリー・データの処理に関して 3 つのストラテジーを用意しています。

- BYTE
- BYTE_64
- BYTE_64_URL

これら 3 つの静的定数のうちのいずれかを、`BinaryDataStrategy` から `withBinaryDataStrategy()` メソッドに渡します。リスト 12 に一例を記載します。

### リスト 12. バイナリー・データ・ストラテジー

```
new JsonbConfig()
     .withBinaryDataStrategy(BinaryDataStrategy.BASE_64_URL);
```

### I-JSON の構成

「JSON-B のデフォルト設定」で説明した例外を別として、デフォルト設定の大部分は、I-JSON プロファイルに従っています。さらに厳格に I-JSON プロファイルに準拠させる必要があるシナリオには、`withStrictIJSON(true)` メソッドを使用することができます。

## 高度なカスタマイズ

場合によっては、アノテーションもランタイム構成も役に立たないことがあります。例えば、通常は、サード・パーティー・クラスをアノテーションでマークすることは不可能です。また、デフォルトのコンストラクターが定義されていないクラスも面倒の種になります。このようなシナリオに対する 2 つのソリューションとして、JSON-B ではアダプターとシリアライザー/デシリアライザーを使用できるようになっています。

### アダプターを使用して JSON-B をカスタマイズする

アダプターを使用すると、カスタム Java オブジェクトを作成して、JSON コードをシリアライズすることができます。アダプター・クラスでは `JsonbAdapter` インターフェースを実装すること、そしてこのインターフェースの `adaptToJson()` と `adaptFromJson()` という 2 つのメソッドに対応するコードを指定することが必要です。

リスト 13 では、`Booklet` オブジェクト (リスト 14 を参照) を JSON ストリングに変換するコードを使用して `adaptToJson()` メソッドを実装しています。

## リスト 13. adaptToJson() メソッドを実装する

```
public JsonObject adaptToJson(Booklet booklet) throws Exception {
   return Json.createObjectBuilder()
        .add("title", booklet.getTitle())
        .add("firstName", booklet.getAuthor().getFirstName())
        .add("lastName", booklet.getAuthor().getLastName())
        .build();
}
```

リスト 13 では、JSON-P の `JsonObjectBuilder` を使用していることに注目してください。以下に、`Booklet` オブジェクトを記載します。

## リスト 14. Booklet

```
public class Booklet {
    private String title;
    private Author author;
    // Plumbing code removed for brevity
}
```

ご覧のように、`adaptToJson()` メソッドは `Author` オブジェクトを 2 つのプロパティー (`firstName` と `secondName`) に「フラット化」します。この方法は、アダプターを使用してシリアライズをカスタマイズするさまざまな方法の一例でしかありません。

## JSON-B アダプターを登録する

JSON-B アダプターを登録するには、以下のように `JsonbConfig` インスタンスを使用します。

```
new JsonbConfig().withAdapters(new bookletAdapter())
```

あるいは、特定のクラスに対して以下のアノテーションを使用するという方法もあります。

```
@JsonbTypeAdapter(BookletAdapter.class)
```

リスト 15 に、この変換によって生成された JSON ドキュメントを記載します。

## リスト 15. Booklet クラスのシリアライズ

```
{
  "title": "Fun with Java",
  "firstName": "Alex",
  "lastName": "Theedom"
}
```

`adaptFromJson()` メソッドはここには記載されていませんが、この記事に関連する GitHub リポジトリー内にサンプル・コードが用意されています。

## シリアライザーとデシリアライザーを使用して JSON-B をカスタマイズする

JSON-B のシリアライザーとデシリアライザーは、最下位レベルで利用できるカスタマイズ手段です。この 2 つを使用することで、JSON Processing API 内にあるパーサーやジェネレーターにアクセスできるようになります。

この手段を使用するには、`JsonbDeserializer` と `JsonbSerializer` を実装して、該当する `serialize()` または `deserialize()` メソッドをオーバーライドする必要があります。その上で、必要な処理を行うカスタム・コードを作成し、そのコードをメソッド内に配置します。

シリアライザーとデシリアライザーを `JsonbConfig` インスタンスに登録するには、`withDeserializers()` または `withSerializers()` のいずれか該当するメソッドを使用します。JSON-B のシリアライザーとデシリアライザーを作成して使用する例については、この記事の GitHub リポジトリーを参照してください。

## まとめ

Java EE 8 で JSON-B をリリースするために行ったエキスパート・グループの懸命な努力は、称賛に値します。この API により、Java EE の JSON API スイートは完全なものになったからです。今回の記事では、JSON Binding API の包括的な機能の表面をかじっただけに過ぎません。以降の 3 回の記事で、Java EE に追加されたこの最も歓迎すべき API の本質的側面について詳しく探っていきます。

## 学んだ知識をテストしてください

1. 次のうち、どのアノテーションを組み合わせて使用すると、null 値を維持し、プロパティーを辞書式順序とは逆の順序で並べ、プロパティー「cost」の名前を変更するように JSON 出力をカスタマイズできますか？
   a. `@JsonbNillable`
   b. `@JsonbNullable`
   c. `@JsonbPropertyOrder(PropertyOrderStrategy.REVERSE)`
   d. `@JsonbPropertyOrder(PropertyOrderStrategy.RESERVED)`
   e. `@JsonbPropertyName("cost")`
   f. `@JsonbProperty("cost")`
2. 厳格な I-JSON プロファイルを使用し、月/日/年としての日付形式と英語ロケールを指定し、JSON を整形して出力するカスタム構成を作成するには、次の `JsonbConfig` ビルダー・メソッドのうち、どのメソッドを組み合わせて呼び出しますか？
   a. `withNonStrictIJSON(false)`
   b. `withStrictIJSON(true)`
   c. `withDateFormat(Locale.ENGLISH, "MM/dd/yyyy")`
   d. `withDateFormat("MM/dd/yyyy", Locale.ENGLISH)`
   e. `withFormatting(true)`
   f. `withPrettyFormatting(true)`
3. アダプターが `OrderAdapter.class` という名前になっている場合、このアダプターを使用するように指定する方法は、次のうちどれですか？
   a. `new JsonbConfig().withAdapters(new orderAdapter())`
   b. `new JsonbConfig().withTypeAdapters(new orderAdapter())`
   c. `@JsonbTypeAdapter(OrderAdapter.class)`
   d. `@JsonbAdapter(OrderAdapter.class)`
   e. どれも当てはまらない
4. 次のうち、シリアライズとデシリアライズのデフォルトの可視性構成はどれですか？
   a. public アクセサーおよびミューテーター・メソッドのみ

   b. public アクセサーおよびミューテーター・メソッドと、public フィールドのみ
   c. public フィールドのみ
   d. public フィールドと private フィールド
   e. public/protected アクセサーおよびミューテーター・メソッドと、public/protected フィールド
5. 次のうち、JSON ドキュメントからのオブジェクトの作成をカスタマイズするために使用できる手段はどれですか？
   a. `@JsonCreator` アノテーションを使用したカスタム・コンストラクター
   b. `@JsonCreator` アノテーションを使用した static ファクトリー・メソッド
   c. JSON B アダプター
   d. JSON B シリアライザー
   e. このカスタマイズは不可能

このリンク先のページで正解を確認してください。

## 関連トピック

- Java EE 8 の新機能
- この記事のコードは GitHub で入手可能です。
- JSON Binding API のソースコード
- Yasson: JSON-B のリファレンス実装
- JSON-B のオープン・イシューの一覧を入手してください。
- JSON Binding 使用に関するフィードバックを投稿してください。