Java EE 8 の新機能

Java セキュリティー、JSON バインディングおよび処理、HTTP/2 などに対応するための新しい API および機能の紹介

Alex Theedom
Senior Java developer
Consultant

2018年 4月 26日

クラウドとリアクティブ・プログラミング向けに作成された次期 Java Platform Enterprise Edition は、今後数年のうちに、エンタープライズ・アプリケーション開発を方向付けることになるでしょう。この記事では、Java エンタープライズ・セキュリティー、JSON バインディング、HTTP/2 サーバー・プッシュなどに対応するための新しい API とアップグレードについて紹介します。

待望の Java EE 8 がもうすぐリリースされます。このリリースは、2013 年 6 月以来初となる Java Platform Enterprise Edition のリリースであり、最終的には Java EE 9 で締めくくられる 2 回のリリースの前半となっています。Oracle はクラウド・コンピューティング、マイクロサービス、リアクティブ・プログラムをサポートするテクノロジーを重視して戦略的に Java EE を新しく位置付けしています。その結果、多くの Java EE API の基礎構造にリアクティブ・プログラミングが織り込まれ、JSON 交換フォーマットがコア・プラットフォームを支えるようになっています。

この記事では、Java EE 8 の主な特徴を 1 つひとつ紹介していきます。ハイライトとなっているのは、既存の API の更新と新しい API の導入、そして HTTP/2、リアクティブ・プログラミング、JSON を対象とした新しいサポートなどです。この先数年のうちに間違いなくエンタープライズ Java プログラミングを方向付けることになる Java EE の仕様とアップグレードを導入してください。

新しい API と更新された API

Java EE 8 では、中核的な API にメジャーおよびマイナーな更新 (Servlet 4.0、Context and Dependency Injection 2.0 など) を取り入れている他、新しい API として Java API for JSON Binding (JSR 367) と Java EE Security API (JSR 375) の 2 つも導入しています。まずは新しく導入された 2 つの API から紹介し、その後、長年存在している Java EE 仕様に加えられた変更を探っていきます。

JSON Binding API

新しく導入された JSON Binding API (JSON-B) は、JAXB (Java API for XML Binding 2.0) との整合性を維持しながらも、Java オブジェクトと RFC 7159 対応 JSON との間のシリアライズとデシリアライ

ズをサポートし、Java のクラスおよびインスタンスと、認められた表記法に従った JSON ドキュメントとの間のデフォルト・マッピングを提供しています。

JSON-B では、開発者がシリアライズとデシリアライズをカスタマイズできるようにもなっています。具体的には、アノテーションを使用して個々のクラスに対してシリアリズ/デシリアライズのプロセスをカスタマイズするという方法、またはランタイム構成ビルダーを使用してカスタム戦略を開発するという方法です。後者の方法では、ユーザー定義のカスタマイズをサポートするためにアダプターも使用します。JSON-B は、後で説明する Java API for JSON Processing (JSON-P) 1.1 と密接に統合します。

仕様の内容

新しい JSON Binding API を使用するためのインターフェースには、`JsonbBinding` と `Jsonb` の 2 つがあります。

- `JsonbBinding` は、JSON Binding API へのクライアント・アクセス・ポイントを提供するために、設定された構成とパラメーターに基づいて `Jsonb` インスタンスを作成します。
- `Jsonb` では、`toJson()` メソッドと `fromJson()` メソッドを使用して、それぞれシリアライズ、デシリアライズを処理できるようになっています。

JSON-B では、外部 JSON バインディング・プロバイダーを接続するための機能についても記述しているため、API に付属のバインディング・ロジックしか使用できないわけではありません。

JsonB を使用したリアライズとデシリアライズ

リスト 1 に、`book` という名前の `Book` クラスのインスタンスをシリアライズする例、次にデシリアライズする例を示します。

リスト 1. 最も単純なシリアライズとデシリアライズの例

```
String bookJson = JsonbBuilder.create().toJson(book);
Book book = JsonbBuilder.create().fromJson(bookJson, Book.class);
```

上記で使用している静的 `create()` ファクトリー・メソッドは、`Jsonb` のインスタンスを返します。そのインスタンスに対して、いくつもの多重定義された `toJson()` および `fromJson()` メソッドを呼び出すことができます。また、この仕様では往復変換での等価性を要件としていないことにも注意してください。したがって、上記の例では、`bookJson` ストリングを `fromJson()` メソッドに取り込んでも同等のオブジェクトにデシリアライズされない可能性があります。

JSON-B は、コレクション・クラスとプリミティブやインスタンスの配列 (多次元配列を含む) のバインディングについても、オブジェクトとほとんど同じような形でサポートしています。

Jsonb のカスタマイズ

`Jsonb` のメソッドのデフォルト動作は、フィールド、JavaBeans メソッド、クラスにアノテーションを付けることでカスタマイズできます。

例えば、`@JsonbNillable` アノテーションを使用して null の処理をカスタマイズしたり、`@JsonbPropertyOrder` アノテーションを使用してプロパティーの順序をカスタマイズしたりできます。これらのアノテーションは、クラス・レベルで指定します。

リスト 2. Jsonb のカスタマイズ

```
@JsonbNillable
@JsonbPropertyOrder(PropertyOrderStrategy.REVERSE)
public class Booklet {

   private String title;

   @JsonbProperty("cost")
   @JsonbNumberFormat("#0.00")
   private Float price;

   private Author author;

   @JsonbTransient
   public String getTitle() {
       return title;
   }

   @JsonbTransient
   public void setTitle(String title) {
       this.title = title;
   }

   // price and author getters/setter removed for brevity
 }
```

`toJson()` メソッドを呼び出すと、リスト 3 の JSON 構造体が生成されます。

リスト 3. カスタマイズされた JSON 構造体

```
{
  "cost": "10.00",
  "author": {
    "firstName": "Alex",
    "lastName": "Theedom"
  }
}
```

別の方法として、`JsonbConfig` というランタイム構成ビルダーを使用してカスタマイズを処理することもできます。

リスト 4. Jsonb のランタイム構成

```
JsonbConfig jsonbConfig = new JsonbConfig()
    .withPropertyNamingStrategy(
        PropertyNamingStrategy.LOWER_CASE_WITH_DASHES)
    .withNullValues(true)
    .withFormatting(true);

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

リスト 4 では、null が存在する場合は `LOWER_CASE_WITH_DASHES` 表記で null を表現して、修飾した JSON を出力するように JSON-B を構成しています。

オープンソースのバインディング

前述のとおり、JSON-B にあらかじめ用意されているバインディング・ロジックを使用しなければならないわけではありません。一例として、リスト 5 にオープンソースのバインディング実装を構成する方法を示します。

リスト 5. オープンソースのバインディング構成

```
JsonbBuilder builder = JsonbBuilder.newBuilder("aProvider");
```

Java EE Security API

サーブレット・コンテナー間でセキュリティー機能の実装方法に統一がとれていないという問題を正すために、新しい Java EE Security API が導入されました。この問題は、特に Java の Web Profile 内で顕著になっていました。その原因は主に、Java EE で要件としているのは、Java EE プロファイル全体で標準の API を実装する方法だけであるためです。この新しい仕様には、既存の API では利用していない CDI といった最新の機能も導入されています。

Java EE Security API の素晴らしい点は、アイデンティティー・ストアや認証メカニズムを構成する代替手段となることです。ただし、この API は既存のセキュリティー・メカニズムに取って代わるものではありません。ベンダー固有のソリューションや独自仕様のソリューションを使用するかどうかを問わず、柔軟に Java EE Web アプリケーション内でセキュリティーを有効にできるようになることは、開発者にとって歓迎すべき状況でしょう。

仕様の内容

Java EE Security API 仕様は、3 つの主要なセキュリティーの懸念に対処します。

- `HttpAuthenticationMechanism` により、サーブレット・コンテナーの認証をサポートします。
- `IdentityStore` により、JAAS `LoginModule` を標準化します。
- `SecurityContext` により、プログラムによるセキュリティーのアクセス・ポイントを提供します。

上記 3 つのコンポーネントそれぞれについて、簡単に説明しておきましょう。

HttpAuthenticationMechanism

Web アプリケーションのユーザーを認証するメカニズムについては、Java EE にはすでに 2 つの仕様があります。一方の Java Servlet 仕様 3.1 (JSR-340) では、アプリケーションの構成に関する宣言型メカニズムを指定しています。もう一方の JASPIC (Java Authentication Service Provider Interface for Containers) では、あらゆるタイプの資格情報を処理する認証モジュールの開発をサポートする、`ServerAuthModule` と呼ばれる SPI を定義しています。

これらのメカニズムは両方とも有意義で効果的ですが、Web アプリケーション開発者の観点からはどちらも限界があります。具体的には、サーブレット・コンテナー・メカニズムは限定的なタイプの資格情報しかサポートしていません。また、JASPIC は非常に強力で柔軟性のある仕様ではありますが、複雑で使いにくいという難点があります。

Java EE Security API はこれらの問題を、`HttpAuthenticationMechanism` という新しいインターフェースで解決しようとしています。本質的に、このインターフェースは JASPIC `ServerAuthModule` インターフェースを単純化してサーブレット・コンテナーの形にしたインターフェースであり、既存のメカニズムを利用すると同時に、これらのメカニズムに伴う制約を緩和しています。

`HttpAuthenticationMechanism` 型のインスタンスは、CDI Bean です。CDI Bean はコンテナーに対する注入に使用できますが、サーブレット・コンテナーのみを対象に指定されています。この仕様では、EJB や JMS などの他のコンテナーを明示的に除外しています。

`HttpAuthenticationMechanism` インターフェースが定義するメソッドには、`validateRequest()`、`secureResponse()`、`cleanSubject()` の 3 つがあります。これらのメソッドは JASPIC `ServerAuth` インターフェース上で宣言されるメソッドに非常によく似ているため、開発者にとって馴染み深いものに感じられるはずです。このうち、オーバーライドする必要があるメソッドは `validateRequest()` だけです。他のすべてのメソッドにはデフォルトの実装があります。

IdentityStore

アイデンティティー・ストアとは、ユーザーのアイデンティティー・データ (ユーザー名、グループ・メンバーシップなど、資格情報を検証するために使用する情報) を保管するデータベースのことです。新しい Java EE Security API 内では、`IdentityStore` という名前のアイデンティティー・ストアの抽象化を使用してアイデンティティー・ストアとやりとりします。その目的は、ユーザーを認証し、グループ・メンバーシップを取得することです。

要件ではありませんが、この仕様が作成されている意図は、`HttpAuthenticationMechanism` の実装から `IdentityStore` を使用することにあります。`IdentityStore` と `HttpAuthenticationMechanism` を併せて使用すると、アプリケーションはそのアイデンティティー・ストアを、移植可能な標準的な方法で制御できるようになります。

SecurityContext

`IdentityStore` と `HttpAuthenticationMechanism` を組み合わせると強力なユーザー認証ツールになりますが、それでもやはり、システム・レベルのセキュリティー要件が原因で、宣言型モデルでは認証に対応しきれない場合があります。そこで力を発揮するのが SecurityContext です。`SecurityContext` のプログラムによるセキュリティーにより、アプリケーション・リソースに対するアクセス権を付与または拒否するために必要なテストを、Web アプリケーションで実行できるようになります。

大幅な更新: Servlet 4.0、Bean Validation 2.0、CDI 2.0

Java EE 8 でメジャー・リリースになったエンタープライズ標準 API には、Servlet 4.0 (JSR 369)、Bean Validation 2.0 (JSR 380)、Contexts and Dependency Injection for Java 2.0 (JSR 365) の 3 つがあります。

それぞれの API の注目すべき機能について見ていきましょう。

Servlet 4.0

Java エンタープライズ開発者にとって、Java Servlet API は最も早くのうちからある API の 1 つで、最も深く根付いたものになっています。1999 年に J2EE 1.2 でデビューした Java Servlet API は、現在 Java Server Pages (JSP)、JavaServer Faces (JSF)、JAX-RS、MVC (JSR 371) で重要な役割を果たしています。

Java EE 8 で Java Servlet API が大幅に改定されている理由は、主に HTTP/2 のパフォーマンス強化機能に対応するためです。これらのパフォーマンス強化機能のうち、サーバー・プッシュは群を抜いて注目度が高い機能となっています。

> **サーバー・プッシュとは**
> サーバー・プッシュでは、サーバーがクライアント・サイドのリソース要件を予測して、それらのリソースをブラウザーのキャッシュにプッシュします。したがって、クライアントがリクエストを送信した後、サーバーからレスポンスを受け取る頃には、必要なリソースがすでにキャッシュ内にあるというわけです。

PushBuilder

Servlet 4.0 では、サーバー・プッシュは `PushBuilder` インスタンスを介して公開されます。リスト 6 に、`HttpServletResponse` インスタンスから取得されてリクエストを処理するメソッドに渡される `PushBuilder` インスタンスを示します。

リスト 6. サーブレット内の PushBuilder

```
@Override
protected void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {

PushBuilder pushBuilder = request.newPushBuilder();
pushBuilder.path("images/header.png").push();
pushBuilder.path("css/menu.css").push();
pushBuilder.path("js/ajax.js").push();

// Do some processing and return JSP that
// requires these resources
}
```

リスト 6 では、`path()` メソッドによって `PushBuilder` インスタンス上で `header.png` のパスが設定され、ここのパスが呼び出し側 `push()` メソッドによってクライアントにプッシュされます。メソッドが戻ると、ビルダーで再利用できるようにパスと条件付きヘッダーがクリアされます。

サーブレット・マッピングのランタイム・ディスカバリー

Servlet 4.0 では、URL マッピングのランタイム・ディスカバリーに新しい API を使用できるようになっています。この `HttpServletMapping` インターフェースの目標は、サーブレットをアクティブにする原因となったマッピングを判断しやすくすることです。この API は内部で以下の 4 つのメソッドを使用して、`ServletRequest` インスタンスからサーブレット・マッピングを取得します。

- `getMappingMatch()` は、一致のタイプを返します。
- `getPattern()` は、サーブレット・リクエストをアクティブにした URL パターンを返します。
- `getMatchValue()` は、一致した `String` を返します。
- `getServletName()` は、リクエストによってアクティブにされたサーブレット・クラスの完全修飾名を返します。

リスト 7. HttpServletMapping インターフェース上の 4 つすべてのメソッド

```
HttpServletMapping mapping = request.getHttpServletMapping();
String mapping = mapping.getMappingMatch().name();
String value = mapping.getMatchValue();
String pattern = mapping.getPattern();
String servletName = mapping.getServletName();
```

以上の更新に加え、Servlet 4.0 ではハウスキーピング処理のための小さな変更も行われ、HTTP Trailer のサポートが追加されています。新しい `GenericFilter` および `HttpFilter` クラスは、フィルターの作成を単純化し、Jave SE 8 に全般的な改善をもたらしています。

Bean Validation 2.0

Bean Validation 2.0 が強化され、幅広い新機能が追加されています。その多くは、Java 開発者コミュニティーから要望されていた機能です。Bean 検証は分野横断的な懸念事項であることから、2.0 仕様ではクライアントからデータベースに至るまでのデータ整合性を確実にするために、フィールド、戻り値、メソッド・パラメーターに含める値に制約を適用します。

これらの機能強化の中には、e-メール・アドレスを検証する制約、数値が正または負であることを確認する制約、日付が過去または現在であるかどうかをテストする制約、フィールドが空または null でないことをテストする制約も含まれます。具体的には、`@Email`、`@Positive`、`@PositiveOrZero`、`@Negative`、`@NegativeOrZero`、`@PastOrPresent`、`@FutureOr`です。現在、さらに幅広い場所で制約を適用できるようになっています。制約により、例えばパラメーター化されたタイプの引数を処理することもできます。リスト 8 に示すように、型引数によってコンテナー要素を検証するためのサポートが追加されています。

リスト 8. 型によるコンテナー要素の検証

```
private List<@Size(min = 30) String> chapterTitles;
```

更新された Bean Validation API は、Java SE 8 の `Date` 型と `Time` 型を改善し、`java.util.Optional` のサポートを提供しています (リスト 9 を参照)。

リスト 9. Bean Validation 2.0 での Date 型、Time 型、Optional のサポート

```
Optional<@Size(min = 10) String> title;

private @PastOrPresent Year released;
private @FutureOrPresent LocalDate nextVersionRelease;
private @Past LocalDate publishedDate;
```

新しい便利な機能としては、コンテナーのカスケード検証も挙げられます。コンテナーの型引数に `@Valid` アノテーションを付けると、親オブジェクトの検証時にすべての要素が検証されることになります。リスト 10 では、すべての `String` と `Book` 要素が検証されるようにしています。

リスト 10. コンテナー型のカスケード検証

```
Map<@Valid String, @Valid Book> otherBooksByAuthor;
```

Bean Validation はカスタム・コンテナー型のサポートを追加するために、値のエクストラクターも組み込みます。組み込みコンテナーは反復可能としてマークされ、リフレクションを使用してパラメーター名が取得されます。デフォルトのメソッドは `ConstraintValidator#initialize()` です。JavaFX の型もサポートされます。

Contexts and Dependency Injection for Java 2.0

Context and Dependency Injection API (CDI) は、基幹テクノロジーとしてバージョン 6 で Java EE に導入されました。それ以来、開発を容易にするための重要な機能となっています。

この API が新しく生まれ変わり、Java SE と連動するように拡張されています。この変更に対応するために、CDI 仕様は 3 つのセクションに分割されました。Java EE と Java SE の両方に共通の概念を取り上げたパート 1、Java SE での CDI のルールだけを取り上げたパート 2、そして Java EE での CDI のルールだけを取り上げたパート 3 です。

CDI 2.0 では、オブザーバーとイベントが動作してやりとりする方法にも重要な変更を加えています。

CDI 2.0 でのオブザーバーとイベント

CDI 1.1 では、オブザーバーの実行順序を定義するメカニズムがないため、イベントが起動されるとオブザーバーが同期的に呼び出されます。このような動作に伴う問題は、オブザーバーが例外をスローしたとすると、以降のオブザーバーは 1 つも呼び出されることなく、その連鎖が止まってしまうことです。CDI 2.0 ではこの問題をある程度緩和する手段として、`@Priority` アノテーションを導入しました。このアノテーションにより、オブザーバーを呼び出す順序を指定できるようになっています (番号が小さいものの順から呼び出されます)。

リスト 11 に、イベントの起動と、優先度が異なる 2 つのオブザーバーを示します。2 つのオブザーバーのうち、`AuditEventReciever1` (優先度 10) が `AuditEventReciever2` (優先度 100) より先に呼び出されます。

リスト 11. オブザーバーの優先度の例

```
@Inject
private Event<AuditEvent> event;

public void send(AuditEvent auditEvent) {
   event.fire(auditEvent);
}

// AuditEventReciever1.class
public void receive(@Observes @Priority(10) AuditEvent auditEvent) {
 // react to event
}

// AuditEventReciever2.class
public void receive(@Observes @Priority(100) AuditEvent auditEvent) {
 // react to event
}
```

指摘すべき点として、同じ優先度を持つ複数のオブザーバーが呼び出される順序は予測できないことがあります。デフォルトの順序は、`javax.interceptor.Interceptor.Priority.APPLICATION + 500` です。

非同期イベント

イベントを非同期で起動できるようになっていることも、オブザーバーに追加された興味深い機能です。この機能をサポートするために、新しい起動メソッド (`Async()`) と、それに対応するオブザーバー・アノテーション (`@ObservesAsyncfire`) が追加されました。リスト 12 に、非同期で起動される `AuditEvent` と、このイベントの通知を受け取るオブザーバー・メソッドを示します。

リスト 12. イベントの非同期起動

```
@Inject
private Event<AuditEvent> event;

public CompletionStage<AuditEvent> sendAsync(AuditEvent auditEvent) {
   return event.fireAsync(auditEvent);
}

// AuditEventReciever1.class
public void receiveAsync(@ObservesAsync AuditEvent auditEvent) {}

// AuditEventReciever2.class
public void receiveAsync(@ObservesAsync AuditEvent auditEvent) {}
```

オブザーバーのいずれか 1 つからでも例外がスローされると、`CompletionStage` は `CompletionException` で完了します。このインスタンスには、オブザーバーの起動中にスローされて抑止されたすべての例外への参照が格納されます。リスト 13 に、このシナリオに対処する例を示します。

リスト 13. 非同期オブザーバーでの例外の対処

```
public CompletionStage<AuditEvent> sendAsync(AuditEvent auditEvent) {
   System.out.println("Sending async");
   CompletionStage<AuditEvent> stage = event.fireAsync(auditEvent)
           .handle((event, ex) -> {
               if (event != null) {
                   return event;
               } else {
                   for (Throwable t : ex.getSuppressed()) {}
                   return auditEvent;
               }
           });

   return stage;
}
```

CDI 2.0 は前述のとおり、ストリーム、ラムダ式、反復可能修飾子などの Java SE 8 機能も全般的な改善をもたらしています。注目に値するその他の追加内容は以下のとおりです。

- 新しい `Configurators` インターフェース
- オブザーバー・メソッドを構成または禁止する機能
- 組み込みアノテーション・リテラル
- プロデューサーに対してインターセプターを適用する機能

すべての変更を網羅したリストは、ここに掲載されています。詳細については、このリンク先のページで仕様の最終ドラフト案を参照してください。

細かな更新: JAX-RS 2.1、JSF 2.3, JSON-P 1.1

Java API for RESTful Web Services (JSR 370)、JavaServer Faces 2.3 (JSR 372)、Java API for JSON Processing 1.1 (JSR 374) にも変更が加えられています。これらの変更は比較的小規模ながらも、特にリアクティブな関数型スタイル・プログラミングの要素を利用しているという点で注目に値します。

Java API for RESTful Web Services 2.1

JAX-RS 2.1 API リリースで重点が置かれている主要な機能は 2 つあります。それは、新しいリアクティブなクライアント API と、サーバー送信イベントのサポートです。

リアクティブなクライアント API

RESTful Web Services にはリリース 1.1 以来、クライアント API が組み込まれており、高位レベルの手段を通じて Web リソースにアクセスできるようになっています。JAX-RS 2.1 ではこの API に、リアクティブ・プログラミングのサポートを追加しています。最も顕著な違いは、クライアント・インスタンスを作成するために使用する `Invocation.Builder` にあります。新しい `rx()` メソッド (リスト 14 を参照) の戻り値の型は `CompletionStage` となっていて、パラメーター化された型のレスポンスを返します。

リスト 14. 新しい rx() メソッドを使用した呼び出しビルダー

```
CompletionStage<Response> cs1 = ClientBuilder.newClient()
       .target("http://localhost:8080/jax-rs-2-1/books")
       .request()
       .rx()
       .get();
```

Java 8 で導入された `CompletionStage` インターフェースは、いくつかの興味深い可能性を提示しています。リスト 15 では、異なる 2 つのエンドポイントを呼び出して、それぞれの呼び出しの結果を結合しています。

リスト 15. 異なるエンドポイントの呼び出し結果の結合

```
CompletionStage<Response> cs1 = // from Listing 14
CompletionStage<Response> cs2 = ClientBuilder.newClient()
       .target("http://localhost:8080/jax-rs-2-1/magazines")
       .request()
       .rx()
       .get();

cs1.thenCombine(cs2, (r1, r2) ->
    r1.readEntity(String.class) + r2.readEntity(String.class))
        .thenAccept(System.out::println);
```

サーバー送信イベント

HTML 5 で W3C によって導入され、WHATWG コミュニティーによって保守されている Server Sent Events (SSE) API を使用すると、サーバーが生成するイベントにクライアントがサブスクライブできます。SSE アーキテクチャーでは、サーバーからクライアントへの片方向チャネルが作成され、このチャネルを介してサーバーが複数のイベントを送信することができます。接続は長時間存続し、サーバーまたはクライアントが閉じるまでオープン状態が維持されます。

JAX-RS API には、SSE 用のクライアントとサーバー API が含まれています。クライアント側からのエントリー・ポイントは、リスト 16 に示されている `SseEventSource` インターフェースです。ここでは、`WebTarget` を構成して、このインターフェースを変更しています。このコード・スニペットにおいて、クライアントはコンシューマーを登録します。登録後、コンシューマーはコン

ソールに出力されてから、接続を開きます。`onComplete` および `onError` ライフサイクル・イベントのハンドラーもサポートされます。

リスト 16. サーバー送信イベントに対応する JAX-RS クライアント API

```
WebTarget target = ClientBuilder.newClient()
    .target("http://localhost:8080/jax-rs-2-1/sse/");

try (SseEventSource source = SseEventSource
    .target(target).build()) {
    source.register(System.out::println);
    source.open();
}
```

もう一方の側では、SSE サーバー API がクライアント側からの接続を受け入れて、接続されているすべてのクライアントにイベントを送信します。

リスト 17. サーバー送信イベントに対応するサーバー API

```
@POST
@Path("progress/{report_id}")
@Produces(MediaType.SERVER_SENT_EVENTS)
public void eventStream(@PathParam("report_id")String id,
                        @Context SseEventSink es,
                        @Context Sse sse) {
    executorService.execute(() -> {
    try {
        eventSink.send(
            sse.newEventBuilder().name("report-progress")
                .data(String.class,
                "Commencing process for report " + id)
                .build());
            es.send(sse.newEvent("Progress", "25%"));
            Thread.sleep(500);
            es.send(sse.newEvent("Progress", "50%"));
            Thread.sleep(500);
            es.send(sse.newEvent("Progress", "75%"));
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    });
}
```

リスト 17 では、`SseEventSink` および `Sse` リソースがリソース・メソッドに注入されます。`Sse` インスタンスは新しいアウトバウンド・イベント・ビルダーを受け取り、進行状況イベントをクライアントに送信します。このメディア・タイプは、イベント・ストリームで排他的に使用される新しい `text/event-stream` タイプであることに注意してください。

サーバー送信イベントのブロードキャスト

複数のクライアントに同時にイベントをブロードキャストすることができます。それには、複数の `SseEventSink` インスタンスを `SseBroadcaster` に登録します。

リスト 18. 登録済みのすべてのサブスクライバーに対するブロードキャスト

```
@Path("/")
@Singleton
public class SseResource {
```

```
    @Context
    private Sse sse;

    private SseBroadcaster broadcaster;

    @PostConstruct
    public void initialise() {
       this.broadcaster = sse.newBroadcaster();
    }

    @GET
    @Path("subscribe")
    @Produces(MediaType.SERVER_SENT_EVENTS)
    public void subscribe(@Context SseEventSink eventSink) {
       eventSink.send(sse.newEvent("You are subscribed"));
       broadcaster.register(eventSink);
    }

    @POST
    @Path("broadcast")
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    public void broadcast(@FormParam("message") String message) {
       broadcaster.broadcast(sse.newEvent(message));
    }
}
```

リスト 18 に記載されている例は、サブスクリプション・リクエストを URI `/subscribe` 上で受け取ります。サブスクリプション・リクエストにより、当該サブスクライバーの `SseEventSink` インスタンスが作成されて、そのインスタンスが該当するリソースの `SseBroadcaster` に登録されます。URL `/broadcast` に送信された Web ページのフォームから、すべてのサブスクライバーにブロードキャストされるメッセージが取得されます。取得されたメッセージは、`broadcast()` メソッドによって処理されます。

JAX-RS 2.1 で行われたその他の更新

JAX-RS 2.1 が導入している細かな変更のうち、以下の変更は触れておく価値があります。

- `Resource` メソッドが戻り値の型として `CompletionStage<T>` をサポートするようになっています。
- Java API for JSON Processing (JSON-P) および Java API for JSON Binding (JSON-B) に対する完全なサポートが追加されています。
- `@Priority` は、エンティティー・プロバイダー (すなわち、`MessageBodyReader` と `MessageBodyWriter`) を含むすべてのプロバイダーでサポートされます。
- Java Concurrency Utilities API をサポートしていないすべての環境に対するデフォルトが削除されました。

JavaServer Faces 2.3

JavaServer Faces (JSF) 2.3 リリースで目的としているのは、コミュニティーから要望された機能に対処し、CDI と WebSocket の統合を強化することです。このリリースでは、かなりの数の問題 (最後に数えた時点では数百の問題) を解消し、小さいながらも解決されていない多数の問題の改善が試みられました。JSF は成熟したテクノロジーです。したがって、現在取り組んでいる問題は、これまで解決するのが難しかった問題や小さな問題です。これらの問題のうち、JSF 2.3 では Java `Date/Time` 型およびクラス・レベルの Bean 検証がサポートされるようになり、Java SE 8 の改善に

寄与しました。完全なリストについては、このリンク先のページから最終リリースの仕様をダウンロードすることをお勧めします。

JSF 2.3 でのサーバー・プッシュ

JSF 2.3 では、Servlet 4.0 でのサーバー・プッシュのサポートを統合しています。サーバー・プッシュの実装に必要なリソースを識別するのに理想的な JSF は、現在、`RenderResponsePhase` ライフサイクル・フェーズの間にそれらのリソースを識別するようにセットアップされています。

JSON Processing 1.1

ポイント・リリースである JSON-P は、最新の IEFT 標準を適用して更新されています。具体的には、JSON Pointer、JSON Patch、JSON Merge Patch がサポートされるようになっています。

`javax.json.streams` で新しく導入された `JsonCollectors` クラスにより、JSON のクエリーも単純化されています。

JSON-Pointer

JSON Pointer は、JSON ドキュメントに含まれる特定の値を識別するストリング表現を定義します。XML に含まれるフラグメントを識別するために使用される XPointer と同様に、JSON Pointer は JSON ドキュメント内の値を参照します。例えば、リスト 19 に示す JSON ドキュメントでは、`topics` 配列に含まれる 2 番目の要素を参照するには、`/topics/1` という JSON ポインター表現を使用します。

リスト 19. 配列からなる JSON オブジェクト

```
{
 "topics": [
   "Cognitive",
   "Cloud",
   "Data",
   "IoT",
   "Java" ]
}
```

エントリー API は `JsonPointer` インターフェースです。インスタンスを作成するには、`Json` クラスの静的ファクトリー・メソッド `createPointer()` を呼び出します。リスト 20 に記載するコード・スニペットでは、`JsonPointer` を作成して、topic 配列内の 2 番目の要素を参照しています。

リスト 20. 配列の要素を参照する JsonPointer

```
Json.createPointer("/topics/1").getValue(jsonTopicData);
```

`JsonPointer` API は、プロパティーを追加、置換、削除することによって JSON ドキュメントに手を加えることもできます。リスト 21 では、値「Big Data」を topics リストに追加しています。

リスト 21. JsonPointer を使用した、配列への値の追加

```
Json.createPointer("/topics/0")
    .add(jsonTopicData, Json.createValue("Big Data"));
```

JSON Patch

JSON Patch は、ターゲット JSON ドキュメントに適用する一連の処理を JSON Pointer 表記で表現します。JSON Patch では、追加、コピー、移動、削除、置換、テストといった処理を行うことができます。

この API へのゲートウェイとなる `JsonPatchBuilder`インターフェースを作成するには、`Json` クラスの静的メソッド `createPatchBuilder()` を呼び出します。JSON Pointer 表現を処理メソッドのいずれかに渡すと、その表現が JSON ドキュメントに適用されます。リスト 22 に、topics 配列の最初の要素を値「Spring 5」で置き換える例を示します。

リスト 22. 配列内の値を置き換える JsonPatchBuilder

```
Json.createPatchBuilder()
    .replace("/topics/0", "Spring 5")
    .build()
    .apply(jsonTopicData);
```

複数の処理を連鎖させることもできます。その場合、前のパッチ結果に次の処理を適用するというプロセスが順に行われます。

JSON Merge Patch

JSON Merge Patch は、ターゲット JSON ドキュメントに対して行う一連の変更を記述する JSON ドキュメントです。表 1 に、使用できる 3 つの処理を記載します。

表 1. マージ・パッチ処理の選択

処理	ターゲット	パッチ	結果
置換	{"color":"blue"}	{"color":"red"}	{"color":"red"}
追加	{"color":"blue"}	{"color":"red"}	{"color": null}
削除	{"color":"red"}	{"color":"blue","color":"red"}	{}

`Json` クラスの静的メソッド `createMergePatch()` によって、`JsonMergePatch` 型のインスタンスが作成されます。作成された `JsonMergePatch` インスタンスにパッチを渡し、このインスタンスの `apply()` メソッドをターゲット JSON に渡すことで、パッチが適用されます。リスト 23 に、表 1 に記載された置換処理を実行する方法を示します。

リスト 23. JSON Merge Patch を使用した値の置換

```
Json.createMergePatch(Json.createValue("{\"colour\":\"blue\"}"))
    .apply(Json.createValue("{\"colour\":\"red\"}"));
```

JsonCollectors

Java 8 で `JsonCollectors` クラスが `javax.json.streams` パッケージに導入されたことにより、JSON のクエリーが大幅に単純化されています。リスト 24 に、`topics` 配列を文字 c でフィルタリングし、その結果を `JsonArray` に収集する方法を示します。

リスト 24. JsonCollectors を使用したフィルタリングと JsonArray への収集

```
JsonArray topics = jsonObject.getJsonArray("topics")
    .stream()
    .filter(jv -> ((JsonString) jv).getString().startsWith("C"))
    .collect(JsonCollectors.toJsonArray());
```

まとめ

Java EE はクラウドを対象として新しく位置付けられています。2 回にわたって予定されているリリースの前半は、クラウド対応の Java EE という目標を推進するテクノロジーを特徴としています。Java EE 8 がリリースされると同時に、Java EE 9 への取り組みが開始されることになります。現在目標とされているのは、1 年以内に Java EE 9 をリリースすることです。

Java EE 9 へのロードマップですでに予定されている Java EE Security API の機能拡張では、Java EE Security 1.0 には統合できなかった機能が追加される予定となっています。また、開発者はクラウド内でのエンタープライズ開発のサポートが強化されることも期待しているかもしれません。レジリエンスとスケーラビリティーを促進するためのブループリントとして、リアクティブ宣言に従った Java EE テクノロジーがさらに追加されるためです。

私が Java EE 9 に望む機能の中には、マイクロサービス対応のテクノロジーの強化も含まれています。マイクロサービスをサポートすることになる API には、Configuration API 1.0 (JSR 382) と Health Check の 2 つがあります。いずれも Java EE 8 では断念されましたが、次回のアップデートに向けてこの 2 つの API の追加が検討されているところです。

また、嬉しいことに、最近 Java EE が Eclipse で採用されるという発表がありました。これにより、このオープンソース・ファウンデーションは、ベンダーや開発者のコミュニティーの関与をさらに受け入れるようになるはずです。Java EE リリースの加速化と、この強力なエンタープライズ仕様に対して現在行われている機能強化に期待しましょう。

関連トピック： JSON Processing 1.1 API ドキュメント JSON Binding のダウンロードとドキュメント JAX-RS 2.1 の新機能 JSF 2.3 リリース・ノート JSF の機能の詳細な資料 Bean Validation 2.0 仕様と参照実装 Hibernate Validator 6.0 の参照実装 What's new in Java EE Security API 1.0? (Arjan Tijms) 公式 Java EE GitHub

著者について

Alex Theedom

May 2017