今まで知らなかった 5 つの事項: MicroProfile 1.3

Eclipse MicroProfile 1.3 に新しく導入された 5 つの API を理解する

Alex Theedom

2018年 9月 27日

つい最近、Eclipse MicroProfile が Java クラウド・ネイティブ・マイクロサービスの開発を目的とした新しい 5 つの API をリリースしました。この記事では、MicroProfile 1.3 に導入された新機能の注目すべき点を説明します。サンプル・コードを参考に、早速これらの API を使えるようになってください。

このシリーズについて

皆さんは自分が Java プログラミングについて知っていると思うかもしれません。しかし実際には、ほとんどの開発者は Java プラットフォームの表面的な部分しか扱っておらず、当面の作業を完了するために十分なことしか学んでいません。この連載では、Java テクノロジーを徹底的に追及する著者たちが Java プラットフォームのコア機能を深く掘り下げ、非常に厄介な Java プログラミングの難題の解決にも役立つ、アドバイスとコツを紹介します。

2016 年に設立された Eclipse MicroProfile は、Java でマイクロサービス・アーキテクチャー向けの新しいツールおよび仕様を迅速に開発することを目標としているコミュニティーです。新しいリリースごとに、Java クラウド・ネイティブ・マイクロサービスの開発に大いに役立つテクノロジーが追加されています。MicroProfile の最新バージョンは、3 回目のリリースとなるバージョン 1.3 です。

この記事では、MicroProfile 1.3 で新しく導入された以下の 5 つの API の注目すべき点を紹介します。

- REST Client 1.0
- Metrics 1.1
- OpenAPI 1.0
- OpenTracing 1.0
- Config 1.2

MicroProfile の API はすべて、以下の Java EE 7 API をベースに作成されています。

- CDI 1.2。すべての API で使用されている、Java プラットフォームの中核です。
- JAX-RS 2.1。RESTful API を作成するためのクライアントおよびサーバー機能を提供します。
- JSON-P 1.0。JSON を処理する機能を提供します。

- Commons Annotation 1.2。共通のセマンティック概念に対応するアノテーションを提供します。

バージョン 1.2 でリリースされた Java MicroProfile API (Fault Tolerance 1.0、Health Check 1.0、および JWT Authentication 1.0) については、このリンク先の MicroProfile プロジェクト・ページを参照してください。

MicroProfile 1.3 をインストールする

MicroProfile 1.3 の最小要件として、Java SE 8 が実行されている必要があります。リスト 1 に、pom.xml に追加される MicroProfile 1.3 の依存関係が示されています。

リスト 1. Maven による MicroProfile 1.3 を対象とした調整

```
<dependency>
   <groupId>org.eclipse.microprofile</groupId>
   <artifactId>microprofile</artifactId>
   <version>1.3</version>
   <type>pom</type>
   <scope>provided</scope>
</dependency>
```

IBM Cloud Developer のツールまたは IBM Cloud の UI を使用して IBM Liberty 対応の MicroProfile プロジェクトを生成することもできます。

コードを入手する

MicroProfile プロジェクトの歴史

Oracle による Java EE 開発が進行していないことを受けて、2016 年、MicroProfile が誕生しました。当時、Java EE のリリース・サイクルが遅かったことが、マイクロサービスとクラウド・ネイティブ・ソリューションのサポートを統合する障害となっていました。そこで、ベンダーとユーザー・グループからなるグループがこの需要に対処すべく Java MicroProfile を提案したところ、瞬く間に提案が本格化しました。

その後、他の多くのベンダーが MicroProfile コミュニティーに参加し、続いて MicroProfile が Eclipse Foundation に移されて、Eclipse プロジェクトとして保守されるようになりました。

Java MicroProfile の仕様を規定するのはコミュニティーの取り組みによってであり、現在、新しいコントリビューターの参加が歓迎されています。このプロジェクトに参加するには、手始めとして、MicroProfile コミュニティーの Google グループ内で行われているディスカッションに参加することができます。開発者は GitHub 上でホストされている MicroProfile プロジェクトにコードを貢献することもできます。

以降のセクションでは、MicroProfile 1.3 について知らなかった 5 つの事項、つまり、このリリースで新たに導入された 5 つの API について明らかにします。

1. Rest Client 1.0

JAR-RS 2.0 をベースとした Rest Client API では、型セーフな方法で HTTP を介して RESTful エンドポイントを呼び出すことができます。リスト 2 に示す `BookClientService` インターフェースは、ISBN を基準に書籍のレコードを作成、取得できるリモート・インターフェースを表します。

リスト 2. BookClientService インターフェースを作成する

```
@Path("/books")
public interface BookClientService {
    @GET
    @Path("/{isbn}")
    Book getBook(@PathParam("isbn") String isbn);


    @POST
    String addBook(Book book);
}
```

リスト 3 では、`BookClientService` クラスを `RestClientBuilder` のビルダー・メソッドに渡した後、インターフェースのメソッドを呼び出して書籍レコードを作成し、取得します。

リスト 3. BookClientService インターフェースを使用する

```
URI bookSrvUri = new URI("http://localhost:8081/bookservice");
BookClientService bookSrv = RestClientBuilder.newBuilder()
            .baseUri(bookSrvUri)
            .build(BookClientService.class);

Book book = new Book ("Fun with Microprofile","Alex Theedom");
String isbn = bookSrv.addBook(book);
Book newBook = bookSrv.getBook(isbn);
```

2. Metrics 1.1

Metrics API は、開発者が移植可能なマイクロサービスのインスツルメンテーションを行う方法になります。この API は、多くの開発者がすでに使い慣れている Dropwizard Metrics API と類似したものになるよう設計されています。Metrics API を使用すると、サード・パーティー・ライブラリーへの依存性がなくなります。

Metrics API は極めて使いやすくなっていて、良く使われている Prometheus 形式を十分理解している開発者であれば、Prometheus をセットアップするだけで、メトリックを取得してそのサービスをモニタリングできます。

Health Check API 1.2 (MicroProfile 1.2 でリリースされた API) と差別化するために、Metrics API ではサービスの正常性をバイナリーの yes/no で表現するだけではなく、サービスのパフォーマンスに関する詳細なメタデータも提供するようになっています。

クラウドおよびマイクロサービス・アプリケーション内でメトリックを有効化する

MicroProfile アプリケーション内でメトリックを有効化するには、メトリックを作成してアプリケーション・レジストリーに登録する必要があります。それには、以下のいずれかの方法を使用できます。

- MetricRegistry を使用する。この方法を使用する場合は、メトリックを明示的に作成して登録する必要があります。
- メトリックを注入してメトリック・アノテーションを使用する。この場合、メトリックは CDI コンテナーによって自動的にインスタンス化されて、MetricRegistry アプリケーションに登録されます。

メトリック値を表示する

メトリックを表示するには、サービス内で URI /metrics エンドポイントを呼び出します (リスト 4 を参照)。`localhost` 上にあるメトリック・エンドポイントは、ポート 9091 上で稼働します。

リスト 4. メトリック・エンドポイント

```
https://localhost:9091/metrics
```

MetricRegistry を使用してメトリックを有効化する

MetricRegistry アプリケーションの目的は、すべてのメトリックとそれらのメトリックのメタデータを保管することです。上記にリストアップしたいずれの方法でも、メトリックとメタデータを登録して取得することができます。2 つの方法のそれぞれについて、詳しく見ていきましょう。

手作業でメトリックを作成して登録する

`MetricRegistry` のインスタンスが使用するクラスに注入されて (リスト 5 を参照)、これらのメトリックを登録および取得するために各種のメソッドが呼び出されます。

リスト 5. MetricRegistry を注入する

```
@Inject
private MetricRegistry registry;
```

`MetricRegistry` クラス上には、リスト 6 に示されている、メトリックを取得するメソッドと登録するメソッドが定義されています。このコード・スニペットに記載されているのは、カウント型のメトリックを作成して登録する方法です。カウント型のメトリックには、カウンター、ゲージ、メートル、ヒストグラム、タイマーの 4 つのタイプがあります。

リスト 6. メタデータ・カウンター・メトリックを新規に作成する

```
Metadata counter = new Metadata(
    "hitsCounter",                          // name
    "Hits Count",                           // display name
    "Number of hits",                       // description
    MetricType.COUNTER,                     // type
    MetricUnits.NONE);                      // units
```

メトリックは、プログラムによって MetricRegistry に登録されます (リスト 7 を参照)。

リスト 7. カウンター・メトリックを登録する

```
Counter hitsCounter = registry.counter(counter);
```

メトリックが登録されると、メトリック値を記録するために使用できるようになります (リスト 8 を参照)

リスト 8. hitsCounter メトリックをインクリメントする

```
@GET
@Path("/messages")
public String getMessage() {
    hitsCounter.inc();
    return "Hello world";
}
```

記録されたメトリック値は MetricRegistry 内に保管され、`/metric` URI の呼び出しによって取得できるようになります。上記で作成したヒット・カウンター・メトリック値を取得するには、メトリック・エンドポイントの URI として `https://localhost:9091/metrics/application/` を呼び出します。

リスト 9 に記載する出力が、REST エンドポイントから生成された結果です。

リスト 9. REST エンドポイントによって生成されたメトリック統計情報

```
https://localhost:9091/metrics/application/hitsCounter
# TYPE application:hits_count counter
# HELP application:hits_count Number of hits
application:stats_hits 499
```

これらのメトリックは Prometheus 形式で表示されます。統計情報を JSON 形式にしたい場合、呼び出すエンドポイントは同じですが、`HTTP Accept` ヘッダーを `application/json` として構成します (リスト 10 を参照)。

リスト 10. JSON 形式で統計情報を取得する

```
https://localhost:9091/metrics/application/statsHits
{"hitsCount":499}
```

メトリックを注入してアノテーションを使用する

アプリケーション内でメトリックを有効化するには、CDI インジェクションのほうが簡単で自然な方法です。メトリックを作成して MetricRegistry に登録することに変わりはありませんが、この場合、メタデータ・インスタンスを作成するのではなく、メトリック構成データを `@Metric` アノテーションのパラメーターとして渡します (リスト 11 を参照)。

リスト 11. @Metric を使用してメトリックを構成する

```
@Inject
@Metric(name="hitsCounter", displayName="Hits Count", description="Number of hits", absolute=true)
Counter hitsCounter;
```

`@Metric` アノテーションにより、サーバーは MetricRegistry 内にカウンターを作成して登録し、このカウンターをアプリケーションで使用するように指定します。

3. OpenAPI 1.0

MicroProfile 1.3 で新たに導入された OpenAPI 1.0 は、SOAP Web サービスの WSDL 契約と似たような契約を RESTful サービスを対象にして提供します。OpenAPI 仕様に基づく OpenAPI 1.0 には、開発者が JAX-RS アプリケーションから OpenAPI v3 ドキュメントを生成するために使用できる一連の Java インターフェース・プログラミング・モデルが揃っています。

OpenAPI は、JAX-RS 2.0 アプリケーションから有効な OpenAPI ドキュメントを生成し、JAX-RS アノテーションのすべて (`@Path` および `@Consumes/@Produces` アノテーションを含む) に加え、JAX-RS 処理の入力または出力として使用される POJO も処理します。

OpenAPI を起動して構成する

OpenAPI 1.0 の使用を開始するには、既存の JAX-RS アプリケーションを MicroProfile サーバー (Open Liberty など) にデプロイして、OpenAPI URI `/openapi` から出力をチェックアウトすればよいだけです。

`/openapi` 呼び出しによる出力を確認した後、より適切なドキュメンテーションになるよう、RESTful リソースを構成することができます。その方法は、RESTful API の各種要素にアノテーションを付けるというものです。

表 1 に、OpenAPI 1.0 仕様の主要なアノテーションを記載します。OpenAPI アノテーションのすべてを網羅したリストは、org.eclipse.microprofile.openapi.annotations パッケージに含まれています。

表 1. 主要な OpenAPI アノテーション

アノテーション	説明
@Operation	処理、または通常は特定のパスに対する HTTP メソッドを記述します。
@APIResponse	API 処理による単一の応答を記述します。
@RequestBody	単一の応答本文を記述します。
@Content	特定のメディア・タイプのスキーマを指定し、サンプルを提供します。
@Schema	入力および出力データの型を定義できます。
@Server	複数のサーバー定義のコンテナーです。
@ServerVariable	サーバー URL のテンプレートでの代入に使用するサーバー変数を表します。
@OpenAPIDefinition	OpenAPI 定義の一般的なメタデータです。

OpenAPI アノテーションを使用する

OpenAPI アノテーションは JAX-RS リソース定義内で機能をマーキングして、そのリソースに関する情報を提供します。リスト 12 では、`@Operation` アノテーションが付けられた `findBookByISBN` に、その関数に関する情報が渡されます。

リスト 12. OpenAPI アノテーションを使用して REST リソースをマーキングする

```
@GET
@Path("/findByISBN")
@Operation(
    summary = "Finds a book by its ISBN",
    description = "Only one ISBN should be provided")
public Response findBookByISBN(...) { ... }
```

リスト 13 に、このアノテーションによる出力を記載します。

リスト 13. REST ドキュメンテーションの YAML 出力

```
/books/findBookByISBN:
  get:
    summary: Finds a book by its ISBN
    description: Only one ISBN should be provided
    operationId: findBookByISBN
```

OpenAPI URI `/openapi` 呼び出しのデフォルトの出力形式は YAML です。`HTTP Accept` ヘッダーを `application/json` に設定してこのエンドポイントを呼び出すと、レスポンスは JSON 形式になります。

4. OpenTracing 1.0

OpenTracing API は OpenTracing Project の実装です。この API を使用すると、サービスの境界を越えてリクエストをトレーシングすることができます。リクエストが多数のサービスを経由する一般的なマイクロサービス環境では、この分散トレーシング機能が特に重要となります。

分散トレーシングを可能にするために、各サービスでは、分散トレース・レコードの保管場所として指定されたサービスに対する相関 ID をログに記録します。この相関 ID を使用することで、サービスはリクエストに関連するトレースを表示することができます。

OpenTracing はアプリケーション・コードによる明示的なインスツルメンテーションを使用してもしなくても動作できます。両方の動作モデルを見ていきましょう。

コードによるインスツルメンテーションを使用しない OpenTracing

この動作モデルでは、JAX-RS アプリケーションに分散トレーシング用のコードを追加しなくても、アプリケーションがトレースの記録に参加できます。この方法を使用すると、開発者はアプリケーションのデプロイ先となるトレーニング環境についての情報さえ把握する必要がなくなります。

コードによるインスツルメンテーションを使用した OpenTracing

場合によっては、アクティブにトレーシングとやり取りする必要があることもあります。その場合は、`@Traced` アノテーションを使用してトレーシング対象のクラスまたはメソッドを指定します。明示的なコードによるインスツルメンテーションとコードを使用しないインスツルメンテーションの両方を組み合わせて使用することができます。シナリオによっては、この 2 つの方法を統合することによって、使用ケースに合わせた適切なソリューションが作成されます。

@Traced アノテーションを使用する

クラスに適用された `@Traced` アノテーションは、そのクラスに含まれるすべてのメソッドに適用されます。メソッドのサブセットだけをトレーシングするには、対象とするメソッドだけにアノテーションを追加します。アノテーションはメソッド先頭のスパンで始まり、メソッド末尾のスパンで終了します。リスト 14 では、クラス・レベルのアノテーション内でトレーシング対象の演算に `Book Operation` という名前を指定しています。`getBookByIsbn()` メソッドはトレーシングから除外されます。それは、アノテーションに false の値を含めて指定しているためです。

リスト 14. @Traced の使用例

```
@Path("/books")
@Traced(operationName = "Book Operation")
public class BookResource {

    @GET
    public Response getBooks() {
      // implementation code removed for brevity
    }

    @Traced(value = false)
    @GET
    @Path("/{isbm}")
    public Response getBookByIsbn(String @Param("isbn") isbn {
      // implementation code removed for brevity
    }

}
```

5. Config 1.2

動作環境が変わるたびにアプリケーションを再パッケージ化する必要を失くすために、これまでさまざまな手法が開発されてきました。マイクロサービスは、さまざまな環境に移植して稼働するように意図されているため、動作環境が変わってもアプリケーションを再パッケージ化しなくて済むようにすることは特に重要となります。

Config 1.2.1 API は各種の個別の `ConfigSource` から構成を集め、それらを集約して単一のビューとしてユーザーに表示します。

構成は `String/String` というキーと値の形式で保管されます。構成キーには、Java パッケージ内の名前空間と同様のドットで区切った表記を使用することもできます。リスト 15 に、Config API 対応の構成ファイルの例を記載します。

リスト 15. 構成ファイルの例

```
com.readlearncode.bookservice.url = http://readlearncode.com/api/books
com.readlearncode.bookservice.port = 9091
```

構成オブジェクトを使用する

`@Inject Config` を使用すると、インジェクションによって自動的に構成オブジェクトのインスタンスを取得できます (リスト 16 を参照)。

リスト 16. 構成に対してインジェクションを使用する

```
public class AppConfiguration {
    public void getConfiguration() {
        Config config = ConfigProvider.getConfig();
        String bookServiceUrl = config.getValue("com.readlearncode.bookservice.url", String.class);
    }
}
```

また、`ConfigProvider` を使用して、プログラムによって取得することもできます (リスト 17 を参照)。

リスト 17. 構成に対して CDI を使用する

```
@ApplicationScoped
public class InjectedConfigUsageSample {

    @Inject
    private Config config;

    @Inject
    @ConfigProperty(name="com.readlearncode.bookservice.url")
    private String bookServiceUrl ;

}
```

まとめ

MicroProfile は、マイクロサービス・ベースのシステムを開発するために不可欠となっている広範な機能を開発者に提供します。短期リリース・サイクルにより、この仕様には定期的に新機能が追加されていくはずです。実際、リリース以来、この仕様に含まれる機能は急激に増加し続けています。いずれの MicroProfile API も業界で広く受け入れられているベスト・プラクティスに従っているため、これらの API は Java ベースのマイクロサービスをアジャイルかつ堅牢にするためにも役立ちます。

関連トピック

- MicroProfile Web サイト
- Eclipse Microprofile プロジェクト・ページ
- MicroProfile の概要
- 今まで知らなかった 5 つの事項... Java 10
- AdoptOpenJDK: ビルド済み OpenJDK バイナリー