

【MicroProfile開発ガイド】 MicroProfile Rest Client

2018/12

日本アイ・ビー・エム システムズ・エンジニアリング株式会社



Disclaimer

- この資料は日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の正式なレビューを受けておりません。
- 当資料は、資料内で説明されている製品の仕様を保証するものではありません。
- 資料の内容には正確を期するよう注意しておりますが、この資料の内容は2018年12月現在の情報であり、製品の新しいリリース、PTFなどによって動作、仕様が変わる可能性があるのでご注意ください。
- 今後国内で提供されるリリース情報は、対応する発表レターなどでご確認ください。
- IBM、IBMロゴおよびibm.comは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。現時点でのIBMの商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。
- 当資料をコピー等で複製することは、日本アイ・ビー・エム株式会社ならびに日本アイ・ビー・エム システムズ・エンジニアリング株式会社の承諾なしではできません。
- 当資料に記載された製品名または会社名はそれぞれの各社の商標または登録商標です。
- JavaおよびすべてのJava関連の商標およびロゴはOracleやその関連会社の米国およびその他の国における商標または登録商標です。
- Microsoft, WindowsおよびWindowsロゴは、Microsoft Corporationの米国およびその他の国における商標です。
- Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。
- UNIXはThe Open Groupの米国およびその他の国における登録商標です。

目次

- MicroProfile Rest Client概要
 - MicroProfile Rest Clientとは
 - 利用シナリオ
 - インターフェースによるRESTクライアントの定義
 - 例外Mapperの利用
 - フィーチャーの有効化
- MicroProfile Rest Clientの使用方法
 - 概要
 - インタフェースの作成
 - 例外Mapperの追加
 - クライアントのインスタンス化と実行
 - CDI+MicroProfile Config
 - RestClientBuilder API
- MicroProfile Rest Clientの各種機能
 - プロバイダー
 - デフォルト例外Mapper
 - 非同期的なサービス間の連携
- 参考リンク

MicroProfile Rest Client概要

MicroProfile Rest Clientとは

MicroProfile Rest ClientはHTTPを介してRESTful サービスを呼び出すためのタイプ・セーフなアプローチを提供します。

WAS LibertyではMicroProfile Rest Client 1.1 仕様に準拠したmpRestClient-1.1 フィーチャーを提供しており、JAX-RSクライアントの機能を拡張し、クライアント・インターフェースを定義してサービス独自のRESTクライアントを作成することができます。

MicroProfile の機能

JAX-RS	CDI	JSON-P	JSON-B	
REST APIおよびクライアント開発	依存性注入とライフサイクル管理	JSONデータの生成と解析	JSONデータとJavaオブジェクトのマッピング	
Config	Fault Tolerance	Health Check	Health Metrics	JWT Propagation
ポータビリティを向上させる構成の外部化	障害に対処するための堅牢な動作	サービスの稼働確認の共通フォーマット	サービス状況をモニタリングするためのエンドポイント	インターオペラブルな認証とロールベースのアクセス制御
Open Tracing	Open API	Rest Client		
複数サービスに跨る分散トレース	Open API仕様 (Swagger) 対応	タイプセーフなREST API呼び出し		

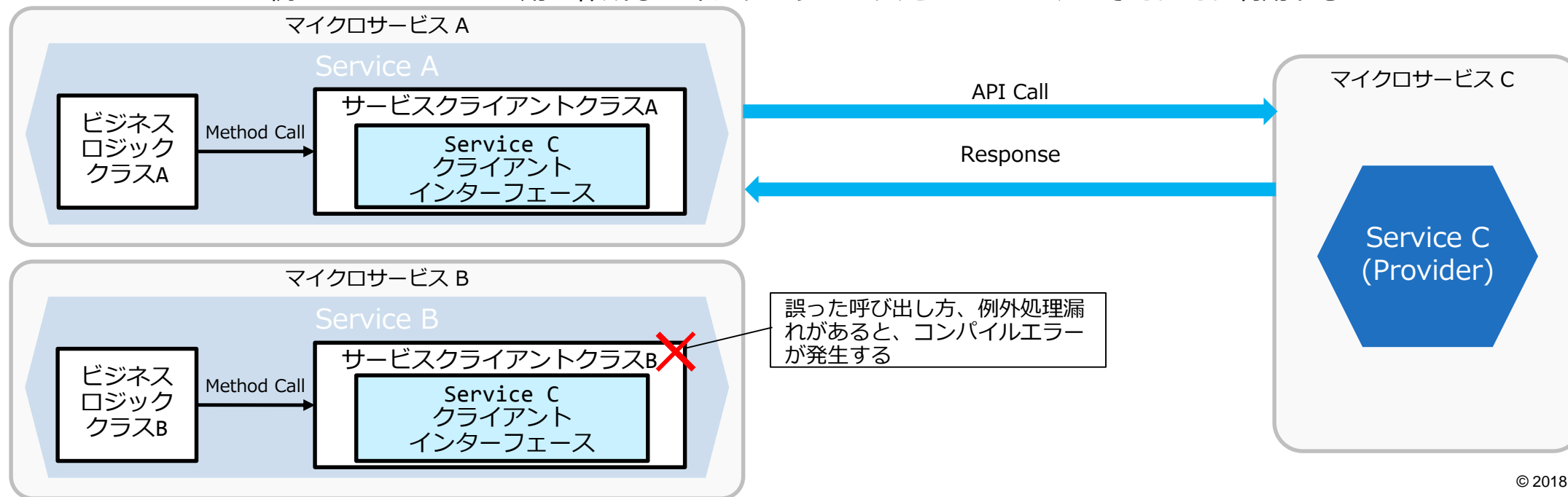
利用シナリオ：インターフェースによるRESTクライアントの定義

マイクロサービスでは、サービスをRESTなどで連携させて一つのアプリケーションを開発するため、サービス毎に用意されたRESTful APIを正確に呼び出す必要があります。

MicroProfile Rest Clientの機能を利用すると、事前にRESTクライアントでAPIの呼び出し方、例外処理をインターフェースで定義することができ、同じインターフェースを複数のサービスで利用できます。

ユーザーはインターフェースに指定されたメソッド、引数、例外に従って、タイプセーフにサービスを呼び出すことができ、誤ったリクエストを実装した場合や例外処理の実装を忘れた際も、実行前にコンパイルエラーとして検出することができます。

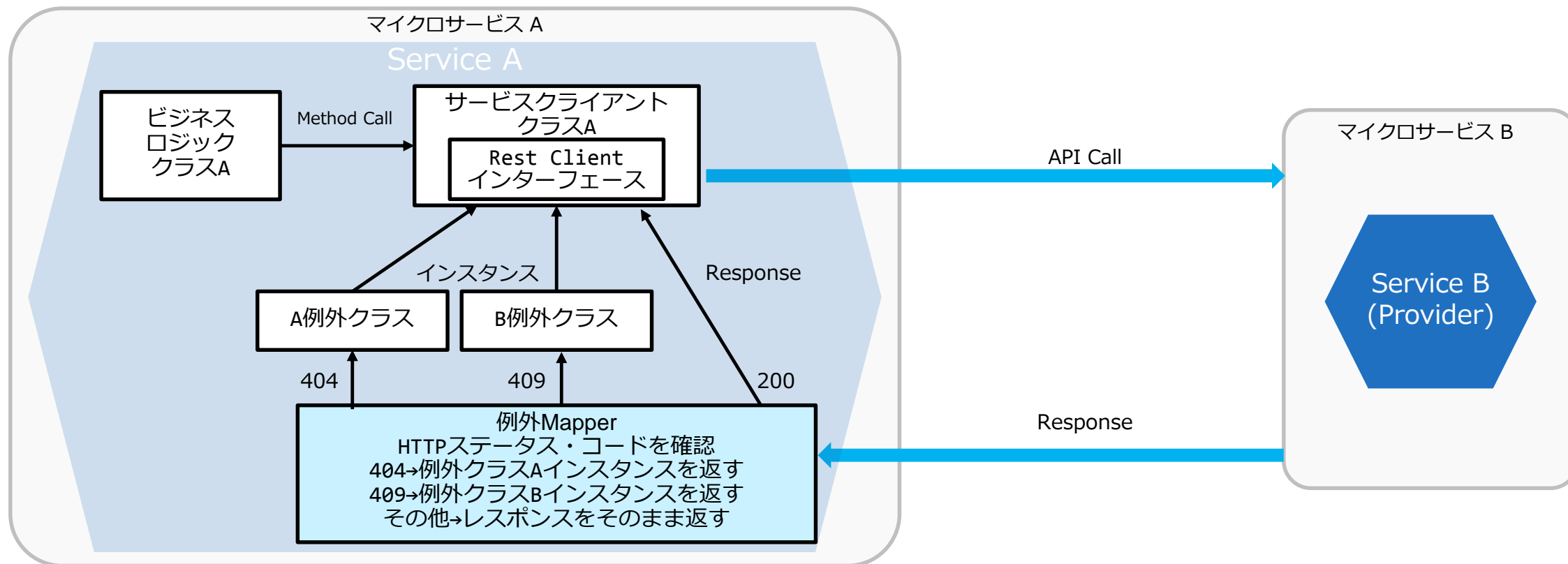
例：Service CのAPI用に作成したインターフェースをService A、Bでそれぞれ利用する



利用シナリオ：例外Mapperの利用

MicroProfileの機能として、独自の例外Mapperを利用することができます。例外Mapperによって、事前にレスポンス（HTTPステータス・コード等）とそれに対応する独自の例外を指定することができ、柔軟に例外処理を作成することができます。

例：例外Mapperでレスポンスのステータスコードを確認し、独自例外インスタンスを返却する



(補足) JAX-RS クライアントとの比較

前述の通り、MicroProfile RestClientはJAX-RS 2.0 Client APIの機能を拡張したものです。
サービス独自のクライアント・インターフェースや例外を作成することができるようになります。

	JAX-RS クライアントAPI	MicroProfile Rest Client
クライアント インターフェース	JAX-RS標準クライアントAPIを常に使用して、 サービスを呼び出す	サービス独自のインターフェースを作成し、それ に従ってタイプ・セーフにサービスを呼び出す
データの受け渡し	JAX-RS標準クライアントAPIを常に使用して、 URLや引数を渡す	サービス独自のインターフェースを利用し、URLや 引数を受け渡す (渡したデータはアノテーションに従って処理さ れる)
例外	JAX-RSデフォルトの例外のみ	独自の例外Mapperにより、独自の例外で処理する ことができる

MicroProfile Rest Clientでは独自のクライアント・インターフェースにより、サービス呼び出しコードの冗長性の軽減が可能です。また、クライアント・インターフェースは複数のサービスで共通に利用できます。

フィーチャーの有効化

WAS Liberty上でMicroProfile Rest Clientの機能を有効化するためには、該当サーバーのserver.xmlにmpRestClient-1.1フィーチャーを設定します。

当該フィーチャーが含まれるMicroProfile-1.4、 MicroProfile-2.0を設定することも可能です。

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>mpRestClient-1.1</feature>
    </featureManager>
</server>
```

mpRestClient-1.1フィーチャーを有効化することにより、以下のフィーチャーも暗黙的ロードにより自動的に有効化されます。

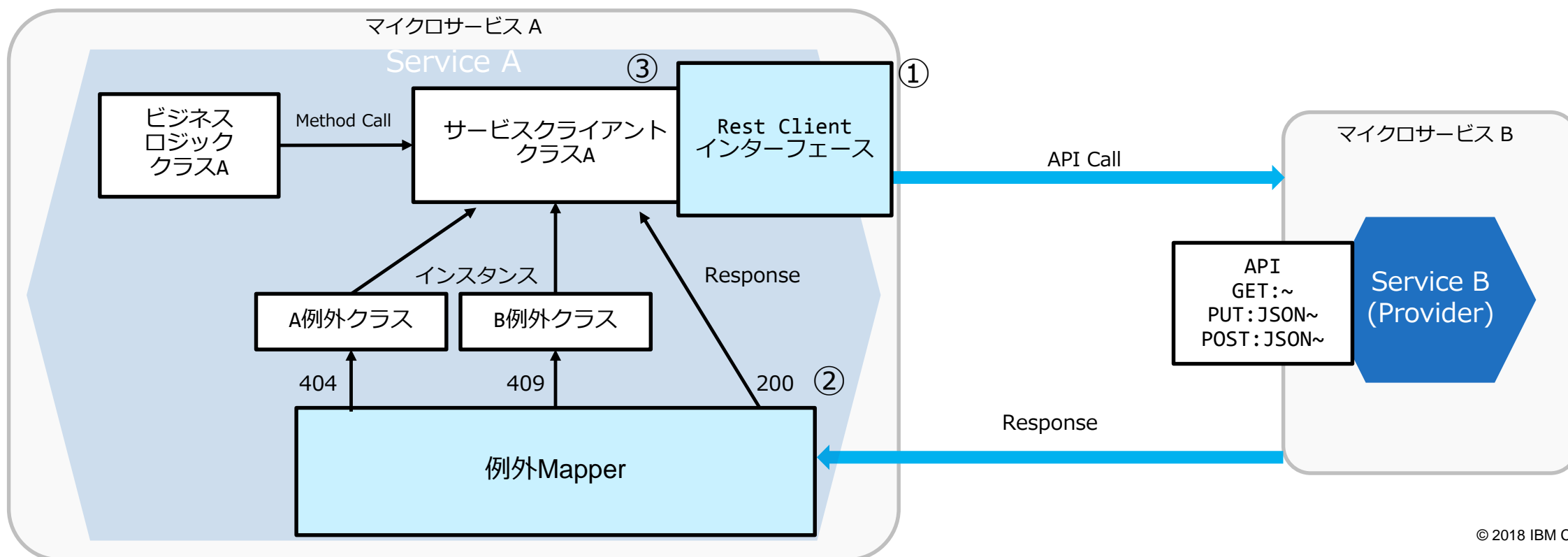
- jaxrsClient-2.0
- json-1.0
- jsonp-1.0
- servlet-3.1

MicroProfile Rest Clientの使用方法

使用方法概要

呼び出し対象のAPIに用意されているRESTful APIのメソッドや扱うエンティティに合わせて、次の手順でクライアントを作成します。

- ① インターフェースの作成
- ② 例外Mapperの追加（オプション）
- ③ クライアントのインスタンス化と実行



インターフェースの作成

まず、独自のクライアント・インターフェースを作成します。インターフェースのメソッドは、エンドポイントのRESTful APIに合わせてHTTPメソッドや受け渡すデータの型などを定義します。使用するアノテーション (@Path、@GET、@Produces等) は、JAX-RSのサーバー用APIで使用するアノテーションと同じです。

コード例

```
....
@Path("/user")
public interface HRServiceClient {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/{userId}")
    public User getUser(@PathParam("userId") int id) throws UnknownUrlException;

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public User createUser(User user) throws ConfrictDataException;

    ....
}
```

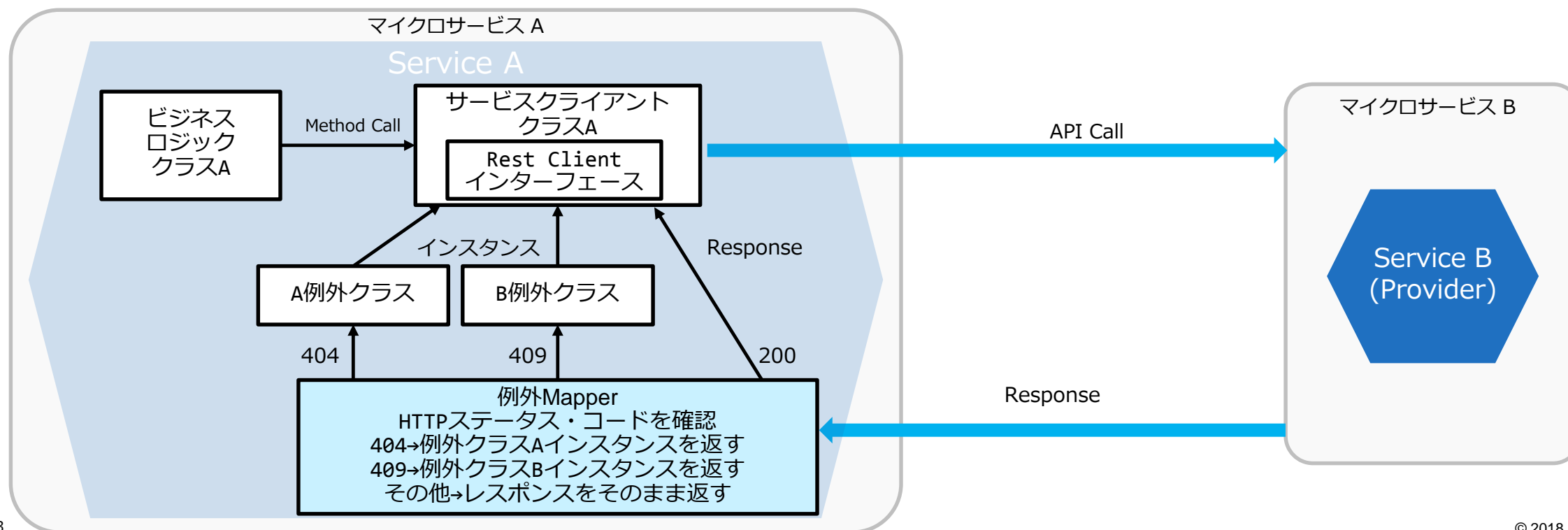
上記の例の場合、getUserメソッドが実行された時にHRServiceClientインスタンスは <baseUrl>/user/{userId} にGETリクエストを送信し、レスポンスをJSON型として処理します。また、各メソッドについてスローする例外を明示的に定義することができます。

例外Mapperの追加

MicroProfile Rest Clientでは、リモート・サービスからのレスポンスに対して、要件に合わせた独自例外の設定ができます。ResponseExceptionMapperのインターフェースを実装した独自の例外Mapperクラスで、レスポンスと例外クラスの対応づけを行います。

また、独自の例外Mapperを作成しない場合に利用されるデフォルトの例外Mapper（P23参照）が提供されています。

例：例外Mapperでレスポンスのステータスコードを確認し、独自例外インスタンスを返却する



例外Mapperの追加：独自例外Mapperクラスの実装

ResponseExceptionHandlerインターフェースを利用し、要件に合わせて独自例外Mapperクラスを作成します。以下のメソッドによって、レスポンスに対する例外を定義します。

■ handlesメソッド

- レスポンスのステータスコードやヘッダーを確認し、例外Mapperクラスで扱うかを決めます。
- 指定した条件に合わないレスポンスは別のMapperクラスに渡されるか、そのまま呼び出し元に返されます。

■ toThrowableメソッド

- handlesメソッドの条件に合うレスポンスのコンテンツに基づいて、用意した例外インスタンスを返却します。

この他に、例外Mapperクラスの優先度を定めるgetPriorityメソッドがあります。

コード例（ステータスコードに応じて返却する例外インスタンスを振り分ける）

```
@Provider
public class HRServiceResponseExceptionHandler
    implements ResponseExceptionHandler<HRServiceResponseException> {

    @Override
    public boolean handles(int status,
                           MultivaluedMap<String, Object> headers) {

        return status == 404 || status == 409;
    }

    @Override
    public HRServiceResponseException toThrowable(Response response) {

        switch(response.getStatus()) {
            case 404: return new UnknownUrlException();
            case 409: return new ConflictDataException();
        }

        return null;
    }
}
```

クライアントのインスタンス化と実行

作成したインターフェースを実装し、クライアントをインスタンス化・実行する方法には、以下 2 通りの方法があります。

■ CDI + MicroProfile Config

- アノテーションを追加し、CDI管理BeanとしてRestClientインスタンスを作成
- baseURLはMicroProfile Configを利用して指定
- シンプルなコードでサービス呼出を実装可能

■ RestClientBuilder API

- RestClientBuilder APIを利用し、RestClientインスタンスを作成
- CDIを利用するよりもコードは冗長になるものの、CDIが使えない場面でも利用可能

クライアントのインスタンス化と実行 CDI+MicroProfile Config (1/3)

MicroProfile Rest Clientは、CDI と MicroProfile Config を使用して実装することができます。アノテーションによる設定で、クライアント作成時のコードの冗長性が軽減されます。

まず、アノテーションを用いてインタフェースをCDI管理Beanに登録します。また、必要に応じて独自例外Mapperなどの機能を追加するプロバイダー・クラス（P22参照）に登録します。

■ MicroProfile Rest Clientアノテーション

アノテーション	クラス	
@ResisterRestClient	org.eclipse.microprofile.rest.client.inject.RegisterRestClient	CDI管理BeanにRestClientとして登録
@ResisterProvider()	org.eclipse.microprofile.rest.client.annotation.RegisterProvider	プロバイダー・クラスの登録

コード例

```
@Path("/user")・・・APIパスを指定
@Consumes("application/json")・・・エンティティを指定
@Dependent・・・スコープに応じたアノテーションを記述（@ApplicationScoped、@RequestScoped等）
@registerRestClient・・・CDI管理BeanにRestClientとして登録
@registerProvider(HRServiceExceptionMapper.class)・・・プロバイダー・クラスの登録
public interface HRServiceClient {
    ...
}
```


クライアントのインスタンス化と実行 CDI+MicroProfile Config (2/3)

CDIで注入するクラスに対してRestClientアノテーションを付与することにより、クライアントをインスタンス化します。クライアントはインターフェースで定義したメソッドを呼び出すことで利用できます。また、各メソッドに対して、必要な引数、例外処理を追加します。

■ MicroProfile Rest Clientアノテーション

アノテーション	クラス	
@RestClient	org.eclipse.microprofile.rest.client.inject.RestClient	Injectで注入したクラスをRestClientとして利用する

コード例

```
....
public class HRServiceManager {

    @Inject・・・CDI管理ビーンを注入
    @RestClient・・・注入したクラスをRestClientとしてインスタンス化
    private HRServiceClient hrServiceClient;

    public User get(int userId) {
        try {
            return hrServiceClient.getUser(userId);
        } catch (UnknownUrlException e) {
            ....
        }
    }
}
```

クライアントのインスタンス化と実行 CDI+MicroProfile Config (3/3)

- CDIを利用してRestClientを実装する場合、リモートエンドポイントのurl(base URL)をMicroProfile Configを利用して設定します。

```
com.exampleclient.demo.user.client.UserClient/mp-rest/url=http://localhost:8080/api/v1/
```

上記のようにbase URLを指定した場合、呼び出すAPIのURLは次のようになります。

```
http://localhost:8080/api/v1/(インターフェースで指定したパス)
```

- MicroProfile Configの詳細は「【MicroProfile開発ガイド】MicroProfile Config」を参照してください。

クライアントのインスタンス化と実行 RestClientBuilder API (1/2)

RestClientBuilder APIは、CDIが使えない場合にも利用することができ、次の4つのメソッドを呼び出し、実装します。

- ① newBuilder() . . . 必須
 - RestClientBuilder API のインスタンスを作成します。
- ② baseUrl() . . . 必須 ※MicroProfile Rest Client 1.1で追加
 - リモート・エンドポイントのURL(baseUrl)を指定します。クライアントをインスタンス化する前に必要です。
- ③ register() . . . オプション
 - 独自例外Mapperを登録します。MessageBodyReaders、MessageBodyWriters、filters、interceptors など、他のプロバイダー・クラスを登録する必要がある場合は、register メソッドを使用して登録します。
- ④ build() . . . 必須
 - 実装するインターフェースを指定し、クライアントをインスタンス化します

コード例

```
URL apiUrl = new URL("http://localhost:8080/api/v1/");
HRServiceClient hrServiceClient = RestClientBuilder.newBuilder() . . . ①
    .baseUrl(apiUrl) . . . ②
    .register(HRServiceExceptionMapper.class) . . . ③
    .build(HRServiceClient.class); . . . ④
```

クライアントのインスタンス化と実行 RestClientBuilder API (2/2)

RestClientBuilder APIを使用してインスタンス化したクライアントは、インターフェースで定義したメソッドを呼び出すことで利用できます。各メソッドに対して、必要な引数、例外処理を追加します。

コード例

```
....  
public User get(int userId) {  
    try {  
        URL apiUrl = new URL("http://localhost:8080/api/v1/");  
        HRServiceClient hrServiceClient =  
            RestClientBuilder.newBuilder()  
                .baseUrl(apiUrl)  
                .register(HRServiceExceptionMapper.class)  
                .build(HRServiceClient.class);  
  
        hrRestClient.getUser(userId);  
    } catch (UnknownUrlException e) {  
        ....  
    }  
}
```

MicroProfile Rest Clientの各種機能

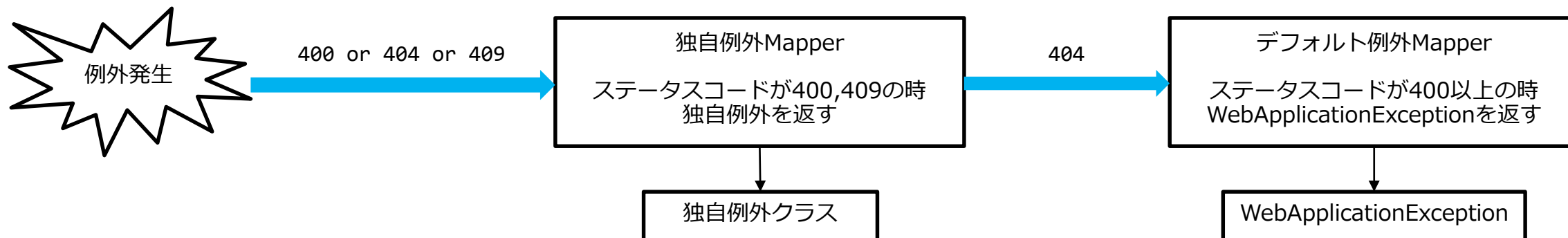
プロバイダー

プロバイダーは、実装することでデータ・バインディングなどの機能を追加することができます。RestClientBuilderインターフェースはJAX-RSのConfigurableインターフェースを継承しているため、独自例外Mapperで使ったResponseExceptionMapperの他にJAX-RSと同じプロバイダーを利用することができます。

- ClientResponse(Request)Filter
 - フィルター：サービスから着信したレスポンスや、サービスへ送信するレスポンスを加工するポイント
- MessageBodyReader(Writer)
 - エンティティ・プロバイダー：Java 型と表現 (representation) の間のマッピング・サービスを提供
 - 指定しない場合、デフォルトの定義が実装されます
- ParamConverter
 - 引数コンバーター：リソース・メソッドの引数に対する独自の型変換処理を追加
- WriterInterceptor
 - エンティティ・インターセプター：MessageBodyWriterに処理を追加

デフォルト例外Mapper

デフォルト例外Mapperは、レスポンスのステータス・コードが400より大きい場合にWebApplicationExceptionを返します。優先度は一番低くなっており、独自例外Mapperでレスポンスを確認し、指定した条件に当てはまらなければデフォルト例外Mapperによって処理されます。



このデフォルト例外Mapperは全てのRest Clientインターフェースに登録され、無効化するためには次の2通りの手順が必要になります。

- MicroProfile Configで無効設定
 - `microprofile.rest.client.disable.default.mapper=true`
- Rest Client Builder呼び出し時にプロパティメソッドで設定
 - `RestClientBuilder.newBuilder().property("microprofile.rest.client.disable.default.mapper",true)`

非同期的なサービス間の連携

MicroProfile Rest Clientでは、REST APIの非同期呼び出しメソッドをインターフェースに定義することが可能です。標準のJAX-RSでは非同期処理のための別スレッド管理の仕組みがないため、スレッド管理機構としてConcurrency Utilities for Java EEを利用します。

- 非同期メソッドを宣言する場合、戻り値の型は`java.util.concurrent.CompletionStage`を利用します。

コード例

```
public interface MyAsyncClient {  
    @GET  
    @Produces(MediaType.APPLICATION_JSON)  
    public CompletionStage<List<User>> getUserAsync() throws UnknownUrlException;  
}
```

- 返された`CompletionStage`は、`CompletableFuture`型に変換することで、処理結果を`get()`メソッドで取得可能です。型の変換には、`CompletionStage`の`toCompletableFuture()`メソッドを使用します。

コード例

```
CompletableFuture<List<User>> future = hrServiceClient.getUserAsync().toCompletableFuture();  
List<user> users = future.get();
```

上記の`get()`メソッドは、非同期で実行されている処理が終わるまで呼び出し元のスレッドをブロックして待ちます。

CompletionStage : <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletionStage.html>
CompletableFuture : <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html#get-->

参考リンク

参考リンク

- MicroProfile Rest Client 1.1

<http://download.eclipse.org/microprofile/microprofile-rest-client-1.1/microprofile-rest-client.pdf>

- IBM Knowledge Center - MicroProfile Rest Client の構成

https://www.ibm.com/support/knowledgecenter/ja/SSAW57_liberty/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/twlp_mp_restclient.html

- Open Liberty - Consuming RESTful services with template interfaces

<https://openliberty.io/guides/microprofile-rest-client.html>

- MicroProfile Rest Client API 1.1 API

<http://download.eclipse.org/microprofile/microprofile-rest-client-1.1/apidocs/overview-summary.html>

End of File