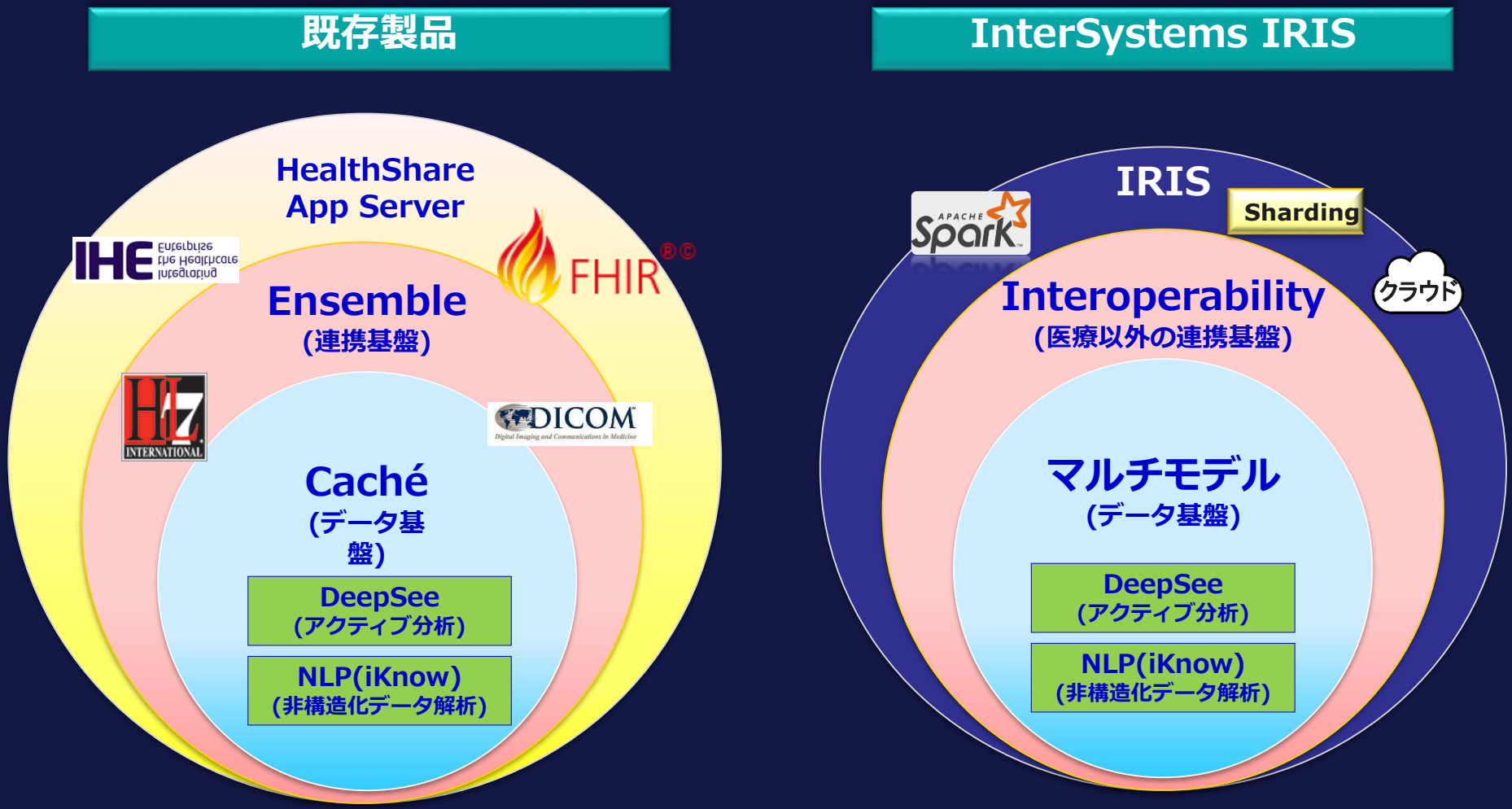
InterSystems IRIS
システム連携基盤
体験コース

InterSystems®
Health | Business | Government

このコースの主な目的

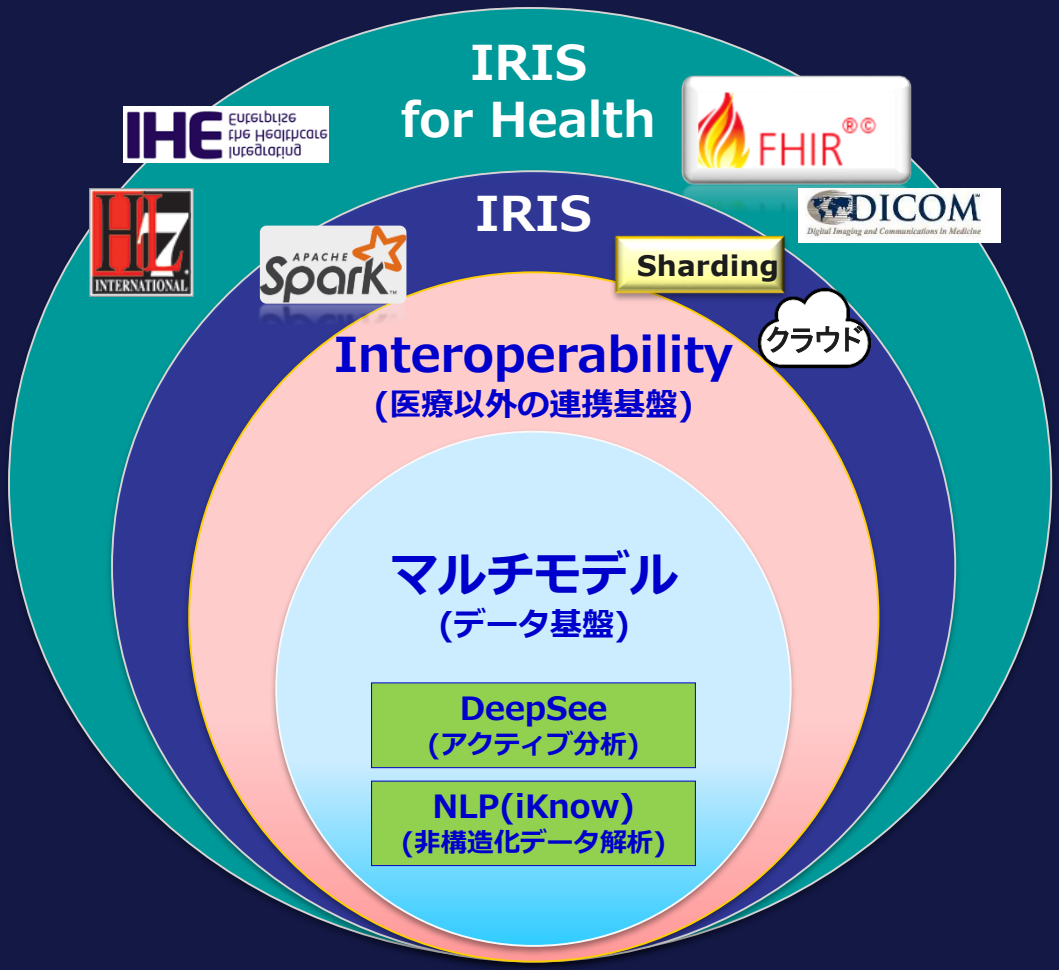
- 本コースでは、開発者様を対象にInterSystems IRIS（以降　IRIS）のシステム連携基盤について、簡単なテーマを使用しながら操作に慣れること、動作概要をご理解いただくことを目的としています。

データプラットフォーム
既存製品と InterSystems IRISとの関係



InterSystems IRISは、Caché に含まれていない機能（シャーディング、クラウドデプロイ用ツール）やEnsembleやHealthShareの持つ相互運用性も含めた、包括的なデータプラットフォームです。
2018年8月、InterSystems IRIS 2018.1.1がリリースされています。

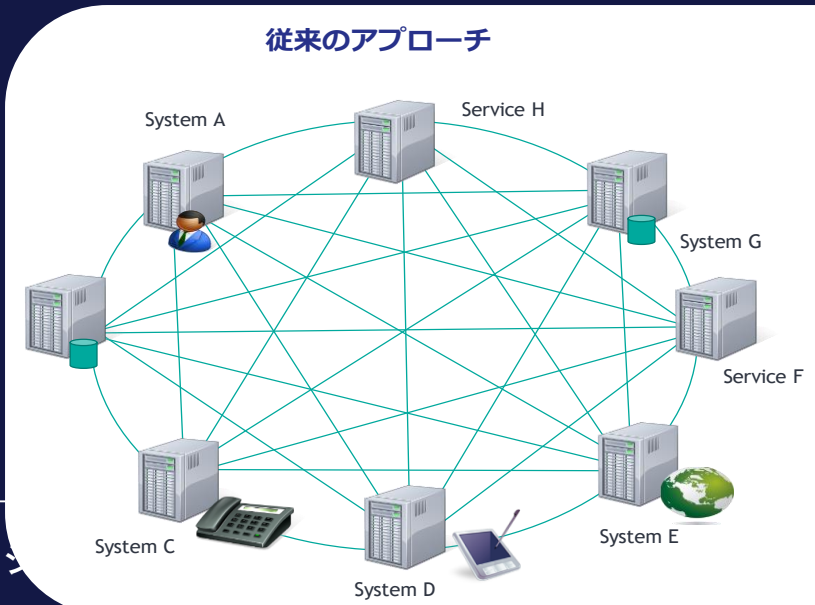
InterSystems IRIS for Health

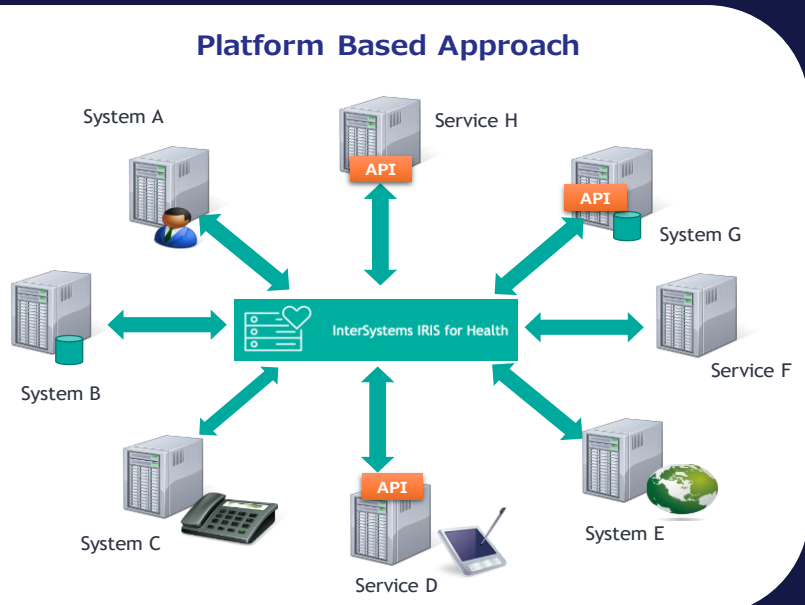


IRIS for HealthはIRISに医療用規格に対応する機能を搭載したIRISの医療向けプラットフォームです。

相互運用性機能（Interoperability）を使いたい理由

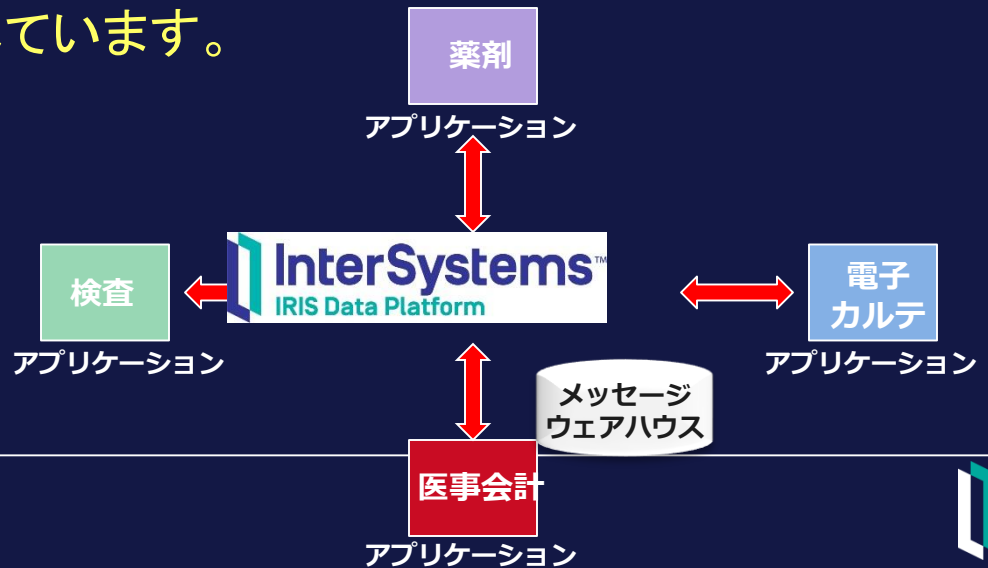
- 既存システムにある情報を利用して新しい仕組みを作りたいけれど、既存システムに手を加えられない。
  - 1度だけでなく定期的にデータを取得したい。
  - 取得した情報を使用したいフォーマットに変換し、データベースに格納しておきたい。
- アラート通知などの機能がほしい。
  - 既存システムは変えられない
    → 送受信するデータからアラート対象を見つけ出し通知したい！
- 異なるシステム間の接続の見通しをよくしたい。
  - リアルタイムに状況を確認したい。
- 連携先の接続断による送信のやり直しを、最初からではなく途中からできるような仕組みが欲しい！





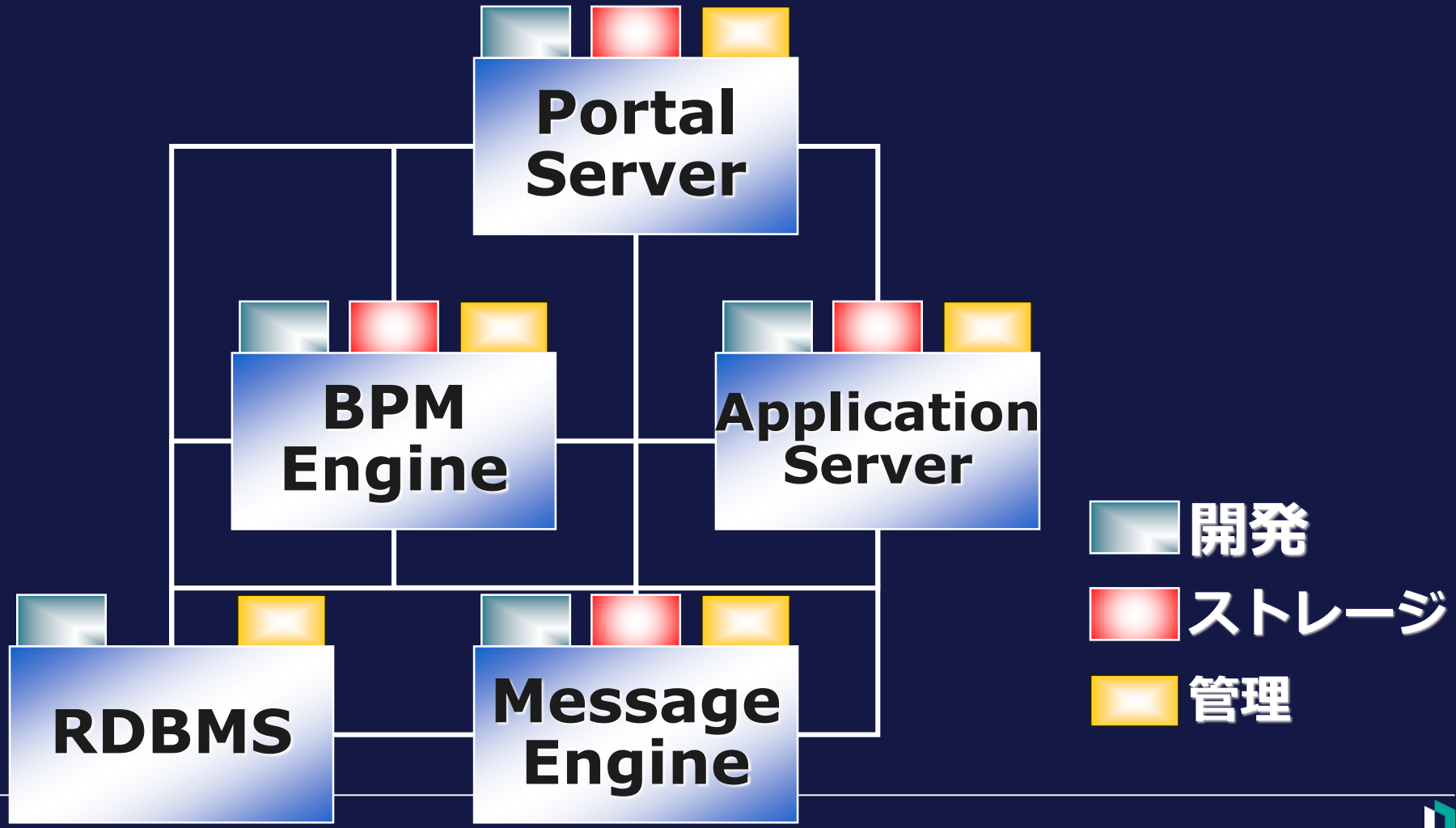
InterSystems IRISのシステム連携基盤で オールインワンの管理

- IRISのシステム連携基盤を利用すると以下の内容を **IRIS 1つで対応できます。**
  - 連携中データ（＝メッセージ）がデータベースに自動的に記録されるため、いつでも状況をトレースしたり、データの再送が行えます。
  - 処理のフローを制御するためのBPM実行エンジンを持っています。
  - IRISのデータプラットフォームを利用することで、2次利用目的のデータを蓄積するデータベースを構築できます。
  - ビジネスアクティビティモニタリング（BAM）を行うための監視や結果表示のための画面が提供されています。



ご参考：他統合製品を利用した場合

- 統合するために、沢山のツールと技術が必要



ご参考：IRISを利用した場合

- オールインワン！

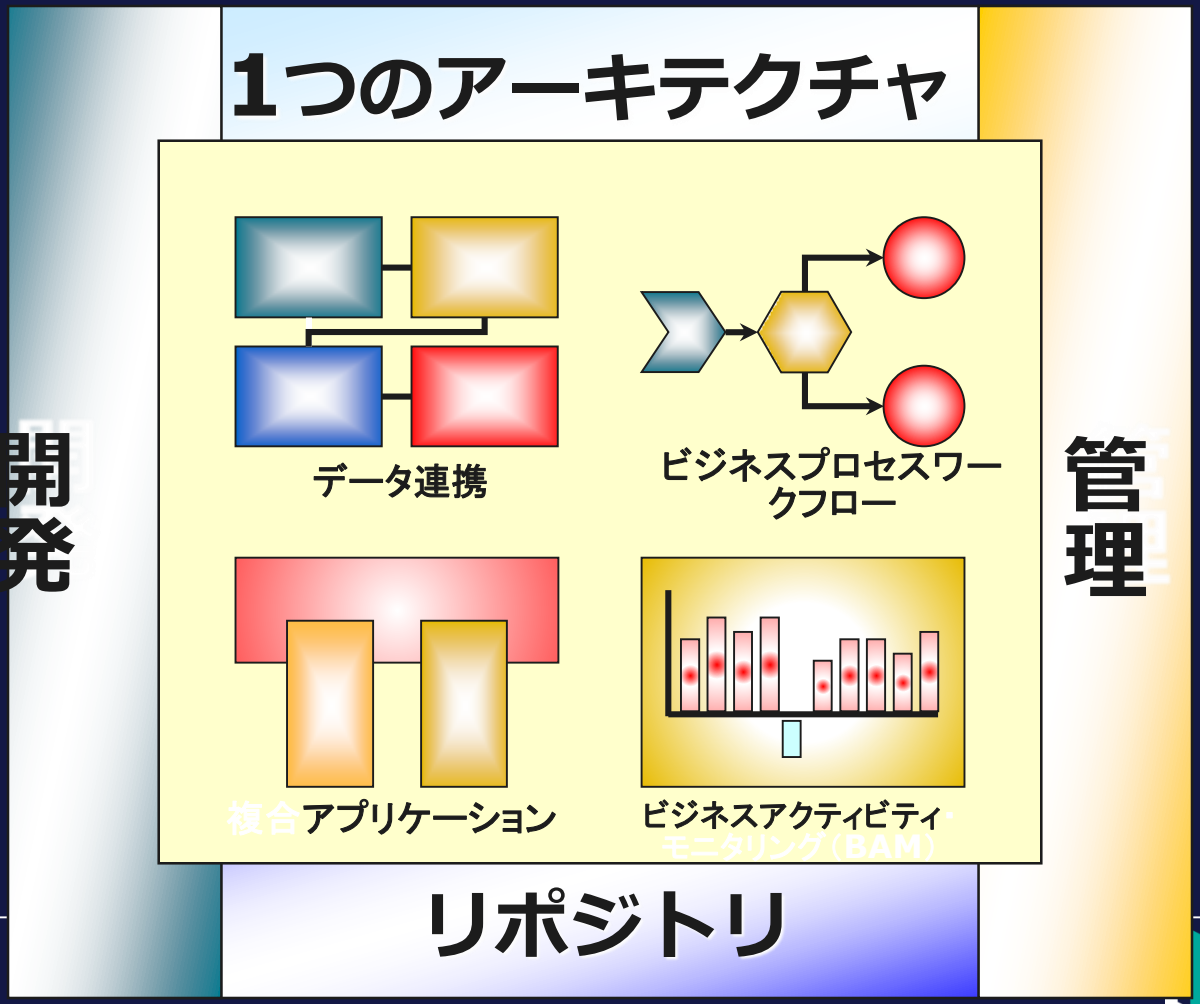
1つ のデータ＋論理モデル

1つ のリポジトリ

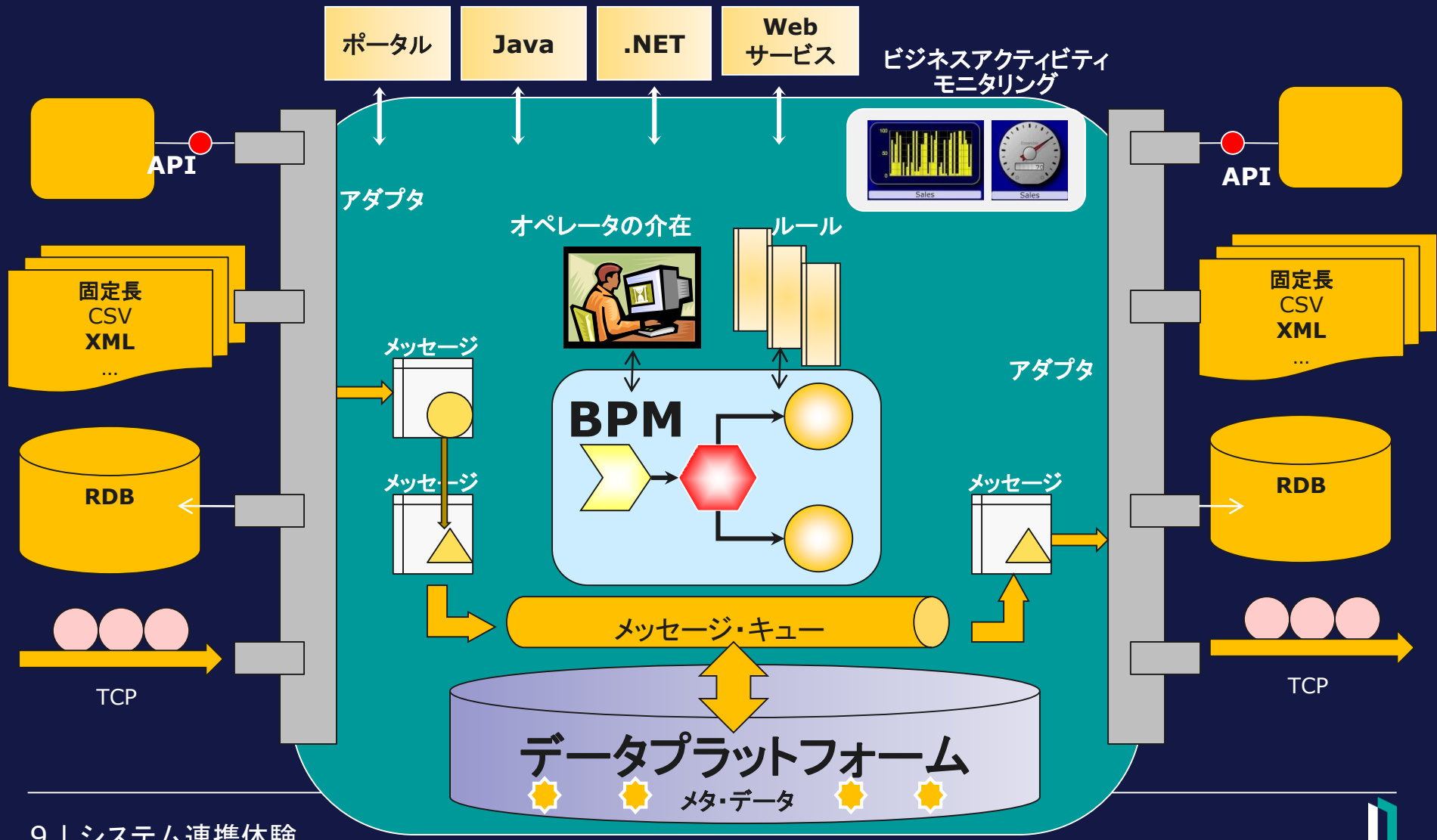
1つ の開発環境

1つ の実行環境

1つ の管理環境

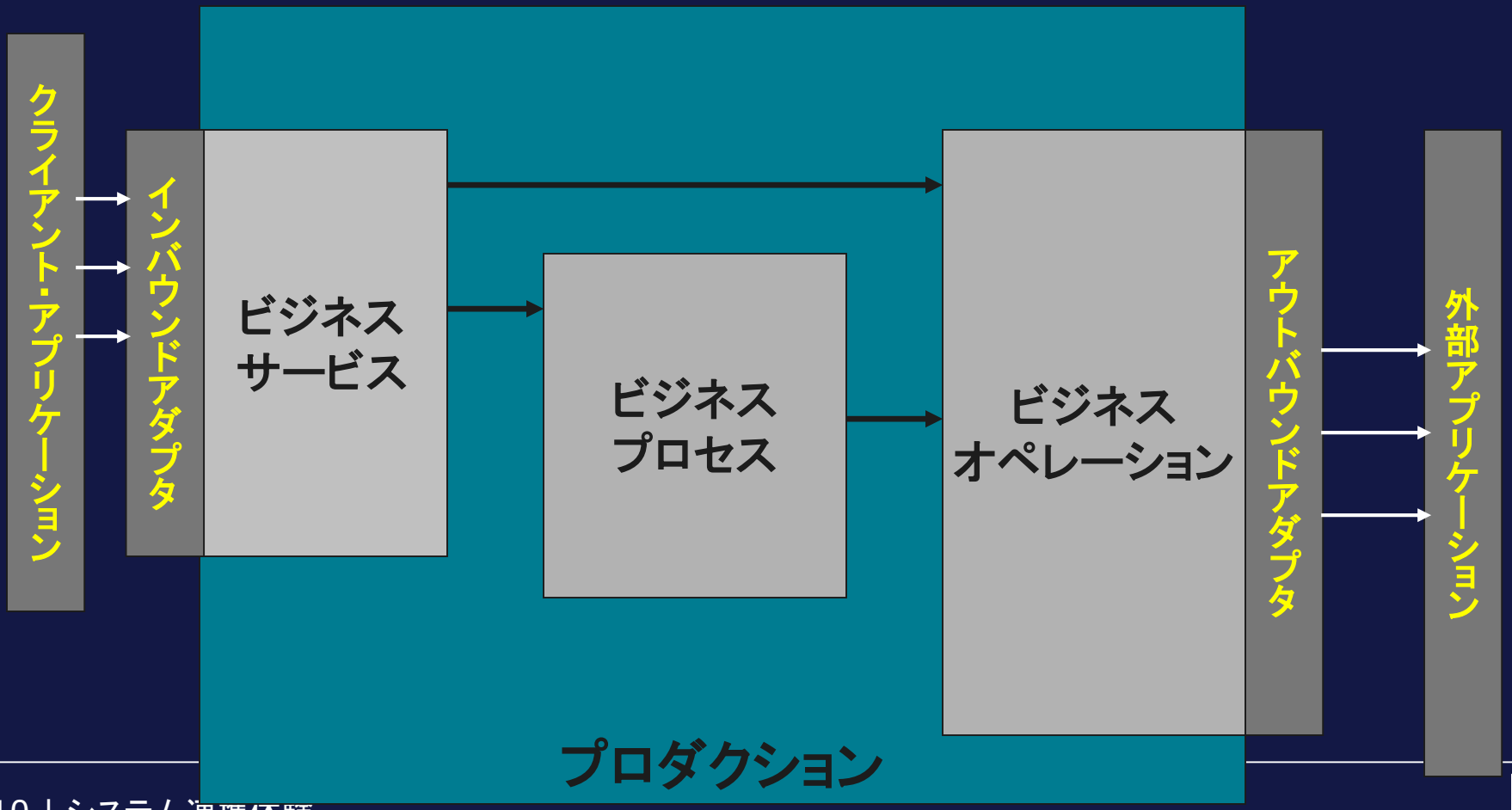


システム連携　イメージ図



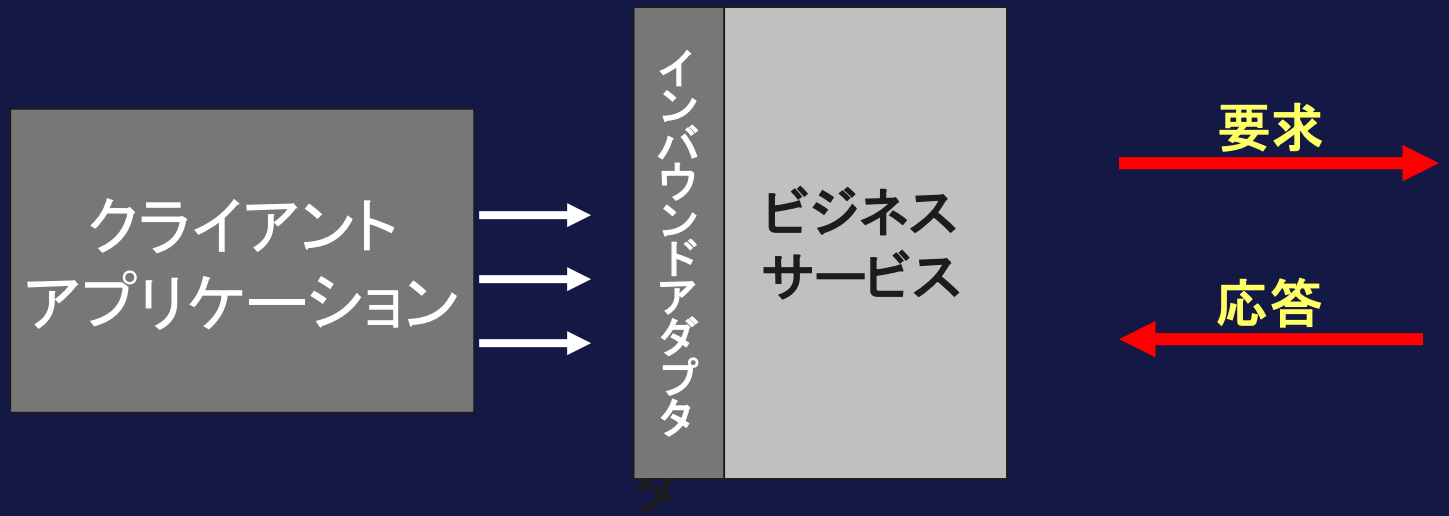
システム連携　簡易図

- 連携するために必要なコンポーネントの集まりを定義したものをプロダクションと呼びます。



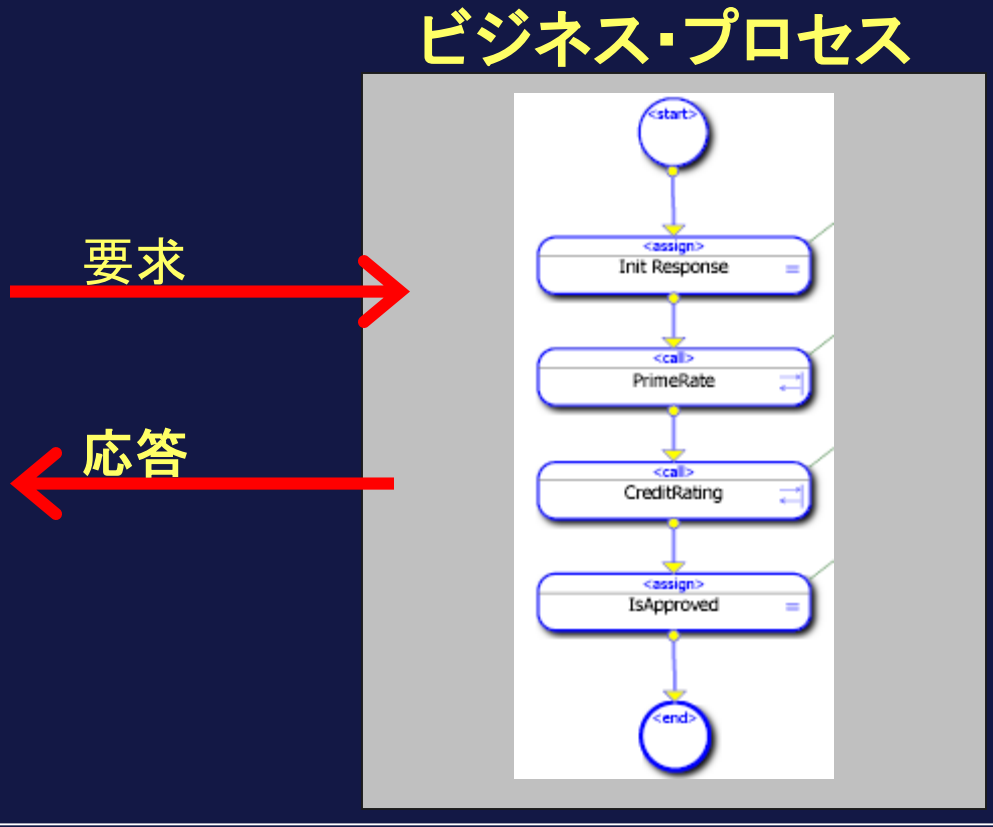
ビジネス・サービス

- ビジネス・サービスは、外部からの情報を受け取るコンポーネントです。
  - データの受信にはインバウンドアダプタを利用する方法が提供されています。
- ビジネス・サービスでは、トランザクションごとの最初のメッセージを作成します。



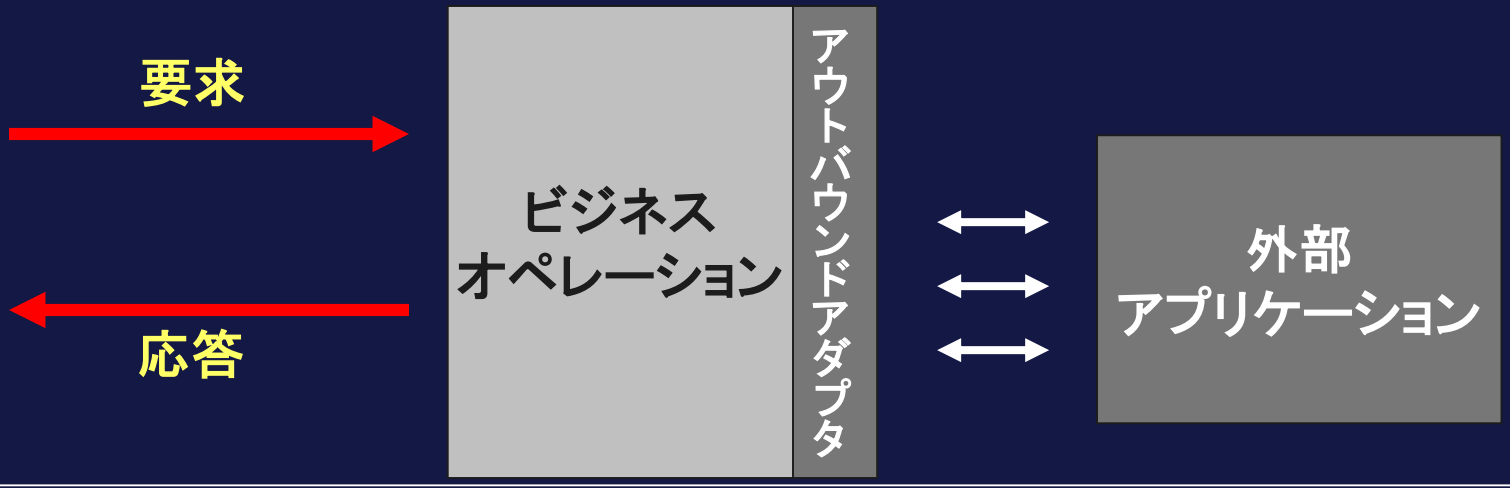
ビジネス・プロセス

- ビジネス・プロセスは、ロジックと処理フローを提供するコンポーネントです。



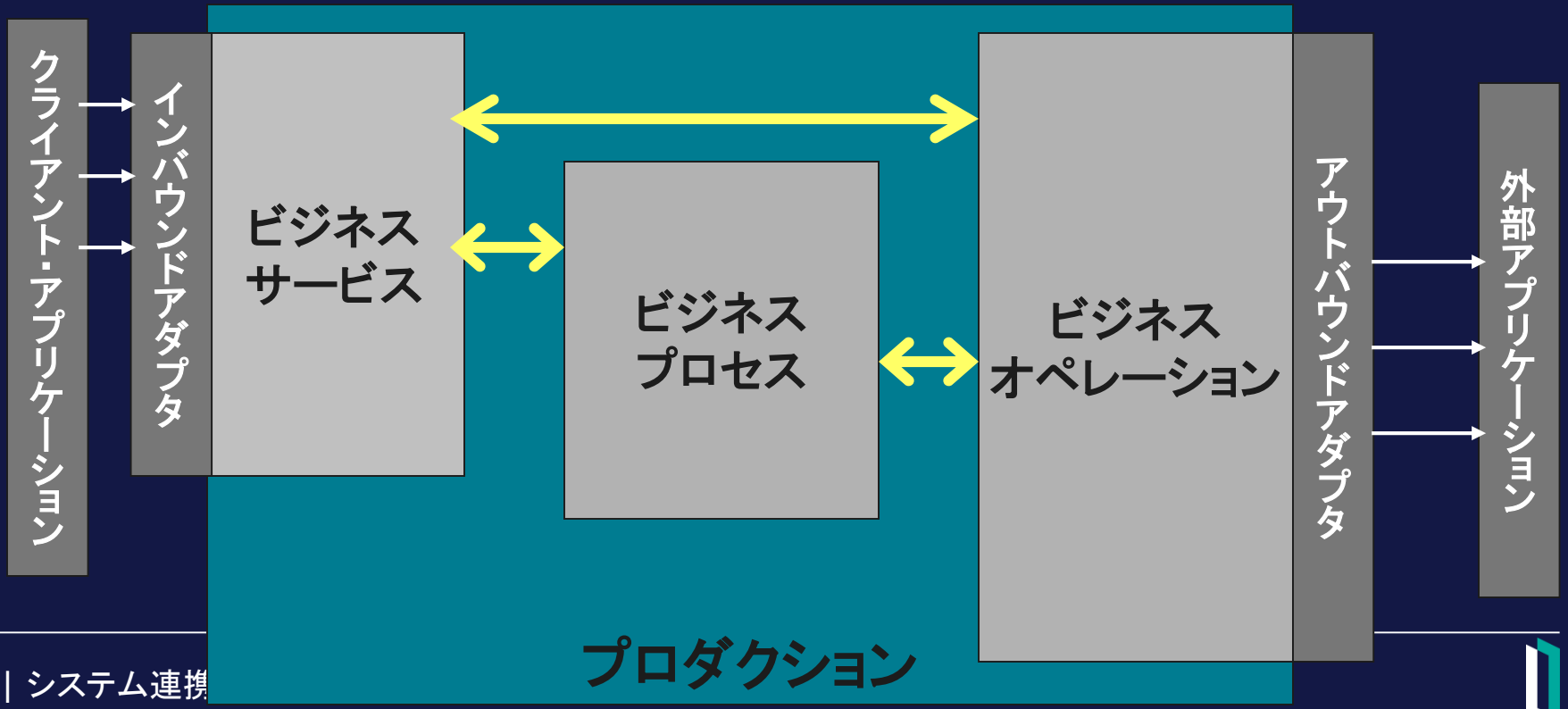
ビジネス･オペレーション

- ビジネス･オペレーションは、外部のアプリケーションに通信し、処理を依頼する役割があります。
  - 情報を渡すためにクエリを指定できます。
  - データを外部に送信します。
- ビジネス･オペレーション毎に1つのインターフェースに接続します。



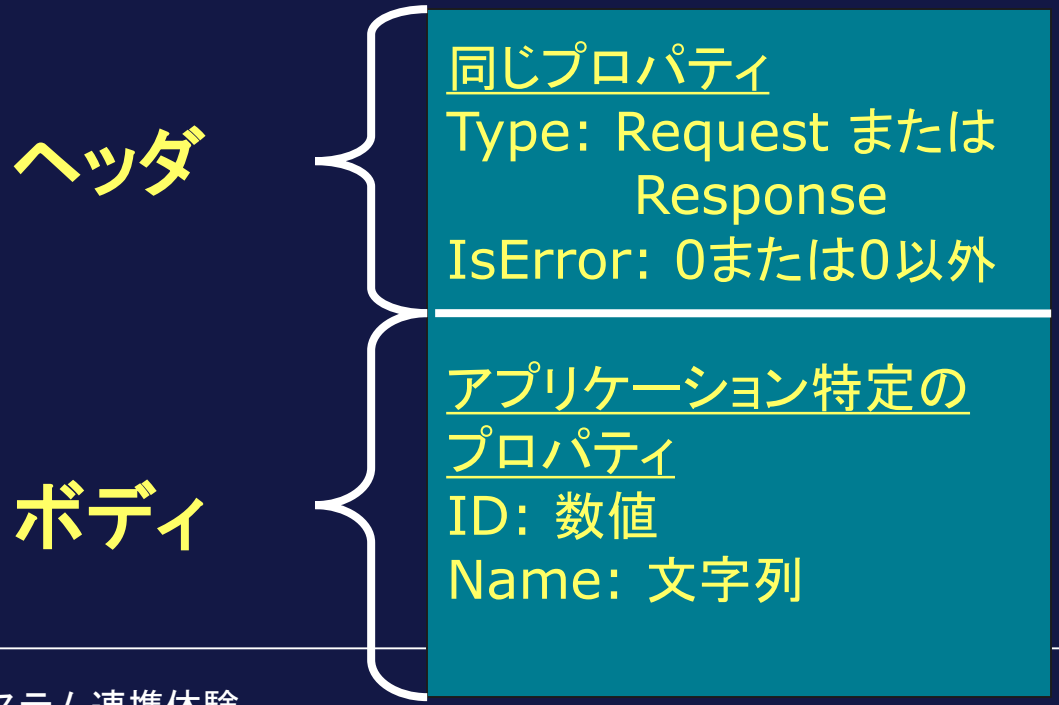
メッセージ

- プロダクションのコンポーネント間で送受信される情報のことをメッセージと呼びます。
- メッセージは外部から送信される情報を元にビジネス・サービスで作成します。
  - メッセージには、受信した情報を全てを登録することも一部抜粋した情報を登録することもできます。
- メッセージの送受信が発生すると自動的にメッセージ をデータベースに格納します。
  - メッセージは永続クラスです。



メッセージの中身

- プロダクションで発生するすべての通信にメッセージを使用します。
- メッセージでは、どこからどこの経路で使用したメッセージであるかを把握できるよう、**システムが管理するヘッダ部分とアプリケーションが管理するボディ部分に分け**、メッセージを管理しています。
  - ボディには、アプリケーションロジックで使用したい（持ちまわりたい）情報が格納できるよう、任意のプロパティを定義できます。

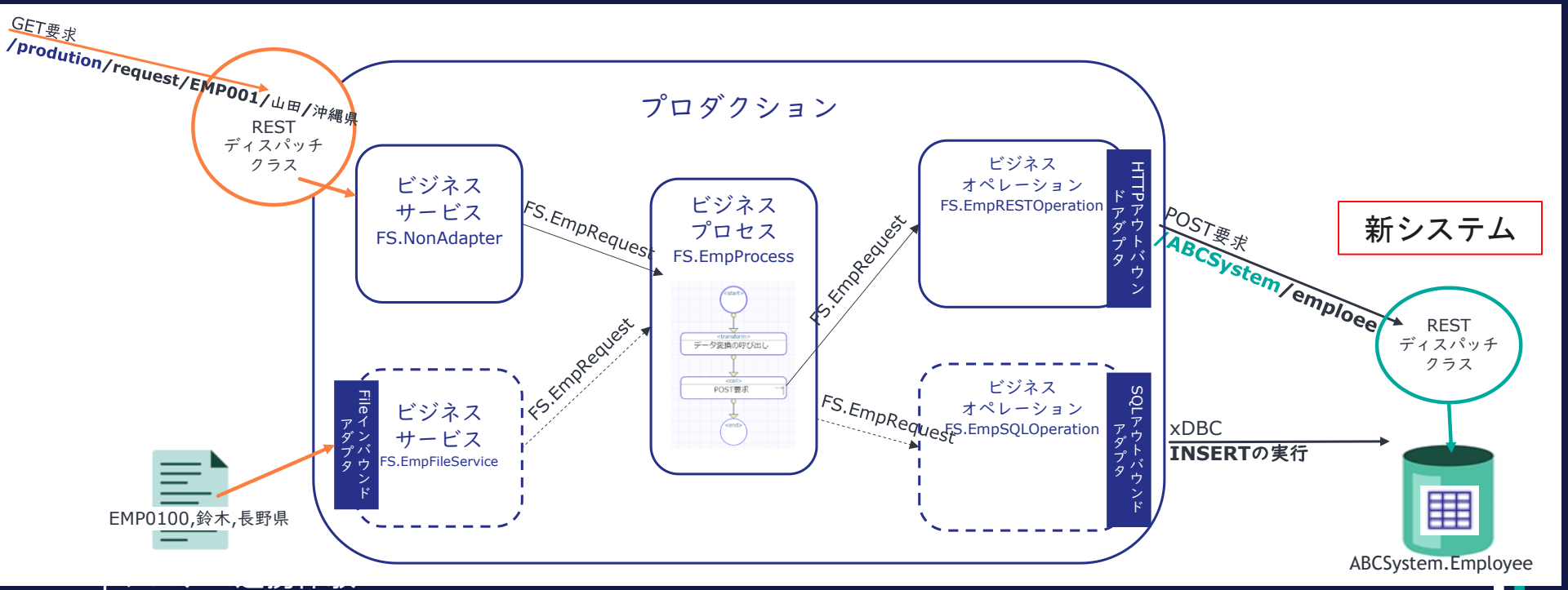


アダプタについて

- 外部システムとの接続を容易にするクラスです。
  - 例）メール、ファイル、SOAP、FTP、HTTP、SQL、TCP
- アダプタを使用することで、**外部システムへ接続するための処理の詳細を意識する必要がありません。**
  - 例1） SQLインバウンド／アウトバウンド　アダプタ
    ODBC経由で指定のDSNへ接続する処理が自動的に提供されるため、接続処理に必要なコードが記載不要です。
    - UPDATEやINSERT、SELECTなどのSQL文は記述します。
  - 例2） ファイルインバウンド／アウトバウンド　アダプタ
    指定ファイルの入出力処理が自動的に提供されるため、ファイル検知のためのディレクトリ監視や、デバイスオープン、クローズ処理などコード中に記載不要です。

演習テーマ

- ある会社に新社内システム（ABCSystem）が導入され、利用申請を済ませた社員情報を自動的に新システムに登録するようにしたいと考えました。
- 新システムは REST API とテーブルが公開されています。
- 利用申請画面から、REST APIを利用して登録できたら？と考えましたが、新システムに必要な情報が不足していることに気が付きました。
- ということで、IRISのInteroperabilityを利用して、REST経由で送付される情報に不足する情報を追加し、新システムのREST APIに送信することにしました。

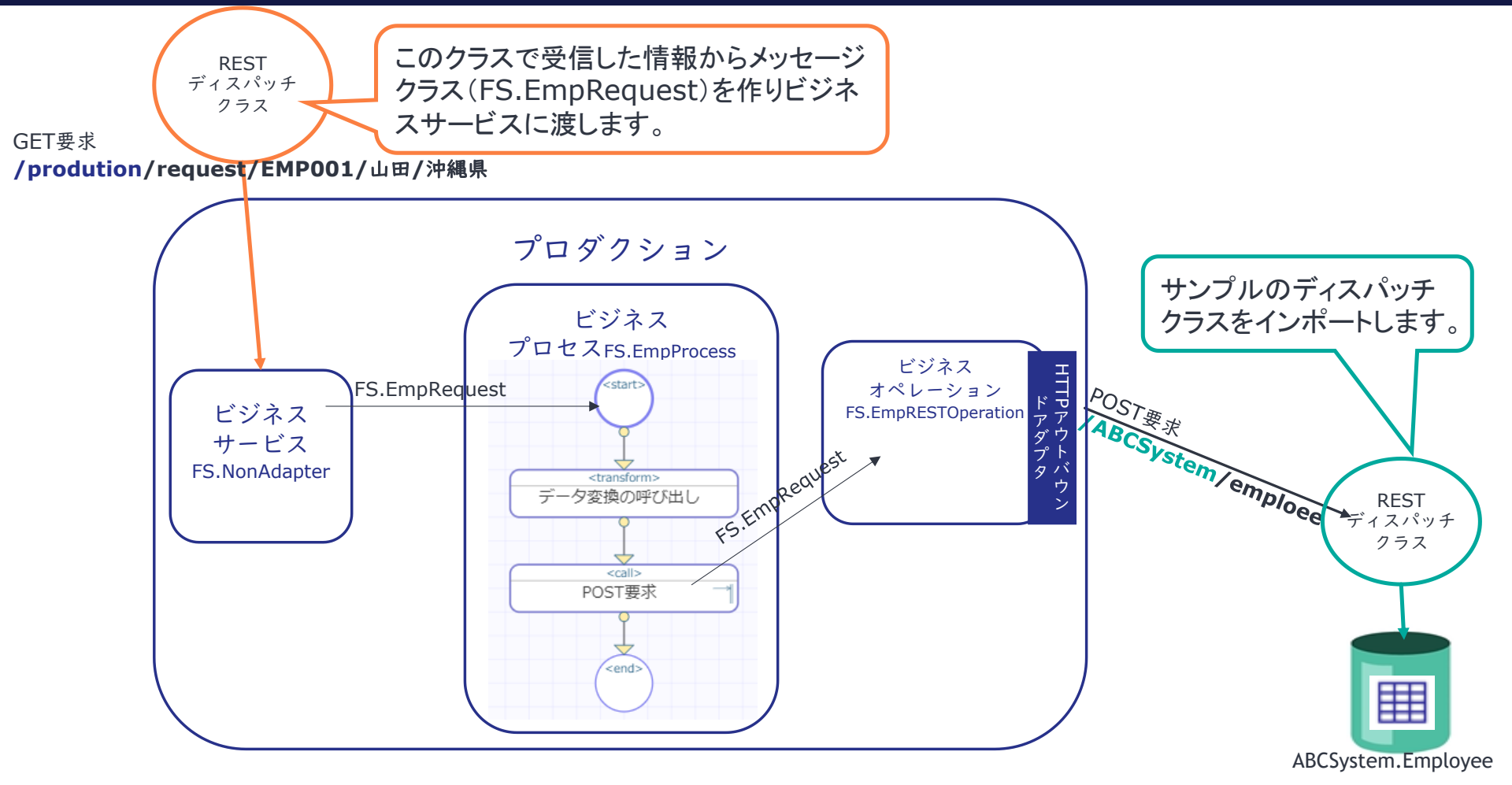


演習テーマ：入出力情報

- プロダクションの入力
  従業員番号（EMPID）、名前（Name）、勤務地（Location）
- 新システムに渡す情報
  従業員番号（EMPID）、名前（Name）、勤務地（Location）、
  サービスエリア（Area）
  - サービスエリア＝勤務地よりも大きな単位で社員をグループ分けするための情報が追加されている。

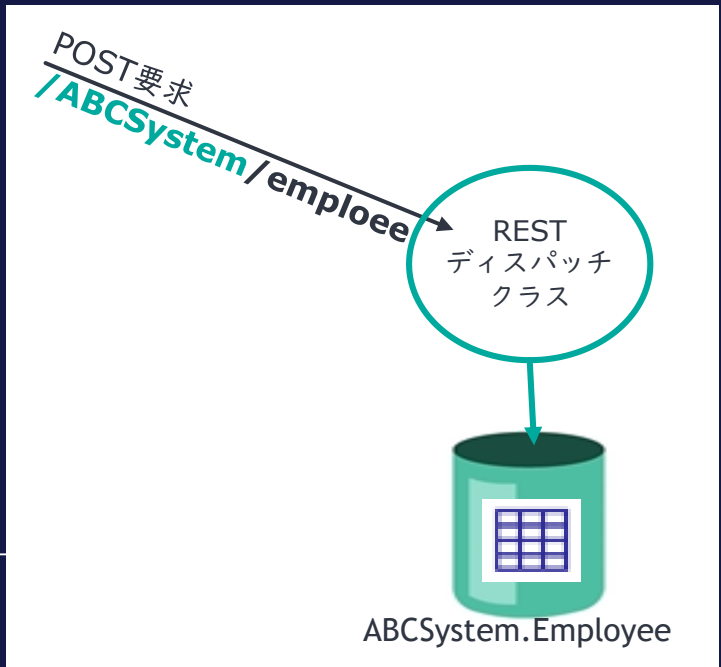
作成順序

1. 要求メッセージ(FS.EmpRequest)
2. REST APIを呼び出すオペレーション(FS.EmpRESTOperation)
3. データ変換
4. データ変換とオペレーションを呼び出すプロセス(FS.EmpProcess)
5. サービス(FS.NonAdapter)
6. サービスを呼ぶRESTディスパッチクラス

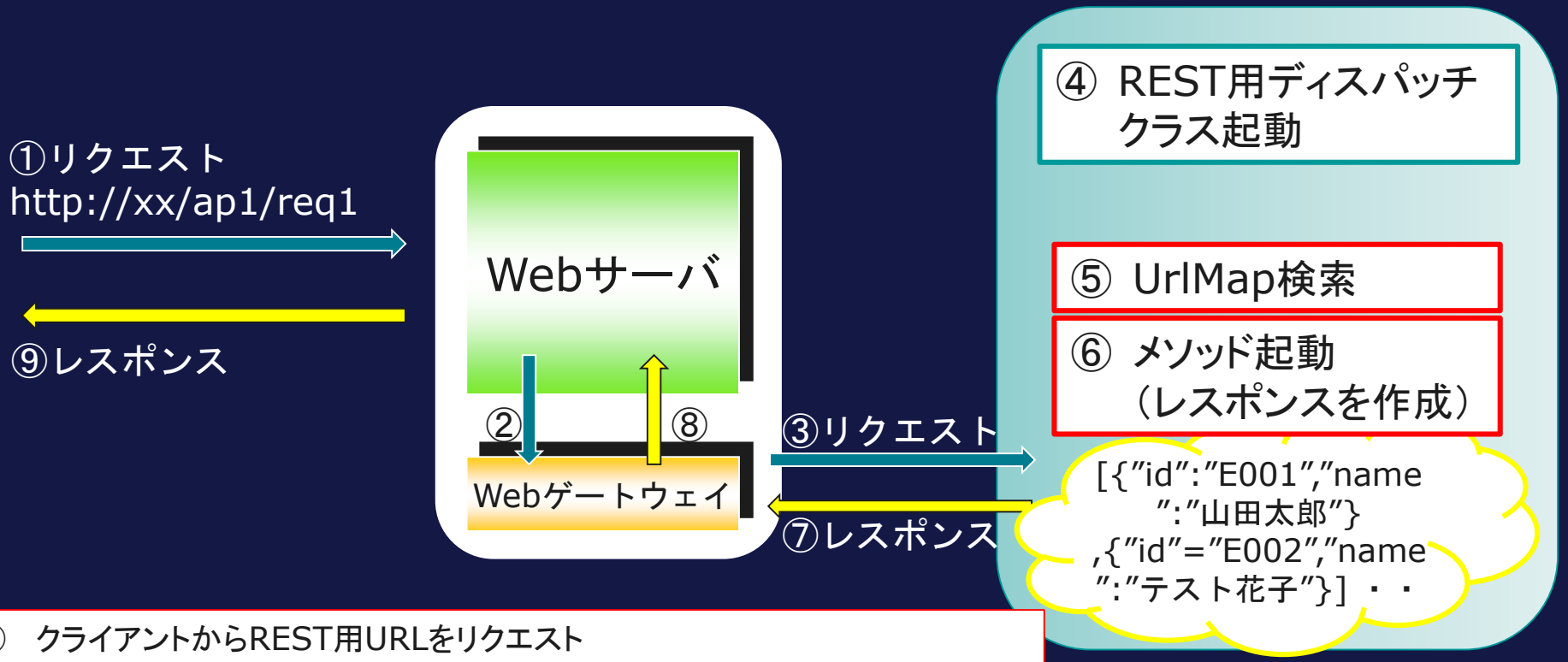


準備

- ABCSystem.Employeeテーブルに対してGETとPOST要求を行うためのRESTディスパッチクラスをインポートします。
- REST要求時に使用するベースURLを設定します。
  - サンプルのRESTディスパッチクラスを指定します。
- テストします。
  - RESTクライアント、または curlコマンドでテストします。



IRISで作成するRESTサーバの仕組み



① クライアントからREST用URLをリクエスト
② Webサーバがリクエストを受け取り、Webゲートウェイへ送信
③ WebゲートウェイがInterSystems製品にリクエスト（正確には空いているWebゲートウェイ⇔サーバ間のプロセスにリクエスト）
④ リクエストのURLからネームスペースとREST用ディスパッチクラスを特定
⑤ ディスパッチクラスのUrlMapから対応するメソッドを特定
⑥ メソッドを起動してHTTPレスポンスを作成
⑦ サーバからWebゲートウェイにレスポンスを返す
⑧ WebゲートウェイはWebサーバにレスポンスを返す
⑨ Webサーバからクライアントへ返す

ここまでの内容の解説ビデオは開発者コミュニティでも公開中！

RESTディスパッチクラスとは

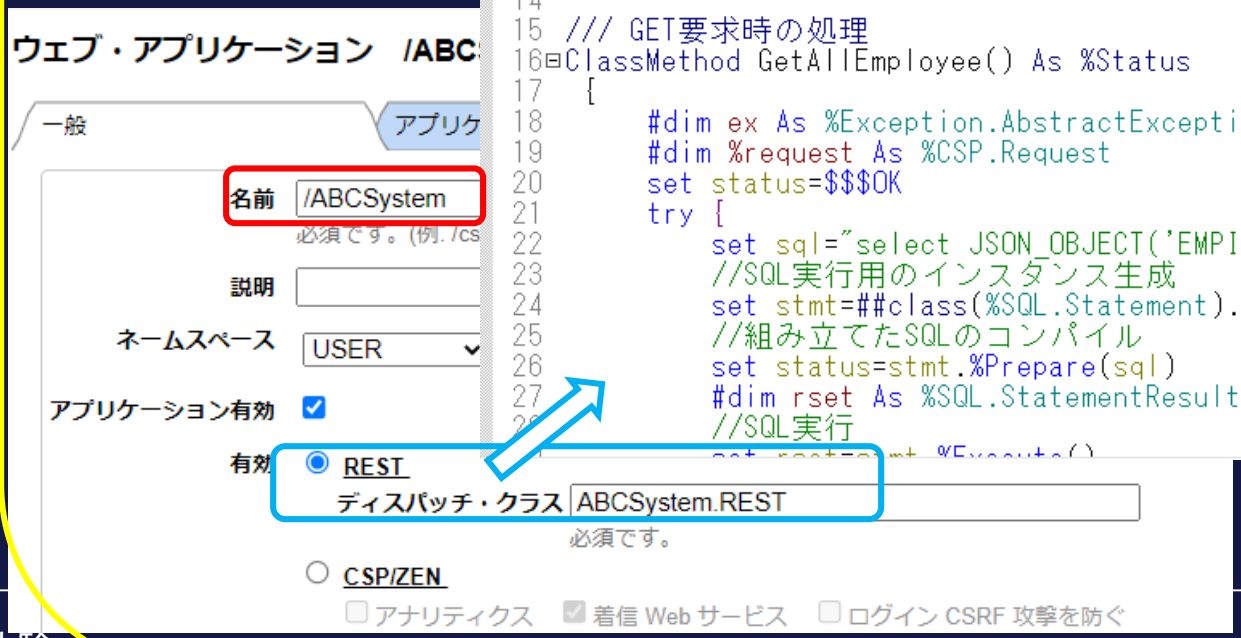
- REST URL と HTTP メソッドに対して実行するクラスメソッドを指定するUrlMapを持つクラスです。

例）GET要求
/ABCSystem/employee

RESTのベースURLの設定（/ABCSystem）

RESTディスパッチクラス

```
1  /// RESTディスパッチクラス
2  Class ABCSystem.REST Extends %CSP.REST
3  {
4
5  /// ベースURL以降のURLに対応するHTTPメソッドと処理（メソッド
6  XData UrlMap [ XMLNamespace = "http://www.intersystems.com/u
7    {
8  <Routes>
9    <Route Url="/employee" Method="POST" Call="CreateEmployee"
10   <Route Url="/employee" Method="GET" Call="GetAllEmployee"
11   <Route Url="/employee/:empid" Method="GET" Call="GetEmploy
12   </Routes>
13   ]
14
15 /// GET要求時の処理
16 ClassMethod GetAllEmployee() As %Status
17   {
18       #dim ex As %Exception.AbstractException
19       #dim %request As %CSP.Request
20       set status=$$$OK
21       try {
22           set sql="select JSON_OBJECT('EMPID':EMPID,'Name':Na
23           //SQL実行用のインスタンス生成
24           set stmt=##class(%SQL.Statement).%New()
25           //組み立てたSQLのコンパイル
26           set status=stmt.%Prepare(sql)
27           #dim rset As %SQL.StatementResult
28           //SQL実行
```



REST用ウェブアプリケーションパス（ベースURL）の作成

- RESTサービス専用のウェブアプリケーションパスを作成し、REST用ディスパッチクラスを定義します。

REST URLの指定方法

- URLは以下の順序で指定します。

Webサーバアドレス/ウェブアプリケーションパス/URLマップに記載したURL

- REST用ディスパッチクラスのテストのため、管理ポータルが使用するWebサーバ(Apache)を利用できます。

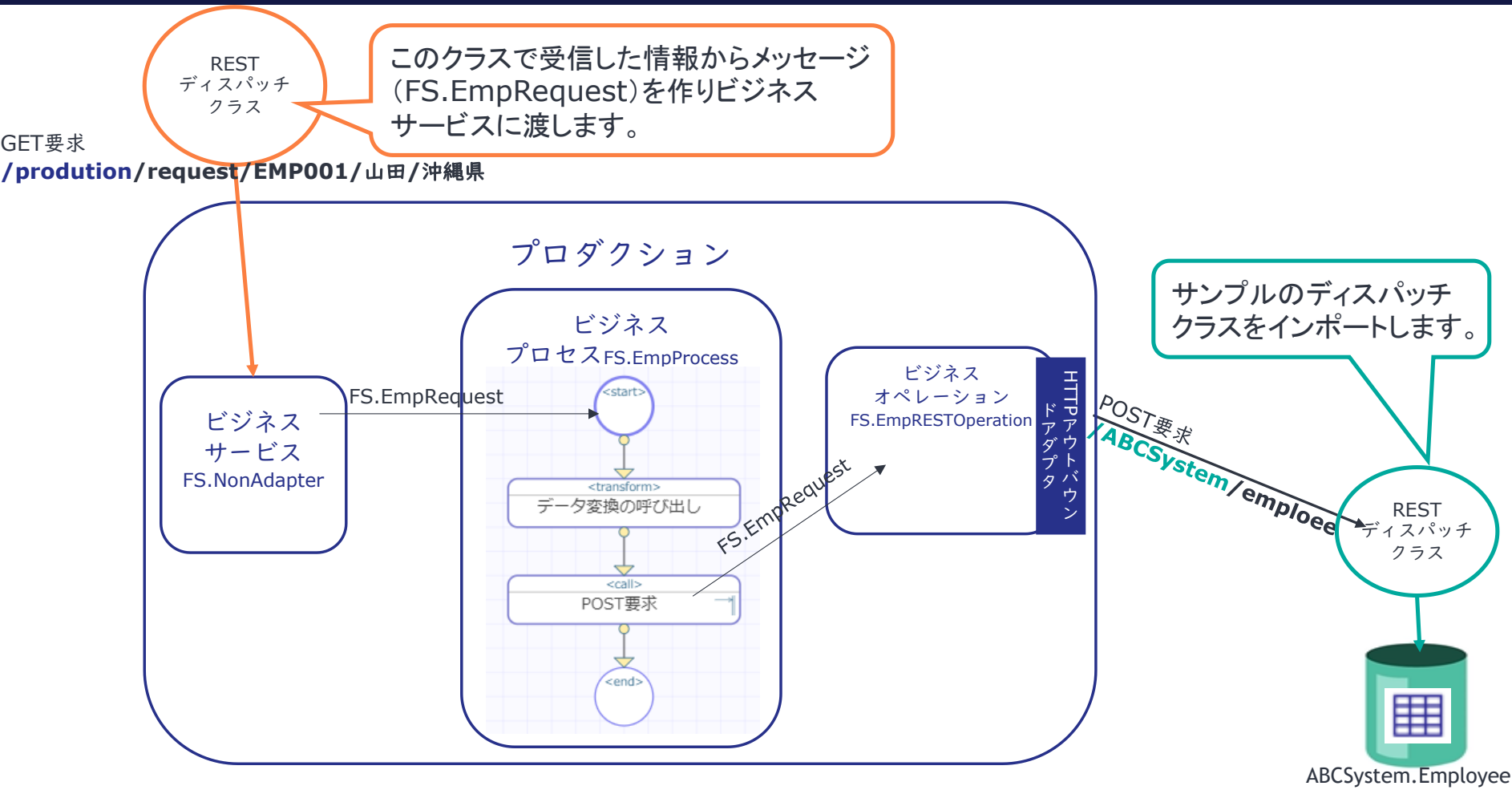
Webサーバアドレス:ポート/ウェブアプリケーションパス/URLマップに記載したURL

  - ウェブアプリケーションパスは前頁で作成した **/ABCSystem**
  - ABCSystem.Employee に対して、GET/POST/PUT/DELETE を行うためのパスを **/employee** とします。
    - ディスパッチクラスの UrlMap にそれぞれのHTTP要求が来た時に動作するメソッドの名称を定義します。
    - サンプルは、GETとPOSTの処理のみ記述しています。
  - 指定EMPIDのEmployeeを返すURLは以下の通りです。

localhost:52773/ABCSystem/emloyee/EMP0001

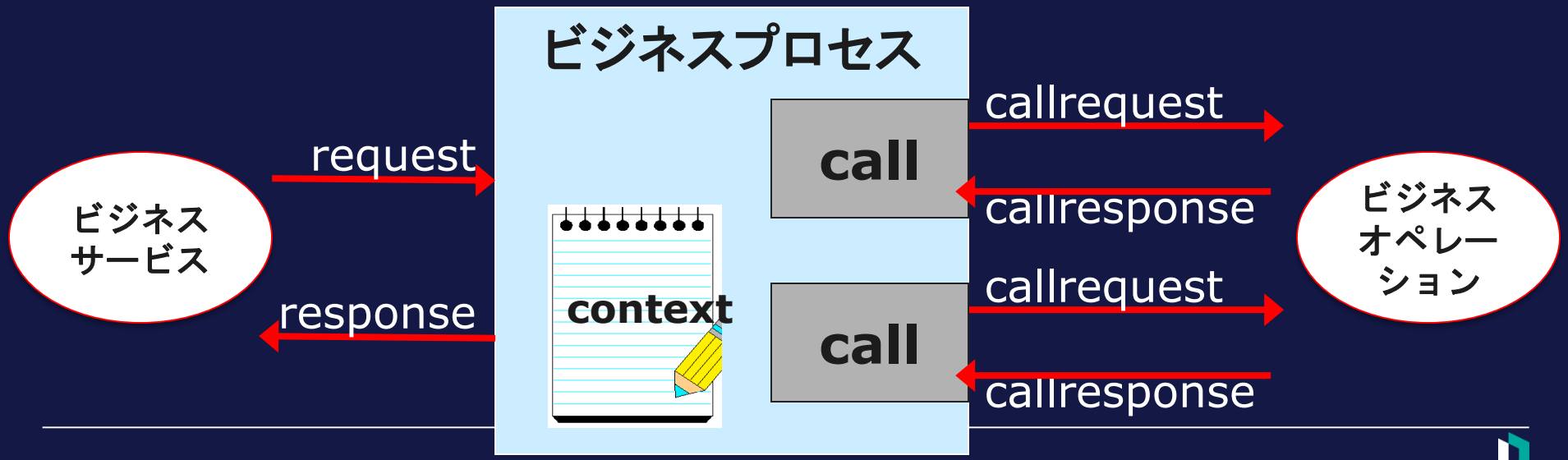
ここからは・・

- いよいよ処理を連携させるため、プロダクションを作成します。



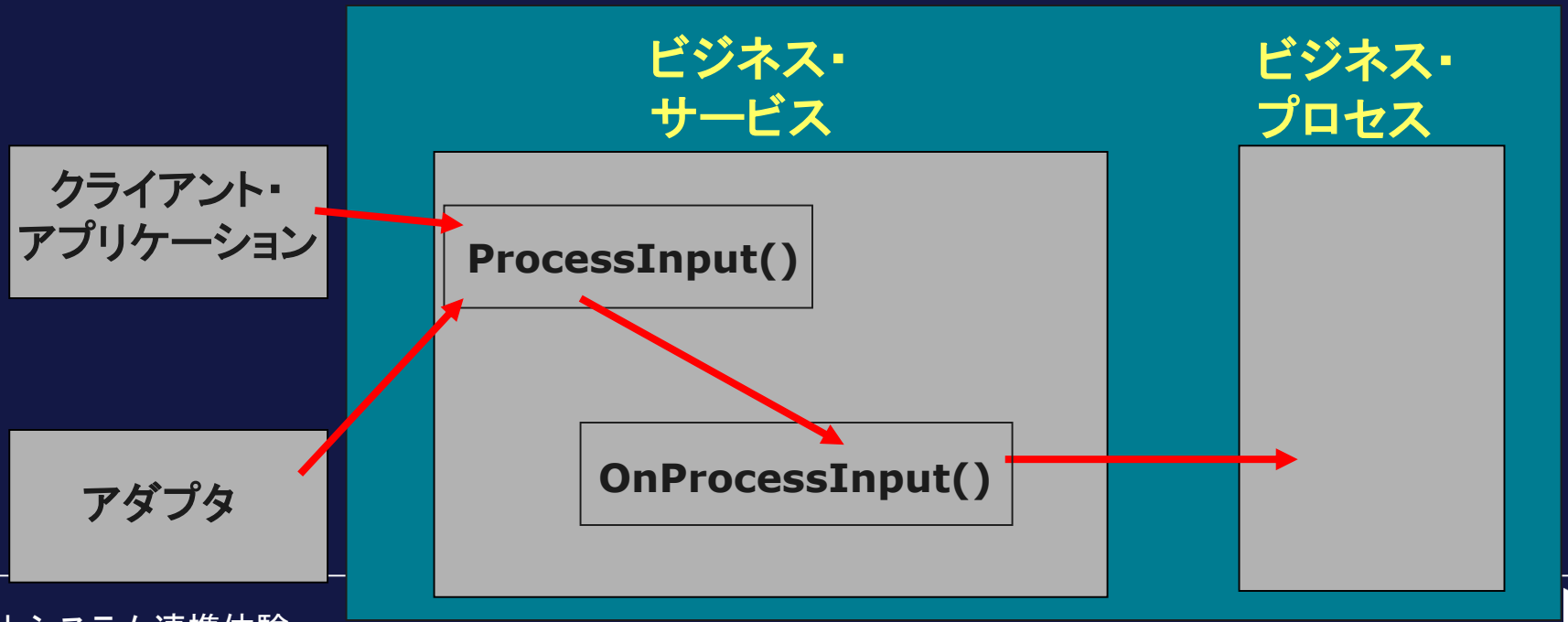
ビジネスプロセス内で利用するメッセージ専用オブジェクト

- コンポーネント内の処理に必要な情報は全て「メッセージ」でやり取りします。
  - ビジネスサービス⇒ビジネスプロセス、ビジネスプロセス⇔ビジネスオペレーションの間に、引き継ぎたい情報がある場合は、メッセージを使用して情報を受け渡します。
- ビジネスプロセスでは、以下オブジェクトを利用し、メッセージを処理します。
  - request、response、callrequest、callresponse、context
    - contextはビジネスプロセスの生存期間中操作できるオブジェクトです。



ビジネス・サービスへの入力処理

- ビジネス・サービスには、以下のメソッドが用意されています。
  - ProcessInput()
    - アダプタまたはクライアント・アプリケーションがビジネス・サービスの呼び出すために使用されるメソッドです。
  - OnProcessInput()
    - 入力処理に合わせて実装する必要があります。



ビジネス・サービスへの入力処理
アダプタを使用しない場合の呼び出し

- アダプタがないビジネス・サービスでは、外部からビジネス・サービスのProcessInput()を呼び出すために、何らかの方法でビジネス・サービスのインスタンスを生成します。
  - **Ens.DirectorのCreateBusinessService()メソッドを使用します。**
    - 第1引数： アダプタ無のビジネス・サービスのプロダクション構成名
    - 第2引数： 参照渡しで変数を指定（メソッド実行後、ビジネス・サービスのインスタンスが格納されます）

```
set st=##class(Ens.Director).CreateBusinessService("FS.NonAdatper",.bs)
```

  - **生成したビジネス・サービスインスタンスを利用して、ProcessInput()を呼び出します。**
    - ProcessInput()はインスタンスメソッドであるため、##class()のクラスメソッドの文法では呼び出せません。
    - ProcessInput()の引数には、ビジネス・サービスのOnProcessInput()で受け取りたい情報を指定します（例は、要求メッセージを作成し渡しています）。

```
4 Method OnProcessInput(pInput As FS.EmpRequest, Output pOutput As %RegisteredObject) As %Statu
5 [
6     set status=..SendRequestAsync("FS.EmpProcess",pInput)
7     Quit status
8 ]
```

←アダプタ無のビジネス・サービス

```
set request=##class(FS.EmpRequest).%New()
set request.EMPID="EMP0202"
set request.Name="佐々木五郎"
set request.Location="福井県"
set st=bs.ProcessInput(request)
```

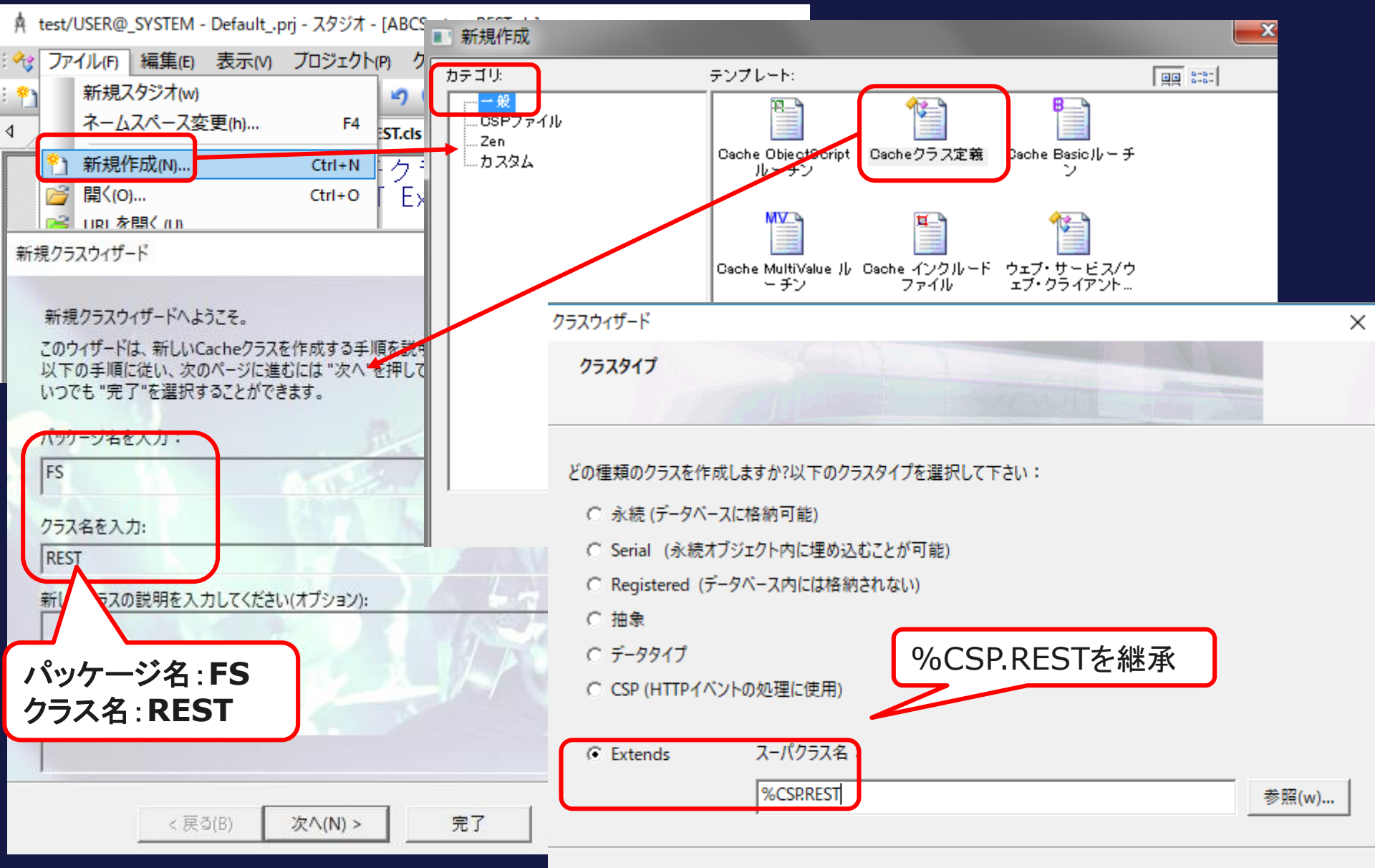
RESTディスパッチクラスの作成

- 作成概要は以下の通りです。
  1. ディスパッチクラスを作成する。
     - **REST用スーパークラス（%CSP.REST）を継承してクラス定義を作成**します。
     - REST要求で送信されるURLに対応して実行されるメソッドを指定するため、**URLMapを定義**します。
     - **URLMapに対応するメソッドを実装**します。
  2. ウェブアプリケーションパスを作成し、1で定義したディスパッチクラスを指定する。
     - ディスパッチクラスを指定したウェブアプリケーションパスはRESTサービス専用のパスとして設定されます。

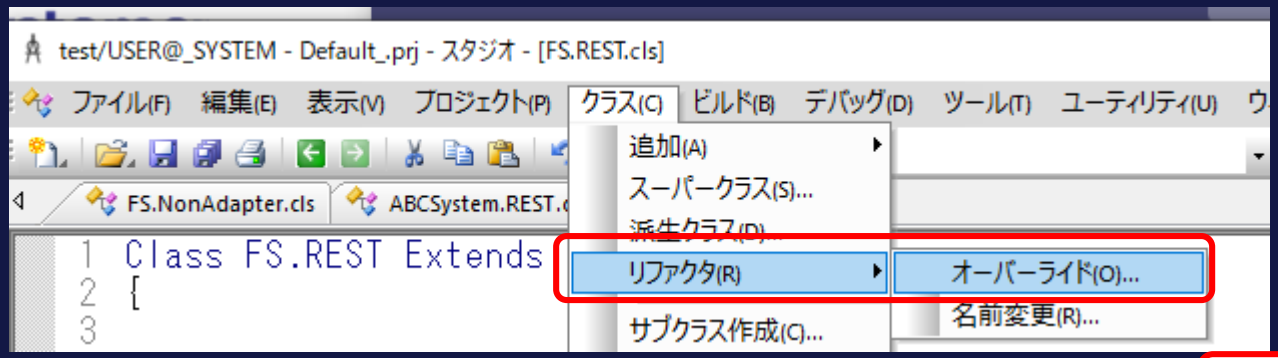
%CSP.REST

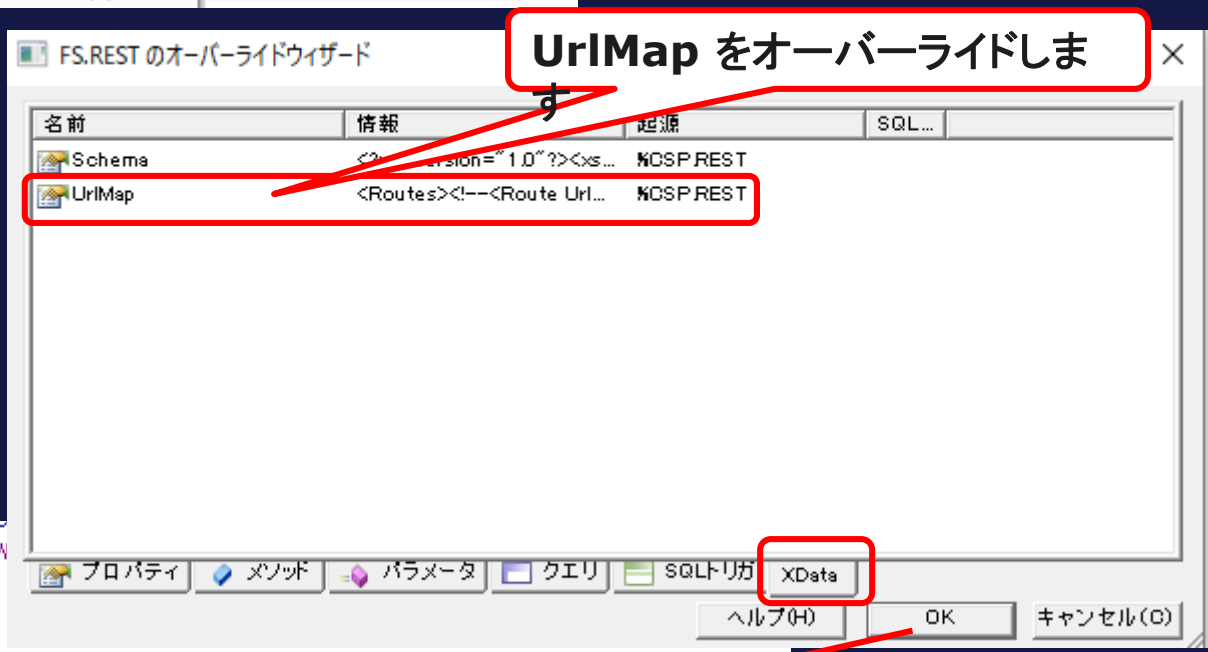
- %CSP.RESTクラスは、**REST用ディスパッチクラスのスーパークラス**で、**HTTP要求を受信し、指定されたURLに対して呼び出すメソッドの割り当てと実際の処理を記述できるクラス**です。
- クライアントから送信される**GET／POST／UPDATE／DELETEのHTTP要求は%requestオブジェクト（%CSP.Request）で操作できます。**
  - ストリームとして渡されるため、**Read()メソッド**を使用して送信データの中身を取り出します。
  - 受信データは HTTP 要求時に指定された Content-Type の charset の指定コードで変換します。
    例）set data=$ZCONVERT**(%request.Content.Read()**,"I","UTF8")
- **クライアントへ返されるHTTP応答は、%responseオブジェクト（%CSP.Response）で操作できます。**
  - 応答時のContent-Typeの指定は **%response.ContentType**="application/json" で指定できます。
  - charset は **%response.CharSet**="utf8" で指定できます。
- HTTP要求と応答のオブジェクトは、メソッドが終了すると自動消去されます。

REST用ディスパッチクラス作成方法　図解



UrlMapのオーバーライドとメソッドの作成





UrlMap をオーバーライドします

UrlMapを実装したい内容に合わせて修正します。

```
28 XData UrlMap [ XMLNamespace = "http://ww
29   {
30 <Routes>
31   <!--
32   <Route Url="/class/:namespace/:classname" Method="GET" Call="GetClass" Cors="true"/>
33   <Map Prefix="/docserver" Forward="%Api.v1.DocServer"/>
34   -->
35 </Routes>
36   }
```

メソッド：記述しやすくするためのコツ

- REST用ディスパッチクラスのメソッドを記述するとき、RESTディスパッチクラス内で自動生成されるオブジェクトに対してスタジオは入力候補を出しません。
- 入力候補があることで記述ミスを減らせるため、**#dim**を使用して専用の変数にどのクラスのインスタンスが格納されるか明示指定します。

変数	#dim	内容
%response	#dim %response As %CSP.Response [使用例] 応答時のContentTypeを指定する set %response.ContentType="application/json"	HTTP クライアントへ返される応答ヘッダ管理用
%request	#dim %request As %CSP.Request [使用例] Bodyの中身を変数にセット set body=%request.Content.Read()	GET／POSTなどの要求で送信される情報など
ex (任意名)	#dim ex As %Exception.AbstractException [使用例] 例外発生時の返送メッセージを取得する set st=ex.AsStatus()　// ステータスへ変換 set msg=$system.Status.GetErrorText(st)	例外オブジェクト

付録：JSON操作

- 開発者コミュニティ（jp.community.intersystems.com）に、JSON操作についての解説ビデオもあります。ぜひご参照ください。
- その他
  - コースでは、スタジオを利用してRESTディスパッチクラスを作成する方法を解説していますが、APIファーストで作成するディスパッチクラスもあります。
  - 詳細は、開発者コミュニティにある「APIファーストで作成するRESTディスパッチクラス」（解説ビデオ）をご参照ください。