「Web拡張、クラウド対応も万全！つなぐシステム構築法」

第36回 エンバカデロ・デベロッパーキャンプ

エンバカデロ・テクノロジーズ
セールスコンサルタント

井之上 和弘

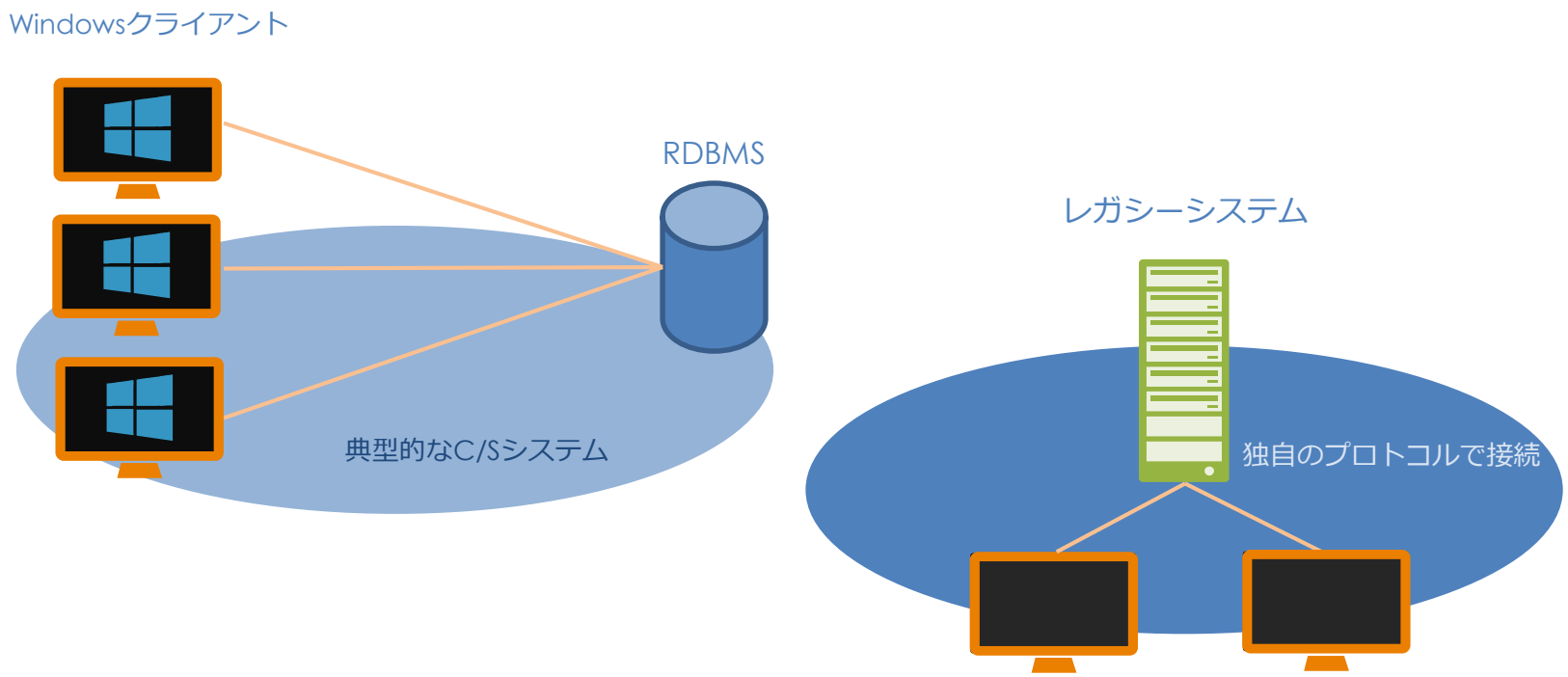
embarcadero®
DEVELOPER CAMP

アジェンダ

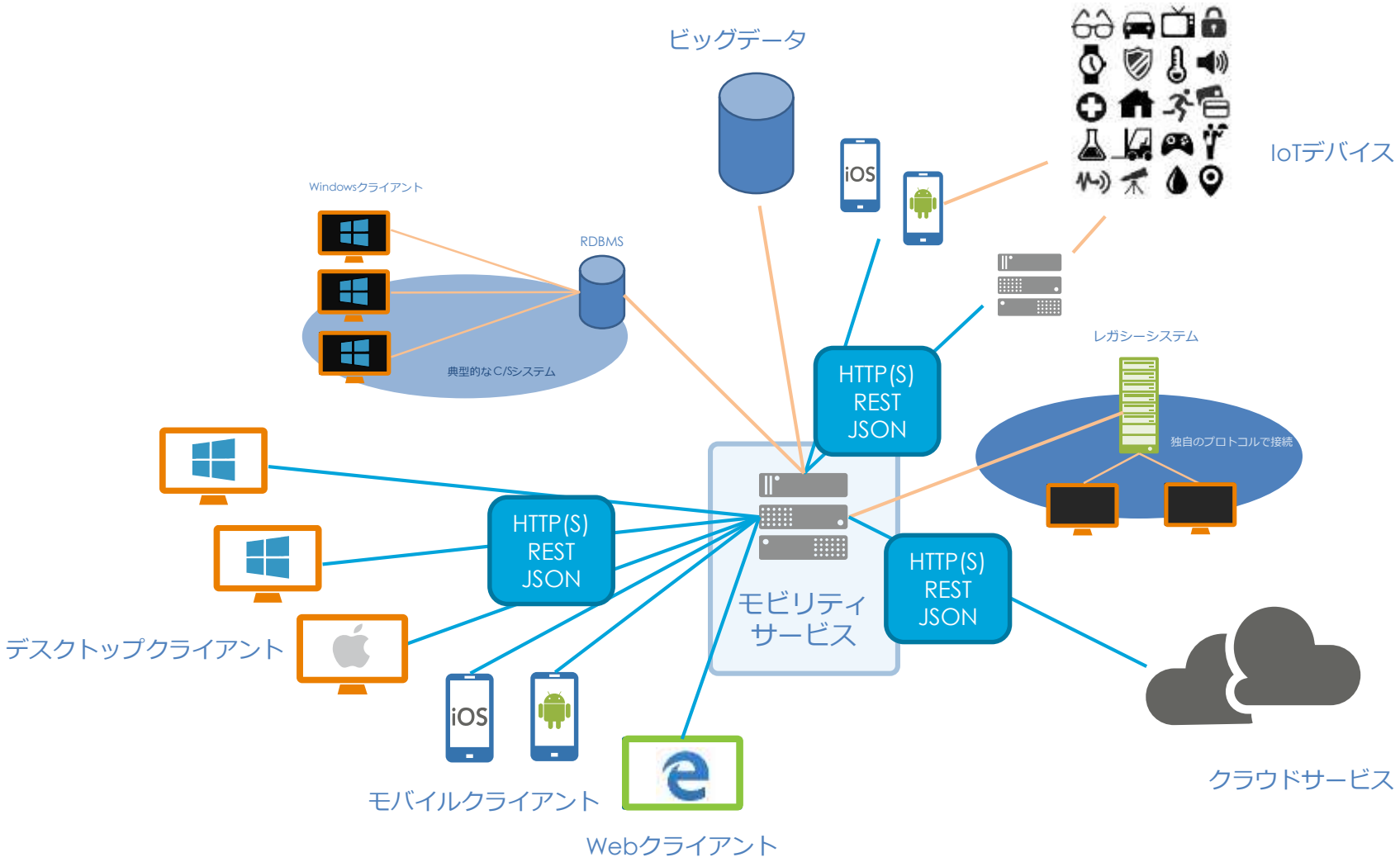
- 今までのソフトウェア資産を新しく刷新してほしい、という要望に対し、どのような方法で実現すればよいか、というプレッシャーがあります
- 既存の資産を捨てて新たに作り直すという選択肢もありますが、それは本当に正しい選択でしょうか？
- 既存のアプリケーション資産を活かすことができれば、工数・コスト削減につながります
- 刷新に伴うWeb拡張やクラウドとの連携による「つなぐシステム」について解説します

既存のC/S = 拡張性が低い隔離されたシステム

- クライアントは特定のデータソースと通信するのみ
- データソースごとにクライアント側の個別実装が必要



つなぐシステム = さまざまなデータソースとつながる



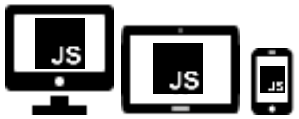
つなぐシステム = さまざまなクライアントからつながる

- ネイティブアプリ

VCL Windows
アプリケーション

FireMoneyマルチデバイス
アプリケーション

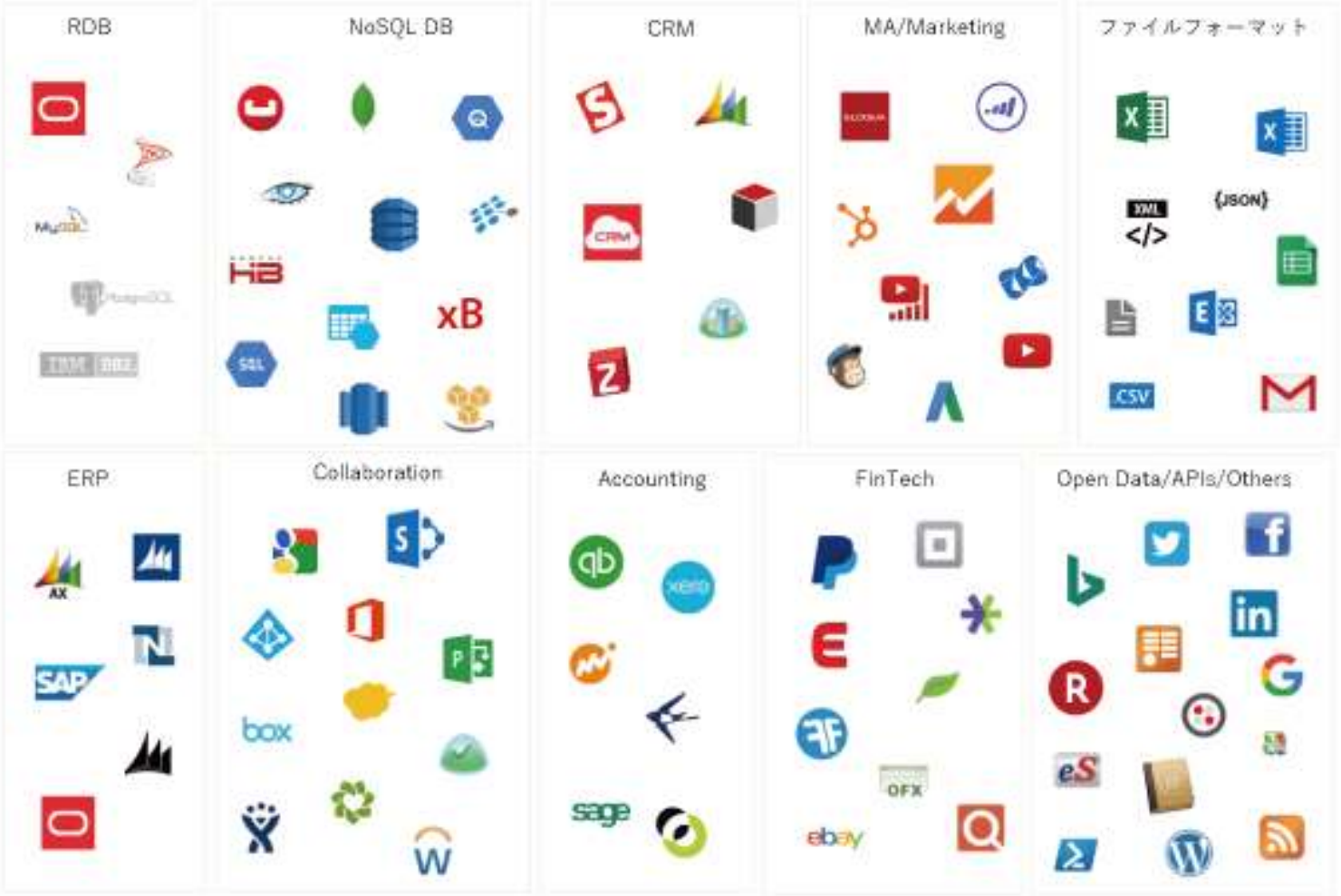
- HTML5/JavaScript アプリ



Sencha ExtJS
Webアプリケーション

- その他のWebクライアント

つなぐシステム = RDBMS以外のデータソースとつなぐ



C/Sと、つなぐシステムの違い

- C/S は所定のデータベースサーバにつなぐのみ

- つなぐシステムは…
  - つながるためのデバイスや時間、場所を選ばない

  - ネイティブアプリからエンタープライズクライドサービスにつなぐ
  - Webアプリからエンタープライズクライドサービスにつなぐ

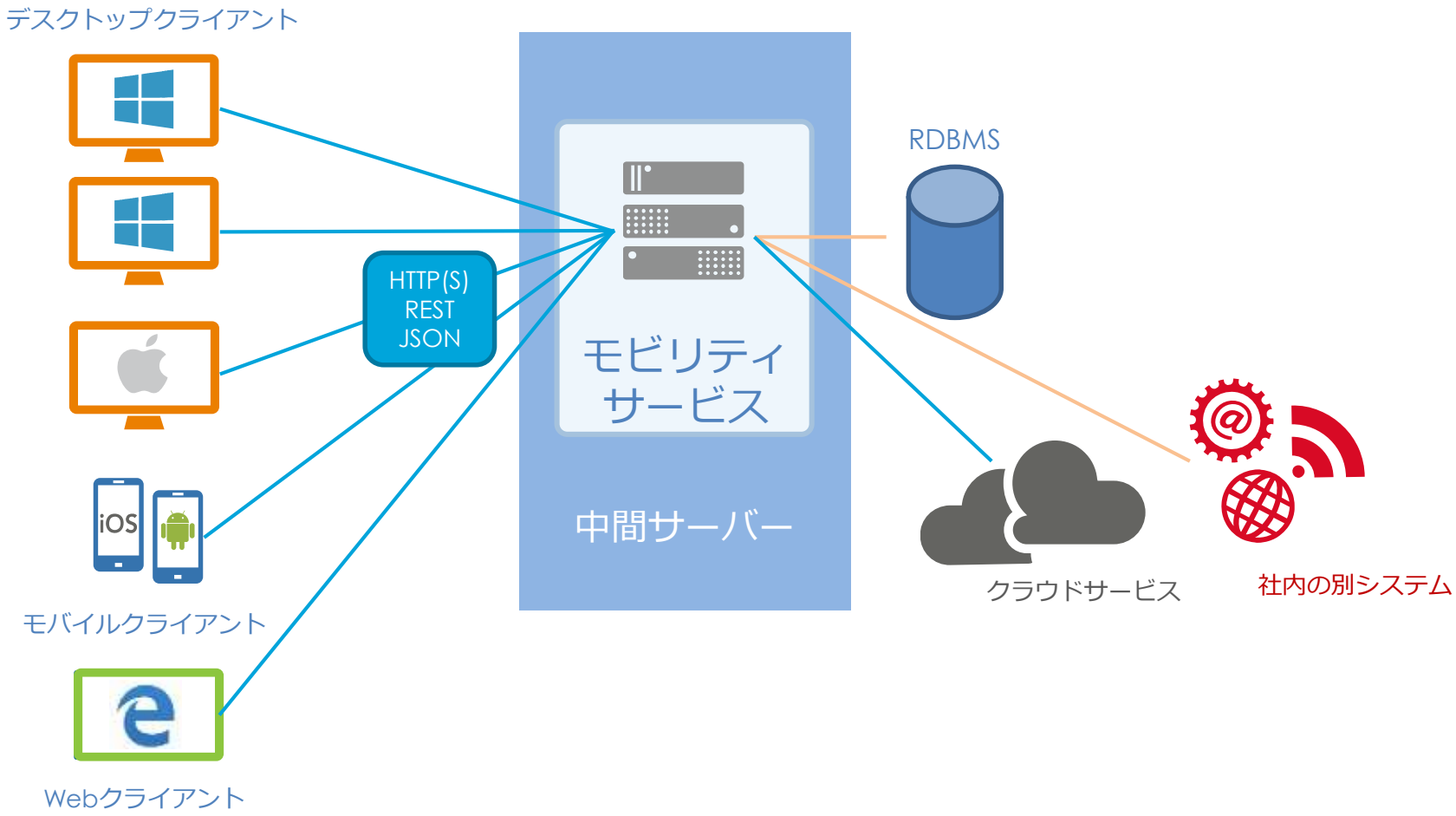
  - コマンドラインツールでバッチ処理的につなぐ

DEMO

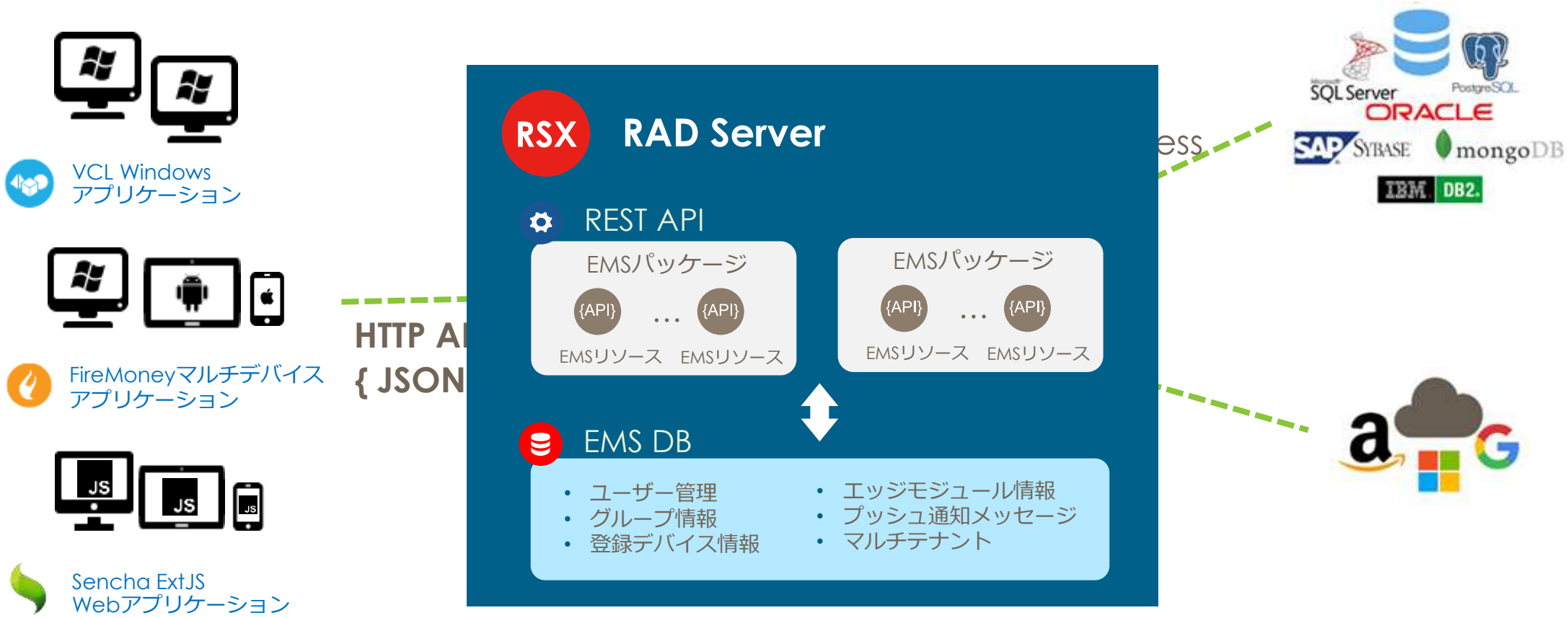
つなぐシステムに必要なこと

embarcadero®
DEVELOPER CAMP

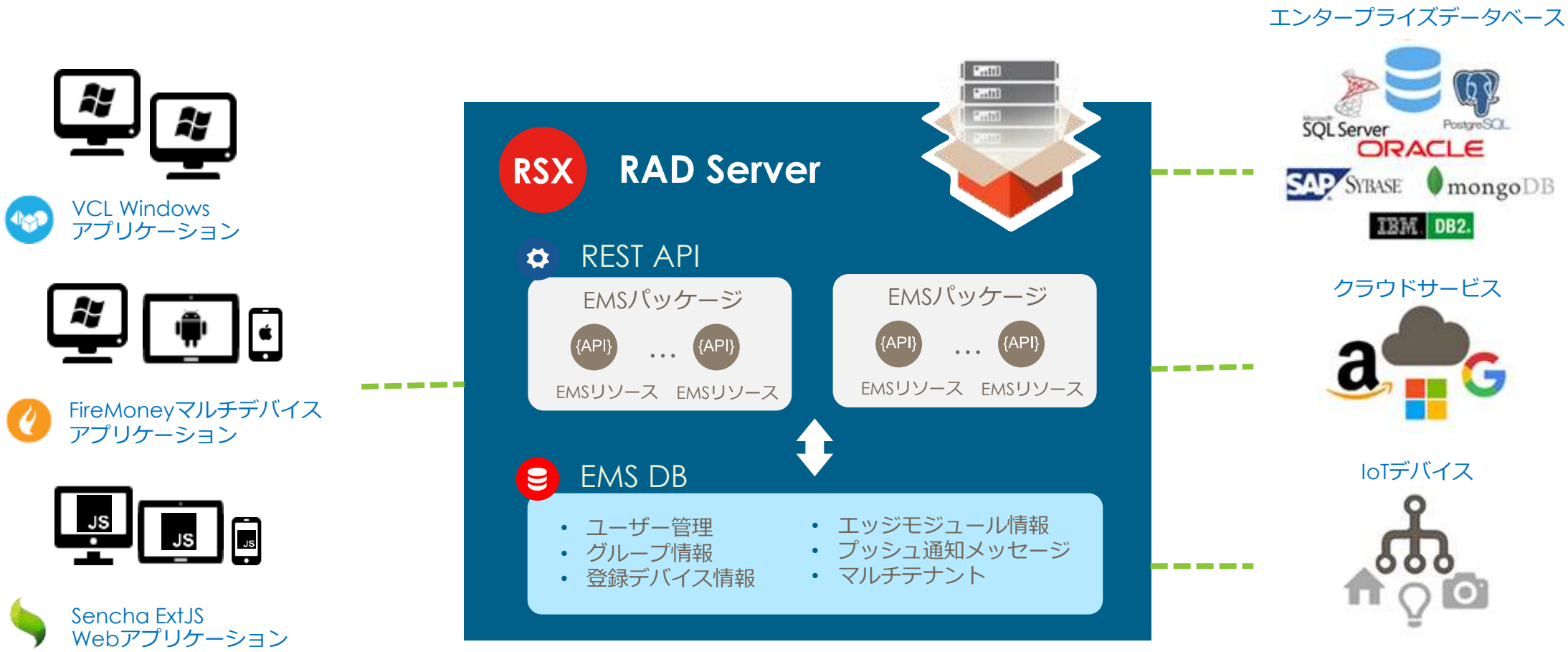
つなぐシステムは多層化で実現



つなぐシステムはオープンなAPIとフォーマットを使う



RAD Server = Delphi/C++Builder での多層化のハブ



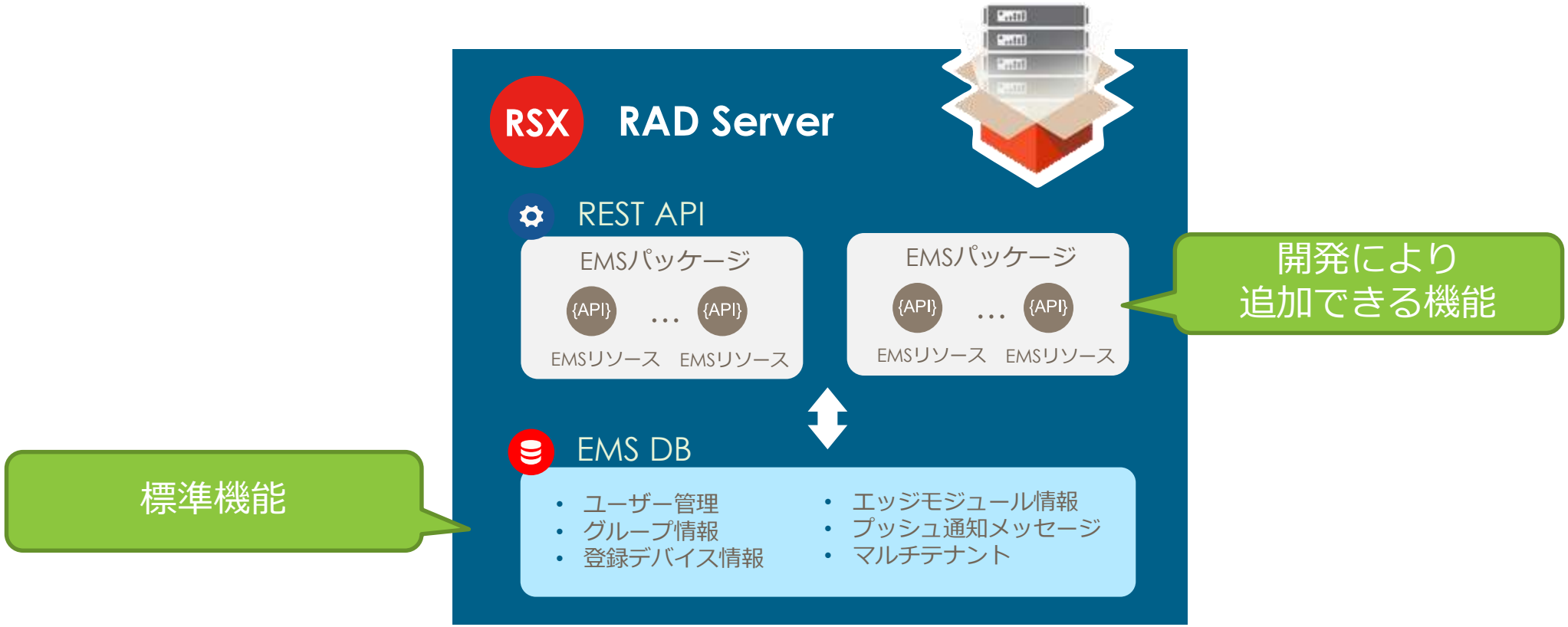
つなぐシステムのハブ
RAD Serverを理解する

embarcadero®
DEVELOPER CAMP

RAD Serverについて理解しておくべきポイント

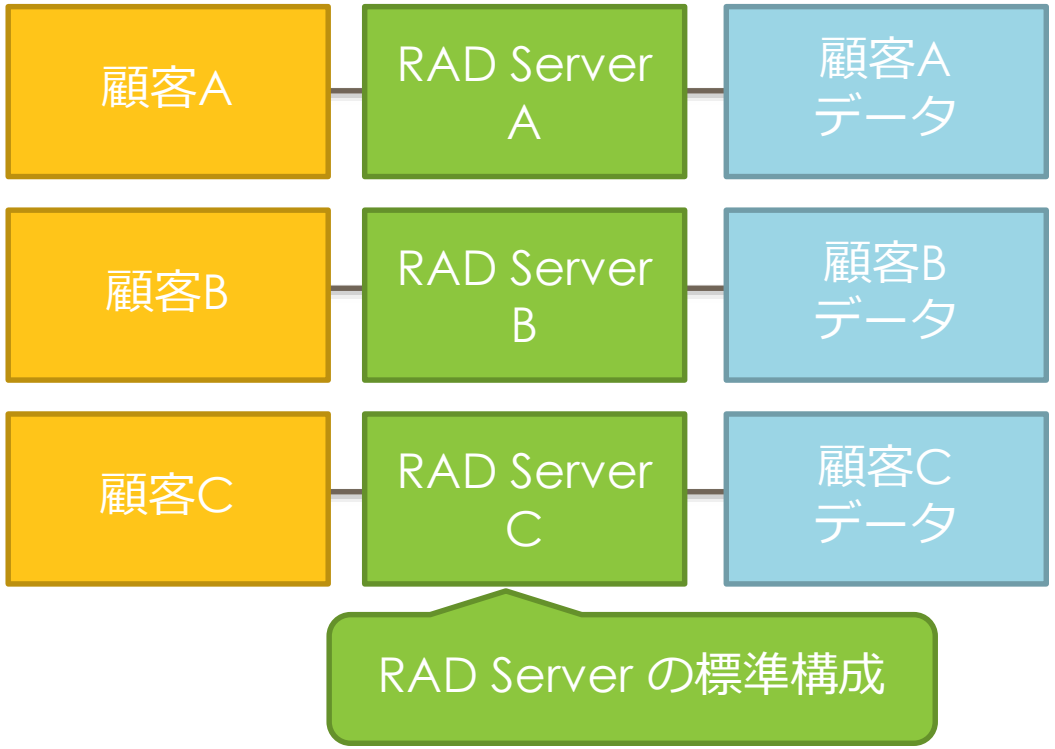
- 標準機能と開発で追加できる機能
- マルチテナント機能
- 開発ライセンス、配置ライセンス（Single, Multi の違い)
- 用途によるマルチテナント、ライセンスの選定

標準で含まれる機能、開発により追加できる機能

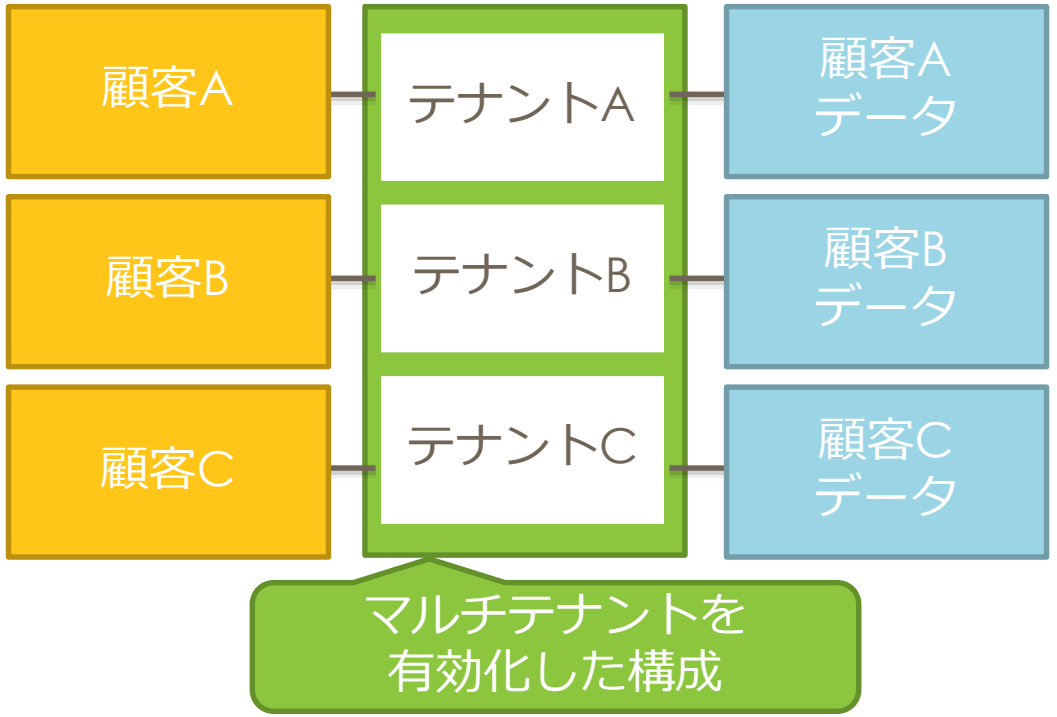


RAD Serverのマルチテナントでできること

- シングルテナント = 組織ごとに独立したサーバでサービス利用する



- マルチテナント = 同じシステムやサービスを複数の組織で共有する

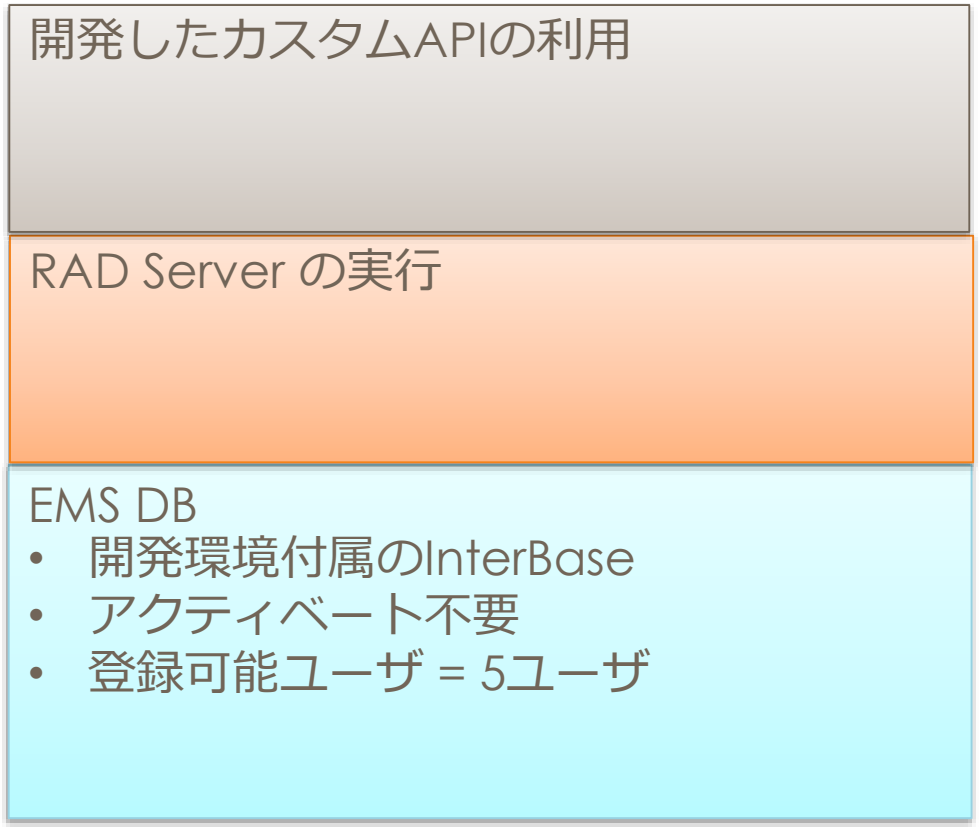


開発ライセンス、配置ライセンス（Single, Multi の違い)

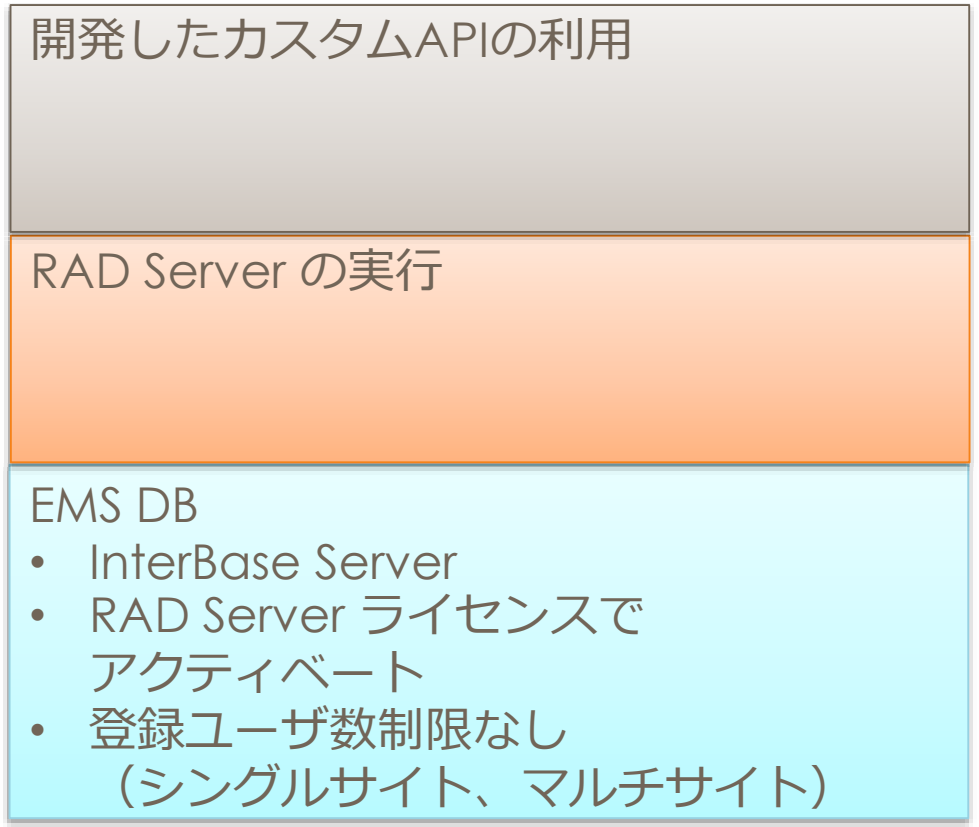
- 開発ライセンス
  - 開発環境でRAD Serverを実行可能
- 配置ライセンス
  - 基本：RAD Server を運用する機材ごとに Single Site License
  - Multi Site License では 1 ライセンスで複数サーバを運用可能

開発環境向けの構成と、本番向けの構成の違い

- 開発環境向け構成



- 運用環境向け構成



開発環境向けの構成と、本番向けの構成の違い

- 開発環境向け構成

- 運用環境向け構成

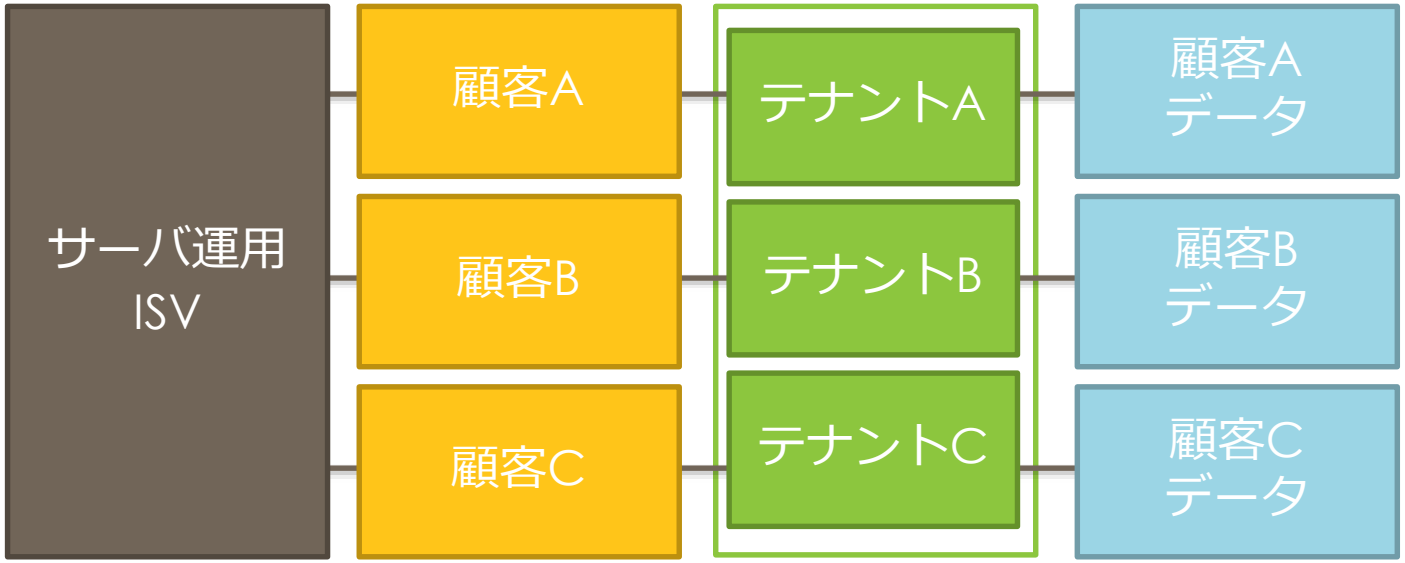
開発環境向けの構成と、本番向けの構成の違い

- 開発環境向け構成

- 運用環境向け構成

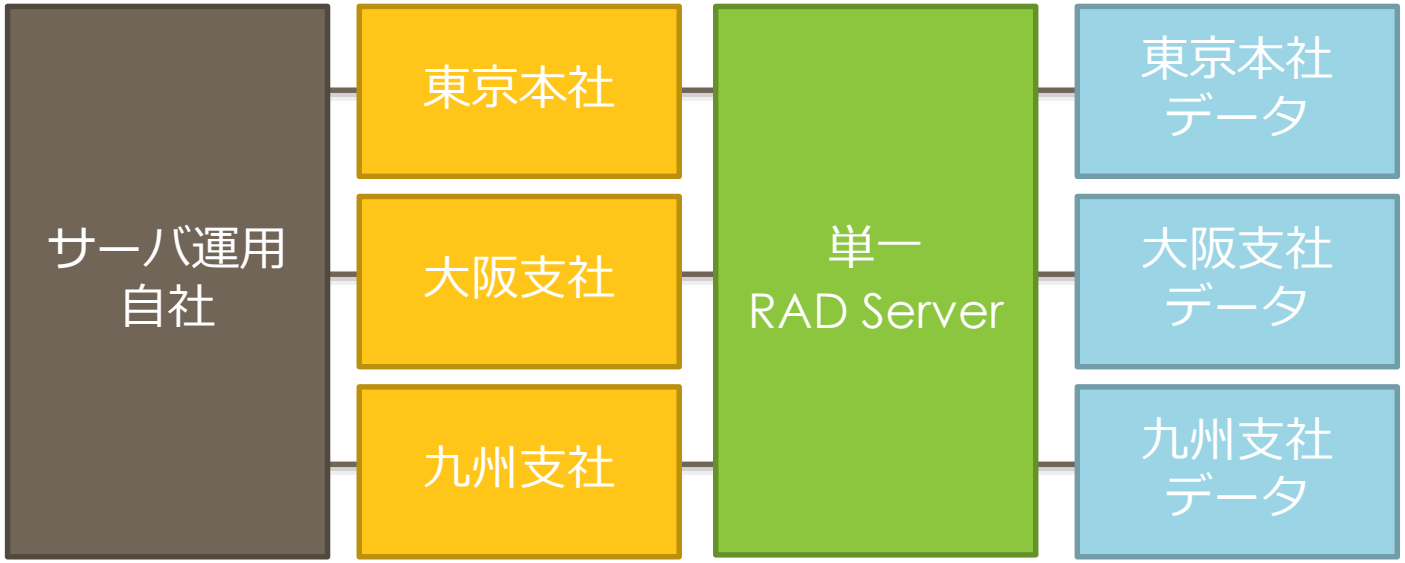
用途によるマルチテナント、ライセンスの選定
ケース１：自社開発したアプリでASPサービス運用

- 1台のRAD Serverでマルチテナント運用
- シングルサイト・ライセンス（冗長性を考慮しない場合）



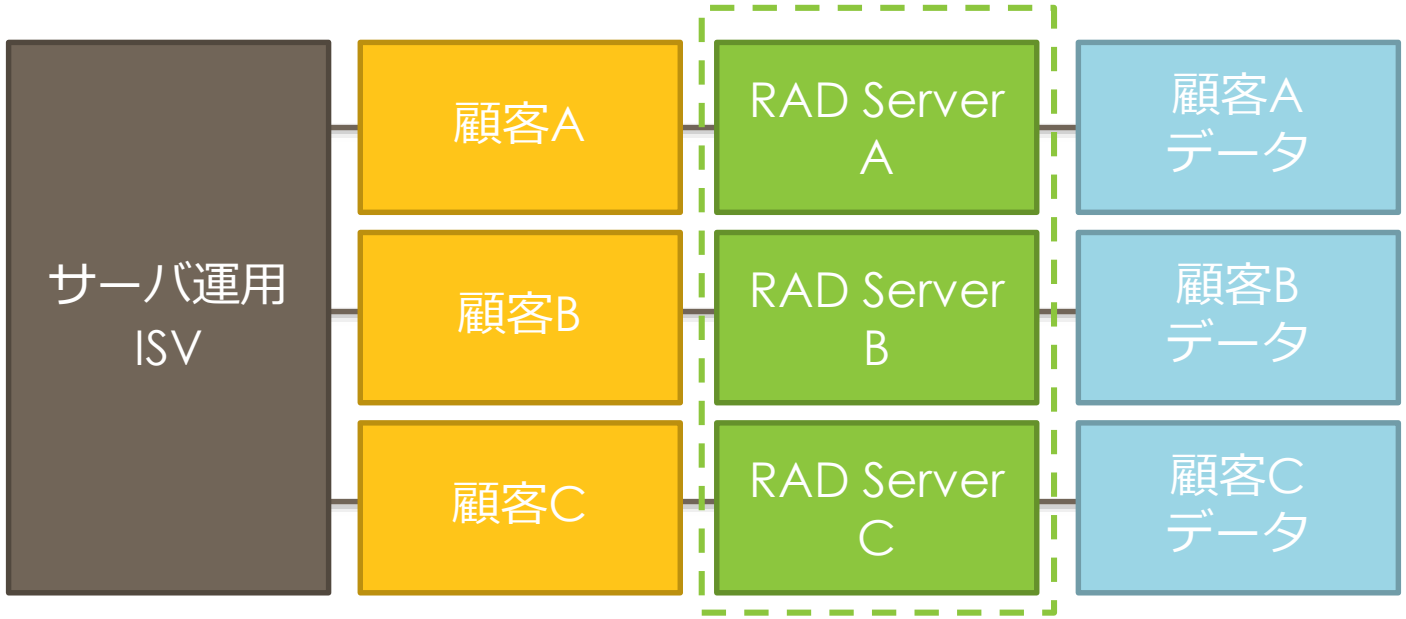
用途によるマルチテナント、ライセンスの選定
ケース２：自社の業務アプリを単一のRAD Serverで提供する

- 複数のオフィスや異なる拠点向けのサービスを
  １台のRAD Serverで運用する
- シングルサイトライセンス
- 必要に応じてマルチテナント機能も利用可能



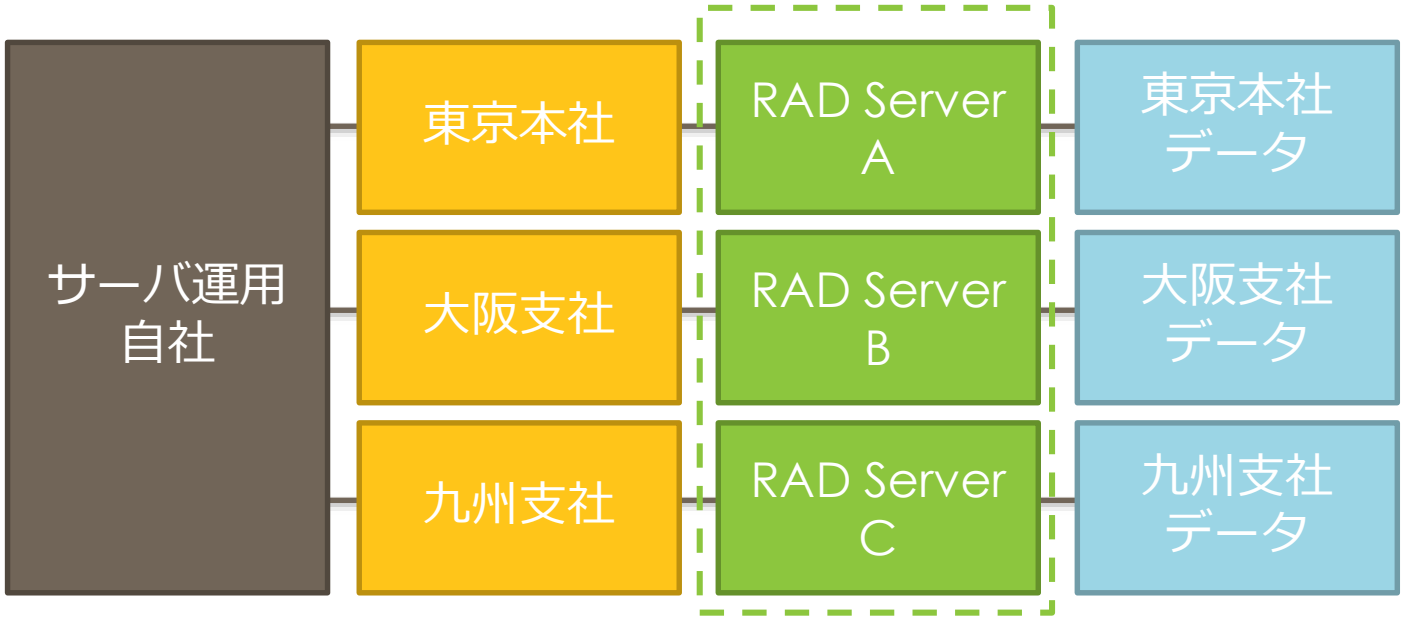
用途によるマルチテナント、ライセンスの選定
ケース３：ASPサービスを顧客ごとに別サーバで運用

- サーバごとにシングルテナントで運用
- マルチサイト・ライセンス
- サーバ台数分のシングルサイト・ライセンス



用途によるマルチテナント、ライセンスの選定
ケース４：自社の複数拠点ごとに個別のサーバを運用

- サーバごとにシングルテナントで運用
- マルチサイト・ライセンス
- サーバ台数分のシングルサイト・ライセンス



つなぐシステムのAPI実装

embarcadero®
DEVELOPER CAMP

RAD ServerでのAPI実装の基本（従来の方法）

- ResourceName でリソース名を指定

```
Type
  [ResourceName('testResource')]
  TTestResourceResource1 = class(TDataModule)
  published
    // GET http://hostname/testResource/customer
   [ResourceSuffix('customer')]
   procedure Get(const AContext: TEndpointContext; ...);

    // POST http://hostname/testResource/customer/{item}
   [ResourceSuffix('customer/{item}')]
   procedure PostItem(const AContext: TEndpointContext; ...);
  end;
```

RAD ServerでのAPI実装の基本（従来の方法）

- ResourceName でリソース名を指定
- ResourceSuffix で個々のリソースを細かく定義

```
Type
  [ResourceName('testResource')]
  TTestResourceResource1 = class(TDataModule)
  published
    // GET http://hostname/testResource/customer
    [ResourceSuffix('customer')]
    procedure Get(const AContext: TEndpointContext; ...);

    // POST http://hostname/testResource/customer/{item}
    [ResourceSuffix('customer/{item}')]
    procedure PostItem(const AContext: TEndpointContext; ...);
  end;
```

RAD ServerでのAPI実装の基本（従来の方法）

- ResourceName でリソース名を指定
- ResourceSuffix で個々のリソースを細かく定義
- 個々の procedure 名で Get, Post, Put, Delete などのメソッドを指定する

```
Type
  [ResourceName('testResource')]
  TTestResourceResource1 = class(TDataModule)
  published
     // GET http://hostname/testResource/customer
    [ResourceSuffix('customer')]
    procedure Get(const AContext: TEndpointContext; ...);

     // POST http://hostname/testResource/customer/{item}
    [ResourceSuffix('customer/{item}')]
    procedure PostItem(const AContext: TEndpointContext; ...);
  end;
```

RAD ServerでのAPI実装（RAD Studio 10.3からの機能）

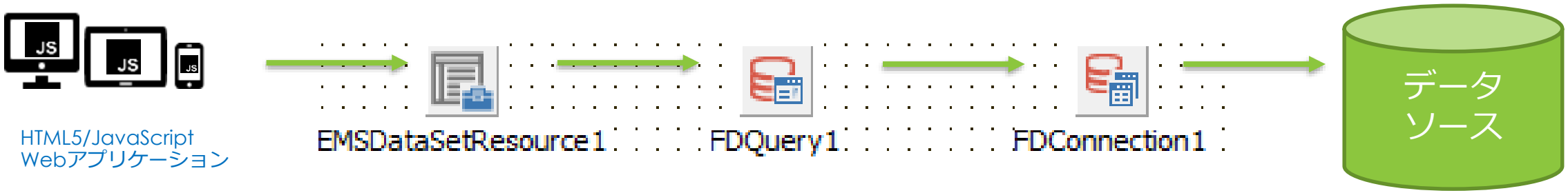
- カスタムメソッド名へのHTTP動詞マッピング

```
// 10.2.3 まで：procedure 名に Get, Post, Put, Delete と明記する
type
 [ResourceName('Test')]
 TTestResource = class(TDataModule)
 public
    procedure GetPrintEmps(const AContext: TEndpointContext;
     const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
 end;
```

```
// 10.3 ： 任意の procedure に EndpointMethod でメソッドを割当可能
type
 [ResourceName('Test')]
 TTestResource = class(TDataModule)
 public
   [EndpointMethod(TEndpointRequest.TMethod.Get)]
   procedure PrintEmps(const AContext: TEndpointContext;
     const ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
 end;
```

RAD Serverでのレスポンス作成（RAD Studio 10.3からの機能）

- **TEMSDataSetResource**
  - リソースに対する Get, Post, Put, Delete 操作をサポートするコンポーネント
  - リクエストではページ単位の取得や指定したキーでのソートをサポート
  - レスポンスではJSON配列を返す
  - API実装をコーディング無しでも実装できる

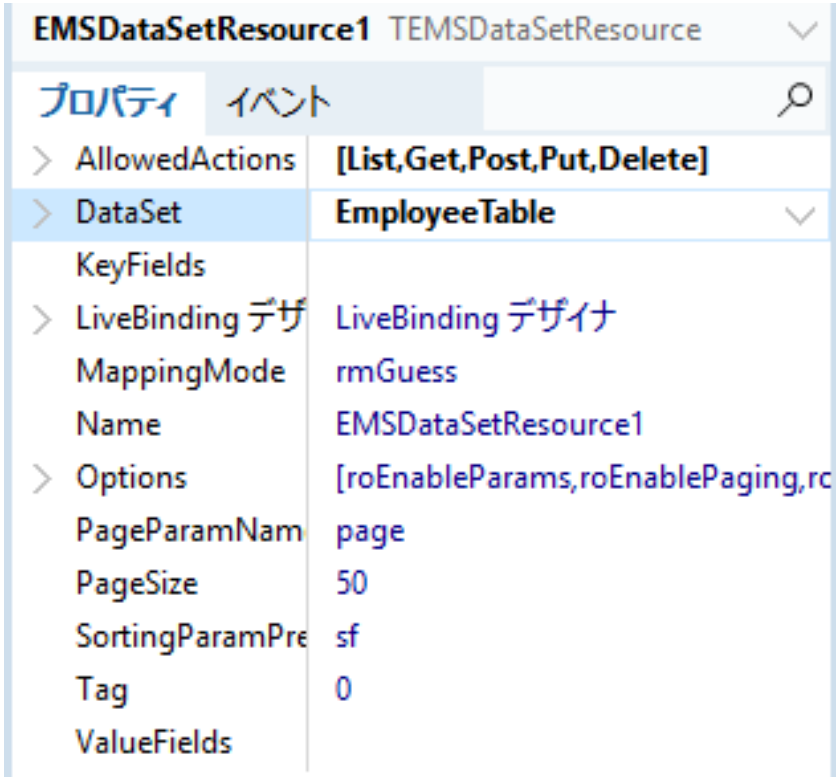


RAD Serverでのレスポンス作成（RAD Studio 10.3からの機能）

- TEMSDataSetResource の設定
  - **DataSet**にFDQueryを割り当てる
  - **AllowedActions** の設定でデータベースのCRUD操作がDataSetに行われる

AllowedActions	CRUD操作
List	SELECT
Get	SELECT where …
Post	INSERT
Put	UPDATE where …
Delete	DELETE where …

  - **KeyFields** や **ValueFields** プロパティでAPI呼び出しパラメータやJSONレスポンスに含めるフィールドを指定可能



RAD Serverでのレスポンス作成（RAD Studio 10.3からの機能）

- TEMSDataSetResource のコード実装
  - **EMSDataSetResource** に **ResourceSuffix** を割り当てる

```
// 個々のHTTPメソッドとパラメータを明示して
// 割り当てる
type
  [ResourceName('testResource')]
  TTestResourceResource1 = class(TDataModule)
    EmployeeConnection: TFDConnection;
    EmployeeTable: TFDQuery;

    [ResourceSuffix('list', '/')]
    [ResourceSuffix('get', '/{id}')]
    [ResourceSuffix('post', '/')]
    [ResourceSuffix('put', '/{id}')]
    [ResourceSuffix('delete', '/{id}')]
  EMSDataSetResource1: TEMSDataSetResource;
  published
  end;
```

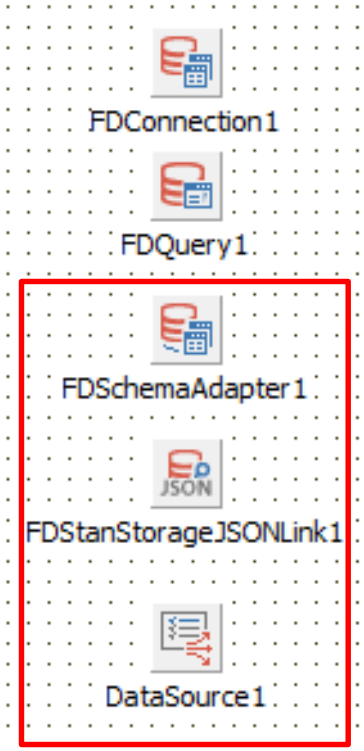
```
// パラメータが {id} の場合は
// 個別のメソッドやパラメータは省略可能
type
  [ResourceName('testResource')]
  TTestResourceResource1 = class(TDataModule)
    EmployeeConnection: TFDConnection;
    EmployeeTable: TFDQuery;

    [ResourceSuffix('/')]
  EMSDataSetResource1: TEMSDataSetResource;
  published
  end;
```

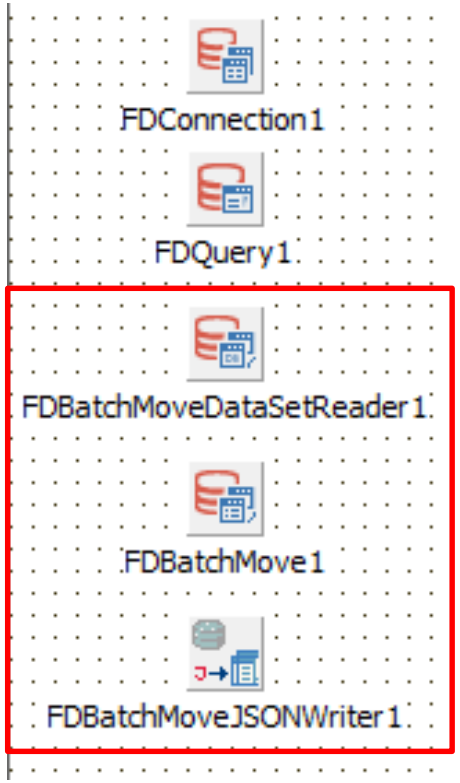
DEMO

RAD Serverでのレスポンス作成（RAD Studio の従来の方法）

- Delphi/C++Builderネイティブクライアント向けJSON



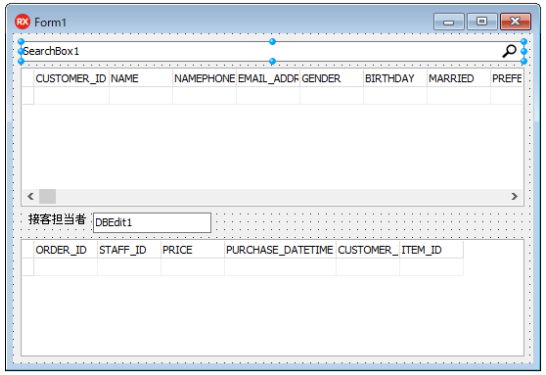
- HTML5/JavaScript 向け汎用



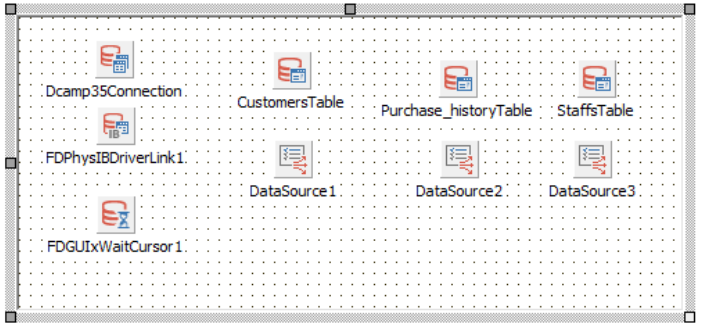
DEMO

既存のアプリケーション資産を活かす

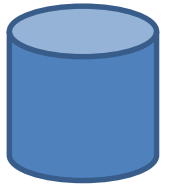
- 表示、ビジネスロジック、データアクセスを分離して多層化

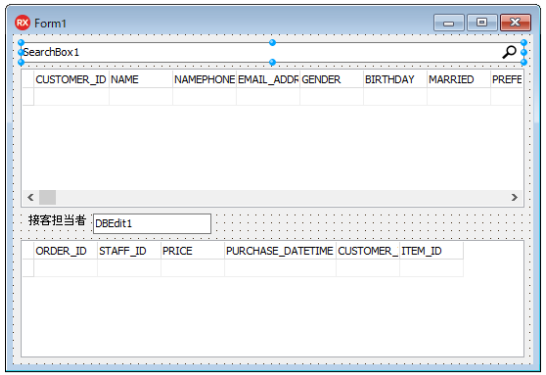


データモジュール
参照

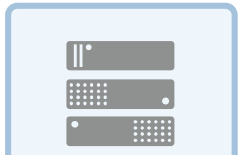


FireDACで
データアクセス

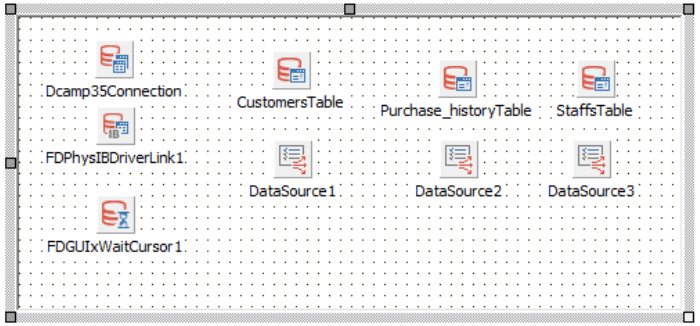




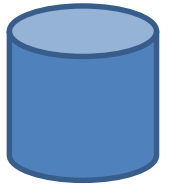
HTTP API
{ JSON }



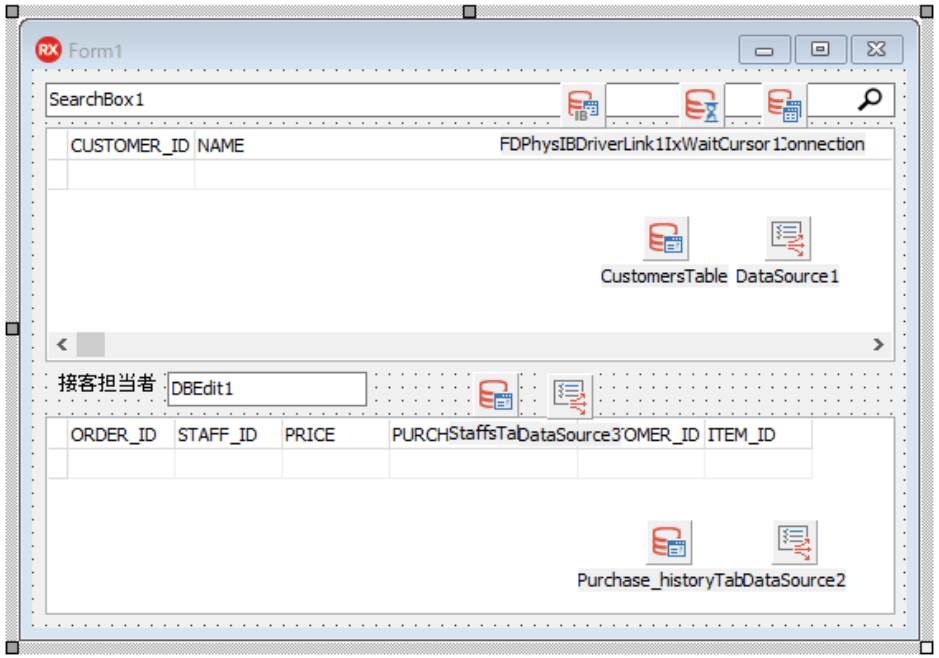
モビリティ
サービス

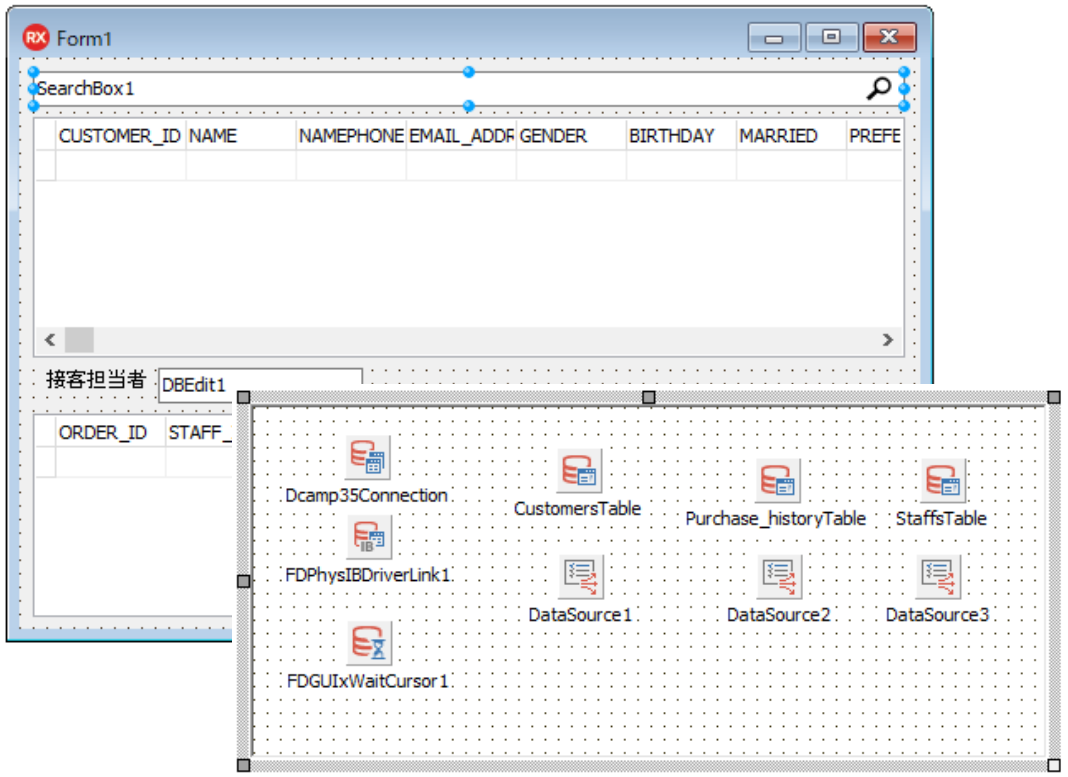


FireDACで
データアクセス



データアクセスとビジネスロジックはデータモジュールに分離



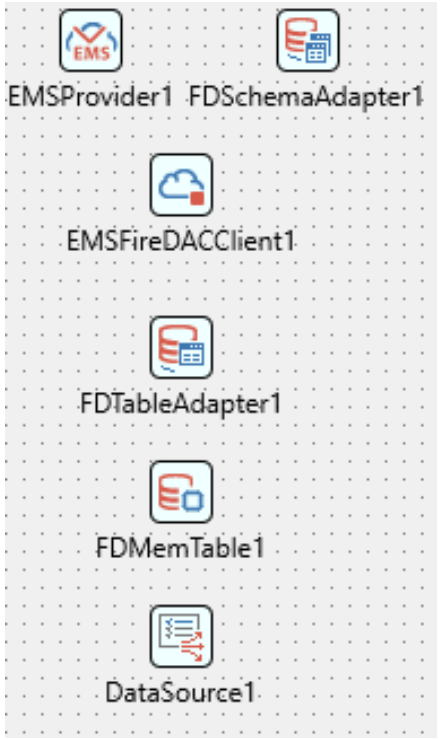


つなぐシステムへの
さまざまなクライアントからの接続

embarcadero®
DEVELOPER CAMP

Delphi/C++Builderのネイティブアプリからつなぐ

- クライアント側は



RAD Server接続と
スキーマアダプタ

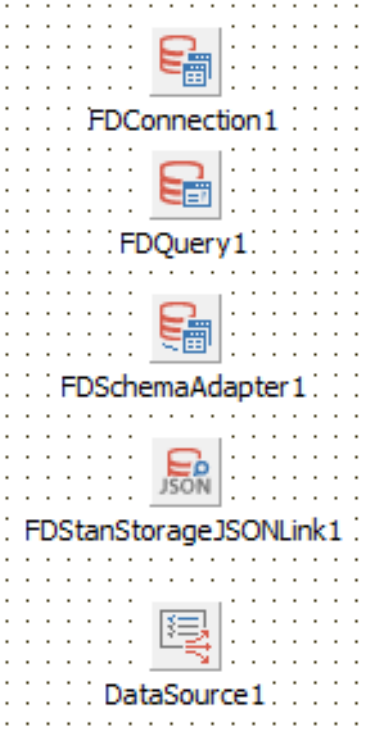
FireDAC形式JSON処理用

テーブルアダプタ

データの一時的なキャッシュ

TDataSource

- サーバ側は



データベース接続

クエリ実行

スキーマアダプタ

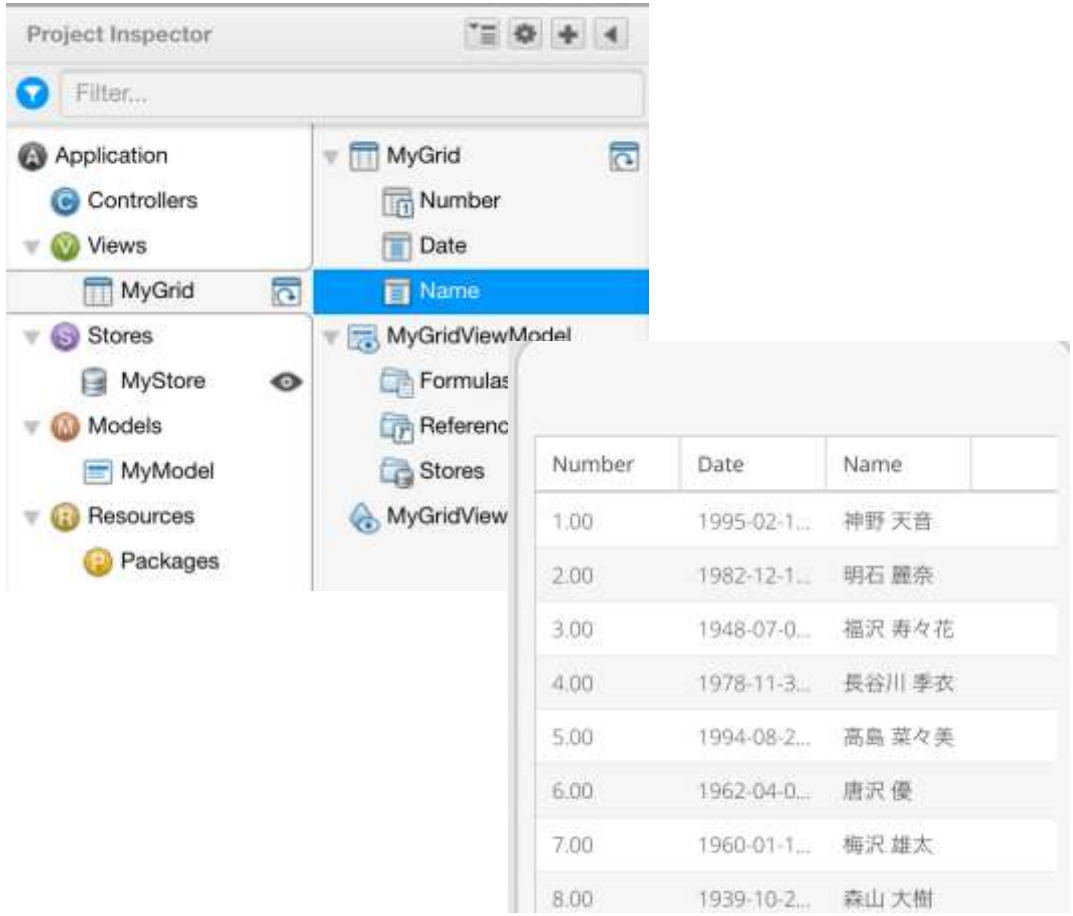
JSONLink

TDataSource

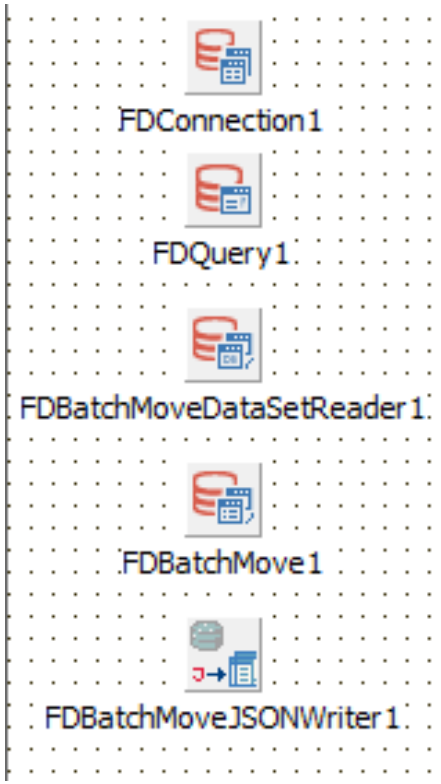
DEMO

Sencha Ext JS（HTML5/JavaScript）のWebアプリや任意のクライアントからつなぐ

- クライアント側は



- サーバ側は



データベース接続

クエリ実行

FDBatchMoveで
FDQueryの結果を
JSONに変換する

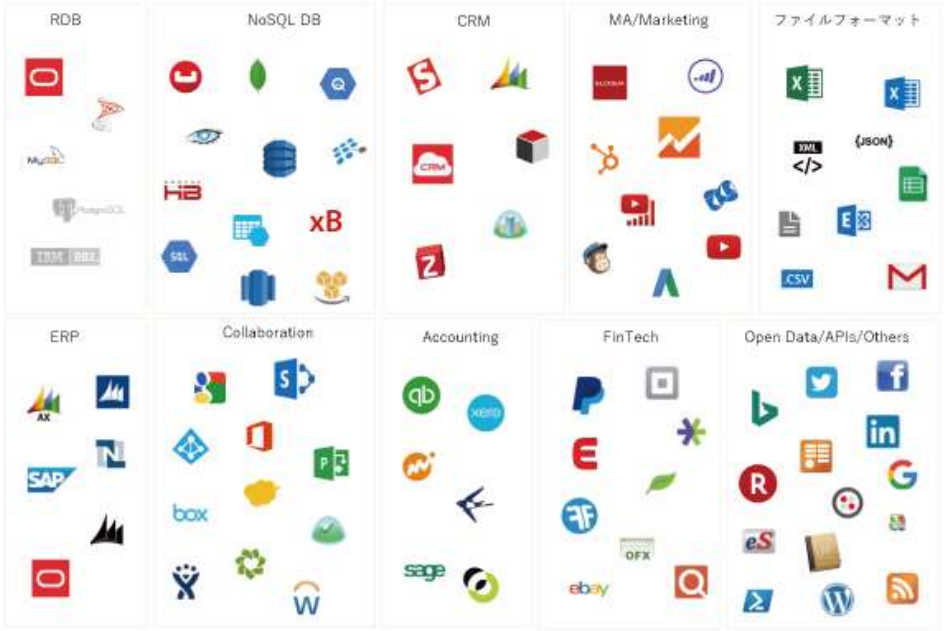
DEMO

つなぐシステムで
クラウドサービスを利用する

embarcadero®
DEVELOPER CAMP

汎用データライブラリの重要性

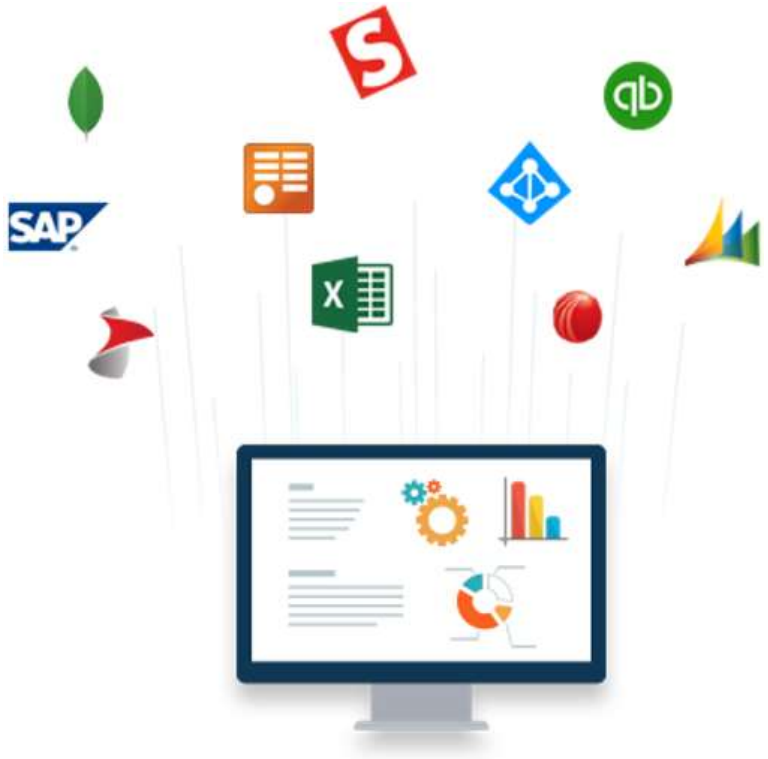
- 多様化するデータソース
  - ローカルDB、RDBMS、NoSQL、クラウドDB、クラウドサービス／エンタープライズサービス（ERP、SFAなど）
  - オンプレミス、クラウド、分散、モバイル、IoT…
- 異なるデータソースに対して異なる接続方式では…
  - データソース変更に伴う追加工数、追加の学習コスト
  - アプリケーションとデータの分離性が低くメンテナンス効率が悪い



そこで汎用的で多様なシーンで利用できるデータライブラリが必要とされてきました

クラウドサービスと容易に連携できる機能を提供

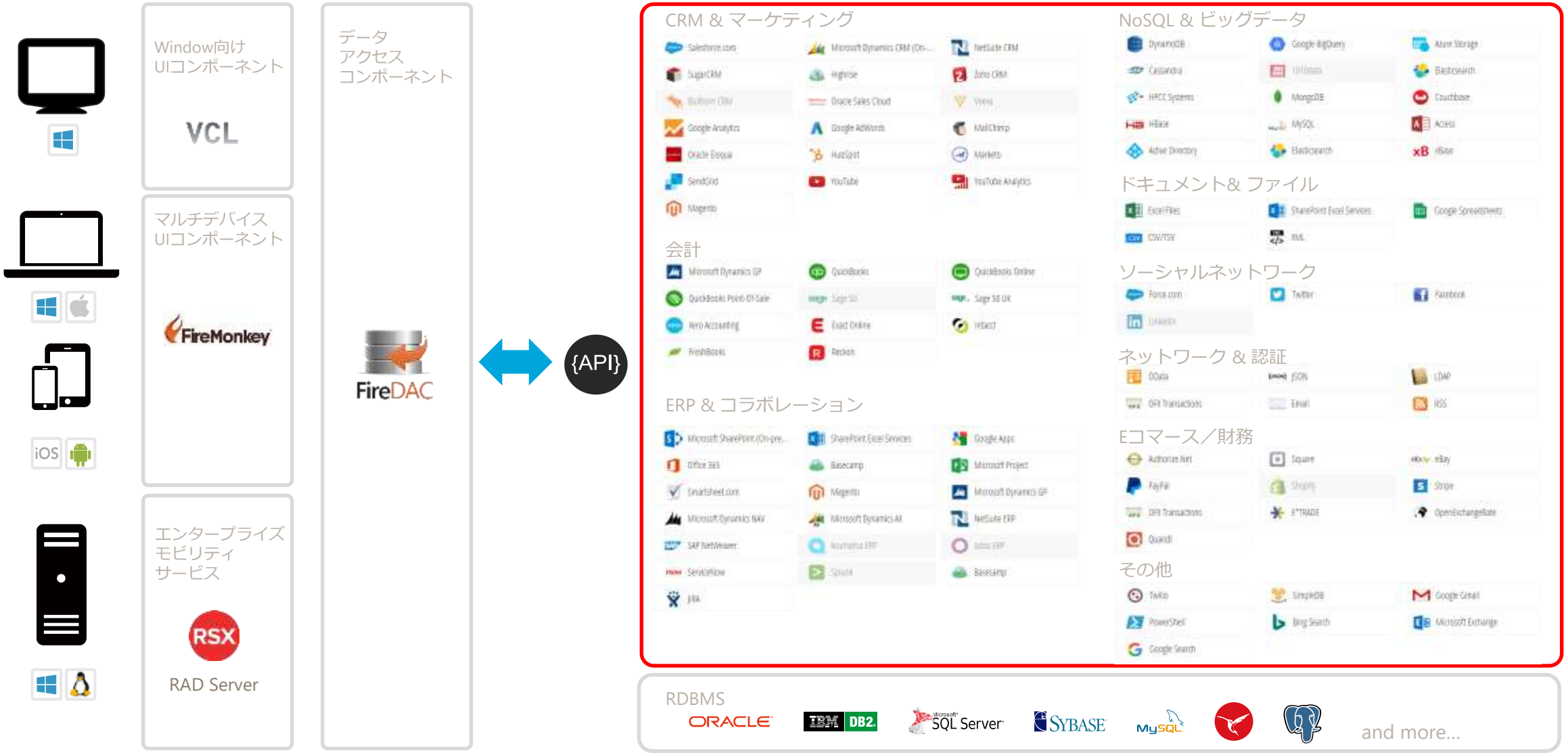
- Enterprise Connectors
  - エンタープライズデータソースへのアクセス性を提供するコンポーネント



Delphi / C++Builderの標準データアクセスフレームワーク「FireDAC」により多様なエンタープライズデータ／クラウドサービスに接続可能

- コンポーネントによる共通アクセス
- TDataSet、TFieldなどのデータ型をそのまま利用可能
- DBコントロールで表示、編集
- データエクスプローラ、フィールドエディタなどを利用可能

FireDACによる共通アクセスはRDBMS以外にも拡張



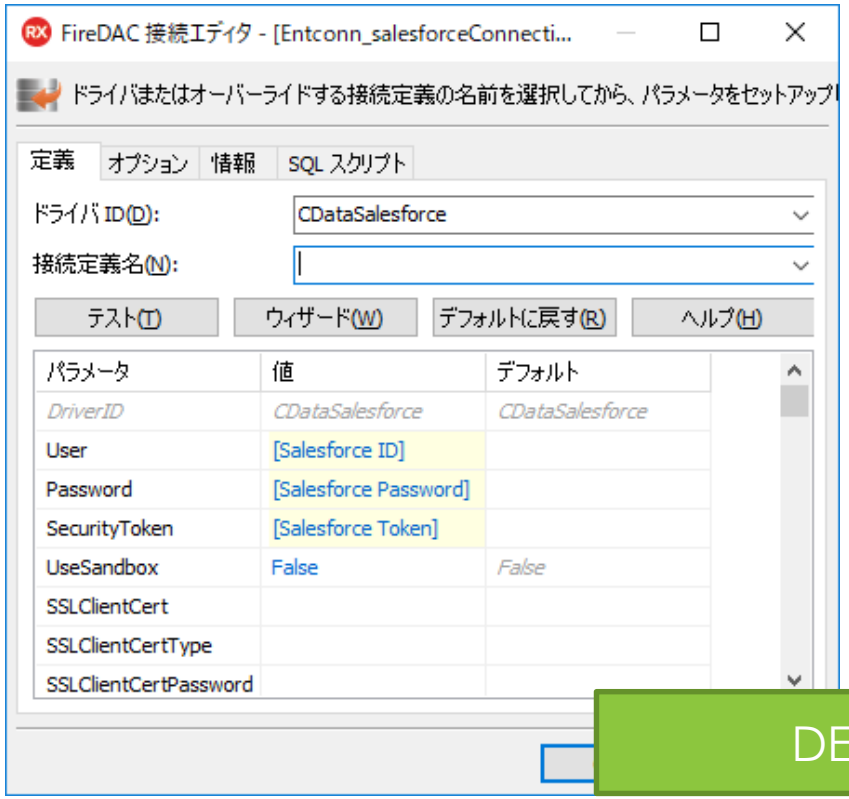
個別のAPIを学習することなくRAD開発で利用できる

- 本来は個々のサービスのAPIをネイティブに使う必要がある
  - 例：Salesforce API

```
POST https://login.salesforce.com/services/Soap/c/40.0 HTTP/1.1
Host: login.salesforce.com
Authorization: Basic xxxxxxx
Content-Type: text/xml; charset=utf-8
SOAPAction: urn:enterprise.soap.sforce.com/login

<?xml version="1.0" encoding="utf-8"?>
<SOAP-ENV:Envelope>
<SOAP-ENV:Body>
   <login xmlns="urn:enterprise.soap.sforce.com">
     <username>[Salesforce ID]</username>
     <password>[Salesforce Password]</password>
   </login>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope
```

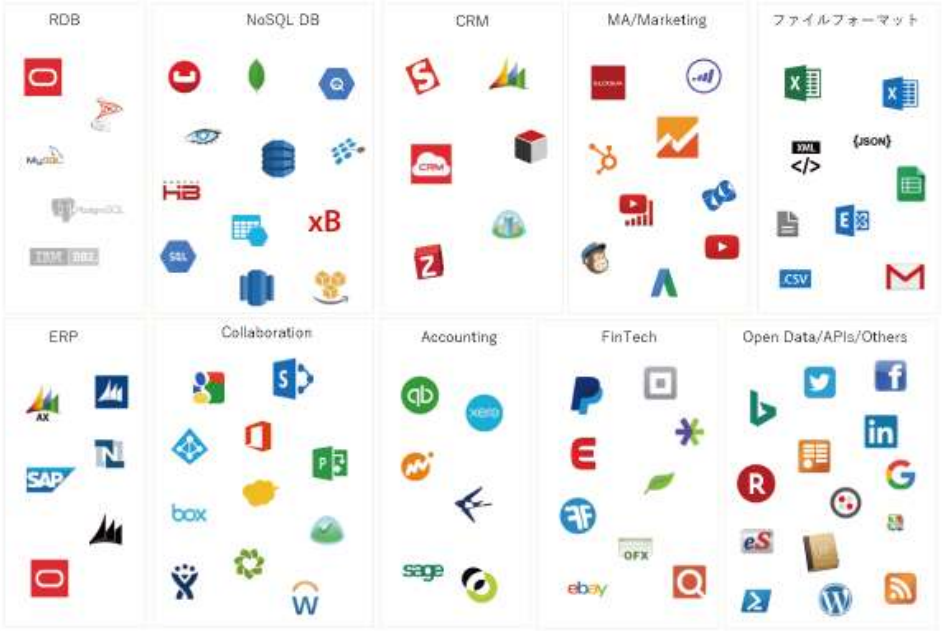
- Enterprise Connectors なら FireDAC の DB接続と同じ方法で利用できる



DEMO

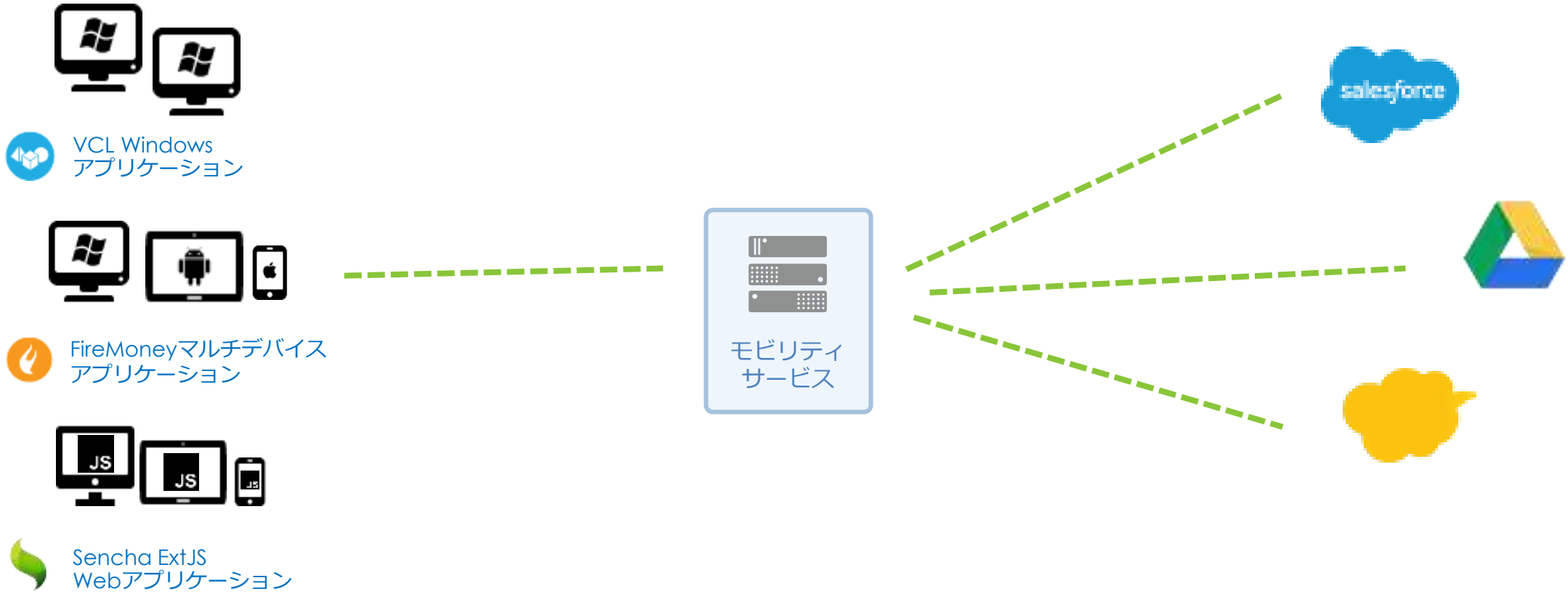
Enterprise Connectorsの利用パターンは主に3つ

1. パッケージソフトウェアで多くのサービスに対応させる
   - BIツールなどは、多くの接続先に対応することが求められる



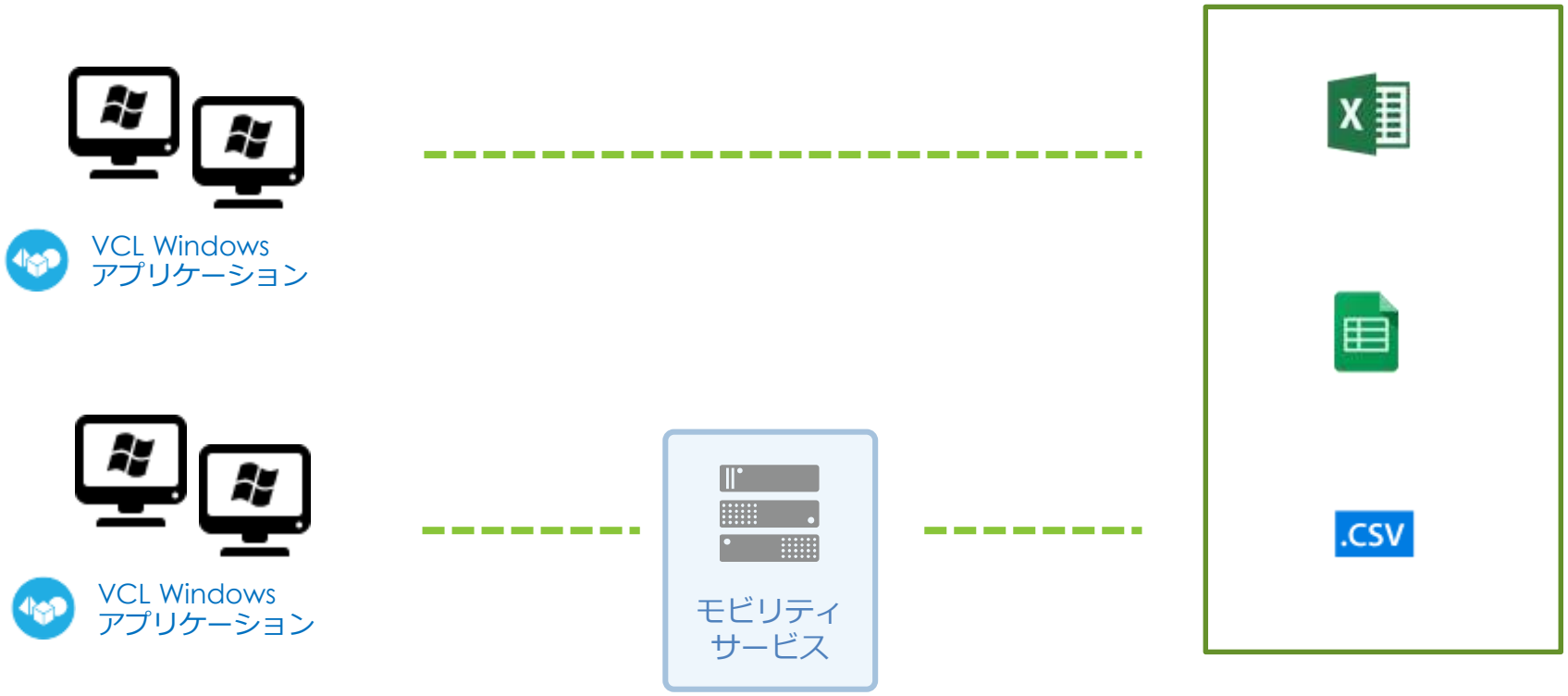
Enterprise Connectorsの利用パターンは主に3つ

2. 業務アプリから利用する特定の接続先コネクタのみ組み込む



Enterprise Connectorsの利用パターンは主に3つ

3. プロトタイプ段階でDBをセットアップせずにExcel、CSVのデータをデータソースとして使いたい



つなぐシステムの運用向けTips

embarcadero®
DEVELOPER CAMP

つなぐシステムがクライアントからリクエストを受けるための APIの設計、開発

- OSや動作環境の制約を確認する
- URLに見えてほしくない文字列は隠蔽する
- CORS（クロスオリジン）の設定を行う
- データ圧縮する

OSや動作環境の制約を確認する

- 利用可能な実行環境はDelphiとC++Builderで異なります

	Windows 32, 64-bit	Linux 64-bit
Delphi	IIS, Apache	Apache
C++Builder	IIS, Apache	–

IISとApacheで設定が必要な項目を理解する

- IISではいくつかの設定で追加モジュールが必要
- Apacheは標準モジュールの設定調整だけでOK

	IIS	Apache
RAD Server基本機能のインストール	必要	必要
URLに見せたくない文字列を隠蔽する	URL Rewriteの追加	不要
CORS（クロスオリジン）の設定を行う	CORS Moduleの追加	mod_headers
データ圧縮する	HTTP圧縮の設定を行う	mod_deflate

IIS：URLで見せたくない文字列を隠蔽する

- IIS で RAD Server をセットアップすると、URLにDLL名が見えてしまう

```
http://localhost/emsserver/emsserver.dll/Version
http://localhost/emsserver/emsserver.dll/Users
```

- URL Rewrites でDLL名が隠蔽できる

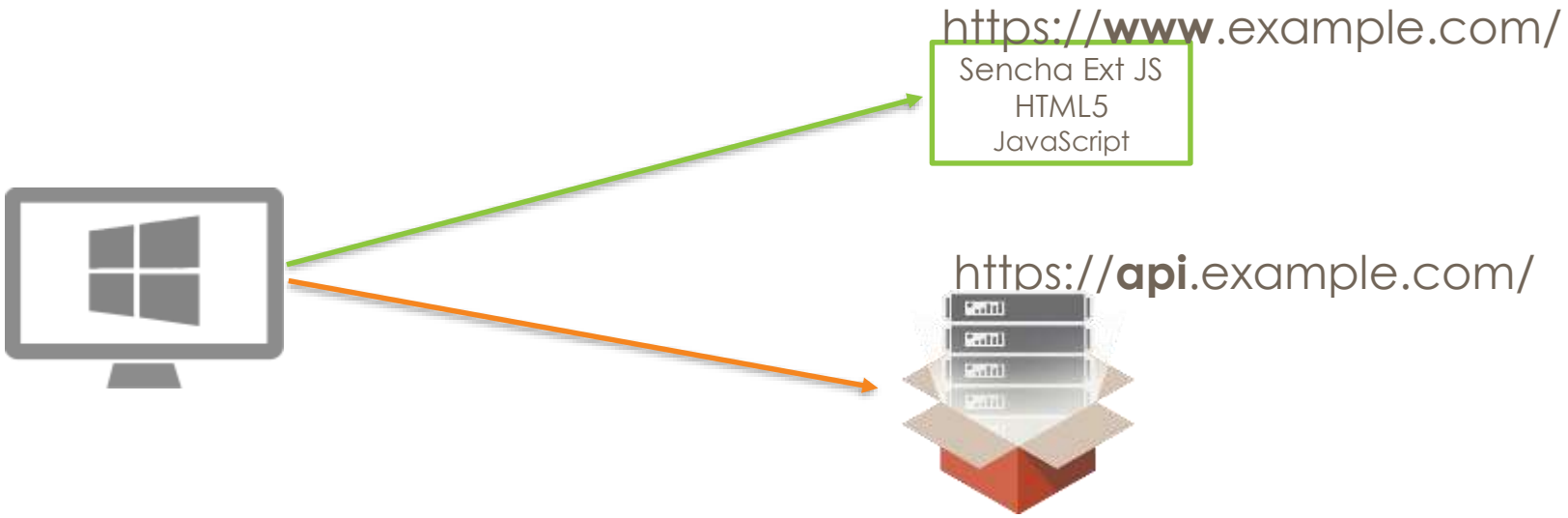
URL Rewrites 導入後のURL

```
http://localhost/emsserver/Version
http://localhost/emsserver/Users
```

- https://www.iis.net/downloads/microsoft/url-rewrite

IIS, Apache：CORS（クロスオリジン）の設定を行う

- Webコンテンツがアクセスするデータは同じサーバであるべき
- HTMLコンテンツとAPIサーバを分かれる場合は、CORSの設定が必要



IIS, Apache：データ圧縮する

- サーバの動的データ圧縮を使えば低帯域でも応答時間を減らせる

- HTTP圧縮なし

192.168.0.1 - - [30/Nov/2017:20:23:16 +0900] "**GET /senchaRAD/customer?page=1&limit=5000** HTTP/1.1" 200 **1510283** "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36" "-"

- HTTP圧縮するとデータ量が小さくなる

192.168.0.1 - - [30/Nov/2017:20:22:04 +0900] "**GET /senchaRAD/customer?page=1&limit=5000** HTTP/1.1" 200 **351026** "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36" "-“

まとめ

- 多層化で、つなぐシステム化
- Delphi/C++BuilderではRAD Serverが、つなぐシステムのハブ
- つなぐシステムはクライアントを選ばない
- つなぐシステムとクラウドは Enterprise Connectors でつながる

- つなぐシステムでアプリの機能を拡張できます

THANKS!

www.embarcadero.com/jp

第36回 エンバカデロ・デベロッパーキャンプ

補足資料

embarcadero DEVELOPER CAMP

RAD Server の認証APIの利用方法（その他の認証）

- ユーザ認証以外にもクライアント側から送信可能な付加情報がある
  - Delphi/C++BuilderアプリではTEMSProviderのプロパティで設定
  - Webクライアントはリクエストヘッダで送信
- マルチテナント情報の送信
  - X-Embarcadero-Tenant-Id
  - X-Embarcadero-Tenant-Secret
- すべてのリソースに対する完全なアクセス権の行使
  - X-Embarcadero-Master-Secret
- 特定のエンドポイントに対する権限の行使
  - X-Embarcadero-App-Secret
- 接続先RAD Serverの識別
  - X-Embarcadero-Application-Id

IIS利用時にURLからDLL名を隠す

- IIS向けの URL Rewrite で RAD Server のURLに出現する emsserver.dll などの DLL名が隠蔽できる

URL Rewrite 導入前のURL

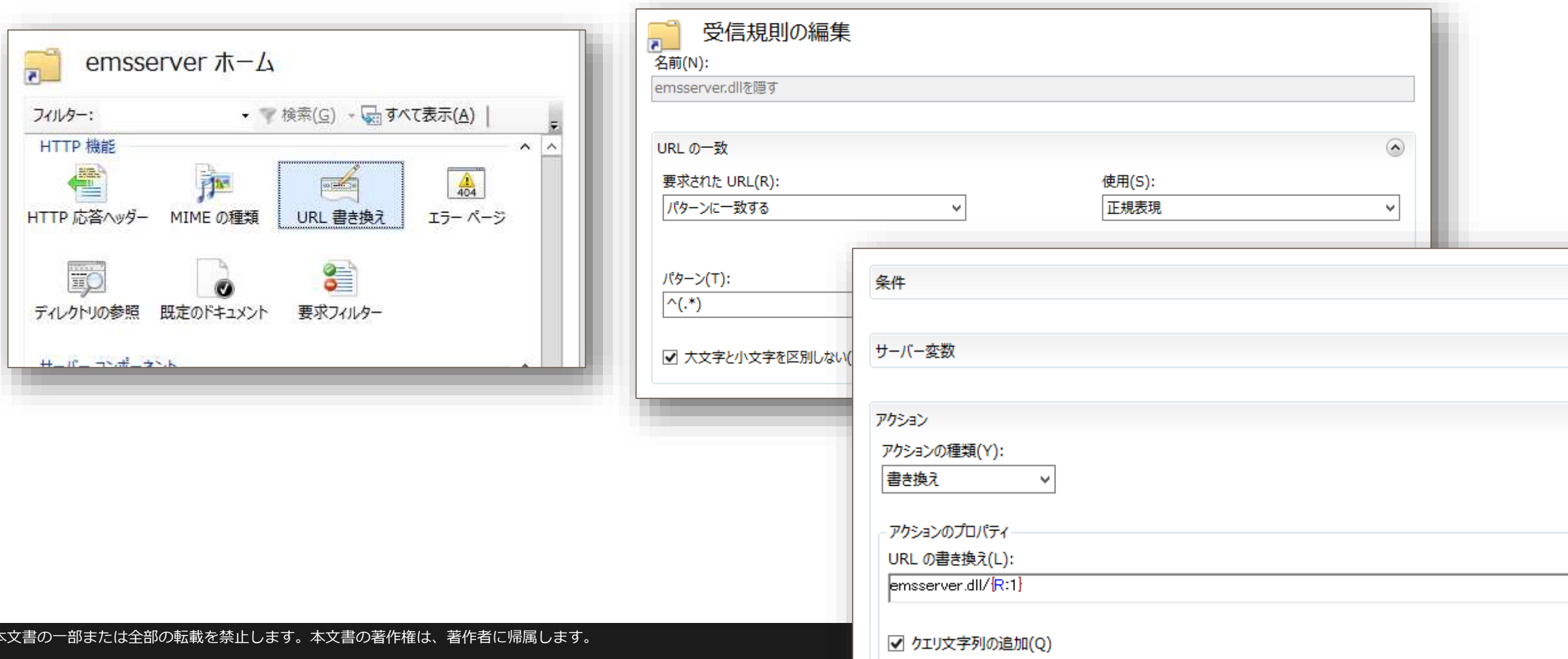
http://localhost/emsserver/emsserver.dll/Version
http://localhost/emsserver/emsserver.dll/Users

URL Rewrite 導入後のURL

http://localhost/emsserver/Version
http://localhost/emsserver/Users

- 配布先：https://www.iis.net/downloads/microsoft/url-rewrite

IISでの URL rewrite の設定例



IIS：CORSを設定する

- IIS CORS Module を追加する
- 以下のいずれかで設定する
  - Configuration Editorを使う
  - applicationHost.confg に設定する
- 右の設定例は applicationHost.confg

```
<location path="" overrideMode="Deny">
  <system.webServer>
  <cors enabled="true" failUnlistedOrigins="false">
        <add origin="*">
          <allowHeaders allowAllRequestedHeaders="true">
            <clear />
            <add header="X-Embarcadero-Application-Id" />
            <add header="X-Embarcadero-App-Secret" />
            <add header="X-Embarcadero-Master-Secret" />
            <add header="X-Embarcadero-Tenant-Id" />
            <add header="X-Embarcadero-Tenant-Secret" />
            <add header="X-Embarcadero-Session-Token" />
          </allowHeaders>
          <allowMethods>
            <add method="GET" />
            <add method="DELETE" />
            <add method="OPTIONS" />
            <add method="POST" />
            <add method="PUT" />
          </allowMethods>
        </add>
      </cors>
  </system.webServer>
</location>
```

Apache：CORSを設定する

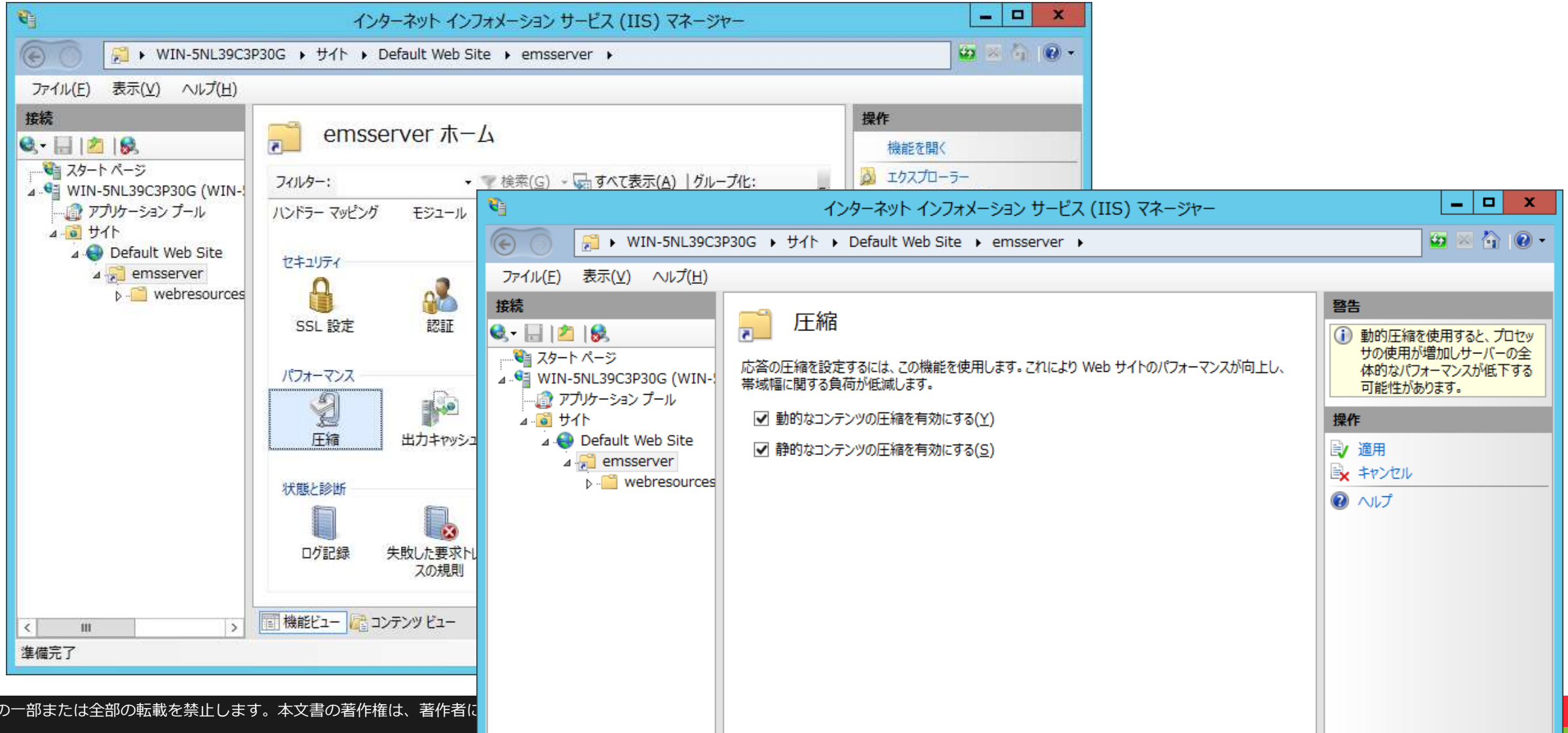
- mod_headers を用いた設定例

```
<Location /emsserver>
  <IfModule mod_headers.c>
    # アクセスを許可するURLを指定
    Header set Access-Control-Allow-Origin “*"

    Header set Access-Control-Allow-Headers "X-Embarcadero-Application-Id"
    Header append Access-Control-Allow-Headers "X-Embarcadero-App-Secret"
    Header append Access-Control-Allow-Headers "X-Embarcadero-Master-Secret"
    Header append Access-Control-Allow-Headers "X-Embarcadero-Tenant-Id"
    Header append Access-Control-Allow-Headers "X-Embarcadero-Tenant-Secret"
    Header append Access-Control-Allow-Headers "X-Embarcadero-Session-Token"

    Header set Access-Control-Allow-Methods "GET"
    Header append Access-Control-Allow-Methods "DELETE"
    Header append Access-Control-Allow-Methods "OPTIONS"
    Header append Access-Control-Allow-Methods "POST"
    Header append Access-Control-Allow-Methods "PUT"
  </IfModule>
</Location>
```

データ圧縮設定例 (IIS)



データ圧縮設定例 (Apache)

- Mod_deflate で以下のような設定を行う

```
<Location />
        AddOutputFilterByType DEFLATE text/html text/plain text/css application/json
</Location>
```

HTTP圧縮の有無によるデータ量の違い

- 5000件のデータを返す例

- HTTP圧縮なし

192.168.0.1 - - [30/Nov/2017:20:23:16 +0900] "**GET /senchaRAD/customer?page=1&limit=5000** HTTP/1.1" 200 **1510283** "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36" "-"

- HTTP圧縮あり

192.168.0.1 - - [30/Nov/2017:20:22:04 +0900] "**GET /senchaRAD/customer?page=1&limit=5000** HTTP/1.1" 200 **351026** "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36" "-“

- この例ではHTTP圧縮によりデータ量に5倍の差