

# 事前準備コース(短縮版) インストール／クラス操作体験



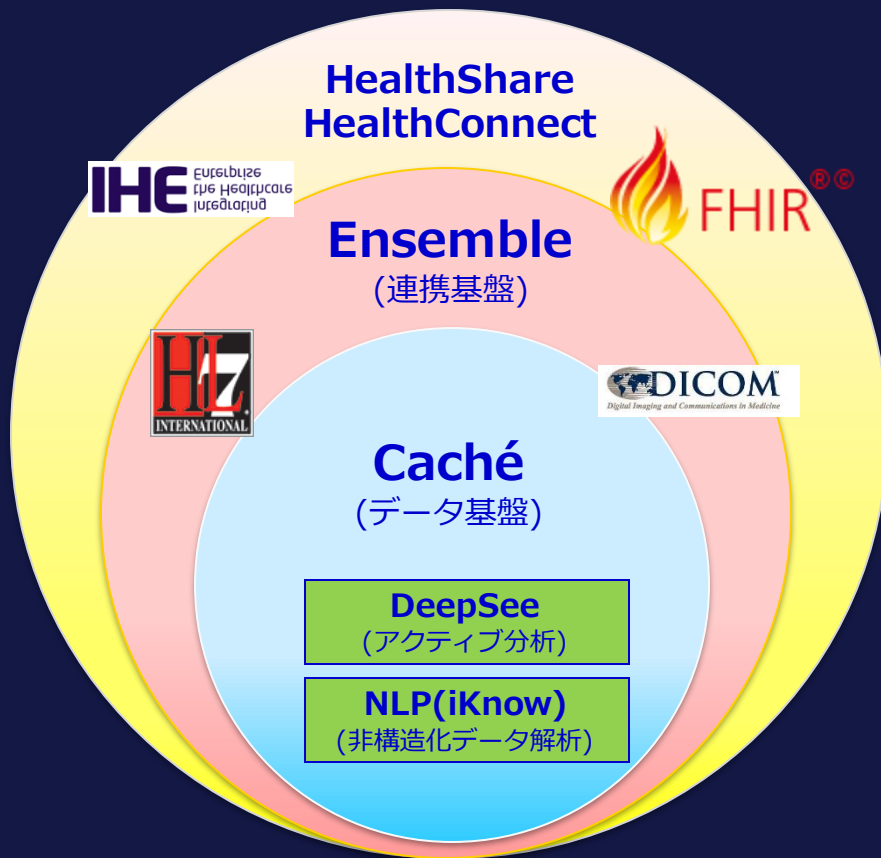
# この資料の主な目標

- この資料では、InterSystems IRIS(以下 IRIS)で開発を行うために必要な基礎知識を操作を交えながら習得いただくため、以下の内容をご説明します。
  - データ基盤の概要
  - インストール
  - 基本構造について
  - IRISへのアクセス
  - クラス定義の作成
  - オブジェクト操作によるデータ登録
  - SQLからのデータ参照／更新
  - レコードデータ／オブジェクト／グローバル変数の関係
  - メソッドの作成(クラスメソッド／ストアードプロシージャ)
  - テーブル定義／クラス定義／ルーチンの関係

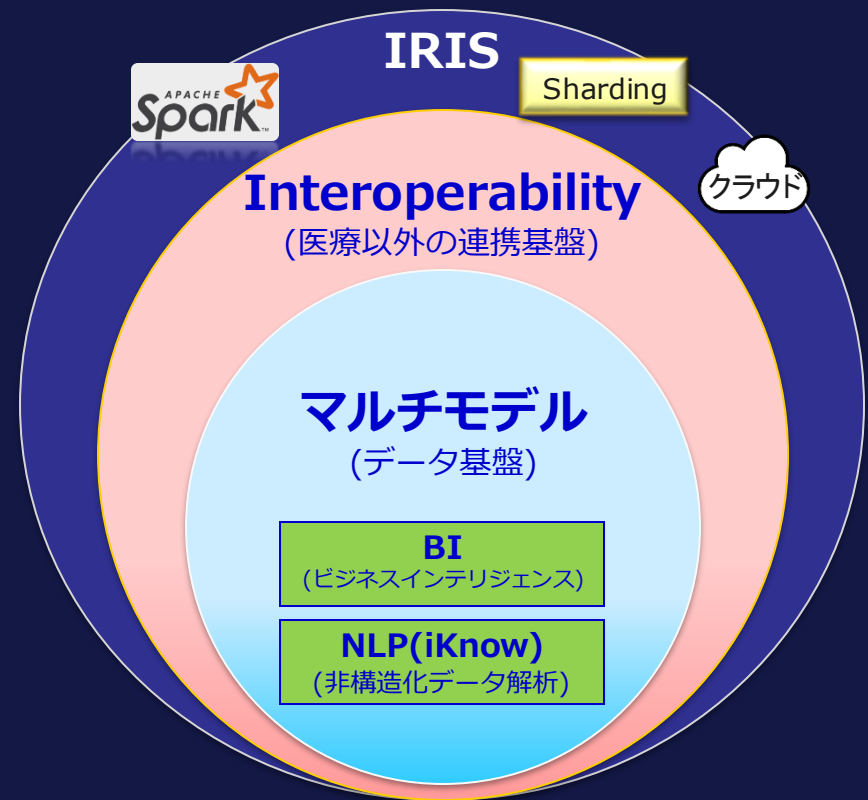


# データプラットフォーム 既存製品と InterSystems IRISとの関係

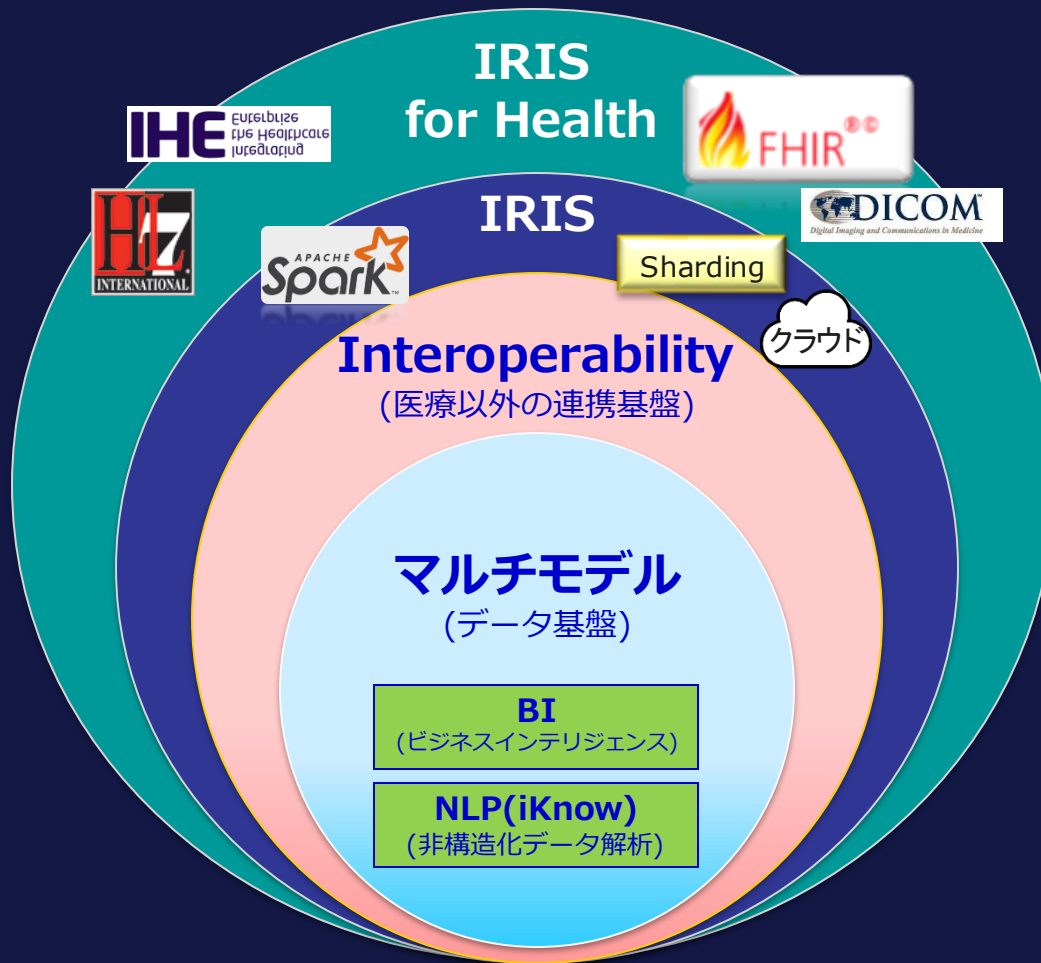
## 既存製品



## InterSystems IRIS



# InterSystems IRIS for Health

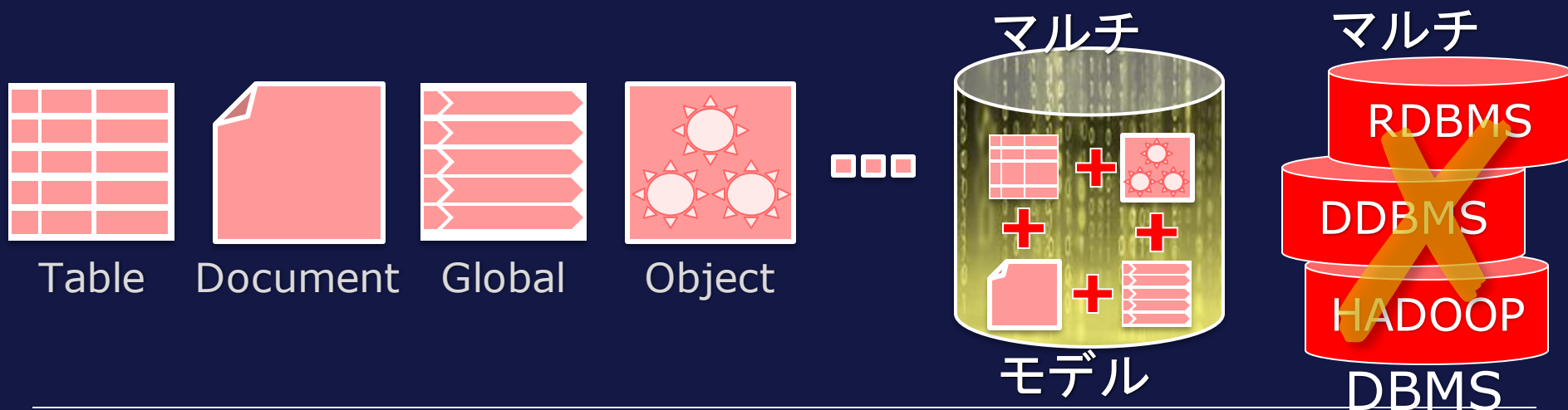


IRIS for HealthはIRISに医療用規格に対応する機能を搭載したIRISの医療向けプラットフォームです。



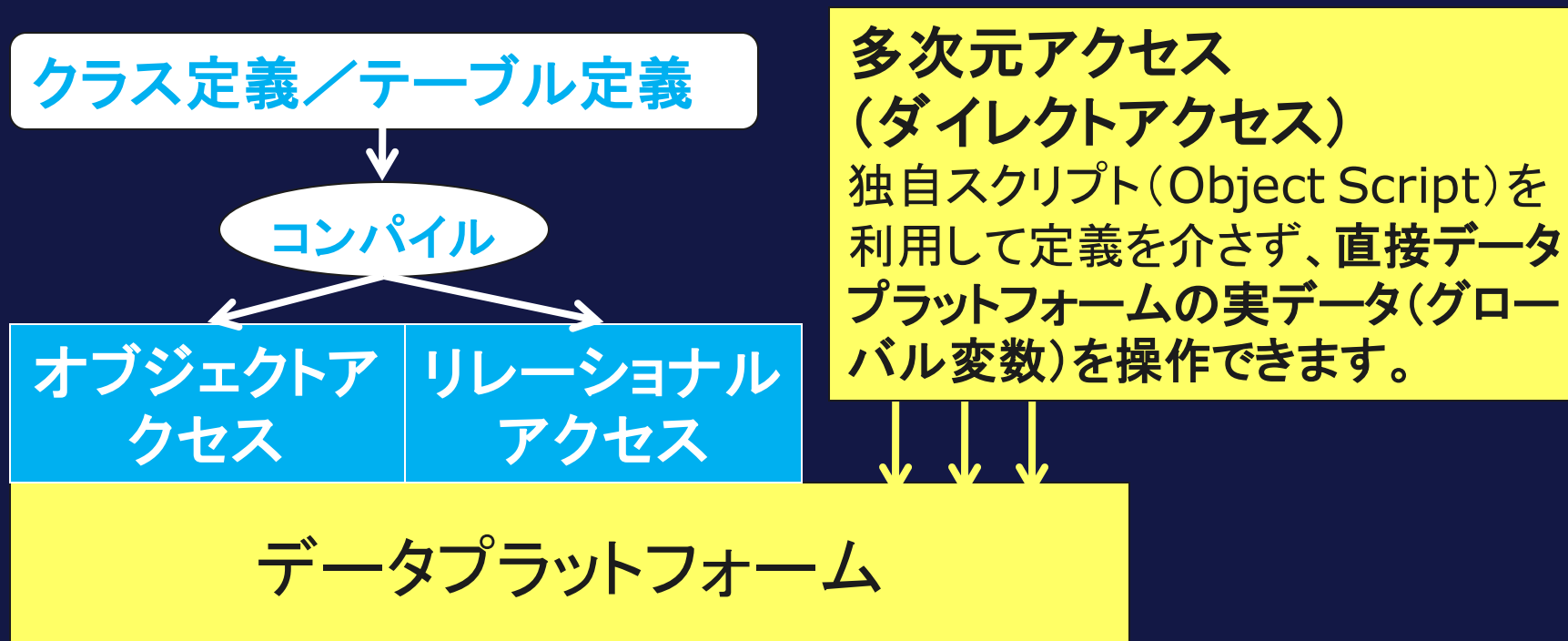
# インターシステムズのデータプラットフォーム

- 高性能で、スケーラビリティが非常に高く、安全性も高い  
マルチモデルデータをサポートできるデータプラットフォームです。
  - 弊社全ての製品 (InterSystems IRIS / Caché / Ensemble / HealthShare) は、Cachéを基盤として開発されているため、データプラットフォームの基本操作は全製品共通です。
    - 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。
  - オブジェクト、リレーショナル、多次元、XMLを完全にサポートしています。



# データプラットフォームへのアクセス

- データプラットフォームにアクセスするために様々なツール、言語、方法を選択できます。
- シンプルに記述すると以下の図のようなアクセスがあります。



# インストール前の確認

- インストールを行うため、管理者権限を持つユーザでOSにログインします。
  - Windowsの場合  
Administratorでログインするか、管理者権限を持つユーザ(Administratorsグループ所属のユーザ)でログインした状態でインストールを行います。
    - Windowsの場合、日本語を含むユーザ名を作成できますが、英数字のみのユーザ名でログインしてください。
  - Unix/Linuxの場合  
rootでログインするか、su/sudoコマンドで管理権限を取得しているユーザでインストールを行います。
- インストール用キットを配置するディレクトリは、英数字のみのディレクトリとしてください。
  - Windows環境では、日本語やスペースを含めたディレクトリを作成できますが、英数字のみのディレクトリに配置してインストールを開始してください。
- 以下URLに記載されたウィルス駆除ソフトをお使いの環境では、ウィルス駆除ソフトの設定を変更いただく必要があります。詳細は以下ページをご参照ください。

一覧: <https://faq.intersystems.co.jp/csp/faq/result.csp?DocNo=115>

対策: <https://faq.intersystems.co.jp/csp/faq/result.csp?DocNo=345>



# インストール時の指定項目

- インストール開始中、以下項目を指定します。
  - インスタンス名(構成名)  
インストール環境名で、1台のサーバでユニークな構成名で設定します  
(デフォルトは製品名の「IRISHealth」が設定されます)。
  - インストール先フォルダ  
デフォルトでは、C:¥InterSystems¥IRISHealth が設定されます。  
変更ボタンより、変更できます。
  - セットアップタイプ  
デフォルトでは、「開発」が選択されています。開発用／サーバー用の全機能がインストールされます。
  - 初期セキュリティ設定  
デフォルトは「通常」が設定されていて、開発ツール、管理ツール使用時にパスワード認証を行います。
    - インストール時の事前定義ユーザ(\_SYSTEM、SuperUser、CSPSystemなど)の初期パスワードを指定します。
    - パスワードは3文字以上の英数字を指定してください。
    - セキュリティ設定はインストール後にも変更できます。





# ライセンスキー

- ライセンスキーは、ファイルで提供されます。
  - ファイル名: **iris.key**
- ライセンスキーファイルは、以下ディレクトリに配置します。

**<インストールディレクトリ>/mgr**

- ライセンスキーファイルを指定の場所にコピーした後で、ライセンス有効化を行います（詳細は次ページ参照）。
- ライセンスキーファイルは、IRIS開始時に反映されるため、IRIS再開（停止→開始）を行うことでも有効化できます。



# ライセンス有効化

- ライセンスキーの反映を行うためには、「ライセンス有効化」を行う必要があります。
  - システム管理 > ライセンス > ライセンスキー > 新しいキーを有効化 > ライセンスキーファイルの選択 > 有効化

# 弊社製品が使用するポート番号

- インストール中に、以下の用途でポート番号を設定しています。
  - ファイアウォールを有効としている環境では、以下のポートをブロックしないよう、事前に設定してください。
- スーパーサーバー(ネットワーク)ポート
  - **1972**番をデフォルトで設定します。
    - インストール後に管理ポータルから番号を修正できます(再起動を伴います)。

システム管理→構成→システム構成→メモリと開始設定→スーパーサーバポート番号

- Webサーバーポート
  - **52773**番をデフォルトで設定します。
    - インストール後に管理ポータルから番号を修正できます。

システム管理→構成→追加の設定→開始→WebServerPort



# 管理ポータルとは？

- ブラウザで操作できるIRISの管理ツールです。
- アドレスは

ホスト名: **ポート番号**/csp/sys/UtilHome.csp

localhost: **52773**/csp/sys/UtilHome.csp



# 管理ポータルポート番号？

- 管理ポータルポート番号は、IRISインストール時に一緒にインストールされる Apache のポート番号です。
  - Apacheは、インストールディレクトリ以下 httpd ディレクトリにインストールされます。
  - 管理ポータルやクラスリファレンス参照用に用意されたWebサーバです。
  - Webアプリケーション（REST APIなど）のテスト実行には向いていますが、簡易インストールで用意されたWebサーバであるため、本番運用には向いていません。
    - 負荷テストなどにも向いていません。
  - Webアプリケーションを開発される場合は、別途 Webサーバ（Apache／IIS）をご準備ください。



# 管理ポータル 基本メニュー

- システム管理
  - IRISが使用するデータベースキャッシュサイズや、ネームスペース、データベースの構成設定が行えます。
- システムエクスプローラ
  - データやコードのインポート／エクスポートや参照用画面を提供します。
- システムオペレーション
  - タスク設定やデータベース管理用ユーティリティ、プロセス管理ユーティリティなど運用管理メニューを提供します。



# (ご参考) 管理ポータル その他メニュー

## ■ Interoperability

- 相互運用性メニューで、異なるシステム間の接続を中継したり、中継前にデータ変換を行うなど、様々な操作を定義し実行させることができます。
- 構成したプロダクションを流れる情報(=メッセージ)は、全てIRIS内データベースに自動登録されるためいつでも必要な時にメッセージをトレースできます。
  - 指定期間分を残し、古いメッセージの自動削除もできます。

## ■ Analytics

- スタースキーマを利用したモデル定義を作成するエディタ、ピボットテーブルを作成する画面、分析結果を表示するためのユーザポータルを提供しています。
- 自然言語処理を利用するときに使用するドメイン定義作成用エディタ、ドメイン定義で指定した入力文書を利用し、抽出されたエンティティを確認できるナレッジポータルを提供しています。

## ■ Health

- FHIR リポジトリ作成画面など、FHIR関連のツールが含まれています。



# 仮想の作業環境＝ネームスペースについて

- アプリケーションやユーザは、IRISにアクセスするために必ず仮想の作業環境である「ネームスペース」を指定してアクセスします。
- ネームスペースで作成するものは何？
  - スキーマ(クラス／テーブル)
  - ロジック(メソッド／ストアド)
  - データ(永続オブジェクト／レコード)
- これらの内容を永続的に利用するためにはデータベースが必要です。





# データベースの中身は？

- スキーマ(クラス／テーブル)
- データ(永続オブジェクト／レコード／グローバル)
- ロジック(メソッド／ストアドプロシージャ／ルーチン)
- 上記全ての内容がデータベースに格納されます。



# ネームスペースとデータベースの関係

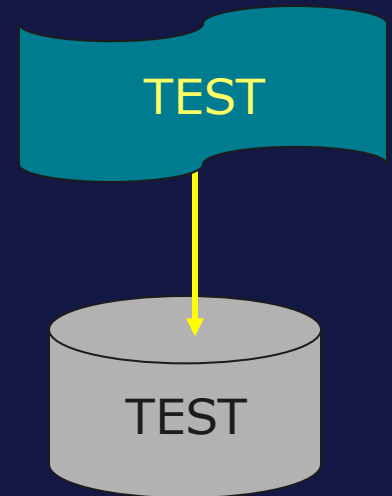
- ネームスペースは、仮想の作業環境で使いたいデータベースを指定する論理定義
- データベースは、データやロジック、クラスやテーブルのスキーマを格納する場所

ネームスペースを特定する



データベースが特定できる

**使いたい情報をすべて利用できる**



ネームスペースとデータベースの名称を同一名に設定するルールはありませんが、慣習としてよく利用される設定方法です。

# ネームスペースを作ってみましょう

- 演習環境を作成してみましょう。
  - IRISのインストールで空のデータベースを使用するUSERネームスペースが作成されますが、本コースでは、このページで作成するネームスペースを利用して演習を行います。
- ネームスペース:TEST、データベース:TESTを作成します。
  - データベースを配置するディレクトリは、  
<インストールディレクト>/mgr/test を指定してください。

**TEST**  
ネームスペース

**TEST**  
データベース




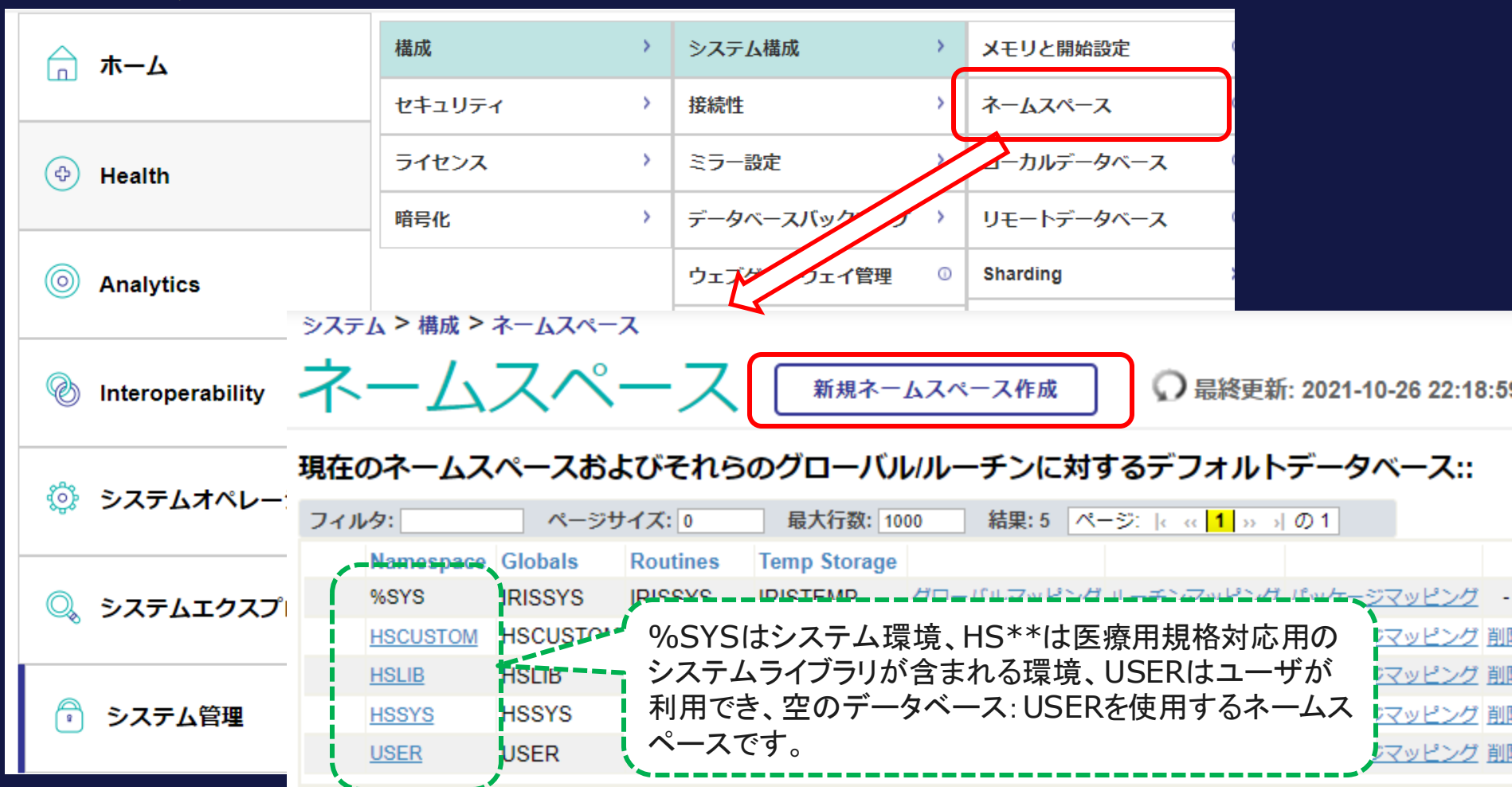
TESTネームスペース上でアクセスするデータ・定義・ルーチンは、TESTネームスペースが参照するTESTデータベースに格納されます。  
TESTデータベースの実体は、  
`c:¥InterSytsems¥IRISHealth¥mgr¥test`  
にある **IRIS.DAT** です。

`C:¥intersystems¥IRISHealth¥mgr¥test¥IRIS.DAT`



# ネームスペース:TESTの作成

1.  ランチャーから管理ポータルを起動します(ユーザ:\_system、パスワード:SYS)。
2. 管理ポータル > システム管理 > 構成 > システム構成 > ネームスペース に移動します。
3. 新規ネームスペース作成 ボタン押下します。



The screenshot shows the system management portal interface. On the left is a sidebar with icons for Home, Health, Analytics, Interoperability, System Operator, System Explorer, and System Management. The main content area shows a breadcrumb trail: システム > 構成 > ネームスペース. Below this, the title 'ネームスペース' is displayed in large green characters. To the right of the title is a button labeled '新規ネームスペース作成' (Create New Namespace), which is highlighted with a red box. Below the title, there is a section titled '現在のネームスペースおよびそれらのグローバルルーチンに対するデフォルトデータベース::' (Default database for current namespaces and their global routines::). This section contains a table with columns: Namespace, Globals, Routines, and Temp Storage. The table lists several namespaces: %SYS, HSCUSTOM, HSLIB, HSSYS, and USER. The 'USER' namespace is highlighted with a green dashed box. To the right of the table, there is a text box explaining the namespaces: '%SYSはシステム環境、HS\*\*は医療用規格対応用のシステムライブラリが含まれる環境、USERはユーザが利用でき、空のデータベース:USERを使用するネームスペースです。' ( %SYS is the system environment, HS\*\* is the system library environment that includes medical specifications, USER is the namespace that can be used by the user, and the empty database:USER is used. ).

Namespace	Globals	Routines	Temp Storage
%SYS	IRISSYS	IRISSYS	IDISTEMP
<a href="#">HSCUSTOM</a>	HSCUSTOM		
<a href="#">HSLIB</a>	HSLIB		
<a href="#">HSSYS</a>	HSSYS		
<a href="#">USER</a>	USER		

# ネームスペース:TESTの作成(新規データベースの作成)

4. 新規ネームスペース画面のネームスペース名にTEST と入力します。
5. 「グローバルのための既存のデータベースを選択」右横の 新規データベース作成... ボタンを押下
6. データベース作成ウィザードに沿ってTEST データベースを作成します。

システム > 構成 > ネームスペース > 新規ネームスペース - (構成設定)\*

## 新規ネームスペース

保存

キャンセル

英数字で新しいネームスペース名を指定します(大小文字の区別無し)。

下記のフォームを使用して新規ネームスペースを作成してください。:

ネームスペース名

TEST

必須です。

コピー元



このネームスペースでグローバルのデフォルト・データベースは

☒ ローカル・データベース

☐ リモート・データベース

グローバルのための既存のデータベースを選択

必須です。

「新規データベース作成」ボタンをクリック

新規データベース作成...

英数字でデータベース名、ディレクトリを入力します(存在しないディレクトリの場合は作成しながら進めます)。

ユーザ \_SYSTEM

ネームスペース %SYS

このネームスペースにデフォルトのウェブアプリケーション:

データベースを作成するお手伝いをします。

次からネームスペースマッピングのデータベースの名前を入力してください

TEST

必須です。

相互運用プロダクション用にネームスペース

データベースディレクトリ

c:\InterSystems\IRISHHealth\mgr\test

参照...

ディレクトリが存在しません! ディレクトリを作成したくない場合は変更してください。

# ネームスペース:TESTの作成 (新規データベースの初期サイズ設定)

7. データベースサイズの初期サイズ設定を行います(デフォルトは1MB)。

※ データベースサイズが足りなくなった場合、自動拡張を行います。  
本番環境などでは、ファイルシステム上のフラグメントを防ぐ目的で、自動拡張を行わせないように適切なサイズを新規作成時に予め確保しておきます。

8. データベース作成ウィザードを完了したあとで、新規ネームスペース画面で「保存」ボタンを押下し、作成完了です。

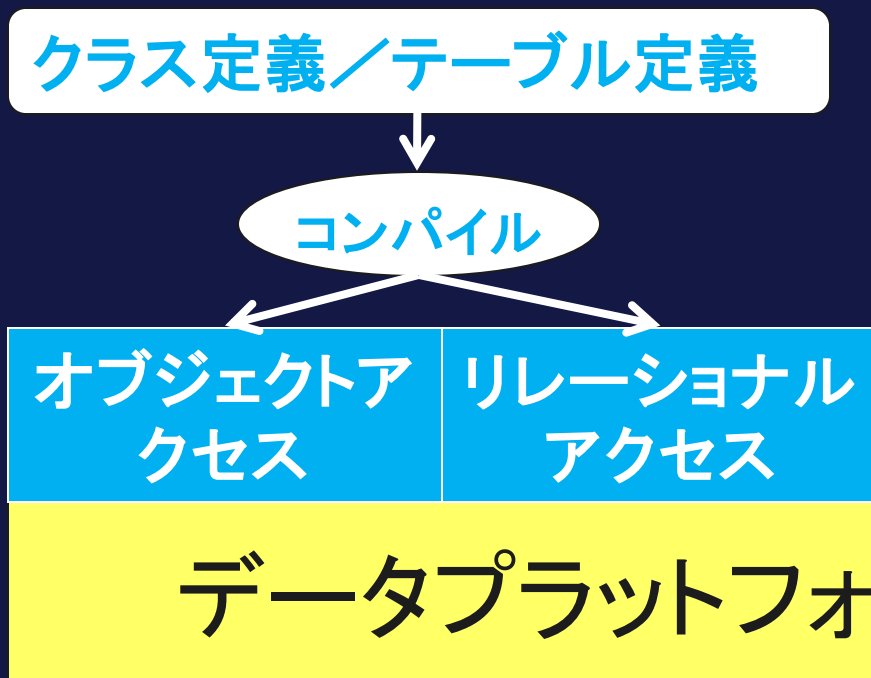
The screenshot shows the 'New Name Space' configuration interface. The breadcrumb path is 'システム > 構成 > ネームスペース > 新規ネームスペース - (構成設定)'. The title is '新規ネームスペース'. There are '保存' (Save) and 'キャンセル' (Cancel) buttons at the top right. The main form contains the following fields and options:

- データベースウィザード**: A section on the left with a red dashed box around the '初期サイズ (MB)' field set to '1'. A callout box points to it, stating: 'データベースウィザードで作成したデータベース名が登録されます。' (The database name created in the wizard is registered.)
- ネームスペース名**: A text field containing 'TEST'. A red dashed box highlights it with a callout: '新規ネームスペース画面の「保存」ボタンを押下するまで、ネームスペース作成が完了しません。必ず保存してください。' (Until you press the 'Save' button on the new name space screen, the name space creation is not complete. Please save.)
- コピー元**: A dropdown menu.
- このネームスペースでグローバルのデフォルト・データベースは**: Radio buttons for 'ローカル・データベース' (selected) and 'リモート・データベース'.
- グローバルのための既存のデータベースを選択**: A dropdown menu with 'TEST' selected. A red dashed box highlights it with a callout: 'Interoperabilityメニューを利用する場合はチェックしたままにします。' (If you use the Interoperability menu, leave the check as is.)
- このネームスペースでルーチンのデフォルト・データベースは**: A dropdown menu.
- このネームスペースにデフォルトのウェブアプリケーションを作成**: A checkbox.
- 次からネームスペースマッピングをコピー**: A dropdown menu.
- 相互運用プロダクション用にネームスペースを有効化**: A checkbox that is checked.
- 完了ボタン**: A button at the bottom left. A red dashed box highlights it with a callout: '完了ボタン押下のタイミングで指定したディレクトリ以下にIRIS.DATが作成されます。' (At the timing of pressing the completion button, IRIS.DAT is created in the specified directory and below.)

Other visible text includes 'データベースウィザード', 'データベース名', '初期サイズ (MB)', '1', 'これはデータベースの初期サイズを指定します', 'このデータベースのブロックサイズ', '8KB', 'ブロックサイズは、データベースが使用するブ', 'グローバルをジャーナル?', 'はい', '暗号化デー', 'used to co', 'システム > 構成 > ネームスペース > 新規ネームスペース - (構成設定)', '新規ネームスペース', '保存', 'キャンセル', 'データベースを作成してください。:', 'ネームスペース名', 'TEST', '必須です。', 'コピー元', 'このネームスペースでグローバルのデフォルト・データベースは', 'ローカル・データベース', 'リモート・データベース', 'グローバルのための既存のデータベースを選択', 'TEST', '必須です。', 'このネームスペースでルーチンのデフォルト・データベースは', 'ルーチンのための既存のデータベースを選択', 'このネームスペースにデフォルトのウェブアプリケーションを作成', '次からネームスペースマッピングをコピー', '相互運用プロダクション用にネームスペースを有効化', '完了', '戻る', '次へ'.

# ここからは・・・

- 簡単なクラス定義を作成して、オブジェクトアクセス／リレーショナルアクセスを体験します。
- 作成したデータが全てグローバル変数として格納されていることも確認します。



## 多次元アクセス (ダイレクトアクセス)

独自スクリプト (ObjectScript) を利用して定義を介さず、直接データプラットフォームの実データ (グローバル変数) を操作できます。



データプラットフォーム



# クラスとは？

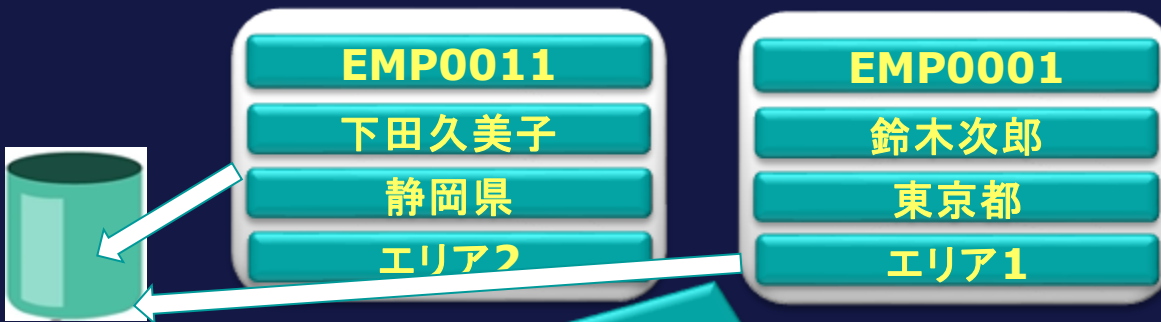
- 最も簡単なクラスは、メソッドだけを定義するメソッドのコンテナとしたクラスです。
  - プロパティを持たないクラスでロジック(メソッド)だけを定義します。
  - アプリケーションロジックのユーティリティコードや、ストアドプロシージャ用ロジックを定義するクラスとして使用します。
- メソッドの他にデータ要素であるプロパティを定義するクラスもあります。
  - データベースに格納する機能を持つクラス(Persistent＝永続クラス)
    - 演習で作成します。
  - データベースに格納しないクラス(Registered＝メモリ上にインスタンスは作成できるけど、データベースには保存できないクラス)
  - 永続クラス(Persistent)のあるプロパティに埋め込みことが前提のクラス(Serial＝埋め込みクラス)
- メソッドは、特定の動作を実行するためのコードです。
  - メソッドコード中で使用する引数や変数は、プライベートスコープです。
- スタジオを使用して、クラスを作成します。
- メソッドの記述には、ObjectScript を使用します。





# オブジェクト

- オブジェクトは、クラスのプロパティに特定のデータをセットしたクラス定義から構築される実体です。
  - オブジェクトの雛形がクラス定義とも表現できます。
- IRISのオブジェクトは、メモリ上でしか動作できないオブジェクト (Registered) とデータベースに保存できるオブジェクト (Persistent＝永続オブジェクト) を作成できます。
  - クラス作成時「クラスタイプ」の項目で指定します。
- 永続オブジェクトは、データベースの新規保存時、ユニークなID番号が割り振られます。
  - テーブルのレコードID (ROWID) と同等です。



## ABCSystem.Employee

EMPID As %String  
Name As %String  
Location As %String  
Area As %String

CreateEmail()

# プロパティとは

- プロパティは、オブジェクトのデータ要素がどのような値であるかを定義する設定項目です。

- 定義文は以下の通りです。

Property Name **As %String**;

%Stringの正式名称は  
**%Library.String** クラスで、  
データタイプもクラス定義として実装されています(データを持たないタイプで定義されています)。

- **As** の後ろにデータタイプやそのプロパティを表現するためのクラス名を指定できます。
  - データタイプの例: %String、%Integer、%Dateなど
  - オブジェクト参照を行う場合は、参照先クラス名をAsの後ろの指定します。
  - %Stringに独自のチェック項目をつけたユーザ定義のデータタイプも指定できます。その場合は、作成したデータタイプクラス名を指定します。
- (ご参考)リスト、リレーションシップ定義など、このほかにもオブジェクトよりの定義も実装できます。



# メソッドとは

- メソッドはクラスに関連したロジックを記述できる定義です。
  - ObjectScript を使用して記述します。
  - 2種類あります。
    - インスタンスメソッド
      - 実行時、メモリ上にメソッド実行対象であるインスタンスが存在していないと実行できないメソッド  
例) オブジェクトを保存するときに使用する%Save()など
    - クラスメソッド
      - **##CLASS()**の文法を使用し、メモリ上のインスタンスの有無にかかわらずいつでも実行できるメソッド
      - SqlProc属性をTrueにすることでストアードプロシージャ化できるメソッド
  - メソッド内に記載するサーバ側コードには、オブジェクト／SQL／ダイレクト、全ての文法が記述できます。
  - ローカル変数のスコープは、ローカル変数を使用しているメソッド内のみです。



# パッケージとは

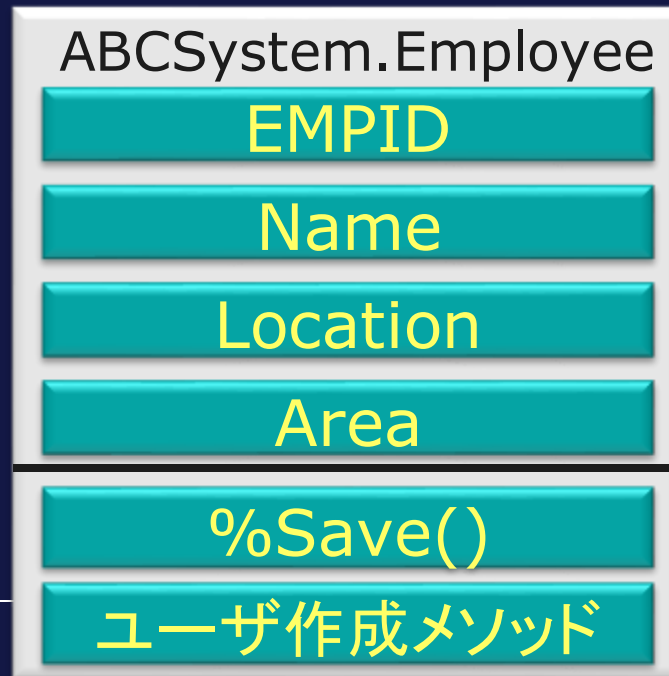
- パッケージはクラス定義にとってフォルダの役割です。
  - ネームスペース内のクラス定義をひとまとめにします。
- クラスの完全名は、パッケージ名.クラス名 です。
  - 大文字小文字の区別のある文字または、文字列で始まる数字を指定します。
- %付きパッケージは、InterSystemsが提供するシステムクラスを含むパッケージです。
  - %付きパッケージは、任意のネームスペースから自動的に利用可能です。
  - 様々な機能を持ったクラスが提供されていてクラスリファレンスやスタジオを使用して定義を確認できます。
- %Libraryパッケージは、クラスの“構築基盤”として使われるクラスを含んでいます。
  - %Library.Persistentや%Library.RegisterdObject、データタイプとして利用する%Library.Stringや%Library.Date
    - **%Library.** を省略でき、%Persistent や%String のように記述できます

クラスリファレンスの開き方  
スタジオ→表示→クラスドキュメントの表示



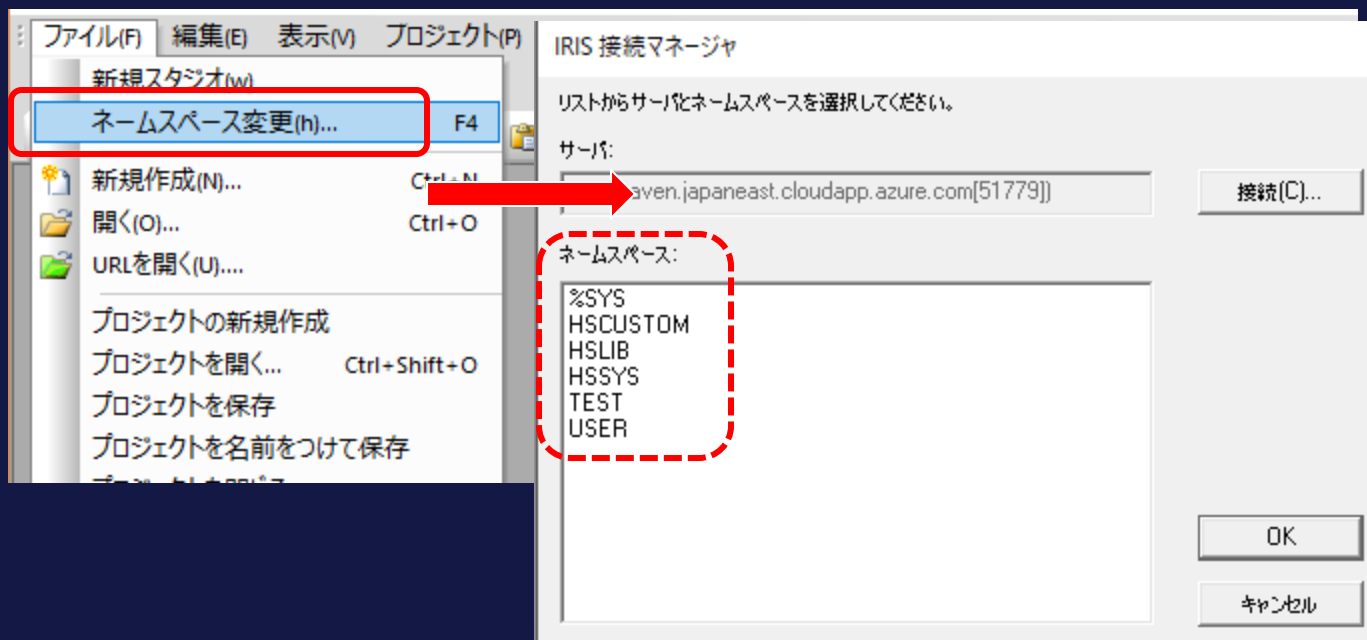
# ABCSystem.Employeeクラスは 永続(Persistent)クラス

- 永続クラスとは、データベースに保存する機能を持ったクラスです。
  - データベースに保存されると、ユニークなID番号が付与されます。
- 永続クラス(クラスタイプ: Persistentを選択したクラス)は、%Library.Persistentクラスを継承します。



# クラス定義作成のための準備

- スタジオを開き、操作対象のネームスペースに移動します。
  - ファイル→ネームスペース変更



# 演習: ABCSystem.Employeeクラスの作成

- スタジオの **ファイル→新規作成→Cacheクラス定義** からクラスを作成します。

test/TEST@\_SYSTEM - Default\_system.prj - スタジオ

ファイル(F) 編集(E) 表示(V) プロジェクト(P) ビルド(B) デバッグ(D) ツー...

新規スタジオ(w)  
ネームスペース変更(h)... F4  
新規作成(N)... Ctrl+N

新規クラスウィザード

新規クラスウィザードへようこそ。  
このウィザードは、新しいCacheクラスを作成する手順を説明します。  
以下の手順に従い、次のページに進むには「次へ」をクリックしてください。  
いつでも「完了」を選択することができます。

パッケージ名: **ABCSystem**  
クラス名: **Employee**

パッケージ名を入力:  
ABCSystem

クラス名を入力:  
Employee

新しいクラスの説明を入力してください(オプション):  
このエリアは説明文を記入できます。

カテゴリ: 一般  
CSPファイル  
プロダクション  
メッセージ  
Zen  
カスタム

テンプレート: Cache ObjectScript ルーチン  
Cache クラス定義  
Cache Basic ルーチン

クラスウィザード

クラスタイプ

データベースに保存する「永続」を選択

どの種類のクラスを作成しますか?以下のクラスを選択して下さい:

- ☒ 永続 (データベースに格納可能)
- ☐ Serial (永続オブジェクト内に埋め込むことが可能)
- ☐ Registered (データベース内には格納されない)
- ☐ 抽象
- ☐ データタイプ
- ☐ CSP (HTTPイベントの処理に使用)
- ☐ Extends

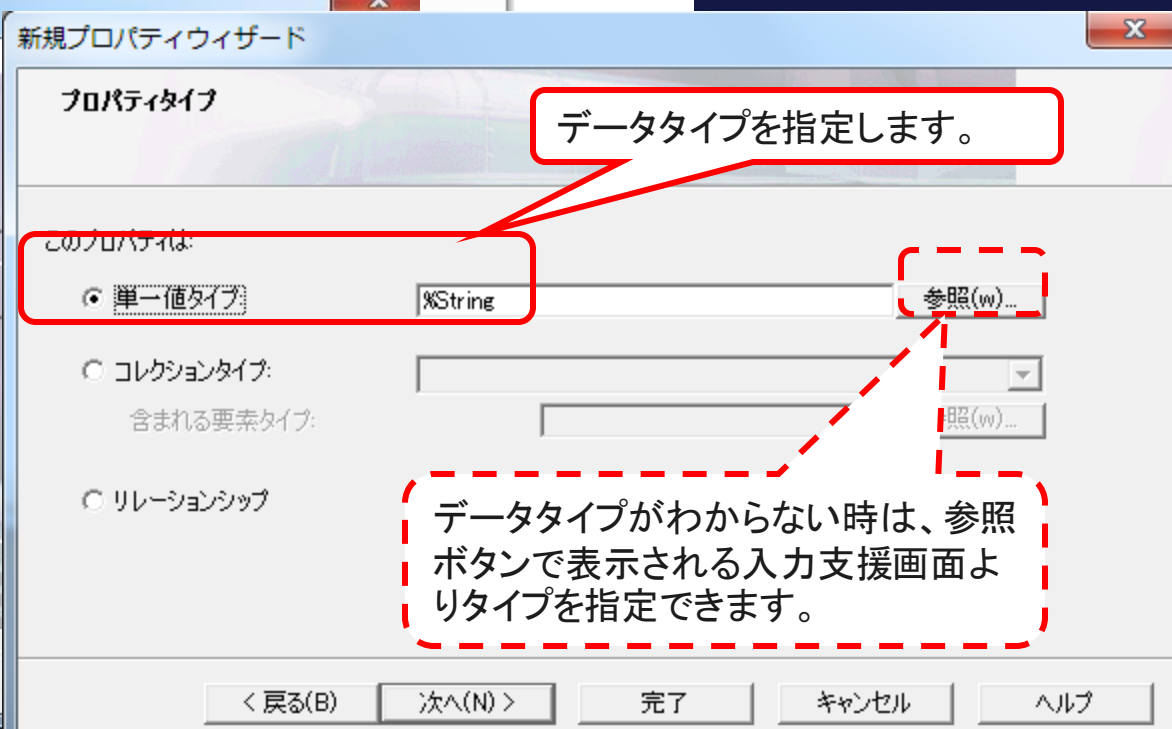
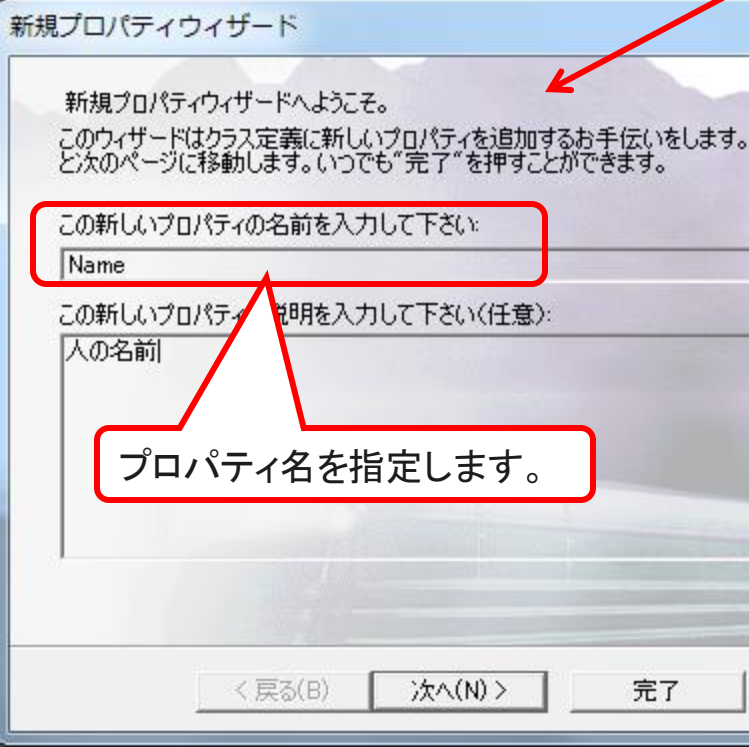
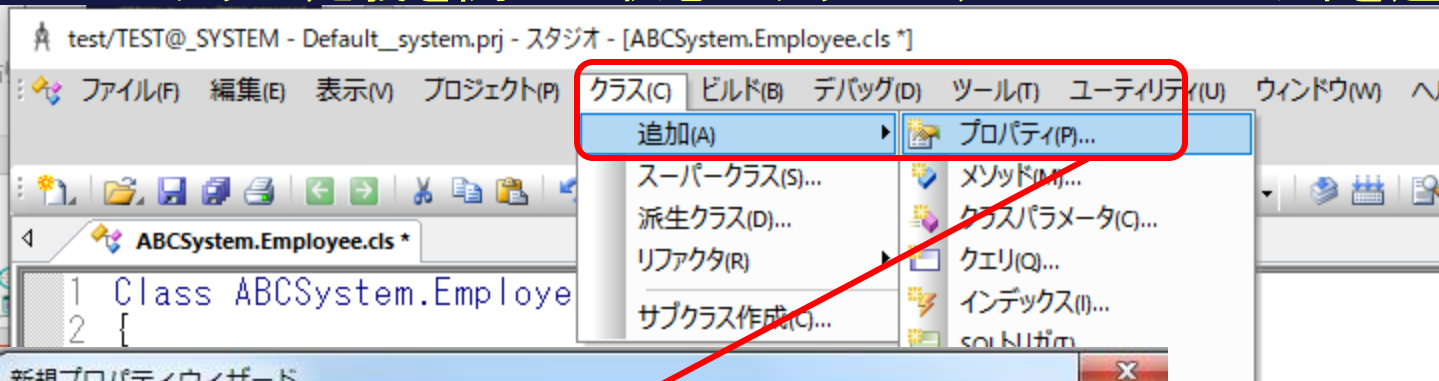
Extendsにチェックを入れるとスーパークラスを指定できます。

スーパークラス名:

< 戻る(B) 次へ(N) > 完了 キャンセル

# 演習: プロパティの作成

- クラス定義を開いた状態でクラス→追加→プロパティを選択します。





# 作成するプロパティ

- Nameと同様にデータタイプに%Stringを設定する以下のプロパティを作成します。

プロパティ名	データタイプなど
EMPID	%String ユニークの属性を付けます（図）
Location	%String
Area	%String

新規プロパティウィザード

プロパティ属性

☐ 必須 このプロパティは必須(ヌル以外)

☒ インデックス このプロパティでインデックスを作成

☒ ユニーク このプロパティでユニークインデックスを作成

☐ 計算 このプロパティはメモリ上の記憶域を割り当てられません。

☐ 計算。このプロパティは計算された値を持ちます。

SQLフィールド名(オプション):

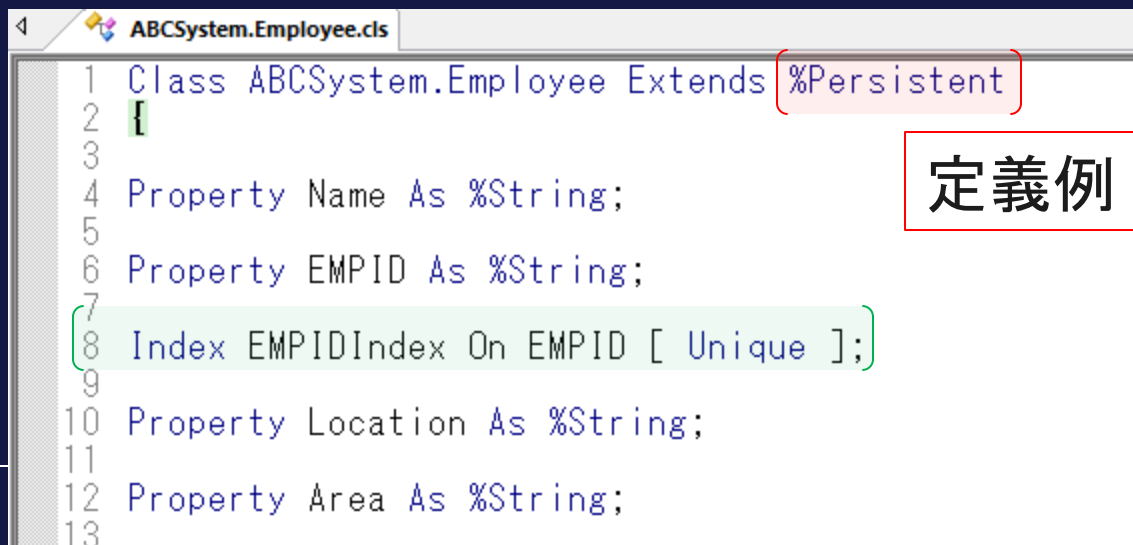
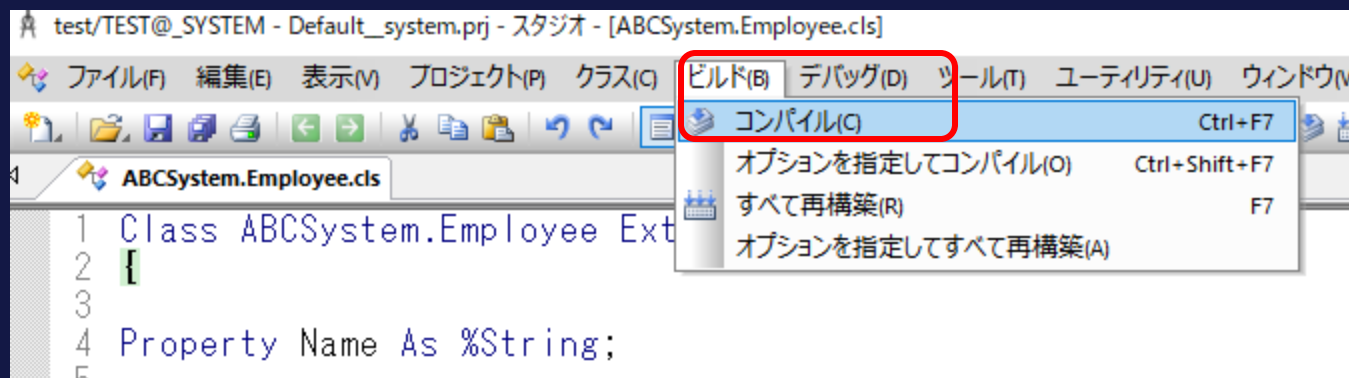
< 戻る(B)    次へ(N) >    完了    キャンセル    ヘルプ

データタイプ指定の次の画面に「**ユニーク**」  
をチェックする項目が出てきます。  
EMPID作成時のみ、チェックしてください



# 完成予定図

- クラス定義の作成が完了したら、スタジオのメニューバーにある **ビルド > コンパイル** でコンパイルを行います。



# クラスを利用したプログラミング

- オブジェクト操作には **##class** 構文を使用します。

- 新規インスタンス生成

```
Set obj=##class(パッケージ名.クラス名).%New()
```

- 既存オブジェクトオープン

```
Set obj=##class(パッケージ名.クラス名).%OpenId(id)
```

- クラスメソッドの実行

```
Set modori=##class(パッケージ名.クラス名).メソッド名(引数n)
```

- インスタンスメソッドの実行

```
Set modori=obj.メソッド名(引数n)
```

- オブジェクトのメモリからの解放

```
Set obj="" または kill obj
```

- プロパティへのアクセス

35 | 事前準備(短縮版) `set obj.プロパティ名="値"`

`write obj.プロパティ名`



# 永続メソッド

- 永続クラスに用意されている主なメソッドは以下の通りです。

メソッド	目的	成功時	失敗時
%New()	メモリー上に新規オブジェクトを作成	object	""
%Save()	オブジェクトを保存	1	エラーステータス
%Id()	object IDを返す	object ID	""
%OpenId( <i>id</i> )	保存済みオブジェクトを取得し、メモリーに展開する	object	""



# ご参考: 永続メソッド(つづき)

メソッド	目的	成功時	失敗時
%DeleteId( <i>id</i> )	保存済みオブジェクトを削除	1	エラー ステータス
%DeleteExtent()	全ての保存済みオブジェクト(とその関連オブジェクト)を削除	1	エラー ステータス
%KillExtent() (開発中のみ使用)	全ての保存済みオブジェクトを削除	1	""
%ClassName(1)	オブジェクトの完全クラス名を表示	クラス名	



ご参考:

## ObjectScriptの変数について

- サーバ側ロジックの記述に使用するObjectScriptの変数(ローカル／グローバル両方)は 型がなく(タイプレス)、変数タイプを宣言する必要がありません。
  - 内部的には、数字か文字列かのどちらかで保持されます。
  - 動的で弱い型に分類されます。
  - 変数自身は、タイプレスですが、クラス定義のプロパティには、データタイプを指定する必要があり、一般的なデータ型も含めて内部に用意されたデータタイプクラスを使用して定義します。



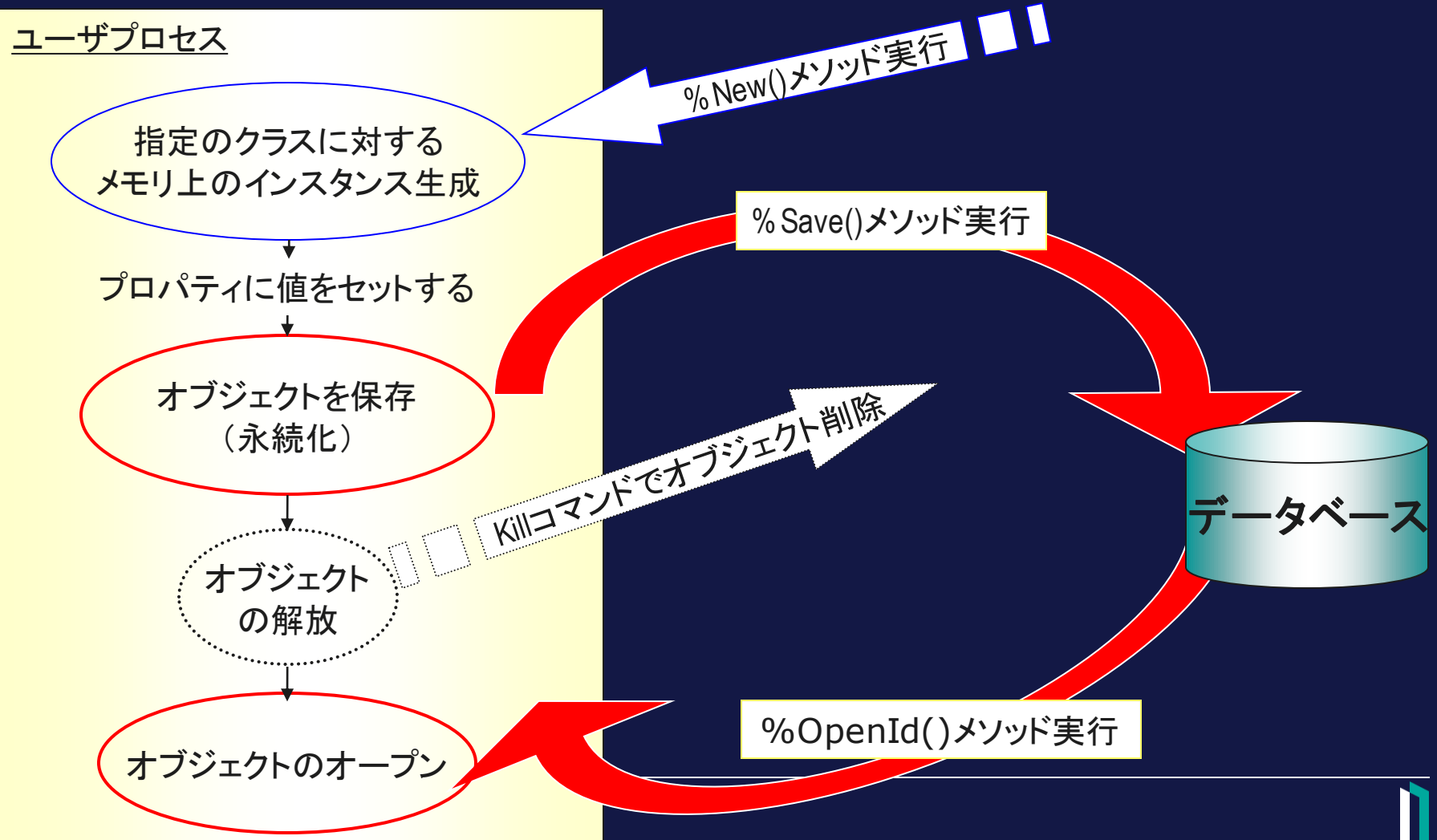
# ご参考： ObjectScript よく使うコマンド

コマンド	内容	例文
SET	変数に値を割り当てる	Set val="あいうえお" 文字列は二重引用符で括ります。(SQL文では一重引用符です。)
WRITE	標準出力	Write val,! ターミナルでは！は改行を出力します。 引数無のWRITEはメモリ上の全変数の出力
KILL	変数の消去	Kill val 指定した変数を削除します。 引数無のKILLはメモリ上の全変数の消去
QUIT	プログラムの終了、FOR、WHILEのループの終了	QUIT retunVal 戻り値がある場合には引数を指定します。
DO	メソッド、プロシージャの実行	Do ##class(Training.Person).Method() クラスメソッドの実行
ZWRITE	Writeと似ている ターミナル用コマンド	オブジェクトリファレンス、配列の全添え字情報出力ができる便利コマンド(デバッグ用)



# 演習： インスタンス生成から保存の流れ

- インスタンス生成／解放、データベースへの保存／オープンの流れは以下の通りです。





# データの作成例(オブジェクト)

```
USER>set $namespace="TEST" //ネームスペース移動

TEST>set emp=##class(ABCSysSystem.Employee).%New() //インスタンス生成

TEST>set emp.Name="テスト従業員" //プロパティ設定

TEST>set emp.EMPID="EMP0001"

TEST>set emp.Location="東京都"

TEST>set st=emp.%Save() //保存

TEST>write st //保存結果の確認 1:成功、それ以外はエラーステータス
1
TEST> // 以下お時間あればお試しください

TEST>set emp2=##class(ABCSysSystem.Employee).%New() //2つ目のインスタンス生成

TEST>set emp2.EMPID="EMP0001" //ユニークではない値を設定

TEST>set st=emp2.%Save() //保存

TEST>write st //エラーステータスが返ります(文字列ではないので読みにくい出力結果)
0 ° MABCSysSystem.Employee:EMPIDIndex: ^ABCSysSystem.EmployeeI("EMPIDIndex",
EMP0001")→#%SaveData+19^ABCSysSystem.Employee.1TEST?'e^%SaveData+19^ABCSysSystem.Employee.1^8.e^%
SerializeObject+6^%Library.Persistent.1^2#e^%Save+4^%Library.Persistent.1^2e^^^0
TEST>write $system.Status.GetErrorText(st) //エラー文字列を出力する
エラー #5808: キーが一意ではありません: ABCSysSystem.Employee:EMPIDIndex: ^ABCSysSystem.EmployeeI("EMPIDIndex",
EMP0001")
TEST>
```



# テーブル

- 永続オブジェクトは、リレーショナルテーブルでは、行になります。
  - オブジェクトid番号は、IDカラムです。
  - パッケージ名は、スキーマ名になります。
    - 一部例外で、*User*パッケージは、*SQLUser*スキーマになります。
- ABCSystem.Employeeテーブルでは以下の通りです。
  - ABCSystem.Employeeクラスのデータと同等です。

ID	EMPID	Name	Location	Area
1	EMP0001	山田太郎	東京都	エリア1
2	EMP0002	鈴木一郎	大阪府	エリア2
3	EMP0003	佐々木小次郎	静岡県	エリア1



# 管理ポータル SQL画面の使い方 ネームスペースの切り替え

- 管理ポータル > システムエクスプローラ > SQL

The screenshot shows the InterSystems Management Portal interface. At the top, the logo for InterSystems IRIS Data Platform is visible. The main header area displays '管理ポータル' (Management Portal) and 'サーバ 6f45803d364c'. Below this, the breadcrumb path 'システム > SQL' is shown. The left sidebar contains a list of navigation links: 'テーブル' (Tables), 'ビュー' (Views), 'プロシージャ' (Procedures), and 'クエリキャッシュ' (Query Cache). The main content area is titled 'ネームスペース %SYS 変更' (Name Space %SYS Change). A red box highlights the '変更' (Change) link. A red callout bubble points to this link with the text: '接続先ネームスペースを確認します。' (Check the target name space). Below the main content area, there is a section titled 'ネームスペース選択' (Name Space Selection). This section includes a list of '利用可能なネームスペース' (Available Name Spaces): %SYS, HSCUSTOM, HSLIB, HSSYS, TEST, and USER. The 'TEST' entry is highlighted with a red box. A red callout bubble points to this box with the text: '「変更」のリンクを押下し、ネームスペース一覧から接続先を切り替えます。' (Press the 'Change' link and switch the connection target from the name space list). At the bottom right of the dialog, there are 'キャンセル' (Cancel) and 'OK' buttons.

InterSystems™  
IRIS Data Platform

管理ポータル

サーバ 6f45803d364c

ネームスペース %SYS 変更 ユーザ \_SYSTEM

システム > SQL

フィルタ 適用先 すべて

システム ☐ スキーマ

> テーブル

> ビュー

> プロシージャ

> クエリキャッシュ

ネームスペース選択

ネームスペース選択

利用可能なネームスペース

%SYS  
HSCUSTOM  
HSLIB  
HSSYS  
TEST  
USER

「変更」のリンクを押下し、  
ネームスペース一覧から  
接続先を切り替えます。

キャンセル OK

# 管理ポータル SQL画面の使い方

## テーブルの操作

- 管理ポータル > システムエクスプローラ > SQL

システム > SQL

フィルタ  適用先  すべて

システム ☐ スキーマ ☐

▼ テーブル

① > ABCSystem.Employee

▼ ビュー

▼ プロシージャ

▼ クエリキャッシュ

《 ウィザード » アクション » ② テーブルを開く ツール »

カタログの詳細 クエリ実行 参照 SQLステートメント

テーブル: ABCSystem.Employee ● テーブル情報 ○ フィールド ○ マ

テーブルタイプ	TABLE
所有者	_SYSTEM
最終コンパイル	2021-10-27 06:51:03
外部	0
読込専用	
タプル名	

テーブルを開く 更新 ウィンドウを

データベース TEST 中の ABCSystem.Employee 最終更新: 2021-10-27 09:00:19.356

#	ID	Area	EMPID	Location	Name
1	1		EMP0001	東京都	やまだたろう
2	3	エリア2	EMP0011	大阪府	山田太郎
3	4	エリア2	EMP0901	沖縄県	佐々木花子

完了

カタログの詳細 クエリ実行 参照 SQLステートメント

実行 プラン表示 履歴を表示 クエリビルダ 表示モード ▼ 最大 1000

```
SELECT
ID, Area, EMPID, Location, Name
FROM ABCSystem.Employee
```

クエリ実行タブでは、任意のSQL文を実行できます。



# クラス定義／テーブル定義 言葉の対応

オブジェクト指向プログラミング (OOP)	構造化クエリー言語 (SQL)
パッケージ	スキーマ
クラス	テーブル
プロパティ	カラム
メソッド	ストアド・プロシジャ
2つのクラス間のリレーションシ ップ	外部キー制約、組み込みJOIN
オブジェクト（メモリー上とディ スク上）	行（ディスク上）



# データとグローバル変数

- データプラットフォームへのアクセス方法は3種類(SQL／オブジェクト／ダイレクト)ありますが、実体のデータは「グローバル変数」と呼ばれる永続多次元配列で格納されます。
  - SQLの場合はテーブルとして操作しているデータも、グローバル変数として格納されます。
- オブジェクト／リレーショナルアクセスでは、クラス／テーブル定義に合わせ、初回コンパイル時に格納するグローバル変数の構造を決定しています。
  - ストレージ定義を初回コンパイル時に自動的に作成します。

コンパイルにより自動生成

SQL  
オブジェクト

生成コード

クラス／テーブル定義

ストレージ定義

グローバル変数

コンパイルにより自動生成

ダイレクト

データの参照・更新＝グローバル変数の参照・更新

どのアクセス方法で操作してもデータの実体はグローバル変数と呼ばれる永続多次元配列に格納されます。

# 補足:オブジェクト／レコードとストレージ定義

```
14 Storage Default
15 {
16   <Data name="EmployeeDefaultData">
17     <Value name="1">
18       <Value>%%CLASSNAME</Value>
19     </Value>
20     <Value name="2">
21       <Value>Name</Value>
22     </Value>
23     <Value name="3">
24       <Value>EMPID</Value>
25     </Value>
26     <Value name="4">
27       <Value>Location</Value>
28     </Value>
29     <Value name="5">
30       <Value>Area</Value>
31     </Value>
32   </Data>
33   <DataLocation>^ABCSystem.EmployeeD</DataLocation>
34   <DefaultData>EmployeeDefaultData</DefaultData>
35   <IdLocation>^ABCSystem.EmployeeD</IdLocation>
36   <IndexLocation>^ABCSystem.EmployeeI</IndexLocation>
37   <StreamLocation>^ABCSystem.EmployeeS</StreamLocation>
38   <Type>%Storage.Persistent</Type>
39 }
```

```
1 Class ABCSystem.Employee Extends %Persistent
2 {
3
4   Property Name As %String;
5
6   Property EMPID As %String;
7
8   Index EMPIDIndex On EMPID [ Unique ];
9
10  Property Location As %String;
11
12  Property Area As %String;
13 }
```

**ID=1**のデータを参照

```
>write $LIST(^ABCSystem.EmployeeD(1),2)
テスト従業員
>write $LIST(^ABCSystem.EmployeeD(1),3)
EMP00001
```

TEST>zwrite ^ABCSystem.EmployeeD

^ABCSystem.EmployeeD=4

^ABCSystem.EmployeeD(1)=\$lb("","テスト従業員","EMP0001","東京都","")

^ABCSystem.EmployeeD(3)=\$lb("","山田太郎","EMP0011","大阪府","エリア2")

^ABCSystem.EmployeeD(4)=\$lb("","佐々木花子","EMP0901","沖縄県","エリア2")

**ID=1**のデータを更新

```
set $LIST(^ABCSystem.EmployeeD(1),2)="やまだたろう"
```

# 機能の追加(JSONアダプタ)

- クラス定義は多重継承が行えます。
- %JSON.Adapterを追加して動作を確認してみましょう。
  - プロパティ名を利用してJSONオブジェクトのインポート、エクスポートが行えます。

```
1 Class ABCSystem.Employee Extends (%Persistent, %JSON.Adaptor)
```

```
TEST>set emp=##class(ABCSystem.Employee).%OpenId(1)
```

↑ または ↓

```
TEST>set emp=##class(ABCSystem.Employee).EMPIDIndexOpen("EMP0001")
```

```
TEST>do emp.%JSONExport()
```

```
{"Name":"やまだたろう","EMPID":"EMP0001","Location":"東京都"}
```

```
TEST>do emp.%JSONExportToString(.moji)
```

```
TEST>write moji
```

```
{"Name":"やまだたろう","EMPID":"EMP0001","Location":"東京都"}
```

```
TEST>do emp.%JSONExportToStream(.stream)
```

エクスポート例

```
TEST>write stream.Read()
```

```
{"Name":"やまだたろう","EMPID":"EMP0001","Location":"東京都"}
```



# 機能の追加(JSONアダプタ): インポートの例

- プロパティ名に合わせたJSONオブジェクトをインポート  
(UTF8で保存したファイルを使用している例)。

```
1 {  
2   "Name": "JSON太郎",  
3   "EMPID": "EMP0202",  
4   "Location": "東京都",  
5   "Area": "エリア1",  
6 }
```

input.jsonの中身

```
TEST>set emp=##class(ABCSysSystem.Employee).%New()
```

```
TEST>do emp.%JSONImport("c:¥temp¥input.json")
```

```
TEST>zwrite emp
```

```
emp=1@ABCSysSystem.Employee ; <OREF>
```

```
+----- general information -----
```

```
|   oref value: 1
```

```
|   class name: ABCSysSystem.Employee
```

```
| reference count: 2
```

```
+----- attribute values -----
```

```
|   %Concurrency = 1 <Set>
```

```
|   Area = "エリア1"
```

```
|   EMPID = "EMP0202"
```

```
|   Location = "東京都"
```

```
|   Name = "JSON太郎"
```

```
+-----
```



## ここからは・・・

- 独自スクリプト「ObjectScript」を使用したプログラミングを体験します。
- ObjectScriptは、クラス／テーブル定義を作成した時点で自動生成されるコードに使用されていますが、それ以外にも、ビジネスロジックの記述に利用できます。
- 以降のページでは、ABCSystem.Employeeクラスにクラスメソッドを追加しながら、スクリプトの記述方法を練習します。



# ObjectScriptとは

- インターシステムズ製品の独自スクリプトです。
  - ANSI/JISで標準化された言語をベースとし、完成度としては非常に高い言語です。(Mumps言語と互換性があります。)
  - データ操作以外にも一般言語と同等の操作ができます。
- スクリプトのほとんど全てに大文字小文字の区別があります。
  - 変数、メソッド、ルーチン、パッケージ、クラスは大文字小文字の区別があります。
  - SQLは区別がありません。(テーブル名、カラム名、ストアド名)
  - コマンドは、大文字小文字の区別はありません。
  - クラスの操作に使用する `##class` 指示文は、大文字小文字の区別はありません。
- 最初のうちは、全てに大文字小文字の区別があると考えておきましょう。



# まずはターミナルで Hello world !

- ObjectScriptは、ターミナルでインタラクティブにコマンド実行とその結果の確認が行えます。
- ターミナルを開き、WRITEコマンドを利用して、文字列「Hello world!」を出力してみましょう。

```
USER>write "Hello world!"  
Hello world!  
USER>
```

- 書き方:
  - コマンドに引数を指定するときはスペースを1つ入れます。
    - コマンドは大文字小文字どちらも対応できます。
  - 文字列は二重引用符で括ります。



# スタジオでプログラミング

- ターミナルは、コマンド実行の確認だけでなく、作成したプログラムのテスト実行にも最適です。
- プログラムの記述はターミナルでは無理なので、スタジオで記述します。
- スタジオは統合開発環境のため作成する用途に合わせエディタが異なります。
- プログラミングはルーチン単位で作成することもできますが、ストアードプロシージャなど外部からの呼び出しに備え、本資料ではクラスメソッドの作成を中心にご説明します。
  - 既存ルーチンをストアードプロシージャやメソッドとして外部から実行したい場合は、クラスメソッドを作成し、その中からルーチンを呼び出すことで実装できます。



# スクリプトの記述ルール

- ObjectScript の記述ルールは以下の通りです。
- メソッドにコードを追加する際、行頭にタブを挿入してからコマンドを書き始めます。
  - 行頭から書き始めた文字はラベルと認識されます。
- シンタックスは以下の通りです。  
*command argument1, argument2, ..., argumentN*
  - コマンドと引数の間には、1つのスペースを記入します。
  - 複数の引数を持てるコマンドの場合、引数をカンマで区切ります。
    - Set, Do, Job, Lock など複数引数を持てるコマンドがありますが、基本は `command arg1,arg2` は `command arg1 command arg2` と同等です。
- スクリプトには **\$** から始まる様々な関数があります。
  - 文字列操作関数
    - `$piece()` : 区切り文字のデータを操作する関数
    - `$length()` : 文字の長さを調べる関数
    - `$extract()` : 指定個所の部分抽出を行う関数
  - 変数チェック用
    - `$Get()` : 変数が存在しない場合、空("")を返します。
    - `$Data()` : 変数の存在や配列変数に下位のノードが存在するかチェックできます。
  - レコードデータをダイレクトアクセスで操作するときに使用する関数
    - `$LIST()`／`$LISTBUILD()`／`$LISTLENGTH()`



# ObjectScript

## よく使うコマンド

コマンド	内容	例文
SET	変数に値を割り当てる	Set val="あいうえお" 文字列は二重引用符で括ります。(SQL文では一重引用符です。)
WRITE	標準出力	Write val,! ターミナルでは！は改行を出力します。 引数無のWRITEはメモリ上の全変数の出力
KILL	変数の消去	Kill val 指定した変数を削除します。 引数無のKILLはメモリ上の全変数の消去
QUIT	プログラムの終了、FOR、WHILEのループの終了	QUIT retunVal 戻り値がある場合には引数を指定します。 ※次のループへジャンプするにはCONTINUEを使います。
DO	メソッド、プロシージャの実行	Do ##class(Training.Visit).Method() クラスメソッドの実行
ZWRITE	Writeと似ている ターミナル用コマンド	オブジェクトリファレンス、配列の全添え字情報 出力ができる便利コマンド(デバッグ用)

# コメントの記述方法

```
ClassMethod comment()
```

```
{
```

```
    // 1行コメント
```

```
    ; 1行コメント
```

```
    write 1,!
```

```
    /*
```

```
    複数行コメント
```

```
    */
```

```
    write 2,!
```

```
    #; 1行コメント + 中間コードに記載されないコメント文
```

```
    write 3,!
```

```
}
```

```
870 zcomment() public {  
871     // 1行コメント  
872     ; 1行コメント  
873     write 1,!  
874     /*  
875     複数行コメント  
876     */  
877     write 2,!  
878     write 3,! }
```

生成された中間コード





# ループの記述

- FORの記述は以下の通りです。

```
for カウンタ用変数=開始値:増(減)分値:終了値 {}  
for カウンタ用変数=開始値,増(減)分値,終了値 {}
```

- FORのオプション指定

- 終了値のみ指定なしでも、FOR文は増分または減分します。
- 引数を何も指定しない場合は、無限ループになります。
- 終了値の指定がない場合は、QUITでループを終了します。

- WHILEの記述は以下の通りです。

```
while (条件) { コード } または、do { コード } while (条件)
```

- do-whileは、1回は実行されます。

- FORとWHILE共通で、ある条件で次のループに移動する場合は、CONTINUEコマンドを使用します。



# ループの記述 ContinueとQuitの違い

```
ClassMethod loopContinue()
```

```
{  
    for i=1:1:5 {  
        if i=3 {  
            write "i=3 のとき continue → ループをスキップ",!  
            continue  
        }  
        write "i=",i,!  
    }  
    write "メソッドの終わり",!  
}
```

```
ClassMethod loopQuit()
```

```
{  
    for i=1:1:5 {  
        if i=3 {  
            write "i=3 のとき quit → ループを停止",!  
            quit  
        }  
        write "i=",i,!  
    }  
    write "メソッドの終わり",!  
}
```

```
USER>do ##class(Script.Sample).loopContinue()  
i=1  
i=2  
i=3 のとき continue → ループをスキップ  
i=4  
i=5  
メソッドの終わり
```

```
USER>do ##class(Script.Sample).loopQuit()  
i=1  
i=2  
i=3 のとき quit → ループを停止  
メソッドの終わり
```

# ループの記述 QuitとReturnの違い

ClassMethod loopQuit()

```
{  
  for i=1:1:5 {  
    if i=3 {  
      write "i=3 のとき quit → ループを停止",!  
      quit  
    }  
    write "i=",i,!  
  }  
  write "メソッドの終わり",!  
}
```

ObjectScript ではプログラムの終了にQuitまたはReturnを利用できますが動作が異なります (Returnは2013.1以降のバージョンで利用可)。

```
USER>do ##class(Script.Sample).loopQuit()  
i=1  
i=2  
i=3 のとき quit → ループを停止  
メソッドの終わり
```

ClassMethod loopReturn()

```
{  
  for i=1:1:5 {  
    if i=3 {  
      write "i=3 のとき return → メソッド停止",!  
      return  
    }  
    write "i=",i,!  
  }  
  write "メソッドの終わり",!  
}
```

```
USER>do ##class(Script.Sample).loopReturn()  
i=1  
i=2  
i=3 のとき return → メソッド停止
```



# メソッドの作成 (Areaを返すメソッドを作成します)

- クラス > 追加 > メソッド よりウィザードを起動します。

test/TEST@\_SYSTEM - Default\_system.prj - スタジオ - [ABCSystem.Employee.cls]

ファイル(F) 編集(E) 表示(V) プロジェクト(P) クラス(C) ビルド(B) デバッグ(D) ツール(T) ユーティリティ(U) ウィンドウ(W)

追加(A) プロパティ(P)...  
スーパークラス(S)... メソッド(M)...  
派生クラス(D)... クラスパラメータ(C)...

ABCSystem.Employee.cls

1 Class ABCSystem.Employee  
2 {

新規メソッドウィザード

メソッドシグニチャ

この新メソッドの戻りタイプ (任意) と引数リスト (任意)

戻りタイプ(T): %String

引数リスト:

名前	タイプ	デフォルト値	渡し方
input	%String		Value

名前: input  
タイプ: %String  
デフォルト:   
渡し方: ☒ 値 ☐ 参照

OK  
キャンセル(C)

メソッド名を指定します。

戻り値がある場合はタイプを指定します。

引数リストの設定

< 戻る(B) 次へ(N) > 完了 キャンセル ヘルプ

# メソッドの作成 つづき

- クラスメソッドとして定義したメソッドは、ストアドプロシージャ化することができます (SqlProc属性の利用)。

新規メソッドウィザード

メソッド属性

このメソッドの追加の性質を選択することができます:

- ☐ Private このメソッドはこのクラスに対してprivateです
- ☐ Final このメソッドはfinalです
- ☒ クラスメソッド このメソッドはクラスメソッドです
- ☒ SQLストアドプロシージャ このメソッドはSQLストアドプロシージャとして

言語:  
cache

戻る(B) 次へ(N) > 完了

新規メソッドウィザード  
インプリメンテーション

この新しいメソッドのソースコードを入力できます:  
%String(input:%String)

```
//引数が指定されなかったら  
if $get(input)="" {  
    return ""  
}  
//関数を利用して戻す値を指定します  
return $case(input,"東京都":"エリア1"  
    ,"大阪府":"エリア2"  
    ,"沖縄県":"エリア2"  
    |,:"エリア99")
```

メソッドコードはスタジオの主画面で修正もできます。

先頭文字にタブを1つ挿入してからコマンドの記述を開始します。

クラスメソッドにチェックを入れると「SQLストアドプロシージャ」にチェックができるようになります。

# SqlProc属性＝ストアドプロシージャ

- ストアドプロシージャを作成したい場合は、クラスメソッドを作成し、**SqlProc属性**を**True**に設定します。
- 定義したストアドプロシージャは、以下命名規則で実行します。  
スキーマ名.テーブル名\_メソッド名
  - 別名に変更することもできます(メソッド定義のSqlName属性に別名を登録します)。
- ストアドプロシージャの呼び出しには、CALL文を使用します。  
例) **call ABCSystem.Employee\_ProcName()**
- 結果セットを返さない場合、関数実行のようにSQL文中に指定できます。

**select ABCSystem.Employee\_GetArea('東京都')**

```
14 ClassMethod GetArea(input As %String) As %String [ Language = objectscript, SqlProc ]
15 {
16     //引数が指定されなかったら
17     if $get(input)="" {
18         return ""
19     }
20     //関数を利用して戻す値を指定します
21     //第1引数の値と第2引数以降のcaseを比較し、最初に一致する caseの値を返します。
22     //caseは左から評価されます。 case:値 で指定します
23     return $case(input,"東京都":"エリア1","大阪府":"エリア2","沖縄県":"エリア2",:"エリア99")
24 }
```



# 実行してみよう！

- ターミナルでテストする場合、戻り値をWRITE文で確認すると簡単です。

```
TEST>write ##class(ABCSysSystem.Employee).GetArea("大阪府")
エリア2
```

- 管理ポータルのSQL画面を利用する場合は、SELECT文で確認できます。

```
select ABCSysSystem.Employee_GetArea('東京都')
```

- 管理ポータルの「プロシージャ実行」画面でもテストできます。

InterSystems™  
IRIS Data Platform

管理ポータル

ホーム 概要 ヘルプ

サーバ 6f45803d364c ネームスペース TEST 変更 ユーザ \_SYSTEM ライセンス先 InterSystems IRIS Community インスタンス IRIS

システム > SQL

フィルタ ABCSystem\* 適用先 すべて

システム ☐ スキーマ ABCSystem

テーブル

ビュー

プロシージャ

ABCSystem.Employee\_Extent

ABCSystem.Employee\_GetArea

クエリキャッシュ

ウィザード > アクション > テーブルを開く ツール > ドキュメント

カタログの詳細 クエリ実行 参照 SQLステートメント

プロシージャ: ABCSystem.Employee\_GetArea ストアドプロシージャ情報

クラス名	ABCSysSystem.Employee	クラスのドキュメント
プロシージャタイプ	function	
メソッドまたはクエリ名	GetArea	プロシージャ実行
説明		
入力パラメータ数	1	
入力/出力 パラメータの数	0	
出力パラメータ数	0	
戻り値	1	
プロシージャ・インターフェース	2 戻り値を持つ関数 結果セットなし	

# 補足：外部I/Fからメソッドを実行したい！ という場合の一例

- データプラットフォームに定義したクラスメソッドは、SQLベースの外部I/Fからでは、直接実行することができません。
- 演習で体験したように「ストアードプロシージャ」属性をクラスメソッドに付与することで、外部I/Fに対してストアードプロシージャとして公開され、どこからでも呼び出すことができます。





# クラスコンパイルによる生成コード

- データプラットフォームでは、クラス定義＝テーブル定義として取り扱えます。
- クラス／テーブル定義コンパイル時、オブジェクトとSQLそれぞれのアクセスに対応した実行コードを生成しています。

スキーマ名.テーブル名.番号

```
10/27/2021 06:46:13 に修飾子 'cukb /checkuptodate=expandedonly' でコンパイルを開始しました。  
クラスのコンパイル中 ABCSystem.Employee  
テーブルのコンパイル中 ABCSystem.Employee  
ルーチンのコンパイル中 ABCSystem.Employee.1  
コンパイルが正常に終了しました (所要時間: 0.342秒)。
```

スタジオ出力画面

- クラスのコンパイルごとに生成ルーチンは、削除／再作成されます。
  - 小さな変更では、差分コンパイルが行われ、既存ルーチンを削除することなく小さな追加のルーチンを生成します。
- クラス操作の中でエラーが発生した場合は、生成ルーチンの行番号でエラー発生個所を示します。
  - 表示 > 他のコードを表示 をクリックすると、クラスの生成コードをスタジオで開くことができます(参照のみ)。



# 演習内容のイメージ

Objectで更新

## ABCSys~~tem~~.Employee

Name As %String  
EMPID As %String  
Location As %String  
Area As %String

ID	Name	EMPID	Location	Area
INTEGER	VARCHAR	VARCHAR	VARCHAR	VARCHAR

GetArea()

= ストアド: ABCSystem.Employee\_GetArea()

生成コード

生成コード+  
ユーザコード

グローバル変数

^ABCSys~~tem~~.EmployeeD、^ABCSys~~tem~~.EmployeeI

ID	Area	EMPID	Location	Name
1		EMP0001	東京都	やまだたろう
3	エリア2	EMP0011	大阪	山田太郎
4	エリア2	EMP0901	沖縄県	佐々木花子

SQLで更新

```
1: ^ABCSystem.EmployeeD = 4
2: ^ABCSystem.EmployeeD(1) = $!b("", "やまだたろう", "EMP0001", "東京都", "")
3: ^ABCSystem.EmployeeD(3) = $!b("", "山田太郎", "EMP0011", "大阪府", "エリア2")
4: ^ABCSystem.EmployeeD(4) = $!b("", "佐々木花子", "EMP0901", "沖縄県", "エリア2")
```

```
1: ^ABCSystem.EmployeeI("EMPIDIndex", "EMP0001", 1) = ""
2: ^ABCSystem.EmployeeI("EMPIDIndex", "EMP0011", 3) = ""
3: ^ABCSystem.EmployeeI("EMPIDIndex", "EMP0901", 4) = ""
```

# エラーステータス(%Status)について

- システム提供クラスでは、処理の成功／失敗のステータス(%Status)をメソッドの戻り値や引数で返すクラスが多数あり、実行結果の成功可否の判断に使用します。
  - %Statusが設定されている場合、成功したときは1、エラー時はオブジェクトでエラー情報が渡されます。 例) set status = human.%Save()
  - メソッドの実行結果を取得し損ねた場合は、デフォルトで用意される%objlasterror変数にステータスが格納されます。
  - %objlasterrorは、直近で発生したオブジェクトのエラーステータスを格納するローカル変数です(明示的に中身をクリアしない限り情報は残ります)。
- エラーステータスの解析には**%SYSTEM.Statusクラス**を利用します。

\$SYSTEM.Status.メソッド名()  
で実行できます。

\$system.Status.xxx	用途
IsError(st)	引数のステータスコードがエラーであるとき1、それ以外は0を返す。
GetErrorText(st)	引数のステータスコードのエラーテキストを返す。複数エラーが発生した場合は、エラーテキストをCR+LFで区切って返す。
GetOneStatusText(st,#)	複数エラーが発生した場合、第2引数に指定した番号のエラーテキストのみを返す。
DecomposeStatus(st,.err)	エラーステータスを分解して第2引数の変数に代入する。
StatusToSQLCODE(st,.msg)	エラーステータスからSQLCODEを返します。第2引数はエラーメッセージが設定される出力引数です。

# エラー処理: 便利なマクロ

- クラス／テーブル定義では、オブジェクトのステータス確認に利用するマクロなど、**%occStatus.inc** から自動的に提供されます。
  - 何も継承しないクラス定義(メソッドのコンテナ)では提供されません。

例

```
ClassMethod A() As %Status
{
  set st=$$$OK // 戻り値の初期値設定
  // 入力アシスト機能を利用するための指定
  #dim ex As %Exception.AbstractException
  try {
    // コードの記述
    /* %Statusを戻すメソッドの呼び出しなど */
    $$$ThrowOnError(st)
  }
  catch ex {
    set st=ex.AsStatus() //エラー時の戻り値を設定
    /* 任意のエラー処理 */
  }
  quit st //戻り値を持つ場合に記述
}
```

%occStatus内マクロ(一部)	内容
\$\$\$ISERR()	%Statusコードがエラーの場合1を返します。
\$\$\$ThrowOnError(st)	ステータスがエラーの場合例外を生成しCATCHブロックへTHROWします。
\$\$\$OK	%StatusでOK(=1)を返すときに使用します。



# クラスリファレンス(クラスドキュメント)

- 保存したクラスのクラス定義ドキュメントを自動的に作成します。
- スタジオでは、以下メニューから起動できます。
  - ワークスペースウィンドウのクラス名を右クリックし、クラスドキュメントの表示をクリックします。
  - クラスを編集集中に表示→クラスドキュメントの表示をクリックします。
- ドキュメンテーションホームページから、クラスリファレンス情報をクリックします。

%付きクラス詳細も確認できます。

INTERSYSTEMS クラスリファレンス Training.Person

ドキュメント | 検索

[kiso] > [Training] > [Person]

作成したクラス参照できます。

persistent class **Training.Person** extends [%Persistent](#)

▼ Inventory

Parameters	Properties	Methods	SystemMethods	Queries	Indices	ForeignKeys
	2	4			1	

▼ Summary

プロパティ

Email	Name
-------	------

メソッド

%AcquireLock	%AddToSaveSet	%AddToSyncSet	%E
%CheckConstraints	%CheckConstraintsForExtent	%ClassIsLatestVersion	%C
%ComposeOid	%ConstructClone	%ConstructCloneInit	%L
%DeleteData	%DeleteExtent	%DeleteId	%L
%DispatchGetModified	%DispatchGetProperty	%DispatchMethod	%L

# ここまでのまとめ

オブジェクトをデータベースに格納できる機能を持ったクラスを作成できること

永続化の機能を持ったクラス定義は、テーブル定義としても取り扱えること  
(=マルチモデルデータのサポート)

オブジェクトもレコードデータもデータプラットフォームの中ではグローバル変数と呼ばれる単純な配列変数として格納されること

グローバル変数に対してSQLでもオブジェクト操作でもなく直接アクセスすることができること(=ダイレクトアクセス)

独自スクリプト(=ObjectScript)を使用してデータ操作以外の操作も簡単に実装できること

(ストアドプロシージャとして定義することで、外部I/Fからは一般的な呼び出し方法と変わりなく呼び出すことができます。)

弊社製品に関する良くあるご質問とその回答をFAQサイト([faq.intersystems.co.jp](http://faq.intersystems.co.jp))に掲載しています。

開発者同士の交流の場として技術的な質問&回答などが行える開発者コミュニティもあります([jp.community.intersystems.com](http://jp.community.intersystems.com))