Caché実践プログラミングガイド

**Current Document Version**

|  |  |  |
| --- | --- | --- |
| Version 0.9.1 | 2014-11-27 | Hiroshi Sato |

# Document Modifications

| **Version** | **Description of Change** | **Modified By** | **Date** |
| --- | --- | --- | --- |
| 0.9 | First Beta Version | Hioshi Sato | 2014-11-26 |
| 0.9.1 | Some changes | Hiroshi Sato | 2014-11-27 |

**Table of Contents**

[Caché実践プログラミングガイド 1](#_Toc404781960)

[Document Modifications 2](#_Toc404781961)

[はじめに 6](#_Toc404781962)

[インターシステムズテクノロジーの情報ソース 7](#_Toc404781963)

[Caché開発ガイド 7](#_Toc404781964)

[FAQサイト 7](#_Toc404781965)

[技術ガイド 8](#_Toc404781966)

[インターシステムズ・ユーザー・グループ 8](#_Toc404781967)

[モデルアプリケーションの設計 9](#_Toc404781968)

[1. データモデルクラス図の設計 10](#_Toc404781969)

[Project 11](#_Toc404781970)

[Customer 14](#_Toc404781971)

[Address 15](#_Toc404781972)

[Phase 16](#_Toc404781973)

[Activity 16](#_Toc404781974)

[Party 17](#_Toc404781975)

[Organization 17](#_Toc404781976)

[Person 17](#_Toc404781977)

[Member 18](#_Toc404781978)

[Manager 18](#_Toc404781979)

[開発基本編 20](#_Toc404781980)

[1. スタジオを使用したクラス定義 20](#_Toc404781981)

[クラス定義、プロパティ定義 21](#_Toc404781982)

[リレーションシップ定義 23](#_Toc404781983)

[List定義 24](#_Toc404781984)

[データベース登録、検索フォーム作成 24](#_Toc404781985)

[CSP 24](#_Toc404781986)

[Zen 27](#_Toc404781987)

[Zen Mojo 30](#_Toc404781988)

[開発に関連するその他の作業 51](#_Toc404781989)

[印刷 51](#_Toc404781990)

[Zen Reportウィザード 52](#_Toc404781991)

[テスト 56](#_Toc404781992)

[UnitTestクラスの作成 56](#_Toc404781993)

[テストデータ生成フレームワーク 74](#_Toc404781994)

[SQLのINSERT文によるデータ生成 74](#_Toc404781995)

[%Populateクラスを使ったデータ自動生成 76](#_Toc404781996)

[%Populateクラスを使った少し複雑なデータ自動生成 84](#_Toc404781997)

[ソース世代管理 89](#_Toc404781998)

[コーディング規約 90](#_Toc404781999)

[ヘッダー情報 90](#_Toc404782000)

[ソース管理ツール 91](#_Toc404782001)

[見栄え 91](#_Toc404782002)

[命名基準 92](#_Toc404782003)

[コーディングの実際 94](#_Toc404782004)

[避けるべきこと 105](#_Toc404782005)

[エラー処理 106](#_Toc404782006)

[データの関連を処理する 107](#_Toc404782007)

[その他 108](#_Toc404782008)

[計算フィールド 108](#_Toc404782009)

[クエリー実行 108](#_Toc404782010)

[ファイルI/O 109](#_Toc404782011)

[Acrtive Analytics 110](#_Toc404782012)

[モデル作成 110](#_Toc404782013)

[前準備 110](#_Toc404782014)

[キューブの作成 110](#_Toc404782015)

[メジャーの定義 111](#_Toc404782016)

[ディメンジョンの定義 111](#_Toc404782017)

[詳細リストの定義 111](#_Toc404782018)

[計算メンバーの定義 111](#_Toc404782019)

[データ探索 112](#_Toc404782020)

[ダッシュボード作成 113](#_Toc404782021)

[パフォーマンス、スケーラビリティ 114](#_Toc404782022)

[性能を管理するために行うべきこと 114](#_Toc404782023)

[アプリケーション指標 114](#_Toc404782024)

[システムレベル指標 115](#_Toc404782025)

[よくあるパフォーマンス問題 121](#_Toc404782026)

[データロード時間 121](#_Toc404782027)

[インデックス構築 121](#_Toc404782028)

[ジャーナル、トランザクション 122](#_Toc404782029)

[クエリーパフォーマンス 122](#_Toc404782030)

[パラレルクエリー 123](#_Toc404782031)

[ローカル変数使用 123](#_Toc404782032)

[スケーラビリティ 123](#_Toc404782033)

[スケールアップ　or　スケールアウト 123](#_Toc404782034)

[ECP 124](#_Toc404782035)

[その他 125](#_Toc404782036)

[Windows Large Page問題 125](#_Toc404782037)

[ロックエスカレーション 126](#_Toc404782038)

[補足資料 127](#_Toc404782039)

[サンプルデータ 127](#_Toc404782040)

[サンプルファイルの構成 127](#_Toc404782041)

[セットアップ方法 130](#_Toc404782042)

[Zen Formの日本語化 131](#_Toc404782043)

[UnitTestサンプル 131](#_Toc404782044)

# はじめに

Cachéがリリースされて15年以上の年月が立ちました。

Cachéは元々シンプルな構成ながらも様々なことに柔軟に対応できる開発プラットフォームとして発展してきました。

しかしながら多岐にわたる外部環境の変化、顧客の要望に対する継続的な機能拡張によって数々の機能が追加されていき、結果としてシンプルさが失われたといわないまでも新らしくCachéを学ぶ人にとってどこから始めたらよいかわからないという声をよく聞くようになりました。

たしかに今ではCachéが用意している技術マニュアルは膨大な量になっており、とても短時間で全てを読み切れる分量ではありません。

そこでCachéを使ったシステム開発を一通りこなせるようになるための速習ガイドを企画しました。

もちろん限られた枚数の中で全てを網羅することはできませんが、少なくともこのガイドを習得した皆さんがインターシステムズのテクノロジーを活用したシステム開発の最初の大きな第一歩を進めるための一助となるよう有用な情報の提供を行っていきたいと思います。

# インターシステムズテクノロジーの情報ソース

このガイドでCachéの基本を習得いただいた後も技術の継続的な研鑽が必要です。

この章では、インターシステムズテクノロジーに関する技術的な情報を取得する際に有用となる情報源について紹介します。

## Caché開発ガイド

HTML版

<http://docs.intersystems.com/Cache20141j/csp/docbook/DocBook.UI.Page.cls?KEY=GORIENT>

PDF版

<http://docs.intersystems.com/documentation/cache/20141j/pdfs/GORIENT.pdf>

Cachéによるアプリケーション開発に関する内容を網羅的にカバーしたドキュメントです。

わかりやすく懇切丁寧に記述されているので、是非一読することをお勧めします。

なおurl上のcache20141jという部分は、今後Cachéのバージョンが上がるたびに更新版がリリースされ、それに伴いその部分が変わっていきます。

詳細は、以下のページを参照ください。

<http://www.intersystems.co.jp/cache/downloads/documentation.html>

## FAQサイト

<http://faq.intersystems.co.jp/csp/faq/FAQ.FAQTopicSearch.cls>

よくあるお問い合わせ内容をまとめたサイトです。

全文検索機能を持っているので、任意のキーワードで探したい内容を検索できます。

## 技術ガイド

<http://www.intersystems.co.jp/support/guide.html>

技術テーマ別に用意された自習ガイドです。

## インターシステムズ・ユーザー・グループ

<https://groups.google.com/forum/#!forum/intersystems-japan-usergroup>

インターシステムズのテクノロジーに関する情報交換を行うコミュニティサイトです。

# モデルアプリケーションの設計

Cachéのような開発プラットフォームを素早く理解するには、それを利用して実際にアプリケーションを構築するのが一番です。

その際に、ありきたりな単純なアプリケーションでは、用意された様々な機能を使うこともなく開発できてしまいます。

一方、本格的な実務に耐えうるようなアプリケーションでは、仕様策定を含めて詳細な作りこみが発生して構築に時間がかかりすぎてしまいます。

そこで、その間を取って実際のアプリケーションにありがちな要素を加味しながら、詳細にはあまり深入りしないようアプリケーションモデルを構築してみたいと思います。

そして、モデルを構築するにしても特定のドメインに特化するのも多くの読者の理解を難しくしてしまう懸念があるため、アプリケーション開発を経験された方であれば必ず何等かの形でなじみのあるプロジェクト管理をテーマとしたサンプルアプリケーションを構築していきたいと思います。

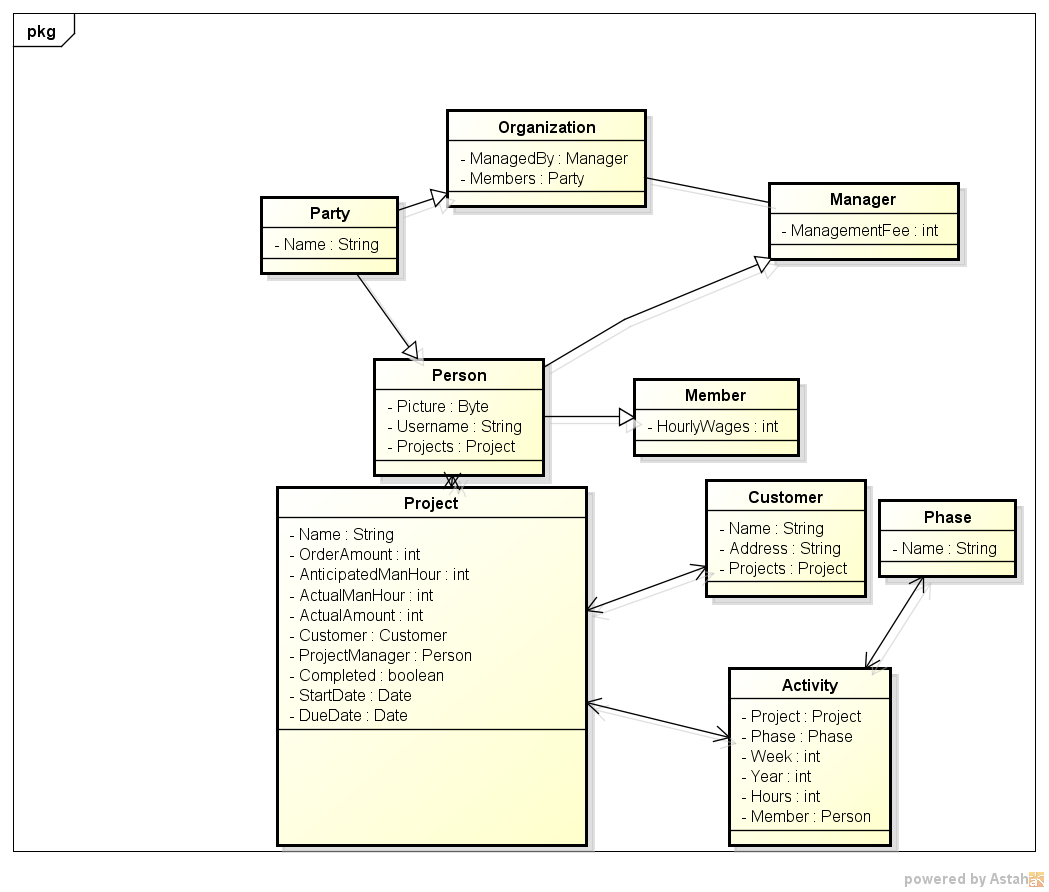
実際に自分でプログラムを入力しながら、逐次実行することで理解を深めることができますので、この文書に書かれている内容を入力（コピー＆ペースト）しながら試していただくことを推奨します。

但しそんな時間がないという方もいらっしゃると思いますので、この文書に記載しているクラス定義の最終形をひとまとめにしたファイルもこの文書と一緒に提供します。

その内容を確認しながら実際に動作させてみるというのでも一定の理解の手助けにはなると思います。

## データモデルクラス図の設計

以下に今回のアプリケーションのデータモデルの全体図を示します。



データモデルの作成方法にはいろいろな手法がありますが、ここではオブジェクト指向開発で一般的なUMLのクラス図で作成していきます。

UMLを記述するツールは無償、有償を含めていろいろなツールがあります。

残念ながらそれらのUMLツールとCachéは連携できていませんので、UMLツールでクラス図を作成した後、Cachéのスタジオを使って別途定義を入力する必要があります。

### Project

本システムの中心クラスです。

ここで定義するクラスは全てデータベースに格納するクラスになるので、%Persistentという型を継承するクラスとなります。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Name | 文字列型 | プロジェクトの名前 |
| OrderAmount | 整数型 | 受注額 |
| AnticipatedManHour | 整数型 | 予想工数（時間） |
| ActualAmount | 整数型 | 実際にかかっている工数（時間） |
| Completed | 論理型 | プロジェクトが完了しているか |
| StartDate | 日付型 | 開始年月日 |
| DueDate | 日付型 | 完了予定日 |

Cachéではクラスの属性のことをプロパティと呼びます。テーブルに対するカラム、フィールドとほぼ同等の意味です。

データ型は、システムが用意している型以外に自分でクラス定義を行うことにより拡張することが可能です。

ここでは、文字列型として%String、整数型として%Integer、論理型として%Boolean、

日付型として%Dateを使います。

これらはあらかじめシステムに組み込まれたシステムクラス（データ型）です。

#### 他クラスとの関係性を表現する属性

Cachéでは他クラスとの関係性を表現するためにもプロパティ（またはその派生形であるリレーションシップ）定義を使うことができます。

ここでは一対他のリレーションシップを使って、クラス間の関係性を定義します。

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| Customer | PM.Customer | Customerが一でProjectが多の関係 |
| ProjectManager | PM.Person | ProjectManagerが一でProjectが多の関係 |
| Activities | PM.Activity | Projectが多でActivityが多の関係 |

クラス間の関係性を定義する方法がリレーションシップのほかにもあります。

* 一対一の関係を表現する方法

1. 参照オブジェクト

プロパティの型を別のPersistent型を継承したクラス定義の名前を設定することにより参照オブジェクトを定義できます。

1. 埋め込みオブジェクト

プロパティの型を別のSerial型を継承したクラス定義の名前を設定することにより埋め込み型のオブジェクトを定義できます。

1. 一対一リレーションシップ

一対多のリレーションシップに制約を加えることで実現できるリレーションシップです。

多側のリレーションシップ定義に対してインデックス定義を作成し、そのインデックスにUnique属性を付加することにより、一対一のリレーションシップをエミュレーションできます。

* 一対多の関係を表現する方法

1. List Of 参照オブジェクト

あるクラスインスタンスへの参照を複数持つことを表現できます。

1. Array Of 参照オブジェクト

Listと同様あるクラスインスタンスへの参照を複数持つことを表現できます。

Listとの違いは、Listは格納順に特別な意味を持ちませんが、Arrayの場合は各要素にキーを設定してそのキーを使って要素を取得します。

1. List Of 埋め込みオブジェクト

対象が参照オブジェクトから埋め込みオブジェクトに変わる以外はList Of 参照オブジェクトと同じです。

1. Array Of 埋め込みオブジェクト

対象が参照オブジェクトから埋め込みオブジェクトに変わる以外はArray Of 参照オブジェクトと同じです。

1. 親子リレーションシップ

一対多リレーションシップよりも関係がより緊密な場合に親子リレーションシップというものを使うことができます。

一対多の関係の場合、片方が消滅したとしても相手方のほうが関係を切りさえすれば存続が可能なモデルに使えます。

一方親子リレーションシップは、ライフサイクルが同期する関係を表現するのに適切です。

つまり関係の片方が消滅する（削除）場合に相手側のインスタンスも一蓮托生の形で消滅するようなデータモデルに使用することが適切と考えられます。

一方実際にデータモデルを設計する際に、データの関係性をどのように定義するのが良いのかという点に関しては絶対的に正しい回答というものは存在しません。

とはいえ、何等かのガイドラインがあるとありがたいところです。

そこで以下のようなガイドラインを適用するのが一般的です。

|  |  |  |  |
| --- | --- | --- | --- |
| 関係モデル | ライフサイクル | 関係の濃度 | 方向 |
| 埋め込みオブジェクト | 依存 | 一対一 | 片方 |
| 参照オブジェクト | 非依存 | 一対一 | 片方 |
| 一対一リレーションシップ | 非依存 | 一対一 | 両方 |
| List Of 参照オブジェクト | 非依存 | 一対多 | 片方 |
| Array Of 参照オブジェクト | 非依存 | 一対多 | 片方 |
| 一対多リレーションシップ | 非依存 | 一対多 | 両方向 |
| List Of 埋め込みオブジェクト | 依存 | 一対多 | 片方 |
| Array Of 埋め込みオブジェクト | 依存 | 一対多 | 片方 |
| 親子リレーションシップ | 一対多 | 一対多 | 両方向 |

### Customer

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Name | 文字列型 | 顧客の名前 |
| Address | PM.Address | 住所（埋め込みオブジェクト） |

ここでは、住所のデータ型としてAddressクラスを定義しています。

#### 他クラスとの関係性を表現する属性

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| Projects | PM.Project | Customerが一でProjectが多の関係 |

### Address

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Zipcode | 文字列型 | 郵便番号 |
| Prefecture | 文字列型 | 県名 |
| City | 文字列型 | 都市名 |
| Street | 文字列型 | 町名、番地など |

Addressクラスは、%SerialObjectを継承しています。

%SerialObjectはいわゆる埋め込みオブジェクトを定義するために使われる基底クラスです。

%Persistent型のクラスと違い、自分自身で永続化する能力はありません。

%Persistent型のクラスのプロパティの型として定義することにより、その永続クラスをコンテナとして実体を生成することができます。

住所データの実装には非常に良く使われるデータであるにも関わらず、一般的に定まった定石

のような方法があるわけではなく、アプリケーションの設計者の裁量で設計されているケース

がほとんどだと思います。

結果として、住所１、住所２といった適当な区分をカラムとして追加したり、逆に県、市町村、町名、番地、ビル名など詳細な区分を使用するケースだったり、さらに同一アプリケーションの入力フォーム毎に仕様が異なったりします。

何故こういうことになるのでしょうか？

住所情報に構造を持たせようとすると、リレーショナルデータベースの観点で考えると

住所テーブルというものを作るという結論となります。（第一正規化の規則により）

しかし、実際に住所テーブルを作るとなると、非常に難しいというか厄介な問題があります。

住所テーブルのレコードは、例えば集合住宅の住人の住所となると、番地だけではなく

その集合住宅の号室まで必要になります。

大規模集合住宅では何棟、何館などの区分もあるかもしれません。

結局現実的に実用的な住所テーブルを作るというのは非常に手間がかかってコストに

見合わないケースがほとんどだと思います。

こういうケースに埋め込みオブジェクトを使用すると、上記の課題に柔軟に対応できます。

リレーショナルデータベースのモデルでは、この構造は正規化の制約条件（第一正規化）よりNGですが、Cachéではこのモデルを適切に使用することによってより柔軟に現実的なデータモデルを構築できると思います。

１データ１箇所の原則に反することになりますが、逆にこの原則を厳密に守ろうとすると、

データモデルの柔軟性を失う例があるということだと思います。

### Phase

プロジェクトの各フェーズを定義するクラスです。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Name | 文字列型 | Phaseの名前 |

### Activity

プロジェクトメンバーの作業を管理するクラスです。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Week | %Integer | 何週目の作業かを示す |
| Year | %Integer | 西暦 |
| Hours | %Integer | 作業時間 |
| Phase | PM.Phase | フェーズ |

#### 他クラスとの関係を表現する属性

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| Project | PM.Project | Activityが多でProjectが一の関係 |
| Member | PM.Person | Activityが多でMemberが一の関係 |

### Party

ここで組織的な構造をモデル化する際に良く利用されるオブジェクトモデリングのコンポジットパターンを使ってクラスを定義していきます。

Partyは人や組織の集合を表す抽象的なクラスです。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Name | 文字列型 | 人や組織の名前 |

### Organization

組織を表すクラスです。

PM.Partyを継承します。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Members | List Of PM.Party | Partyクラス型のリスト |

人または組織が所属できることを表現しています。

#### 他クラスとの関係性を表現する属性

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| Manager | PM.Manager | この組織を管理するマネージャ |

### Person

PM.Partyを継承します。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| Picture | %Stream.GlobalBinary | 写真データ |
| Username | %String | システムにログインするユーザー名 |

写真の様なバイナリーデータを保持するタイプとして%Stream型が用意されています。

%Stream型のデータはいくつか種類がありますが、ここでは、データを直接データベース内に格納するGlobalBinary型を使用します。

#### 他クラスとの関係性を表現する属性

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| Projects | PM.Project | Personが一でProjectが多 |
| Activities | PM.Activity | Personが一でActivityが多 |
|  |  |  |

### Member

PM.Personを継承します。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| HourlyWages | %Integer | 時給 |

### Manager

PM.Personを継承します。

#### 一般的な属性

|  |  |  |
| --- | --- | --- |
| プロパティ名 | データ型 | 説明 |
| ManagementFee | %Integer | 一般的なmemberと異なり固定給 |

#### 他クラスとの関係性を表現する属性

|  |  |  |
| --- | --- | --- |
| リレーションシップ名 | データ型 | 説明 |
| ManagedOrganization | PM.Organization | Managerが一でOrganizationが多 |

# 開発基本編

## スタジオを使用したクラス定義

前章で設計したクラスを実際にCachéのクラスとして定義してみましょう。

クラス定義の方法もいくつか用意されていますが、ここでは一番良く使われるスタジオを使った定義を行います。

スタジオを使うためには、まずCachéのインストレーションが必要ですが、ここでは詳細は説明しません。

インストレーションの方法は以下をご参考下さい。

<http://docs.intersystems.com/cache20141j/csp/docbook/DocBook.UI.Page.cls?KEY=GCI>

またここでは全てのクラス定義の詳細を説明することはしません。

クラス定義を行うにあたってポイントとなる点を例を示しながら説明していきたいと思います。

Windowsのツールバーに表示されているCachéキューブ（ブルーの立方体のアイコン）をクリック（右クリック、左クリックどちらでもOK）します。

そうすると、メニューが表示されますので、そこからスタジオ(d)をクリックします。

ユーザー名、パスワードを求められるケースがあると思いますが、

ユーザー名　\_system

パスワード　SYS（大文字）

を入力してください。

もしスタジオにログインできなければ、ユーザーの設定が変更されている可能性があります。

（この場合には実際に設定を行った人に確認が必要です。）

スタジオのウィンドウが表示されるはずですが、一番上のタイトルの所にCache/USER@XXX のように表示されているか確認してください。

もしUSERの所が違っていれば、以下の操作を行います。

ファイル（F）>ネームスペース変更(h)...をクリック

表示されるダイアログボックスのネームスペースメニューからUSERをクリックします。

### クラス定義、プロパティ定義

ファイル（F）>新規作成(N)

または

メニューバーの下に表示されるツールバーの一番右側の新規作成アイコン

（カーソルをアイコンの近くに持っていくとツールチップに新規作成（Ctrl+N）が表示される）

左側のカテゴリペインから一般をクリック

右側のテンプレートペインからCachéクラスをクリック

OKボタンをクリック

新規クラスウィザードが表示されるのを確認

パッケージ名　PMと入力

クラス名 Phaseと入力

次へ(N)>ボタンをクリック

クラスタイプ Persistentを選択

次へ(N)>ボタンをクリック

追加の属性

XML有効とデータ生成のチェックボックスをチェック

完了ボタンをクリック

クラス(C)>追加(A)>プロパティ(P)をクリック

新規プロパティウィザード

この新しいプロパティの名前を入力して下さい：Name

プロパティタイプ

単一値タイプ：　%String

完了ボタンをクリック

ビルド(B)>コンパイル(C)をクリックするか

ツールバー上のコンパイルボタン（真ん中のコンボボックスの右隣）をクリック

他のクラスも同上の手順でクラス定義とプロパティ定義を行うことができます。

プロパティ定義は慣れてくるとわざわざプロパティメニューを毎回クリックするのが面倒になってきます。

その場合は、表示されているプロパティ定義をコピー＆ペーストし、必要な内容（プロパティ名、タイプ）を修正することで新しいプロパティ定義を作ることもできます。

### リレーションシップ定義

リレーションシップもプロパティと同様に定義できます。

ここではPM.PersonとPM.Acitvityのクラス定義がある程度終わっていることを前提として説明します。

ファイル(F)>開く(O)

PM.Personをクリック

クラス(C)>追加(A)>プロパティ(P)

名前:　Activities

プロパティタイプ リレーションシップを選択

次へ(N)ボタンをクリック

このリレーションシッププロパティの参照:

多(M): 他の多くのオブジェクトを選択

このリレーションシッププロパティは次のオブジェクトを参照

PM.Activityと直接入力するか

参照(R)ボタンを押してPM.Activityをクリック

参照するクラスの対応するプロパティの名前

Memberと入力

他のリレーションシップも同様の手順で定義できます。

### List定義

ファイル(F)>開く(O)

PM.Organizationをクリック

クラス(C)>追加(A)>プロパティ(P)

名前:　Members

プロパティタイプ コレクションタイプを選択

横のコンボボックスからlistを選択

含まれる要素タイプ

PM.Partyを直接入力するか参照ボタンを押してPM.Partyを選択

完了ボタンをクリック

## データベース登録、検索フォーム作成

Cachéでデータベース登録するためのフォームは様々な方法で作成が可能です。

クライアントサーバー全盛の時代では、クライアントの画面設計ツール、例えばマイクロソフトのビジュアルスタジオなどを使い画面設計し、.NETのプログラミング言語（C##やVisual Basic.Net）とCachéが通信する方式が主でしたが、昨今では、Web技術を使ったフォーム設計が良く使われるようになってきました。

CachéはWebでのアプリケーション開発を支援するためのいくつかのフレームワークを持っています。

### CSP

CSP（Caché Server Pages）は最も基本的なWebアプリケーション開発支援フレームワークです。

類似技術としてJavaのServletとJSPの組み合わせあるいはASP.NETがあります。

サーバー側でプログラムを実行し、データベースをアクセスした結果をHTML（あるいはXML）形式に整形し、クライアントのブラウザに表示させる（またはその逆にクライアントから入力されたデータを使ってデータベースにデータを登録、更新する）という技術になります。

HTMLによる描画を含めて全てプログラミングによって制御することにより非常に柔軟な画面設計、処理を実装できる反面、プログラミングの手間が多い手法です。

また昨今アプリケーション設計上望ましいと考えられているMVC（Model View Controler）手法によるインタフェースとモデルおよびビジネスロジックの完全な分離が難しくなります。

そこでプログラミングの手間の軽減を目的としたCSPフォームウィザードというものを用意しています。

このウィザードを使用することによりマスターメンテナンスのような比較的単純な入力フォームを簡単に生成することができます。

#### ウェブフォームウィザード

スタジオ>ファイル>新規作成

カテゴリペインからCSPファイルを選択

Caché Server Pageをクリック

<!-- Put your page code here -->の下の

My page bodyを削除

カーソルは<!-- Put your page code here -->の直下に置く

スタジオ>ツール(T)>テンプレート>テンプレートをクリック

ウェブフォームウィザードをクリック

PM.Phaseをクリック

次へをクリック

Nameをクリック

次へをクリック

完了をクリック

エディタ上にコードが生成されるのを確認

表示（V）>ブラウザで表示（b）をクリック

名前を付けて保存の所でcsp/userをダブルクリック

ファイル名をphase.CSPとして名前を付けて保存ボタンをクリック

ファイルの種類がCachéサーバページになっていることを確認

デフォルトブラウザが立ち上がり、シンプルなウェブフォームが表示されるのを確認

### Zen

ZenはCSP上でコンポーネントを主体としたWebアプリケーションを構築することを支援する目的で開発されたフレームワークです。

CSPのようにHTMLの描画をプログラミングで行うのではなく、XML形式でレイアウトとそこに配置するコンポーネント（ウィジェット）の表示定義を行い、それらのコンポーネントに対するアクションをプログラミングしていきます。

クライアント側はJavaScriptでサーバー側はCaché ObjectScript(Caché Basicも選択できますが、ほとんど利用されていません)で処理を実装していきます。

サーバー（Caché）側とクライアント（ブラウザ）側でフォームのDOM(Document Object Model)を同期しながらその内容を維持管理していきます。

つまりDOMの要素をサーバー側、クライアント側両方で操作することができます。

またMVCのフレームワークも含まれています。

#### Zen フォーム

CSPウェブフォームウィザードと同じようにZen フォームを使って簡単なデータ入力フォームを作成することができます。

Zenフォームを使用するためには、事前にデータモデルを定義しておく必要があります。

通常MVCの観点からデータモデルとデータソースクラスを分けることが推奨されますが、ビューとデータソースがほぼ一致するケース（マスターメンテナンス用フォームのような）では便宜的にデータソースクラスをデータモデルとして見せることができます。

そうするためには、ソースクラスを%ZEN.DataModel.Adaptorを継承するようにします。

ここでは、PM.Phaseクラスを%ZEN.DataModel.Adaptorを継承するように修正します。

スタジオでクラス定義のExtendsの所で%ZEN.DataModel.Adaptorを追加します。

Class PM.Phase Extends (%Persistent, %Populate, %XML.Adaptor, %ZEN.DataModel.Adaptor)

修正後、コンパイルを行います。

スタジオからファイル（F）>新規作成(N)をクリック

カテゴリZenをクリックし、テンプレート Zen フォームをクリック

OKボタンをクリック

Zen Form Wizardダイアログボックス上で以下を入力

パッケージ名: PM

新規クラス名: PhaseEntry

アプリケーション？　なし

次へをクリック

以下のコンボボックスからPM.Phaseをクリック

フォームデータのパッケージとクラス:

フォームタイトル: Phase Input

完了ボタンをクリック

スタジオからPM.Phaseを開く（ファイル（F）>開く（O））

表示（V）> ブラウザで開く（b）をクリック

デフォルトのブラウザに簡単なフォーム画面が表示されることを確認

残念ながらZenフォームウィザードは日本語化されていません。

そのため、日本語化を支援するツールを別途用意しています。

（最終形を含むクラスに含まれます。詳しくは補足資料を参照してください。）

### Zen Mojo

モバイルをターゲットにZenを拡張したZen Mojoがリリースされました。

当初Zen Mobileという名称だったのですが、モバイルだけではなくデスクトップ用の開発にも使えるということになり名称が変更されました。

ZenはHTML5やCSS3などWeb開発の基本となる技術を使用しています。

しかし、なるべくそれらの詳細を隠ぺいし、CachéのオブジェクトモデルをベースとしたZenフレームワークを主体とした開発環境を提供します。

そうすることで開発者の作業負担を軽減することをゴールに設計されました。

つまり、Webアプリケーションに内在する煩雑なサーバークライアント間のデータ交換、セッション管理等をオブジェクトとして隠ぺいすることで開発者にアプリケーションロジックにあまり関係ない処理の記述に余計な時間を割く必要がない環境を提供しています。

結果としてサーバー側で描画を含めた様々なコードを自動生成し、それをクライアントのブラウザーに返信するというのが基本的なアーキテクチャになります。

しかし、モバイルをクライアントとした環境では、通信が無線主体であるという特性からなるべくネットワーク帯域を浪費しないアーキテクチャーが求められるようになりました。

サーバーで必要な描画情報をHTMLで生成し、それをクライアントに返信するというZenのアーキテクチャーがモバイルの要件に合わない部分です。

さらにJQueryなどの様々なJavaScriptのフレームワーク（ライブラリー）が利用されるようになってきました。

ここでも、なるべくZen側でいろいろなコントロールを行うというアーキテクチャーが、外部のJavaScriptライブラリーとの共存を難しくしています。

こういう時代背景に答えるために新たにZenの拡張フレームワークとして開発されたのが、Zen Mojoになります。

Zen Mojoの特徴は

* 描画は基本的にクライアントが担当
* 外部JavaScriptライブラリと共存ができるようにプラグイン形式で外部JavaScriptライブラリーの機能を取り込めるように設計
* サーバーとクライアント間のデータ交換にはJSONを使用
* クライアント側の描画コンポーネントのデータはJSONオブジェクトから取得、入力したデータはJSONオブジェクト経由でサーバーに渡される。
* シングルページアーキテクチャを採用し、ページ遷移は、個々のページを積み上げる方法（またはその逆）で実装

となります。

結果としてZen以上にHTML5、CSS3、各種JavaScriptライブラリーの知識が求められます。

お互い一長一短ありどちらが絶対に良いとはなかなか言いにくい状況ですが、時代の趨勢はZen Mojoに傾きつつあるかなと思います。

もちろんZen Mojoも生まれたての技術になりますので、今後も様々な機能拡張が行われていき、Zenに対して不利、不足な面も改善されていく可能性があります。（当然その方向に進化していくと信じたいところです。）

それでは、Zen Mojoを使った簡単なアプリケーションを作ってみましょう。

以下の作業の前にZen Mojoのインストールが必要です。

なおZen Mojoは現時点(2014年11月）では標準Cachéのキットには含まれません。

キットの入手にはインターシステムズジャパンのカスタマーサポートセンターまでお問い合わせいただく必要があります。（jpnsup@intersystems.com）

#### Zen Mojoを使用したデータ入力フォームの開発

Zen Mojoには現状まだフォームウィザードは用意されていませんので、プログラミング作業が必要です。

Zen Mojoアプリケーションは、通常以下の３つのクラスで構成されます。

* ページクラス
* テンプレートクラス
* アプリケーションクラス

この内アプリケーションクラスは必須ではありません。

ページクラスとテンプレートクラスを通常のクラス定義と同様に行うことも可能ですが、ひな形を生成してくれるウィザードがスタジオに用意されています。

以下の手順は、Zen Mojoのインストレーションが終了した前提で記載しています。

ファイル(F)>新規(N)

カテゴリZenをクリック

Zen Mojoサイトをクリックし、OKボタンをクリック

Zen Mojo Wizardのダイアログボックスで以下を入力

Package Name: PM

Page Class Name: ActivityEntryPage

Template Class Name: ActivityEntryTemplate

1. ActivityEntryPage（ページクラス）

ページの内容をどういう形で提供するかを定義します。

レイアウトを表示する方法

データを取得する方法

使用するプラグインおよび関連するインクルードファイル(CSS,JS)

|  |
| --- |
| /// Name of the default template class that this page uses.  Parameter TEMPLATECLASS = "PM.ActivityEntryTemplate";  Parameter JSINCLUDES As STRING = "dojo-release-1-9-1/dojo/dojo.js";  Parameter CSSINCLUDES As STRING = "dojo-release-1-9-1/dijit/themes/claro/claro.css,dojo-release-1-9-1/gridx/resources/claro/Gridx.css";  /// This XML block defines the contents of the pageContents pane.  XData pageContents [ XMLNamespace = "http://www.intersystems.com/zen" ]  {  <pane xmlns="http://www.intersystems.com/zen"  xmlns:mojo="http://www.intersystems.com/zen/mojo" layout="none">  <mojo:documentView id="mainView"  developerMode="false"  ongetlayout ="return zenPage.getContent('mainViewLayout',key,criteria);"  ongetdata ="return zenPage.getContent('data',key,criteria);">  <mojo:dojo-1.9.1-PageManager>  <mojo:HTML5Helper/>  <mojo:dojo-1.9.1-DijitHelper/>  <mojo:mojoDefaultHelper/>  </mojo:dojo-1.9.1-PageManager>  </mojo:documentView>  </pane>  } |

dojoのコンポーネントを使用するためにdojoのプラグインを読み込んでいます。

HTML5のコンポーネントも使用するためにhtml5のプラグインを読み込んでいます。

1. ActivityEntryTemplate（テンプレートクラス）

実際の細かいページ描画や必要なデータ注入処理はテンプレートクラスで行います。

onGetContentメソッドでレイアウトの取得方法およびデータの取得方法を記述します。

データの取得処理に何も記述しない場合には、サーバーからデータを毎回取得します。

|  |
| --- |
| ClientMethod onGetContent(providerName, key, criteria) [ Language = javascript ]  {  var content = null;  // dispatch to convenient methods  switch(providerName) {  case 'mainViewLayout':  content = this.myGetMainViewLayout(key,criteria);  break;  case 'data':  // always fetch data from server for this sample  }  return content;  } |

onGetContentで指定したレイアウトの取得メソッドの記述方法は以下のようになります。

指定できるレイアウトコンポーネントの種類（$で始まる名前）および指定できる属性は、プラグインによって変わります。

プラグイン毎にどういうコンポーネント、属性定義が利用可能かは、プラグインドキュメンテーションで確認してください。

<http://localhost:57772/csp/samples/%25ZEN.Mojo.PluginDocumentation.HomePage.cls>

|  |
| --- |
| ClientMethod myGetMainViewLayout(key, criteria) [ Language = javascript ]  {  var myLayoutGraph = {};  //The standard technique is to have a switch/case construct based on the key argument.  //In this case, the layout is not key-specific layout, so there is no need to branch.  myLayoutGraph = {  children: [  {type:'$ContentPane',key:'layoutContainer-1',style:'width:100%;height:100%;',children:[  {type: '$header', $content: 'Activity Input',style:'font-size: xx-large;text-align: center;'},  {type: '$br'},  {type: '$label', $content: 'MemberId: ',labelclass:'.labelAlign300px',style:'width:100;'},  {type:'$input',key:'memberid-txt',inputType: 'text',readonly: 'yes',value:'=[memberId]'},  {type: '$br'},  {type: '$label', $content: 'MemberName: ',labelclass:'.labelAlign300px',style:'width:100;'},  {type:'$input',key:'membername-txt',inputType: 'text',readonly: 'yes',value:'=[memberName]'},  {type: '$br'},  {type:'$Select',key:'project-cbx',baseclass:'labelclass',label:'Project: ',labelclass:'.labelAlign300px',title:'Select Project',valueList:'=[projectList]'},  {type: '$br'},  {type:'$Select',key:'phase-cbx',baseclass:'labelclass',label:'Phase: ',labelclass:'.labelAlign300px',title:'Select Phase',valueList:'=[phaseList]'},  {type: '$br'},  {type: '$label', $content: 'Working Hours: ',labelclass:'.labelAlign300px',style:'width:100;'},  {type: '$input',key:'activitytime-input',inputType: 'text'},  {type: '$br'},  {type:'$DateTextBox',key:'activity-datetxt',label:'Select a Week: ',labelclass:'.labelAlign300px',title:'Select Week', value:'=[today]',style:'position:absolute;left:300;'},  {type: '$br'},  {type:'$Button',key:'activitysave-btn',label:'Save',title:'Save the content',value:'SaveButton'}  ]}  ]}    return myLayoutGraph;  } |

データを取得するためのサーバー側のメソッド%OnGetJSONContent()を記述します。

取得したデータはJSONオブジェクトの形式にまとめます。

|  |
| --- |
| ClassMethod %OnGetJSONContent(pWhich As %String, pKey As %String, ByRef pParms, Output pObject As %RegisteredObject, pCriteria As %RegisteredObject, pLoad As %Boolean = 0) As %Status  {  Try {  set tSC = $$$OK  #dim sql As %String = ""  #dim tsc As %Status = $$$OK  #dim result As %SQL.StatementResult    set pObject = ##class(%ZEN.proxyObject).%New()  set pObject.projectList = ##class(%ListOfObjects).%New()    set statement = ##class(%SQL.Statement).%New()  set sql = "SELECT ID, Name FROM PM.Project"  set tsc = statement.%Prepare(sql)  if $$$ISERR(tSC) $$$ThrowStatus(tSC)  set result = statement.%Execute()    while result.%Next() {  set project = ##class(%ZEN.proxyObject).%New()  set project.value = result.%Get("ID")  set project.text = result.%Get("Name")  do pObject.projectList.Insert(project)  }    set pObject.phaseList = ##class(%ListOfObjects).%New()  set sql = "SELECT ID, Name FROM PM.Phase"  set tsc = statement.%Prepare(sql)  if $$$ISERR(tSC) $$$ThrowStatus(tSC)  set result = statement.%Execute()    while result.%Next() {  set phase = ##class(%ZEN.proxyObject).%New()  set phase.value = result.%Get("ID")  set phase.text = result.%Get("Name")  do pObject.phaseList.Insert(phase)  }  set pObject.today = $ZDate($H,3)  set member = ##class(PM.Person).UsernameIndexOpen($UserName,,.tSC)  If $$$ISERR(tSC) $$$ThrowStatus(tSC)  set pObject.memberName = member.Name  set pObject.memberId = member.%Id()  }  Catch tE {  Set tSC2 = ##class(PM.Error).StoreErrorInformation(tE)  }  quit tSC  } |

1. Caché ObjectScriptによるプログラミング解説

$$$OK

名前の先頭に$を３個つけるとマクロ宣言となります。

システムインクルードファイル%occStatus.inc内のOKマクロを参照します。

コンパイルの段階で、まずマクロ内で定義されている内容でプログラミング上の記述が置換されます。

OKマクロは1と定義されていますので$$$OKが1というリテラル値に置換されます。

リテラルは変数より処理を高速に実行できる利点があります。

（昨今のCPUのスピードの速さを考えれば無視できる差ですが）

しかし、数値をそのままプログラムの中に埋め込むとその意味するところがわかりにくいという問題があります。

マクロを使うことで意味を提示しながら実行時の効率性も担保できます。

#dim

Caché ObjectScript言語では、変数の宣言は必要ありませんが、他言語で変数宣言することに慣れている開発者向けに変数宣言の方法も用意されています。

Caché ObjectScript言語で変数宣言をする際には#dim宣言を使用します。

##class()

Caché ObjectScript言語でクラスにアクセスするための宣言です。

set pObject = ##class(%ZEN.proxyObject).%New()

ここでは、%New()メソッドというクラスメソッドを呼び出して新規に%ZEN.proxyObjectクラスのインスタンスを生成しています

set pObject.projectList = ##class(%ListOfObjects).%New()

新規に%ListOfObjectsというオブジェクトのリストを格納するクラスのインスタンスを生成し、そのOREF（Object Reference）を%ZEN.proxyObjectクラスのインスタンスのprojectListというプロパティに設定しています。

|  |
| --- |
| set statement = ##class(%SQL.Statement).%New()  set sql = "SELECT ID, Name FROM PM.Project"  set tsc = statement.%Prepare(sql)  if $$$ISERR(tSC) $$$ThrowStatus(tSC)  set result = statement.%Execute()    while result.%Next() {  set project = ##class(%ZEN.proxyObject).%New()  set project.value = result.%Get("ID")  set project.text = result.%Get("Name")  do pObject.projectList.Insert(project)  } |

クエリーを発行するために%SQL.Statementクラスのインスタンスを生成し、クエリーを設定します。

クエリーはPM.projectから全件取得するクエリーです。

取得した結果を個数分繰り返し処理し、projectListに挿入していきます。

phaseListもほぼ同様の処理で実装します。

|  |
| --- |
| set pObject.today = $ZDate($H,3)  set member = ##class(PM.Person).UsernameIndexOpen($UserName,,.tSC)  If $$$ISERR(tSC) $$$ThrowStatus(tSC)  set pObject.memberName = member.Name  set pObject.memberId = member.%Id() |

pObject.todayに今日の日付を設定します。

$HはCaché ObjectScript言語での日付と時間を内部表現するための内部変数です。

このままの値では、人が理解するのが難しいので$ZDate関数を使って変換します。

ここでこの関数の第2引数は、日付形式を指定します。

形式3はYYYY-MM-DDです。

|  |
| --- |
| set member = ##class(PM.Person).UsernameIndexOpen($UserName,,.tSC)  If $$$ISERR(tSC) $$$ThrowStatus(tSC)  set pObject.memberName = member.Name  set pObject.memberId = member.%Id() |

あるクラスのインスタンスを取得する方法として、IDを指定してオープンする方法（%OpenId()メソッド）が一般的ですがユニークキーを指定してオープンする方法もあります。

取決め事項として

ユニークインデックスが設定されているプロパティ名+IndexOpenという名前のメソッドをその目的で使うことができます。

ここではUsenameというプロパティにユニークインデックスが設定されているのでUsernameIndexOpen()メソッドが利用可能です。

If $$$ISERR(tSC) $$$ThrowStatus(tSC)

UsernameIndexOpen()メソッドが何等かの理由でエラー終了した場合にはtSC変数にステータスオブジェクトのOREFが設定されます。

その値を$$$ThrowStatus()マクロに渡すことで例外を発生します。

例外はTry Catchのメカニズムで捕捉することができます。

Catch tE {

Set tSC2 = ##class(PM.Error).StoreErrorInformation(tE)

}

例外を捕捉した場合には、その例外情報をPM.Errorというクラスのインスタンスとしてデータベース化する処理を組み込んでいます。

データを入力してSave（保存）ボタンでサーバーデータベースに登録する処理は以下のようになります。

|  |
| --- |
| ClientMethod onselect(key, value, docViewId) [ Language = javascript ]  {  console.log('select '+key);  var mainView = zen(docViewId);  var realKey = key.split(':')[0];  switch(realKey) {  case 'activitysave-btn':  var id = mainView.getControlValue('memberid-txt');  var name = mainView.getControlValue('membername-txt');  var project = mainView.getControlValue('project-cbx');  var phase = mainView.getControlValue('phase-cbx');  var activitytime = mainView.getControlValue('activitytime-input');  var activitydate = mainView.getControlValue('activity-datetxt').toJSON();    var response = zenPage.submitData('activitysave-btn',{id:id,name:name,project:project,phase:phase,activitytime:activitytime,activitydate:activitydate});  if (response && response.error) {  alert(response.errorMsg);  }  else {  alert('Saved!');  }  zenPage.getContentProvider().invalidate('data');  //mainView.popDocument(true,true);  break;  }  } |

|  |
| --- |
| /// Submit data handler for content provider.<br/>  /// Overridden by subclasses.  ClassMethod %OnSubmitData(pKey As %String, pID As %String, pSubmitObject As %RegisteredObject, ByRef pResponseObject As %RegisteredObject) As %Status  {  #Dim tSC = $$$OK  set pResponseObject = ##class(%ZEN.proxyObject).%New()  Try {  if (pKey = "activitysave-btn") {  set activitydate = pSubmitObject.activitydate  set activity = ##class(PM.Activity).%New()  set activity.Hours = pSubmitObject.activitytime  set project = ##class(PM.Project).%OpenId(pSubmitObject.project)  set phase = ##class(PM.Phase).%OpenId(pSubmitObject.phase)  set activity.Project = project  set activity.Phase = phase  set activitydate = pSubmitObject.activitydate  set year = $Extract(activitydate,1,4)  set month = $Extract(activitydate,6,7)  set week = $system.SQL.WEEK(activitydate)  set activity.Member = ##class(PM.Person).%OpenId(pSubmitObject.id)  set activity.Week = week  if ((week = 52) && (month = "01")) {  set year = year - 1  }  set activity.Year = year  set tSC = activity.%Save()  if $$$ISERR(tSC) $$$ThrowStatus(tSC)  }  }  Catch(tE) {  Set tSC2 = ##class(PM.Error).StoreErrorInformation(tE)  set pResponseObject.error = 1  set pResponseObject.errorMsg = ##class(%SYSTEM.Status).GetErrorText(tE.AsStatus())  }  Quit tSC  } |

JavaScriptのzenPage.submitData()で渡されたJSONオブジェクト

{id:id,name:name,project:project,phase:phase,activitytime:activitytime,activitydate:activitydate}

は、サーバー側で呼び出される%OnSubmitData()メソッドのpSubmitObjectオブジェクトとして渡されます。

その際Zen Mojoが自動的にJSONオブジェクトに変換してくれますので、

set activitydate = pSubmitObject.activitydate

のようにCaché Object言語の標準クラスインスタンス形式でアクセス可能です。

#### Zen Mojoを使用したデータ表示フォームの開発

複数のアクティビティの一覧から１つのアクティビティを表示するフォームを作成してみましょう。

ここでは、JQueryのプラグインを利用します。

|  |
| --- |
| Parameter TEMPLATECLASS = "PM.ActivityQueryTemplate";  /// Comma-separated list of additional CSS3 include files for the page.  /// If this is defined \*and\* this page is using CSS3, then the CSSINCLUDES parameter is ignored.  /// If this is not defined \*and\* this page is using CSS3, then the CSSINCLUDES parameter is used.  Parameter CSSINCLUDES As STRING = "jquery.mobile-1.3.2.min.css";  /// Comma-separated list of additional JS include files for the page.  Parameter JSINCLUDES As STRING = "jquery-1.10.2.min.js,jquery.mobile-1.3.2.min.js"; |

ページクラスのレイアウトとデータの取得の定義は以下のようになります。

|  |
| --- |
| XData pageContents [ XMLNamespace = "http://www.intersystems.com/zen" ]  {  <pane xmlns="http://www.intersystems.com/zen"  xmlns:mojo="http://www.intersystems.com/zen/mojo"  layout="none">  <mojo:documentView id="mainView"  developerMode="false"  initialDocumentKey="home"  initialLayoutKey="home"  ongetlayout = "return zenPage.getContent('layout',key,criteria);"  ongetdata = "return zenPage.getContent('data',key,criteria);"  >  <mojo:jQM-1.3.2-PageManager jQueryAnimation="slide" onPageShow="zenPage.onPageShow(layoutkey,documentkey);">  <mojo:jQM-1.3.2-Helper/>  <mojo:HTML5Helper/>  <mojo:mojoDefaultHelper/>  </mojo:jQM-1.3.2-PageManager>  </mojo:documentView>  </pane>  } |

JQueryのプラグインを定義しています。

レイアウトとデータを取得するためのテンプレートの定義です。

今回レイアウトは2種類用意し、適宜切り替えて表示するようにしています。

（リスト表示と詳細表示）

|  |
| --- |
| ClientMethod onGetContent(which, key, criteria) [ Language = javascript ]  {  //console.log(which + ' - ' + key + ' - ' + criteria);  switch (which)  {  case 'layout':  return this.getLayout(key,criteria);  case 'data':  // always fetch data from server for this sample  }    // returning null -> fetch data from server  return null;  }  ClientMethod onselect(key, value, docViewId) [ Language = javascript ]  {  console.log('select '+key);  var mainView = zen(docViewId);  var realKey = key.split(':')[0];  switch(realKey) {  case 'drill-activity':  mainView.pushDocument('show-activity',{id:value},'show-activity',{id:value});  break;  }  }  /// Creates the layout object for the page  ClientMethod getLayout(key, criteria) [ Language = javascript ]  {  var content = null;      switch(key) {  case 'home' :  content = {  children:[  {type:'$header',caption:'=[sectionHeader]'},  {type:'$listview',value:'=[activityList]',filter:true,children:[  {type:'$listviewitem',key:'drill-activity',value:'=[id]',label:'=[projectname]',content:'=[yearweek]',clickable:true,labelNoWrapper:false}  ]}  ]  };  break;  case 'show-activity' :  content = {  children:[  {type:'$header',caption:'=[sectionHeader]'},  {type:'$form',children:[  {type:'$text',placeHolder:$$$Text('year'),value:'=[year]',fieldcontain:true,label:$$$Text('Year:'),key:'activity-year'},  {type:'$text',placeHolder:$$$Text('week'),value:'=[week]',fieldcontain:true,label:$$$Text('Week:'),key:'activity-week'},  {type:'$text',placeHolder:$$$Text('projectname'),value:'=[projectname]',fieldcontain:true,label:$$$Text('ProjectName:'),key:'activity-project'},  {type:'$text',placeHolder:$$$Text('phasename'),value:'=[phasename]',fieldcontain:true,label:$$$Text('PhaseName:'),key:'activity-phase'},  {type:'$text',placeHolder:$$$Text('hours'),value:'=[hours]',fieldcontain:true,label:$$$Text('Hours:'),key:'person-hours'}  ]},  ]  };  break;  }    return content;  } |

サーバー側データ取得処理です。

ホームスクリーンと詳細スクリーンで取得するデータが異なっているのがわかると思います。

|  |
| --- |
| ClassMethod %OnGetJSONContent(pWhich As %String, pKey As %String, ByRef pParms, Output pObject As %RegisteredObject, pCriteria As %RegisteredObject, pLoad As %Boolean = 0) As %Status  {  #dim sql As %String = ""  #dim tsc As %Status = $$$OK  #dim result As %SQL.StatementResult    set username = $username    set pObject = ##class(%ZEN.proxyObject).%New()    if (pKey = "home") {  set pObject.sectionHeader = "Activity List for "\_username  set pObject.activityList = ##class(%ListOfObjects).%New()    set statement = ##class(%SQL.Statement).%New()  set sql = "select id,project->name as projectname,year,week from pm.activity where pm.activity.member->name = ? order by year,week"  set tsc = statement.%Prepare(sql)  if ($$$ISERR(tsc)) {  quit $$$OK  }    set result = statement.%Execute(username)    while result.%Next() {  set activity = ##class(%ZEN.proxyObject).%New()  set activity.id = result.%Get("id")  set activity.projectname = result.%Get("projectname")  set activity.yearweek = result.%Get("year")\_" "\_result.%Get("week")  do pObject.activityList.Insert(activity)  }    } elseif (pKey = "show-activity") {    set pObject.sectionHeader = "Activity Detail"    set statement = ##class(%SQL.Statement).%New()  // Get any additional details now  set sql = "select id,project->name as projectname,phase->name as phasename,year,week,hours from pm.activity where id = ?"  set tsc = statement.%Prepare(sql)  if ($$$ISERR(tsc)) {  quit $$$OK  }  set result = statement.%Execute(pCriteria.id)    while result.%Next() {  set pObject.id = result.%Get("id")  set pObject.projectname = result.%Get("projectname")  set pObject.phasename = result.%Get("phasename")  set pObject.year = result.%Get("year")  set pObject.week = result.%Get("week")  set pObject.hours = result.%Get("hours")  }    }    quit $$$OK  } |

set username = $username

$usernameはCaché ObjectScriptの特殊変数の１つで、ログインユーザー名が入ります。

|  |
| --- |
| set pObject.activityList = ##class(%ListOfObjects).%New()    set statement = ##class(%SQL.Statement).%New()  set sql = "select id,project->name as projectname,year,week from pm.activity where pm.activity.member->name = ? order by year,week"  set tsc = statement.%Prepare(sql)  if ($$$ISERR(tsc)) {  quit $$$OK  }    set result = statement.%Execute(username)    while result.%Next() {  set activity = ##class(%ZEN.proxyObject).%New()  set activity.id = result.%Get("id")  set activity.projectname = result.%Get("projectname")  set activity.yearweek = result.%Get("year")\_" "\_result.%Get("week")  do pObject.activityList.Insert(activity)  } |

ここではログインしたユーザーのアクティビティを年、週の順番にリストするクエリーを発行し、

その情報をリストとしてJSONオブジェクトとして生成します。

project->name

CachéのSQLでは標準SQLでサポートされていない暗黙結合というものをサポートしています。

これはオブジェクトアクセスの参照と同じ考え方で参照カラムにアクセスする方法です。

明示的に結合を行うことなくクエリーをコンパクトに見通しよく記述することができます。

# 開発に関連するその他の作業

以上で開発作業の基本に関する説明は終わります。

ここでは、開発に付随するいくつかの作業について解説します。

## 印刷

IT化が進んだとはいえ、紙で出力するというニーズはなかなか衰えません。

Cachéで印刷処理を実装する方法は、フォーム入力、表示と同様に様々な方法がありますが、ここではZen Reportを使ったレポートの作成について説明します。

ログインユーザーのアクティビティレポートを作成してみましょう。

### Zen Reportウィザード

スタジオ>新規作成(C)>カテゴリZen>Zen レポートをクリック

Zen Reportウィザード

パッケージ名: PM

クラス名: ActivityReport

次へをクリック

レポートのメインSQLを以下のエリアで入力することもできます。

SELECT YEAR,WEEK,MEMBER->NAME AS NAME,PROJECT->NAME AS PROJECTNAME, PHASE->NAME AS PHASENAME,HOURS FROM PM.ACTIVITY WHERE MEMBER->USERNAME = ? order by year,week

完了をクリック

ReportDefinitionの所で以下のような定義を行います。

|  |
| --- |
| <report xmlns="http://www.intersystems.com/zen/report/definition"  name="ActivityReport" sql="SELECT YEAR,WEEK,MEMBER->NAME AS NAME,PROJECT->NAME AS PROJECTNAME, PHASE->NAME AS PHASENAME,HOURS FROM PM.ACTIVITY WHERE MEMBER->USERNAME = ? order by year,week">  <parameter expression='$USERNAME'/>  <!-- add definition of the report here. -->  <group name="ActivityHeader" breakOnField="name">  <attribute name="name" field="name" />  <aggregate name="activitytotal" type="SUM" field="hours" />  <group name="ActivityLine">  <attribute name="year" field="year" />  <attribute name="week" field="week" />  <attribute name="projectname" field="projectname" />  <attribute name="phasename" field="phasename" />  <attribute name="hours" field="hours" />  </group></group></report> |

<parameter>は、SQLクエリーの実行に必要なパラメータを定義します。

このSQL定義では、ログインユーザー名を入力（プレースホルダ- ?を使用）として求めています。

$USERNAMEはログインしているユーザ名を保持する特殊システム変数です。

<group>でレポートの構造を定義します。

<group>はネストすることができます。

レポートの典型的な構造は、まずヘッダーがあって、あとに明細情報がくるというパターンですので、この例でもヘッダーと明細を定義しています。

クエリの結果を利用する場合は、<attribute>を使用します。

集計項目は<aggregate>で定義します。

この状態で一度クラスをコンパイルし、表示（V）>ブラウザで表示（b）をクリックして、

デフォルトのブラウザで表示してみましょう。

以下のような表示がされればOKです。

|  |
| --- |
| This XML file does not appear to have any style information associated with it. The document tree is shown below.  <ActivityReport>  <ActivityHeader name="UnknownUser">  <ActivityLine year="2014" week="40" projectname="追加プロジェクト　初期" phasename="要件定義" hours="10"/>  <ActivityLine year="2014" week="42" projectname="刷新プロジェクト　最終工期" phasename="結合テスト" hours="10"/>  <activitytotal>20</activitytotal>  </ActivityHeader>  </ActivityReport> |

次にHTMLまたはPDF形式で印刷可能なように表示レイアウトの定義を行います。

レイアウト定義もXData形式で定義します。

|  |
| --- |
| <report xmlns="http://www.intersystems.com/zen/report/display"  name="ActivityReport" >  <!-- Optional Init element inserts custom XSLT instructions at the top level of the generated XSLT stylesheet. -->  <init ></init>  <!-- Optional Document element specifies page layout and style characteristics. -->  <document width="8.5in" height="11in" marginLeft="1.25in" marginRight="1.25in" marginTop="1.0in" marginBottom="1.0in" ></document>  <!-- Optional Pageheader element. -->  <pageheader ></pageheader>  <!-- Optional Pagefooter element. Does not apply in HTML output. -->  <pagefooter ></pagefooter>  <!-- Required Body element. -->  <body>  <!-- add display definition of the report here. -->  <p class="banner1">Activity Report</p>  <group name="ActivityHeader" line="1px">  <table orient="row" width="5in">  <item field="@name" width="2in">  <caption value="Name:" width="2in"/>  </item>  <item field="activitytotal" width="2in">  <caption value="Activity Total:" width="2in"/>  </item>  </table>  <line pattern="empty"/>  <table orient="col" group="ActivityLine" altcolor="#FFDFDF" width="8in">  <item field="@year">  <caption value="Year:"/>  </item>  <item field="@week">  <caption value="Week:"/>  </item>  <item field="@projectname">  <caption value="Project Name:"/>  </item>  <item field="@phasename">  <caption value="Phase Name:"/>  </item>  <item field="@hours">  <caption value="Activity Hours:"/>  </item>  </table> </group></body>  </report> |

<item field="@name" width="2in">

attributeを指定するにはattribute名の先頭に@を付けます。

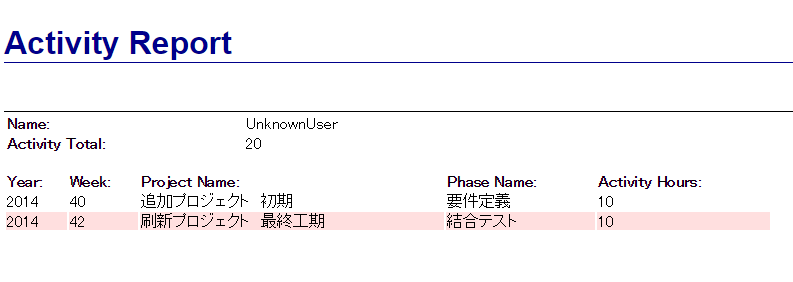
<item field="activitytotal" width="2in">

aggregateを指定するには、そのまま名前を使用します。

パラメータDEFAULTMODEをxmlからhtmlに変更します。

クラスをコンパイルし、表示（V）>ブラウザで表示（b）をクリックして、デフォルトのブラウザで表示してみましょう。

以下のように表示されればOKです。



## テスト

アジャイル開発の普及とともにTest Driven Developmentに関心が集まっています。

プログラムを作成する前にテストプログラムを作ってしまおうという発想です。

テストを行うに当たって、テスト対象となるプログラムは、テストが行いやすい形で作ることを

半ば強制されることになります。

このことは結果として見通しの良いプログラムの作成につながります。

非常に優れた方法論だと思いますので、是非このアプローチを取り入れた開発を推奨します。

CachéにもJavaや.NET環境に用意されているものと同等なテストフレームワークが用意されています。

### UnitTestクラスの作成

UnitTestを実装するクラスです。

ここでは、今回のプロジェクト管理とは関係ありませんが、日付の和暦に変換するプログラムのテストケースを作成していきます。

まずテスト対象となるJDateクラスを作成します。

#### インクルードファイルの作成

いくつかのリテラル値を使いますので、マクロ定義のためのインクルードファイルを作成します。

（スタジオ　新規作成> カテゴリ> 一般> テンプレート> Cachéインクルードファイル）

インクルードファイルの名前は、JDate.incとします。

|  |
| --- |
| #define MeijiStart 9862  #define TaisyouStart 26143  #define SyouwaStart 31404  #define HeiseiStart 54064  #define MeijiYear 1868  #define TaisyoYear 1912  #define SyouwaYear 1926  #define HeiseiYear 1989  #define StartYear 1  #define MeijiStartMonthDate 0101  #define MeijiEndYear 45  #define MeijiEndMonthDate 729  #define TaisyoStartMonthDate 730  #define TaisyoEndYear 15  #define TaisyoEndMonthDate 1224  #define SyouwaStartMonthDate 1225  #define SyouwaEndYear 64  #define SyouwaEndMonthDate 107  #define HeiseiStartMonthDate 108  #define MaxHeiseiYear 99  #define ShortExpression 1  #define LongExpression 2  #define ShortExpressionLength 7  #define LongExpressionLength 11  #define FormatError $System.Status.Error(4001,"Date Format Error")  #define RangeError $System.Status.Error(4002,"Date Range Error") |

ファイル(F)>名前を付けて保存を行います。

#### テスト対象クラスの作成

次にJDateクラスを定義します。

（スタジオ　新規作成> カテゴリ> 一般> テンプレート> Cachéクラス定義）

内容は以下のとおりです。

|  |
| --- |
| Include JDate  Class Sample.JDate Extends %Base  {  ClassMethod LogicalToDisplay(pDate As %Date, pFormat As %Integer = 1, pError As %Status) As %String  {  //明治　1868/1/01 - 1912/7/30 $h 9862 - 26143  //大正　1912/7/30 - 1926/12/25 $h 26143 - 31404  //昭和　1926/12/25 - 1989/1/7 $h 31404 - 54063  //平成　1989/1/8 - $h 54064  //  // 明治、大正、昭和に関しては改元日が重なっているが、その重なっている日を新元号とするケースが多いようである。  //  // Format  // 1 Gyymmdd Gは元号を表すアルファベット  // 2 GGyy年mm月dd日 GGは元号を表す漢字表現    Set pError = $$$OK    Set tDate = $Zdate(pDate,8)    If (+pFormat > $$$LongExpression) || (+pFormat < $$$ShortExpression) Set pFormat = $$$ShortExpression  If pDate < $$$MeijiStart Quit tDate //明治より前は西暦のまま返す    If (pDate > ($$$MeijiStart - 1)) && (pDate < $$$TaisyouStart) {  //明治  Set tYear = ..GengouYear(pDate, $$$MeijiStart)  If (pFormat = $$$ShortExpression) {  Set tDate = "M"\_tYear\_$Extract(tDate,5,8)  }  Elseif (pFormat = $$$LongExpression) {  Set tDate = "明治"\_tYear\_"年"\_$Extract(tDate,5,6)\_"月"\_$Extract(tDate,7,8)\_"日"  }  }    if (pDate > ($$$TaisyouStart - 1)) && (pDate < ($$$SyouwaStart)) {  //大正  set tYear = ..GengouYear(pDate, $$$TaisyouStart)  If (pFormat = $$$ShortExpression) {  Set tDate = "T"\_tYear\_$Extract(tDate,5,8)  }  Elseif (pFormat = $$$LongExpression) {  Set tDate = "大正"\_tYear\_"年"\_$Extract(tDate,5,6)\_"月"\_$Extract(tDate,7,8)\_"日"  }  }    if (pDate > ($$$SyouwaStart - 1)) && (pDate < $$$HeiseiStart) {  //昭和  set tYear = ..GengouYear(pDate, $$$SyouwaStart)  If (pFormat = $$$ShortExpression) {  Set tDate = "S"\_tYear\_$Extract(tDate,5,8)  }  Elseif (pFormat = $$$LongExpression) {  Set tDate = "昭和"\_tYear\_"年"\_$Extract(tDate,5,6)\_"月"\_$Extract(tDate,7,8)\_"日"  }  }  if (pDate >= $$$HeiseiStart) {  //平成  set tYear = ..GengouYear(pDate, $$$HeiseiStart)    If (tYear > $$$MaxHeiseiYear) {  Set pError = $$$RangeError  Set tDate = "" Quit tDate  }  If (pFormat = $$$ShortExpression) {  Set tDate = "H"\_tYear\_$Extract(tDate,5,8)  }  Elseif (pFormat = $$$LongExpression) {  Set tDate = "平成"\_tYear\_"年"\_$Extract(tDate,5,6)\_"月"\_$Extract(tDate,7,8)\_"日"  }  }  Quit tDate  } |

|  |
| --- |
| ClassMethod DisplayToLogical(pDate As %String, pFormat As %Integer = 1, pError As %Status) As %String  {  //明治　1868/1/01 - 1912/7/30 $h 9862 - 26143  //大正　1912/7/30 - 1926/12/25 $h 26143 - 31404  //昭和　1926/12/25 - 1989/1/7 $h 31404 - 54063  //平成　1989/1/8 - $h 54064  //  // 明治、大正、昭和に関しては改元日が重なっているが、その重なっている日を新元号とするケースが多いようである。  Try {  Set pError = $$$OK  Set tH = ""    If (+pFormat > $$$LongExpression) || (+pFormat < $$$ShortExpression) Set pFormat = $$$ShortExpression    If pDate?8N {  Set tYear = $Extract(pDate,1,4)  If tYear < $$$MeijiYear {  Set tH = $ZDH(pDate,8)  Quit  }  Else {  Set pError = $$$FormatError  Quit  }  }  If (pFormat = $$$ShortExpression) {  Set tEra = $Extract(pDate,1)  Set tCheck = $Case(tEra,"M":1,"T":1,"S":1,"H":1,:0)  }  ElseIf (pFormat = $$$LongExpression) {  Set tEra = $Extract(pDate,1,2)  Set tCheck = $Case(tEra,"明治":2,"大正":2,"昭和":2,"平成":2,:0)  }    If 'tCheck Set pError = $$$FormatError Quit  If (pFormat = $$$ShortExpression) {  set tYear = $Extract(pDate,2,3)  set tMonth = $Extract(pDate,4,5)  set tDay = $Extract(pDate,6,7)  If ($Length(pDate) '= $$$ShortExpressionLength) Set pError = $$$FormatError Quit  }  Else {  set tYear = $Extract(pDate,3,4)  set tMonth = $Extract(pDate,6,7)  set tDay = $Extract(pDate,9,10)  Set tNen = $Extract(pDate,5)  Set tGetsu = $Extract(pDate,8)  Set tNichi = $Extract(pDate,11)  If ($Length(pDate) '= $$$LongExpressionLength) Set pError = $$$FormatError Quit  If (tNen '= "年") Set pError = $$$FormatError Quit  If (tGetsu '= "月") Set pError = $$$FormatError Quit  If (tNichi '= "日") Set pError = $$$FormatError Quit  }    If tYear'?2N Set pError = $$$FormatError Quit  If tMonth'?2N Set pError = $$$FormatError Quit  If tDay'?2N Set pError = $$$FormatError Quit  If (+tYear = 0) Set pError = $$$FormatError Quit    If (tEra = "M") || (tEra = "明治") {  Set tYear2 = $$$MeijiYear + tYear - 1  Set tH = $ZDH(tYear2\_tMonth\_tDay,8)  If (+tYear = $$$StartYear) {  If (+(tMonth\_tDay) < $$$MeijiStartMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear = $$$MeijiEndYear) {  If (+(tMonth\_tDay) > $$$MeijiEndMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear > $$$MeijiEndYear) {  Set pError = $$$RangeError  Set tH = ""  }  }  If (tEra = "T") || (tEra = "大正") {  Set tYear2 = $$$TaisyoYear + tYear - 1  Set tH = $ZDH(tYear2\_tMonth\_tDay,8)  If (+tYear = $$$StartYear) {  If (+(tMonth\_tDay) < $$$TaisyoStartMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear = $$$TaisyoEndYear) {  If (+(tMonth\_tDay) > $$$TaisyoEndMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear > $$$TaisyoEndYear) {  Set pError = $$$RangeError  Set tH = ""  }  }  If (tEra = "S") || (tEra = "昭和") {  Set tYear2 = $$$SyouwaYear + tYear - 1  Set tH = $ZDH(tYear2\_tMonth\_tDay,8)  If (+tYear = $$$StartYear) {  If (+(tMonth\_tDay) < $$$SyouwaStartMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear = $$$SyouwaEndYear) {  If (+(tMonth\_tDay) > $$$SyouwaEndMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  ElseIf (+tYear > $$$SyouwaEndYear) {  Set pError = $$$RangeError  Set tH = ""  }  }  If (tEra = "H") || (tEra = "平成") {  Set tYear2 = $$$HeiseiYear + tYear - 1  Set tH = $ZDH(tYear2\_tMonth\_tDay,8)  If (+tYear = $$$StartYear) {  If (+(tMonth\_tDay) < $$$HeiseiStartMonthDate) {  Set pError = $$$RangeError  Set tH = ""  }  }  }  }  Catch tE {  Set pError = $$$RangeError  Set tH = ""  }  Quit tH  } |

|  |
| --- |
| ClassMethod GengouYear(pDate As %Date, pGengouStart As %Date) As %String [ Private ]  {  quit $Translate($Justify($extract($Zdate(pDate,8),1,4)-$Extract($Zdate(pGengouStart,8),1,4) + 1,2)," ",0)  }  } |

ファイル（F）>名前を付けて保存（Sample.JDate.cls）

#### テストケースの作成

%UnitTest.TestCaseクラスを継承したクラスをスタジオで作成します。

Sample.JDateUnitTests.clsというクラス名にします。

Testで始まるテストメソッドを作成します。

ここでは、$Hの日付を和暦に変換するプログラムをテストするメソッドとその逆をテストするメソッドの２つを用意します。

|  |
| --- |
| Method TestLogicalToDisplay()  {  Set tHDate = $ZDH(18671231,8)  Set tDate = 18671231  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking before Meiji 1")  Set tHDate = $ZDH(18680101,8)  Set tDate = "M010101"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Meiji Start 1")  Set tHDate = $ZDH(19120729,8)  Set tDate = "M450729"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Meiji End 1")  Set tHDate = $ZDH(19120730,8)  Set tDate = "T010730"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Taisyo Start 1")  Set tHDate = $ZDH(19261224,8)  Set tDate = "T151224"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Taisyo End 1")  Set tHDate = $ZDH(19261225,8)  Set tDate = "S011225"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Syouwa Start 1")  Set tHDate = $ZDH(19890107,8)  Set tDate = "S640107"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Syouwa End 1")  Set tHDate = $ZDH(19890108,8)  Set tDate = "H010108"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Heisei Start 1")  Set tHDate = $ZDH(20900108,8)  Set tDate = ""  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,1),tDate,"Checking Heisei Range Error")  Set tHDate = $ZDH(18671231,8)  Set tDate = 18671231  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking before Meiji 2")  Set tHDate = $ZDH(18680101,8)  Set tDate = "明治01年01月01日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Meiji Start 2")  Set tHDate = $ZDH(19120729,8)  Set tDate = "明治45年07月29日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Meiji End 2")  Set tHDate = $ZDH(19120730,8)  Set tDate = "大正01年07月30日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Taisyo Start 2")  Set tHDate = $ZDH(19261224,8)  Set tDate = "大正15年12月24日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Taisyo End 2")  Set tHDate = $ZDH(19261225,8)  Set tDate = "昭和01年12月25日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Syouwa Start 2")  Set tHDate = $ZDH(19890107,8)  Set tDate = "昭和64年01月07日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Syouwa End 2")  Set tHDate = $ZDH(19890108,8)  Set tDate = "平成01年01月08日"  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Heisei Start 2")  Set tHDate = $ZDH(20900108,8)  Set tDate = ""  Do $$$AssertEquals(##class(Sample.JDate).LogicalToDisplay(tHDate,2),tDate,"Checking Heisei Range Error")  } |

|  |
| --- |
| Method TestDisplayToLogical()  {  Set tHDate = ""  Set tDate = "18671331"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Before Meiji Format Error")  Set tHDate = $ZDH(18671231,8)  Set tDate = "18671231"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Before Meiji")  Set tHDate = $ZDH(18680101,8)  Set tDate = "M010101"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Meiji Start 1")  Set tHDate = $ZDH(19120729,8)  Set tDate = "M450729"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Meiji End 1")  Set tHDate = ""  Set tDate = "M450730"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Meiji End 1 NO GOOD")  Set tHDate = ""  Set tDate = "T010729"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Taisyo Start 1 NO GOOD")  Set tHDate = $ZDH(19120730,8)  Set tDate = "T010730"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Taisyo Start 1")  Set tHDate = $ZDH(19261224,8)  Set tDate = "T151224"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Taisyo End 1")  Set tHDate = ""  Set tDate = "T151225"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Taisyo End 1 NO GOOD")  Set tHDate = ""  Set tDate = "S011224"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Syouwa Start 1 NO GOOD")  Set tHDate = $ZDH(19261225,8)  Set tDate = "S011225"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Syouwa Start 1")  Set tHDate = $ZDH(19890107,8)  Set tDate = "S640107"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Syouwa End 1")  Set tHDate = ""  Set tDate = "S640108"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Syouwa End 1 NO GOOD")  Set tHDate = ""  Set tDate = "H010107"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Heisei Start 1 NO GOOD")  Set tHDate = $ZDH(19890108,8)  Set tDate = "H010108"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Heisei Start 1")  Set tHDate = ""  Set tDate = "19671231"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Meiji Start 2 NO GOOD")  Set tHDate = $ZDH(18680101,8)  Set tDate = "明治01年01月01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Meiji Start 2")  Set tHDate = $ZDH(19120729,8)  Set tDate = "明治45年07月29日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Meiji End 2")  Set tHDate = ""  Set tDate = "明治45年07月30日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Meiji End 1 NO GOOD")  Set tHDate = ""  Set tDate = "大正01年07月29日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Taisyo Start 1 NO GOOD")  Set tHDate = $ZDH(19120730,8)  Set tDate = "大正01年07月30日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Taisyo Start 2")  Set tHDate = $ZDH(19261224,8)  Set tDate = "大正15年12月24日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Taisyo End 2")  Set tHDate = ""  Set tDate = "大正15年12月25日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Taisyo End 2 NO GOOD")  Set tHDate = ""  Set tDate = "昭和01年12月24日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Syouwa Start 1 NO GOOD")  Set tHDate = $ZDH(19261225,8)  Set tDate = "昭和01年12月25日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Syouwa Start 2")  Set tHDate = $ZDH(19890107,8)  Set tDate = "昭和64年01月07日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Syouwa End 2")  Set tHDate = ""  Set tDate = "昭和64年01月08日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Syouwa End 2 NO GOOD")  Set tHDate = ""  Set tDate = "平成01年01月07日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Heisei Start 2 NO GOOD")  Set tHDate = $ZDH(19890108,8)  Set tDate = "平成01年01月08日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Heisei Start 2")  Set tHDate = ""  Set tDate = "平成01年00月00日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 1")  Set tHDate = ""  Set tDate = "平成00年01月01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 2")  Set tHDate = ""  Set tDate = "平静01年01月01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 3")  Set tHDate = ""  Set tDate = "平成01念01月01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 4")  Set tHDate = ""  Set tDate = "平成01年01日01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 5")  Set tHDate = ""  Set tDate = "平成01年01月01月"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 6")  Set tHDate = ""  Set tDate = "平成01年01月1日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 7")  Set tHDate = ""  Set tDate = "平成01年1月01日"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 8")  Set tHDate = ""  Set tDate = "Z011011"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Format Error 9")  Set tHDate = ""  Set tDate = "H11011"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Format Error 10")  Set tHDate = ""  Set tDate = "H11011"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Format Error 11")  Set tHDate = ""  Set tDate = "H1101011"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,1,.tError),tHDate,"Checking Format Error 12")  Set tHDate = ""  Set tDate = "平成11年01月01日1"  Do $$$AssertEquals(##class(Sample.JDate).DisplayToLogical(tDate,2,.tError),tHDate,"Checking Format Error 13")  } |

ファイル(F) > 名前をつけて保存をクリック

#### テスト環境セットセップ

まずテストスイートを置く場所を決めます。

例えば、c:\UnitTestsというディレクトリを作成します。

次にJDateというサブディレクトリーをその下に作ります。

Cachéターミナルを起動します。

Userネームスぺース上で以下のコマンドを実行します。

USER>Set ^UnitTestRoot = "c:\UnitTests"

スタジオ　ツール（T）>エクスポート(E)をクリック

追加ボタンをクリックして、Sample.JDateUnitTests.clsをリストに追加

ローカルファイルにエクスポートのテキストボックスにc:\UnitTests\JDateを入力

（参照ボタンをクリックしてファイルを選択することもできます。）

OKボタンをクリック

次に以下のメソッドを実行

USER>do ##class(%UnitTest.Manager).DebugLoadTestSuite("JDate")

#### テスト実行

続いて以下のメソッドを実行

do ##class(%UnitTest.Manager).DebugRunTestCase("JDate")

定義したテストが順番に実行されます。

以下のような実行ログが表示されます。

|  |
| --- |
| ===============================================================================  Directory: D:\UnitTests\JDate\  ===============================================================================  JDate begins ...  ディレクトリにあるアイテムのリスト作成を開始 on 10/22/2014 14:35:59 '\*.xml;\*.XML '      ファイル D:\UnitTests\JDate\jdatetest.xml を xml としてリストしています  リスト作成が正常に完了しました。    Sample.JDateUnitTests begins ...  TestDisplayToLogical() begins ...  AssertEquals:Checking Before Meiji Format Error (passed)  AssertEquals:Checking Before Meiji (passed)  AssertEquals:Checking Meiji Start 1 (passed)  AssertEquals:Checking Meiji End 1 (passed)  AssertEquals:Checking Meiji End 1 NO GOOD (passed)  AssertEquals:Checking Taisyo Start 1 NO GOOD (passed)  AssertEquals:Checking Taisyo Start 1 (passed)  AssertEquals:Checking Taisyo End 1 (passed)  AssertEquals:Checking Taisyo End 1 NO GOOD (passed)  AssertEquals:Checking Syouwa Start 1 NO GOOD (passed)  AssertEquals:Checking Syouwa Start 1 (passed)  AssertEquals:Checking Syouwa End 1 (passed)  AssertEquals:Checking Syouwa End 1 NO GOOD (passed)  AssertEquals:Checking Heisei Start 1 NO GOOD (passed)  AssertEquals:Checking Heisei Start 1 (passed)  AssertEquals:Checking Meiji Start 2 NO GOOD (passed)  AssertEquals:Checking Meiji Start 2 (passed)  AssertEquals:Checking Meiji End 2 (passed)  AssertEquals:Checking Meiji End 1 NO GOOD (passed)  AssertEquals:Checking Taisyo Start 1 NO GOOD (passed)  AssertEquals:Checking Taisyo Start 2 (passed)  AssertEquals:Checking Taisyo End 2 (passed)  AssertEquals:Checking Taisyo End 2 NO GOOD (passed)  AssertEquals:Checking Syouwa Start 1 NO GOOD (passed)  AssertEquals:Checking Syouwa Start 2 (passed)  AssertEquals:Checking Syouwa End 2 (passed)  AssertEquals:Checking Syouwa End 2 NO GOOD (passed)  AssertEquals:Checking Heisei Start 2 NO GOOD (passed)  AssertEquals:Checking Heisei Start 2 (passed)  AssertEquals:Checking Format Error 1 (passed)  AssertEquals:Checking Format Error 2 (passed)  AssertEquals:Checking Format Error 3 (passed)  AssertEquals:Checking Format Error 4 (passed)  AssertEquals:Checking Format Error 5 (passed)  AssertEquals:Checking Format Error 6 (passed)  AssertEquals:Checking Format Error 7 (passed)  AssertEquals:Checking Format Error 8 (passed)  AssertEquals:Checking Format Error 9 (passed)  AssertEquals:Checking Format Error 10 (passed)  AssertEquals:Checking Format Error 11 (passed)  AssertEquals:Checking Format Error 12 (passed)  AssertEquals:Checking Format Error 13 (passed)  LogMessage:Duration of execution: .006293 sec.  TestDisplayToLogical passed  TestLogicalToDisplay() begins ...  AssertEquals:Checking before Meiji 1 (passed)  AssertEquals:Checking Meiji Start 1 (passed)  AssertEquals:Checking Meiji End 1 (passed)  AssertEquals:Checking Taisyo Start 1 (passed)  AssertEquals:Checking Taisyo End 1 (passed)  AssertEquals:Checking Syouwa Start 1 (passed)  AssertEquals:Checking Syouwa End 1 (passed)  AssertEquals:Checking Heisei Start 1 (passed)  AssertEquals:Checking Heisei Range Error (passed)  AssertEquals:Checking before Meiji 2 (passed)  AssertEquals:Checking Meiji Start 2 (passed)  AssertEquals:Checking Meiji End 2 (passed)  AssertEquals:Checking Taisyo Start 2 (passed)  AssertEquals:Checking Taisyo End 2 (passed)  AssertEquals:Checking Syouwa Start 2 (passed)  AssertEquals:Checking Syouwa End 2 (passed)  AssertEquals:Checking Heisei Start 2 (passed)  AssertEquals:Checking Heisei Range Error (passed)  LogMessage:Duration of execution: .001116 sec.  TestLogicalToDisplay passed  Sample.JDateUnitTests passed  Skipping deleting classes  JDate passed    Use the following URL to view the result:  http://160.0.9.128:57772/csp/user/%25UnitTest.Portal.Indices.cls?Index=35 |

最後のurlを使ってブラウザで結果を確認することができます。

ブラウザで表示させるためにはセキュリティの設定が必要です。

（一回実行すればOK）

Cachéターミナルでログイン

以下のコマンドを実行します。

USER>ZN "%SYS"

%SYS>set ^SYS("Security","CSP","AllowPrefix","/csp/user/","%UnitTest.") = 1

## テストデータ生成フレームワーク

テストを行うに当たってテストデータが必要なケースが多々あります。

Cachéはテストデータを生成するためのフレームワークを用意しています。

それではテストデータ生成フレームワークを使ったデータ自動生成の方法について紹介します。

### SQLのINSERT文によるデータ生成

一般的なRDBMSと同様INSERT文を使ってデータを生成することができます。

例えばクラス定義のクラスメソッドとしてSQL INSERT文を使用したデータ生成処理を実装することができます。

比較的単純なデータの場合、この方法は簡便で実装しやすい方法です。

PM.Organizationクラスに以下のようなクラスメソッドを追加してみましょう。

|  |
| --- |
| ClassMethod Init()  {  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('インテグレーション事業部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('医療システム部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('医療システム１課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('医療システム２課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('社会システム部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('社会システム１課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('社会システム２課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('社会システム３課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('流通システム部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('流通システム１課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('金融システム部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('金融システム１課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('金融システム２課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('金融システム３課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('製造システム部'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('製造システム１課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('製造システム２課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('製造システム３課'))  &sql(insert into PM.ORGANIZATION (NAME) VALUES ('製造システム４課'))  } |

ここで&sqlは、埋め込みSQL文といってCaché ObjectScriptにSQL文を組み込む１つの方法です。

コンパイル実行後、ターミナル上で以下のコマンドを実行します。

USER>do ##class(PM.Organization).Init()

データが生成されているかどうかを確認する方法もいくつかありますが、管理ポータル上でSQLアクセスする方法が一番手っ取り早い方法です。

管理ポータル>システムエクスプローラ>SQL

左のペインからテーブル>PM.ORGANIZATIONをクリック

右のペインからテーブルを開く

### %Populateクラスを使ったデータ自動生成

大量のデータを生成する必要がある場合にその数に合わせてINSERT文で作成（生成）するのは大変です。

Cachéには乱数やデータの組み合わせで適当なデータを大量生成する仕組みが用意されています。

#### Populate ユーティリティクラス

データ自動生成を支援するPM.PopulateUtilsクラスを作成します。

適当な数のデータのリストを作り、$RANDUM関数によってそのリストから任意の値を取得することおよびそれを組み合わせることで適当にばらついたテストデータを生成することができます。

|  |
| --- |
| Class PM.PopulateUtils Extends %RegisteredObject [ ClassType = "", ProcedureBlock ]  {  /// Returns a random city name.  ClassMethod City() As %String  {  s t1=$lb("大阪市","札幌市","仙台市","大宮市","金沢市","横浜市","川崎市","福岡市","広島市","佐賀市","熊本市","松山市","鹿児島市","山口市","徳山市","岡山市","神戸市","京都市","福知山市","川西市","宝塚市","西宮市","池田市","豊中市","大阪市","奈良市")  Quit $li(t1,$r($ll(t1))+1)  }  /// Returns a random age.  ClassMethod Age() As %Integer  {  Set age=$R(99)  Quit age  }  /// Returns a random bill.  ClassMethod Bill() As %Integer  {  Set bill=$R(99)  Quit bill\*1000  }  /// Returns a random Wage.  ClassMethod Wage() As %Integer  {  Set bill=$R(50)  Set bill = bill \* 1000  If bill < 10000 Set bill = bill + 10000  Quit bill  }  /// Returns a practice name.  ClassMethod Practice() As %String  {  s t1=$lb("内科","外科","小児科","神経外科","皮膚科","眼科","脳外科")  Quit $li(t1,$r($ll(t1))+1)  }  /// Returns sex.  ClassMethod Sex() As %String  {  s t1=$lb("男","女")  Quit $li(t1,$r($ll(t1))+1)  }  /// Returns a random company name.  ClassMethod Company() As %String  {  Set c1=$LB("住井","三友","ＮＴＳ","丸田","タクト","電金","新光","出光","アオキ","東川","富士","ディジタル","コスモ","プライス","暗電","総芝","つばさ","ラックス","東経","セコミ","ビーエスシ","SES","IDGG","小文社","ミック","高地歩","ストラテス")  Set c2=$LB("商事","証券","銀行","損保","製造","機械","石油","情報","研究所","サービス","医療システム","ジャパン","システムズ","コミュニケーションズ","データ","ウェア","総業","工業","建設","技研","薬品")  Set c3=$LB("株式会社","有限会社","株式会社","株式会社","株式会社","株式会社","株式会社")  Quit $LI(c1,$Random($LL(c1))+1)\_$LI(c2,$random($LL(c2))+1)\_$LI(c3,$random($LL(c3))+1)  }  /// Returns a random currency value between <var>min</var>  /// and <var>max</var> (if present).  ClassMethod Currency(min As %Integer = 0, max As %Integer = 10000000) As %Integer  {  Quit ##class(%PopulateUtils).Float(min,max,4)  }  /// Returns a string containing a random first name.  /// <p><var>gender</var> is a optional string used to control the  /// gender of the generated name: 1 is Male, 2 = Female, "" is either.  ClassMethod FirstName(gender As %String = "") As %String  {  #; gender is 1:MALE,2:FEMALE  s:$g(gender)="" gender=$r(2)+1  If (gender = 1) {  Set list = $LB("博康","新太郎","俊哉","実","一二三","俊夫",  "幹夫","正行","渉","雅夫","誠一","博史",  "直弘","孝雄","茂","徹",  "雄三","道元","聡","弘明","敏明","信昭",  "良成","哲治","芳郎","俊介","操",  "英明","道夫","康之","仁孝","浩二郎","和彦",  "一成","道裕","亮","武","英之","勝一郎",  "哲郎","秀和","幸博","豊","道男","司","徹冶",  "高志","昭","明雄","義彦","清司","保之",  "徹也","勇","幸太郎","勝","信弘",  "達也","勝彦","亮一","敏哉","寛文","照美",  "克郎","貴英","正夫","崇","克道",  "誠一","正一","孝","公人","泰久")  }  Else {  Set list = $lb("茜","明子","晶子","あずみ","麻美",  "泉","いずみ","ひとえ","仁美",  "瞳","日登美","美穂","美保",  "香織","和美","一美","馨","エミリ","エミ","恵美",  "紀子","規子","由紀","雪","孝子","貴子","敏子",  "俊子","恵","恵美","愛","藍","三咲",  "美咲","みどり","みさえ","由紀子",  "由貴","裕香","かなえ","幸子","祥子",  "早苗","綾","彩","恵理子","エリカ",  "江美","博美","浩美",  "智子","友子","真紀","真樹","昌枝",  "正枝","静江","順子","淳子","雅子","恭子",  "京子","秀美","秀美",  "伊代","千恵美","智恵美","洋子","陽子","静香","京香","千春")  }  Quit $LI(list,$Random($LL(list))+1)  }  /// Returns a random floating point value between <var>min</var>  /// and <var>max</var> (if present).  ClassMethod Float(min As %Integer = 0, max As %Integer = 100000000, scale As %Integer = 0) As %Integer  {  s float=min+$Random(max-min+1)  q $s((float<max)&scale:+(float\_"."\_$Random(scalemax+1)),1:float)  }  /// Returns a random integer value between <var>min</var>  /// and <var>max</var> (if present).  ClassMethod Integer(min As %Integer = 0, max As %Integer = 10000) As %Integer  {  If min>max Quit 0  Quit min+$Random(max-min+1)  }  /// Returns a string containing a random last name.  ClassMethod LastName() As %String  {  Set x = $R(26)+1  If (x = 1) { Set list = $LB("伊藤","安部","梅田","石川","宇高") }  ElseIf (x = 2) { Set list = $LB("大崎","江原","安藤","榎本","荒川","大田","上村","大谷","石丸","大野","小島","長田") }  ElseIf (x = 3) { Set list = $LB("大林","石橋","石田","鬼塚","岩島","井口","小笠原","内野","大沢","岡本","上田","石村","小倉") }  ElseIf (x = 4) { Set list = $LB("有海","井村","梅沢","大島","井上","上田","奥山","大幡","宇津木") }  ElseIf (x = 5) { Set list = $LB("阿部","岩淵","荒川","大原","赤羽","新井","板谷") }  ElseIf (x = 6) { Set list = $LB("金子","川島","河野","金沢","川越","川下") }  ElseIf (x = 7) { Set list = $LB("甲斐","久保","小林","児玉","木内","亀谷","川原") }  ElseIf (x = 8) { Set list = $LB("柏木","小谷","北川","川西","加藤","吉川","鎌田") }  ElseIf (x = 9) { Set list = $LB("岸田","木村","川口") }  ElseIf (x = 10) { Set list = $LB("佐藤","嵯峨","新庄","鈴木","島","塩田") }  ElseIf (x = 11) { Set list = $LB("斉藤","笹原","正田","品川","杉山","砂川") }  ElseIf (x = 12) { Set list = $LB("桜井","清水","関口","白井","篠田","境","坂口","志村","芝戸","高橋","多久和") }  ElseIf (x = 13) { Set list = $LB("竹林","高松","田畑","高柳","田村","高藤","鷹野","田中","田畑","武田","高岡","滝藤","土井","中村","永尾") }  ElseIf (x = 14) { Set list = $LB("中沢","中村","長島","中本","中元","野田","西山","乗口","野口") }  ElseIf (x = 15) { Set list = $LB("西野","西本","西原","内藤","藤居","成井","波内") }  ElseIf (x = 16) { Set list = $LB("野村","野原","永井","長塚","中武","根本","林","広本","樋口","平田","尾藤","花木") }  ElseIf (x = 17) { Set list = $LB("廣田","本間","藤木") }  ElseIf (x = 18) { Set list = $LB("古川","古田","尾藤","福居","日高") }  ElseIf (x = 19) { Set list = $LB("本田","原田","平山","浜屋","橋本","平本","福嶋","長谷川","平島","吉野","廣瀬","細田") }  ElseIf (x = 20) { Set list = $LB("古館","早川","吉野","森本","松尾","松田","松本") }  ElseIf (x = 21) { Set list = $LB("宮崎","三沢","宮本","前川") }  ElseIf (x = 22) { Set list = $LB("三好","枡屋","武藤","森永") }  ElseIf (x = 23) { Set list = $LB("望月","丸山","森","溝上","三浦","丸谷","山本","山崎","吉村","安田") }  ElseIf (x = 24) { Set list = $LB("山中","柳井","横山") }  ElseIf (x = 25) { Set list = $LB("柳","山野","山原","山口") }  ElseIf (x = 26) { Set list = $LB("渡辺","渡部","渡邊") }  Quit $LI(list,$Random($LL(list))+1)  }  /// Returns a string containing a randomly generated corporate mission statement.  ClassMethod Mission() As %String  {  Set c1=$LB("リーダ ","ディベロッパ ","プロバイダ ","Resellers of ","On-line distributors of ")  Set c2=$LB("advanced ","InterNet ","cutting-edge ","breakthrough ","complex ","high-performance ","scalable ","cross-platform ","just-in-time ","open ","personal ","high-tech ","high-touch ","open-source ","virtual ","interactive ")  Set c3=$LB("quantum ","nano-","hyper-","optical ","financial ","multi-media ","object-oriented ","broad-band ","secure ","digital ","Java ","Enterprise ","Linux-based ","genetic ","wireless ","satellite-based ","ISO 9003-ready ","Y3K-certified ")  Set c4=$LB("devices and ","instrumentation ","graphical ","XML ","InterNet ","application development ","database ","data warehouse ","forecasting ","voice-enabled ","cold-fusion powered ")  Set c5=$LB("services ","technologies ","media ","content ","middle-ware ","connectivity ","consulting ","pharmaceuticals ")  Set c6=$LB("for the InterNet.","for the Financial community.","for discriminating investors.","for the Entertainment industry.","for the home.","for the Fortune 5.","for the Fortune 50.","for the Fortune 500.","for the Fortune 5000.","for the enterprise.","for the desktop.","for the Health Care community.")  Quit $LI(c1,$Random($LL(c1))+1)\_$LI(c2,$random($LL(c2))+1)\_$LI(c3,$random($LL(c3))+1)\_$LI(c4,$random($LL(c4))+1)\_$LI(c5,$random($LL(c5))+1)\_$LI(c6,$random($LL(c6))+1)  }  /// Returns a string containing a random name as <i>lastname,firstname</i>.  /// <p><var>gender</var> is a optional string used to control the  /// gender of the generated name: 1 is Male, 2 = Female, "" is either.  ClassMethod Name(gender As %String = "") As %String  {  Quit ..LastName()\_" "\_..FirstName($g(gender))  }  /// Returns a string value of length <var>len</var>  /// of a random character\_$r(9999).  ClassMethod String(len As %Integer = 1) As %String  {  s:'$g(len) len=1  Set slist=$LB("メトロゴールド","モダンアミュズメント","モンスター","ラブラ","ラブラドル","ランドリー","ルショップシピー","レッドウッド","ロイヤルフラッシュ","６６６","フィラシューズ","フィールドライン","４５ｒｐｍ","フォワード","フラミンゴサルン","ブレイクビーツ","ボイコット","ボーダメイド","ポールスミス","ミリオンエア","メイドインワールド")  s string=$List(slist,$R($LL(slist))+1)  QUIT string  }  /// Project Name  ClassMethod Project() As %String  {  Set slist1=$LB("新規","追加","改修","刷新")  Set slist2=$LB("初期","第一期","第二期","第三期","最終工期")  s string=$List(slist1,$R($LL(slist1))+1)\_"プロジェクト　"\_$List(slist2,$R($LL(slist2))+1)  QUIT string  }  /// Returns a random street address.  ClassMethod Street() As %String  {  s t1=$lb("Maple","Ash","Elm","Oak","Main","First","Second","Washington","Franklin","Clinton","Madison")  s t2=$lb("Street","Avenue","Blvd","Court","Place","Drive")  Quit ($r(9999)+1)\_" "\_$li(t1,$r($ll(t1))+1)\_" "\_$li(t2,$r($ll(t2))+1)  }  /// Returns a random Job Title.  ClassMethod Title() As %String  {  Set t1=$LB("","上級","副","アシスタント","戦略","国際","研究","エグゼクティブ")  Set t2=$LB("エンジニア","営業担当","サポートエンジニア","開発担当","マーケティングマネージャ","アカウント担当","リソースディレクタ","ディレクタ","製品マネージャ","リサーチアシスタント","システムエンジニア","テクニシャン","ウェブマスタ","管理者","製品スペシャリスト","会計士","衛生士")  Quit $LI(t1,$Random($LL(t1))+1)\_$LI(t2,$random($LL(t2))+1)  }  /// Returns a random JPN phone number.  ClassMethod JPNPhone() As %String [ CodeMode = expression ]  {  "0"\_($Random(999))\_"-"\_($Random(9999))\_"-"\_($Random(9999))  }  ClassMethod JPNZip() As %String [ CodeMode = expression ]  {  ($Random(899)+100)\_"-"\_($Random(8999)+1000)  }  } |

#### Populate用クラス定義の変更

PM.Memberの定義を以下の様に変更します。

|  |
| --- |
| /// プロジェクト構成員  Class PM.Member Extends PM.Person  {  /// 時間給  Property HourlyWages As %Integer(POPSPEC = "##class(PM.PopulateUtils).Wage()");  Relationship Activities As PM.Activity [ Cardinality = many, Inverse = Member ];  ClassMethod Init(pNum As %Integer)  {  Do ..Populate(pNum)  } |

POPSPECでどのメソッドを使ってデータを生成するかを定義します。

クラスメソッドInit()でデータの自動生成を行う処理を行います。

..という表記は、自分自身（クラス）がもっているメソッドという意味です。

%Populateクラスを継承すると、Populate()メソッドを呼ぶことができます。

Populate()メソッドを呼び出すと、POPSPEC等の定義に基づきデータを自動生成します。

新しい定義を追加後、コンパイルを実行します。

ターミナル上で以下のコマンドを実行します。

USER>Do ##class(PM.Member).Init(10)

管理ポータルで先ほどOrganizationでデータの確認を行った同じ方法でデータが生成されているか確認しましょう。

### %Populateクラスを使った少し複雑なデータ自動生成

データの関連性（依存データ等）を加味したデータ生成を行うための仕組みが自動データ生成のフレームワークに含まれています。

ここでは、住所のデータを日本郵便がウェブで公開している全国郵便番号データを使って自動生成してみましょう。

以下のurlの全国一括版をダウンロードします。

http://www.post.japanpost.jp/zipcode/dl/kogaki-zip.html

ダウンロードしたZIPファイルを解凍し、ken\_all.csvファイルを適当なディレクトリに置きます。

郵便データを格納するクラスPM.YubinDataクラスを以下のように定義します。

|  |
| --- |
| Class PM.YubinData Extends %Library.Persistent [ Not Abstract, DdlAllowed, Not LegacyInstanceContext, ProcedureBlock ]  {  Parameter ROWTYPE = "DantaiCode VARCHAR(10),OldZipCode VARCHAR(5), ZipCode VARCHAR(7), KenYomi VARCHAR(30), ToshiYomi VARCHAR(30), CyouYomi VARCHAR(30), Ken VARCHAR(30), Toshi VARCHAR(30), Cyou VARCHAR(30)";  Property DantaiCode As %Library.String(MAXLEN = 10) [ SqlColumnNumber = 2 ];  Property OldZipCode As %Library.String(MAXLEN = 5) [ SqlColumnNumber = 3 ];  Property ZipCode As %Library.String(MAXLEN = 7) [ SqlColumnNumber = 4 ];  Property KenYomi As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 5 ];  Property ToshiYomi As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 6 ];  Property CyouYomi As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 7 ];  Property Ken As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 8 ];  Property Toshi As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 9 ];  Property Cyou As %Library.String(MAXLEN = 30) [ SqlColumnNumber = 10 ];  ClassMethod Import(pSelectMode As %Library.Integer = {$zu(115,5)}, pFileName As %Library.String(MAXLEN=""), pDelimiter As %String = ",", pQuote As %String = """", pHeaders As %Integer = 0, ByRef pRecordCount As %Integer) As %Library.Integer [ SqlProc ]  {  set tStatementId = $SYSTEM.Util.CreateGUID(), tCounter = 0, pRecordCount = 0  set tPreparedStatement = ##class(%SQL.DynamicStatement).Prepare(tStatementId,..#ROWTYPE,pDelimiter,pQuote,,,0,"CSV")  if $Isobject(tPreparedStatement) {  set tImporter = tPreparedStatement.%New(tPreparedStatement,,pFileName,pDelimiter,pQuote)  if $Isobject(tImporter) {  do ..%DeleteExtent(,.tDeleted,.tInstances,1)  // burn the column headers  for tPtr = 1:1:pHeaders { do tImporter.%Next() }  while tImporter.%Next() {  set tMe = ..%New()  if 'pSelectMode {  set tMe.DantaiCode = tImporter.%GetData(1)  set tMe.OldZipCode = tImporter.%GetData(2)  set tMe.ZipCode = tImporter.%GetData(3)  set tMe.KenYomi = tImporter.%GetData(4)  set tMe.ToshiYomi = tImporter.%GetData(5)  set tMe.CyouYomi = tImporter.%GetData(6)  set tMe.Ken = tImporter.%GetData(7)  set tMe.Toshi = tImporter.%GetData(8)  set tMe.Cyou = tImporter.%GetData(9)  }  elseif pSelectMode = 1 {  set tMe.DantaiCode = $s('$system.CLS.IsMthd("DantaiCodeOdbcToLogical"):tImporter.%GetData(1),1:tMe.DantaiCodeOdbcToLogical(tImporter.%GetData(1)))  set tMe.OldZipCode = $s('$system.CLS.IsMthd("OldZipCodeOdbcToLogical"):tImporter.%GetData(2),1:tMe.OldZipCodeOdbcToLogical(tImporter.%GetData(2)))  set tMe.ZipCode = $s('$system.CLS.IsMthd("ZipCodeOdbcToLogical"):tImporter.%GetData(3),1:tMe.ZipCodeOdbcToLogical(tImporter.%GetData(3)))  set tMe.KenYomi = $s('$system.CLS.IsMthd("KenYomiOdbcToLogical"):tImporter.%GetData(4),1:tMe.KenYomiOdbcToLogical(tImporter.%GetData(4)))  set tMe.ToshiYomi = $s('$system.CLS.IsMthd("ToshiYomiOdbcToLogical"):tImporter.%GetData(5),1:tMe.ToshiYomiOdbcToLogical(tImporter.%GetData(5)))  set tMe.CyouYomi = $s('$system.CLS.IsMthd("CyouYomiOdbcToLogical"):tImporter.%GetData(6),1:tMe.CyouYomiOdbcToLogical(tImporter.%GetData(6)))  set tMe.Ken = $s('$system.CLS.IsMthd("KenOdbcToLogical"):tImporter.%GetData(7),1:tMe.KenOdbcToLogical(tImporter.%GetData(7)))  set tMe.Toshi = $s('$system.CLS.IsMthd("ToshiOdbcToLogical"):tImporter.%GetData(8),1:tMe.ToshiOdbcToLogical(tImporter.%GetData(8)))  set tMe.Cyou = $s('$system.CLS.IsMthd("CyouOdbcToLogical"):tImporter.%GetData(9),1:tMe.CyouOdbcToLogical(tImporter.%GetData(9)))  }  elseif pSelectMode = 2 {  set tMe.DantaiCode = $s('$system.CLS.IsMthd("DantaiCodeDisplayToLogical"):tImporter.%GetData(1),1:tMe.DantaiCodeDisplayToLogical(tImporter.%GetData(1)))  set tMe.OldZipCode = $s('$system.CLS.IsMthd("OldZipCodeDisplayToLogical"):tImporter.%GetData(2),1:tMe.OldZipCodeDisplayToLogical(tImporter.%GetData(2)))  set tMe.ZipCode = $s('$system.CLS.IsMthd("ZipCodeDisplayToLogical"):tImporter.%GetData(3),1:tMe.ZipCodeDisplayToLogical(tImporter.%GetData(3)))  set tMe.KenYomi = $s('$system.CLS.IsMthd("KenYomiDisplayToLogical"):tImporter.%GetData(4),1:tMe.KenYomiDisplayToLogical(tImporter.%GetData(4)))  set tMe.ToshiYomi = $s('$system.CLS.IsMthd("ToshiYomiDisplayToLogical"):tImporter.%GetData(5),1:tMe.ToshiYomiDisplayToLogical(tImporter.%GetData(5)))  set tMe.CyouYomi = $s('$system.CLS.IsMthd("CyouYomiDisplayToLogical"):tImporter.%GetData(6),1:tMe.CyouYomiDisplayToLogical(tImporter.%GetData(6)))  set tMe.Ken = $s('$system.CLS.IsMthd("KenDisplayToLogical"):tImporter.%GetData(7),1:tMe.KenDisplayToLogical(tImporter.%GetData(7)))  set tMe.Toshi = $s('$system.CLS.IsMthd("ToshiDisplayToLogical"):tImporter.%GetData(8),1:tMe.ToshiDisplayToLogical(tImporter.%GetData(8)))  set tMe.Cyou = $s('$system.CLS.IsMthd("CyouDisplayToLogical"):tImporter.%GetData(9),1:tMe.CyouDisplayToLogical(tImporter.%GetData(9)))  }  set tStatus = tMe.%Save()  if $$$ISOK(tStatus) { set tCounter = tCounter + 1 }  }  }  }  set %sqlcontext.%SQLCODE = 0  set %sqlcontext.%ROWCOUNT = tCounter  set pRecordCount = tCounter  quit tCounter  } |

コンパイルします。

以下のコマンドを実行します。

USER>do ##class(PM.YubinData).Import(,"c:\temp\ken\_all.csv")

管理ポータルで先ほどOrganizationでデータの確認を行った同じ方法でデータが生成されているか確認しましょう。

PM.Addressクラスに以下のメソッドを追加します。

|  |
| --- |
| Method OnPopulate() As %Status  {  Set id=$R($Get(^PM.YubinDataD))+1  Set yubin = ##class(PM.YubinData).%OpenId(id)  Set ..Zipcode=yubin.ZipCode  Set ..Prefecture=yubin.Ken  Set ..City=yubin.Toshi  Set ..Street=yubin.Cyou  QUIT $$$OK  } |

OnPopulate()メソッドはPopulate()メソッドが呼び出された時に自動的に呼び出されるメソッドで、ここにデータ生成のためのカスタムコードを記述することができます。

ここでは郵便番号を乱数で生成した後、その郵便番号に紐づく住所情報をPM.YubinDataクラスのインスタンスから取得しています。

PM.Addressクラスは埋め込みオブジェクトなので、自分自身でデータを生成することはできません

PM.Addressクラスをプロパティとして定義しているPM.Customerを使って住所データが意図通りに生成されるか確認します。

PM.Customerクラスを以下の様に変更します。

|  |
| --- |
| Class PM.Customer Extends (%Persistent, %Populate, %XML.Adaptor)  {  Property Name As %String(POPSPEC = "##class(PM.PopulateUtils).Company()");  Property Address As Address;  Relationship Projects As PM.Project [ Cardinality = many, Inverse = Customer ];  ClassMethod Init(pNum As %Integer)  {  Do ..Populate(pNum)  } |

PM.Customerクラスをコンパイルします。

次に以下のコマンドを実行します。

USER>do ##class(PM.Customer).Init(10)

管理ポータルでPM.Customerクラスのインスタンスが生成されているのを確認します。

## ソース世代管理

複数人で開発するプロジェクトの場合、プログラムの世代管理が重要になってきます。

Cachéにはソースを世代管理する機能は含まれませんが、スタジオには外部のソース世代

管理ツールと連携できるフレームワークが用意されています。

また、よく使われるソース管理ツール用のそのフレームワークを使用したテンプレートもあります。

詳しくは、カスタマーサポートセンターまでお問い合わせ下さい。

## コーディング規約

複数メンバーで開発する場合には、変数命名規約などのコーディング規約を設ける必要があります。

コーディング規約は、開発する組織毎に様々な要件、ニーズがあるためどの組織にも適用できる絶対的な規約は存在しません。

ここでは、実際に規約を決めていく上において参考となる情報の提供を行いたいと思います。

### ヘッダー情報

クラス定義、ルーチンなどの先頭にはそのクラスやルーチンの基本的な情報や目的等を記載した

説明文を載せましょう。

例：

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id

説明： $Horolog値を和暦に変換するメソッド

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### ソース管理ツール

複数人で開発するプロジェクトでは、何等かのソース世代管理ツール（SubVersion, Git等）

を導入し、適切なチェックアウト、チェックインプロセスにより最新版の管理を行うこと

を推奨します。

#### キーワード

コード項目の世代情報をコントロールするためにIdキーワードにはリビジョン番号

を含むことが望ましいです。

例:

Id: JDate.cls#5

　ルーチン

　　ルーチンの最後には、キーワードを出力する以下のようなコードを挿入します。

SrcVer quit "JDate.cls#5"

クラス

　　クラスには以下のようなクラスパラメータを定義します。

Parameter SrcVer = "JDate.cls#5";

### 見栄え

コードを整形する際には一般的な常識に基づき行動しましょう。

なるべくガタガタにならないように読みやすくなるようにしましょう。

#### サイズ

長いコードの項目はなるべく避けましょう。１つの項目が何ページもの長さになる場合には、

より小さく、管理可能な部分に分割しましょう。

全てのメソッドがエディターのウィンドウ内で一覧できれば、可読性は増しますので、コードメソッドは長くならないようにしましょう。

同様にコード行は80または100文字（日本語の場合には1文字2文字で換算）内に収まるようにして、水平スクロールを行うことなしに、全てのコード行を参照できるようにしましょう。

#### 整合性

行とテキスト、空行、コメントの位置が首尾一貫するようにスペースを配置をしましょう。

### 命名基準

ルーチン、クラス、変数などの目的を示唆する名前を使いましょう。

名前に複数の単語を使用することでその使用を示すことが容易になります。

しかしあまりに長い名前は可読性にいつも寄与するとは限りませんし、不必要にコードを

長くしてしまいます。

あくまでも常識の範囲でということを忘れないでください。

コード名と変数名は最初の31文字の範囲でユニークでなければなりません。

クラス名は定義の結果作成されるグローバルの長さの制限により31文字より長くすることは

できません。

クラス名に日本語を使用することは、結果として作成されるグローバル名に関して複雑な制約

があるため、推奨しません。

ルーチン名、変数名、メソッド名、プロパティ名なども日本語を使用することには様々な制約が

あるため推奨しません。

SQLでアクセスする際にテーブル名やカラム名に日本語を使用したい場合には、クラス名、プロパティ名に日本語を使用するのではなくSqlTableNameやSqlFieldNameに日本語の名前を定義する方法を推奨します。

#### 大文字小文字の使用

Cachéのシンタックスから変数とプロシジャを区別するのは容易なので、特定の大文字小文字の区別が名前のタイプを区別するためには必要ないと考えるかもしれませんが、可読性の観点では、変数とプロシジャブロックの名前にはパスカルケース（先頭大文字　例: PatientSurname, GetPatientName）を使うことを推奨します。　代わりに変数名にキャメルケース（先頭を大文字にしない　例: patientName, isNull）を使ってもいいです。

但し、既存のコードとの首尾一貫性は保持すべきです。

PatientFirstNameのように複数の単語の先頭を大文字にすることで可読性は大きく高まります。

#### ルーチン、クラス、CSPファイル名

名前はアプリケーション領域の適当なプレフィックスを含むほうが良いでしょう。

そしてそれに続く名前はそのプログラムの機能を示すようにします。

ルーチン名は、複数の単語を含んでもよいですがスペースは含むことができません。

大文字小文字を組み合わせても良いです。

しかし、ファイルの名前のユニーク性について大文字小文字の区別に頼るべきではありません。

オペレーティングシステムによっては大文字小文字の区別をしないものもあります。

クラスのパッケージ名は各レベルの先頭は大文字で実際のクラス名は複数単語を含んでもよいです。

#### 変数名

変数名にも上記の大文字小文字の区別の内容が適用されます。

十分理解できる範囲で説明可能であれば、短縮形を使ってもよいと思います。

例えば、 lengthの代わりにlen、ずっと使われてきたループのカウンターとして

i,j,kを使うなどの取決め事項など

scはインターシステムズのメソッドから%statusが返される時にステータスコード

として使用することを推奨します。

同様にエラーコード用にerrを使いましょう。

これらの変数はお互いに反対の値を持ちます。

sc=1は成功を意味するのに対してerr=1はエラーがあることを意味しています。

そのほかの提案としては、

一時的なローカル変数名の先頭にtをプレフィックスとして付ける（tVariableName）や

メソッドのパラメータにはpをプレフィックスとして使う（pVariableName）というのも

あります。

### コーディングの実際

ここでは、絶対的に守るべきルールというよりは、首尾一貫性を保つための提案を行い

ます。

#### 一般論

コードは可能な限りわかりやすく、内容を追跡しやすいようにしましょう。

すごく凝ったコードには感銘を受けるかもしれませんが、そのコードの保守が他人の手

に渡ったとたん、彼らにとってそれはありがたくなくなるでしょう。

一行に複雑な表現や複数コマンドを書くことは避けましょう。

一般的には表現を簡略化するためにコマンドを複数行に分割するほうがよいです。

* 空白

空白行を含むことによってテキストは読みやすくなることが多いです。

コードのブロックの間をあけるために空白行を使いましょう。

コード行内でも空白を追加することで項目間の可読性を改善できるかもしれません。

例： set x = (a \* b) \* - c

* インデント

制御構造とコードブロックの認識を支援するために字下げを使いましょう。

関連するコマンドは同じインデント上に並ぶようにすることを強く推奨します。

if、else、whileや中括弧で囲まれたブロックなどのコマンドに使えます。

* コマンド

コマンドと関数は省略形をつかわないようにしましょう。

但し先頭を大文字にするかどうかは自由です。

スタジオで"全て選択"<CtrlA)して<CtrlE>を打つと省略形の全てのコマンドは完全形

に変換します。

* 括弧

優先順位は常に左から右とは限りませんので、括弧を適切に使って計算の優先順位を

はっきり示すために論理部分の理解や妥当性を改善することができます。

例:if ((varA = x) && (varB = y)) || (varX = a) && (varY = b)) do ...

* New

引数なしNewコマンドは使わないようにしましょう。

特定の変数だけを明示的に指定するNewがそのプロセスの途中で使われるかもしれません。

システム変数に影響を与えることがないので一番良い方法です。

しかし必要なものが戻り値だけでその関数のロジックに他の影響を与えたくない場合

には、関数内で排他New（例： new (a,b,c)）を使っても良いです。

但し排他Newは性能的なペナルティが大きい点に注意が必要です。

* Kill

引数なしKillコマンドは使わないようにしましょう。

削除したい項目の名前を明示的に指定しましょう。

#### コメント

コードのセクションの目的を説明するためにコメントを使用しましょう。

特に複雑な処理を行うときにはなおさらです。

コードが複雑になればなるほど何を行おうとしているかを説明するために

より多くのコメントを含めるほうがよいでしょう。

しかし、あまりにたくさんのコメントがあって、全てのテキストの中にコード

が埋もれてしまうのは避けたほうが賢明です。

理解しにくいコードがある時、それが原因であることが多いです。

コメントは問題のコードの直近の前に置くのがよいでしょう

コードと同じ行にコメントは書かないほうが良いです。

左に調整することでコードとテキストを分けることができます。

書き始めをそろえて読みやすくするようにしましょう。

インデントされたコードには同じ量のインデントをコメントにも行い

関連コードと並ぶようにしましょう。

コメントは日本語（国際的な開発の場合は英語）で記述しましょう。

ルーチンとクラスには行コメントとして//を推奨します。

複数行にまたがるコメントには/\*　\*/を推奨します。

クラスリファレンスドキュメンテーションにクラスのコメントを含めるためには

///シンタックスを使いましょう。

クラス内の全ての要素（パラメータ、プロパティ、メソッドなど）にはそのような

コメントを含めることを強く推奨します。

CSPページはクラス形式（%CSPページを継承）とタグ主体の.cspファイルのいずれか

で記述可能です。

コメントのスタイルは選択したコーディングスタイルに適切なものを使いましょう。

つまりクラス形式の場合には//または/\* \*/、タグ主体の場合には、<!-- -->(XMLスタイル)

です。

JavaScriptブロックのコメントはクラス、ルーチンと同様です。

コードの大きな変更の際には、コードの変更のあった場所にコメントを追加することを推奨

します。

ソースコード管理システムで変更管理は可能ですが、顧客先でオンサイトサポートする際に

変更が記載されていると便利な時があります。

一般的にコード行を削除する際にはコメントアウトするのではなく削除しましょう。

ソースコード管理システムで削除した部分の復活は可能です。

#### パフォーマンス

昨今コーディングする際にパフォーマンスを気にすることは主たる関心事では

なくなってきています。

しかし、特に頻繁に呼ばれるコードや大量のデータを処理する時に使っている

テクニックに内在する問題を考慮する必要があります。

新しいオブジェクトスタイルコーディングが推奨されますが、それらがいつでも

古い方法より良いとは限らないことを肝に銘じる必要があります。

例えば、Cachéには古いCaché ObjectScriptコマンドに対応するたくさんのメソッド

が用意されています。

これらのメソッドは、一般的に単純なCaché ObjectScriptコマンドよりも何倍も遅い

です。

例：

現ネームスペースを取得するために

##class(%Library.Functions).NameSpace()または単純に$ZNを使う方法があります。

自分自身のジョブ番号を取得するには

##class(%Library.Function).ProcessID()または単純に$Jを使う方法があります。

Caché ObjectScriptのContainコマンド（[）は##class(Ens.Util.FunctionSet)Contains()

よりずっと高速です。（このメソッドは、Ensemble用です。）

$incrementに内在するロッキングにより他の方法より遅くなることがあります。

また$Pieceは非常に大きなオーバーヘッドがあることをずっと指摘されてきました。

いつも避けることができるとは限りませんが、レコードを構築する際には代わりに$List

コマンドを使ってリストを作ることができます。

最大の性能を得るために厳密なルール適用はできませんので、ここでもコーディングの際

には常識的に考えましょう。

時には可読性や今後のコード保守性の良さのために性能を犠牲にしなければならないかもしれません。

#### エラートラッピング

既存のコードに対してエラーを捕捉して処理する首尾一貫した方法はありません。

新しいオブジェクト指向のコードにはエラーを処理するためにTRY-CATCHメカニズムを使用しましょう。

オブジェクト指向のコードでエラーを処理するためにまさに適した方法です。

この方法を使い、TRYブロックと呼ばれる区切られたコードブロックを作ることができます。

そのコードの実行時にエラーが発生するとCATCHブロックに紐づけられたブロックに制御が

移ります。

ここに例外を処理するためのコードを含みます。

取り組んでいるコードが古いスタイルのコードで、エラーハンドリングをふくんでいなければ、

最初に以下のコード行を追加しましょう。

set $ZT = "^%ETN"

デバッギングの際に役に立つ追加情報がほしい場合には、

set $ZE = "SomeError"

do BACK^%ETN

をコードに追加しましょう。

エラーハンドラーが全ての変数とスタックの情報を集めてくれます。

BACK^%ETNはそのエラートラップを起動した後、それが呼ばれた所に戻って

きます。

単に^%ETNを呼び出した場合、プロセスを停止します。

BACK^%ETNを呼び出すためには$ZEを設定する必要があります。

そうしないと、そこでquitします。

#### プロシジャブロック

プロシジャは、名前があり、中括弧の中にコードブロックがあります。

サブルーチンや関数と同じように振る舞います。

そしてその名前を参照することで呼び出されます。

プロシジャブロックは暗黙的なquitを含んでいますが、常に閉じる中括弧の前

にquitを含めるようにしましょう。

中括弧で囲まれたコードブロックは、If、For、Whileコマンドと一緒に使うこともできます。

これらのブロックに名前はありませんが、Ifなどの後に実行されるコマンドをグループ化します。

たとえ一行のコードしか実行しなくても、中括弧で囲むことでコードを分離することができます。

プロシジャの引数は、自動的にプロシジャ内でローカルスコープとなります。

なのでNewコマンドは必要ありません。

しかし、使用する全ての変数は、使用前に初期化することを推奨します。

古いコーディングスタイルと新しいコーディングスタイルを混ぜないでください。

中括弧内では行タグは使用しないでください。

終了の中括弧は、開始の中括弧を含む行と垂直に並ぶようにしましょう。

if (condition) {

do TrueCode

} Else {

do FalseCode

}

#### プロシジャブロック内でQuitを使う

プロシジャブロック内でのQuitの挙動には注意して下さい。

Quitコマンドは単純に内部プロセスを終了し、外のブロックに制御を戻します。

次の値を処理するために中のループに制御を戻すのではありません。

次の値を処理することを続行したい場合には、Continueコマンドを使用しなければ

なりません。

#### ストアドプロシジャ

ストアドプロシジャに取り組む際には、開発の首尾一貫性を保つために開発グループ

内で取り決めたルールに従う必要があります。

例えば^CacheTempを使う際の命名規則やデータ構造など

#### クラスメソッド

メソッドには期待する入力と出力の説明を含めましょう。

メソッドを使用する際には、１つのエントリーと１つのイグジットポイントを持つようにしましょう。

しかし、データ妥当性のコードをそのメソッドの先頭に含めて何かエラーがあったらすぐにそこでquitしても良いです。

これ以外にもメソッドの途中で戻ることを避け、常にメソッドの最後で終了するようにしましょう。

必要ならば一時的な変数を作成して結果を保持しましょう。

全てのメソッドはせめてステータスだけでも値を返すようにしましょう。

適切ならば、他の値はoutputパラメータとして呼出しコードから参照渡しするようにしましょう。

例えば、%Statusの値は、$$$OKなどのマクロ参照を通して返しましょう。

#### クラスプロパティ

プロパティには既定値を含む説明を含めましょう

コメントに///シンタックスを使って詳細情報をクラスリファレンスドキュメントに含めることが

できます。

プロパティをクラスから削除すると、そのストレージ位置は保持されたまま、そのフィールドは利用できないと解釈します。

またストレージの位置情報も削除することもできます。

但し、そこに何もデータが格納されていないことを確認する必要があります。

そうでない場合には、そのストレージ位置は再利用されて、そこにごみデータが入ることになります。

一般的にはストレージノードはそのままにしておくほうが安全です。

しかし基本環境のコーディングのプロセスの一部として作成された使われていないノードは削除しましょう。

#### パラメータ

コードを書く時にディレクトリやサーバー名や良く使われるデータで、環境によって異なるシステム値をハードコードしないようにしましょう。

その代わりに現時点の値を保持するパラメータグローバルを使用して、プログラムからはそれを参照するようにしましょう。

こうすることで環境毎の柔軟性、より簡単な保守性に繋がり、システム環境の変化に対応できます。

例えば、グローバル^Configをそのようなパラメータデータとして使います。

^Config("WSLOcation",<targetSystem>)がウェブサービスのターゲットシステム名を保持する

という感じで使うことができます。

#### 一時グローバル

一時的なデータには^z\*ではなくて^CacheTempを使うようにしましょう。

最初の添え字には一般的にウェブ用コードでない場合には$Job、ウェブ用の場合には

%Session.SessionIdを使います。

こうすることでユニークなインデックスを使用し、複数のプロセスや複数のセッションが同じストレージを使用してしまうリスクを避けます。

使用開始前にデータが存在していないことを確かめ、終了時点でも使用しているノードレベル

で一時グローバルは削除しましょう。

#### 一時変数

プロセスプライベート変数は、それを作成したプロセスだけがアクセス可能な変数です。

全てのネームスペースからアクセス可能なようにマップされます。

プロセスが終了すると自動的に削除されます。

プロセスプライベートグローバルは、大きなデータに使用でき、そのため^CacheTempの代わり

に使うことができます。

この変数のシンタックスは様々な形式がありますが、最も良く使われるのが^||nameという形式

です。

新しいコードはこの形式を使うようにしましょう。

#### デバッグ

開発の途中でデバッグ用コードを含めた場合は、コードをチェックインするまえに削除した

ことを確かめましょう。

特に一時グローバルの作成をコードの中に残さないようにしましょう。

時間が立つにつれそれがデバッグ目的だったかどうかわからなくなります。

その結果、削除しなくなりずっとデータを収集しつづけることになります。

### 避けるべきこと

以下に開発者がコードを書く際に避けるべきコマンドやテクニックを紹介します。

#### コマンド

* GoTo

GoToはコードのフローを乱し、デバッグを難しくします。

* ..（ドット）記法

可読性のためにドット形式ではなくプロシジャブロックを使いましょう。

ドット記法は、正しくない他のコードやコマンド、コメントの挿入が

予想外のブレークを引き起こしたりするので推奨しません。

#### XECUTEと間接実行

XECUTEと間接実行は両方ともコードを追跡するのが難しいケースがあります。

可能な限り使用しないようにしましょう。

どうしても必要な場合には、明確にコードの機能を説明するコメントを含めるようにしましょう。

プロシジャブロック内の名前による間接実行、引数間接実行、XECUTEいずれもプロシジャ内の

スコープでは実行されない点注意してください。（プロシジャブロックの変数を使うのではない）

またこれらはシステム負荷が高く実行も遅い点にも注意が必要です。

#### ネイキッド参照

グローバル名を参照する際にはいつも完全名を指定しましょう。

以下のようなコマンドは実行しないでください。

set ^TEST(1)=1,^(2)=2

可読性を損ないますし、常に何かを変更するとコードが動かなくなるリスクがあります。

## エラー処理

コーディング規約の章でも述べた通り、エラー処理の実装にはTRY-CATCHメカニズムを

使用することを推奨します。

そして全体として統一されたエラー処理となるよう、全てのエラー処理で共通に行われる

ことを取決め、必ずそれを実行するように実装することを推奨します。

今回のサンプルアプリケーションでは、発生したエラーをエラーデータベースとして

記録する処理を共通処理としています。

|  |
| --- |
| Try {  Do ..%DeleteExtent()  For i = 1:1:pNM {  Set Manager = ..%New()  Set Manager.Name = ##class(PM.PopulateUtils).Name()  Set Manager.MonthlyManagementFee = ($Random(100) \* 10000) + 500000  set Manager.Username = ##class(%PopulateUtils).String()  Set tSC = Manager.%Save()  If $$$ISERR(tSC) $$$ThrowStatus(tSC)  }  }  Catch tE {  Set tSC2 = ##class(PM.Error).StoreErrorInformation(tE)  }  Quit tSC |

PM.Errorクラスのクラス定義です。

|  |
| --- |
| Class PM.Error Extends (%Persistent, %XML.Adaptor)  {  /// アプリケーションエラーを記録するクラス  Property EventDateTime As %TimeStamp;  Property ErrorDescription As %String(MAXLEN = 1000);  ClassMethod StoreErrorInformation(pException As %Exception.General) As %Status  {  Set tSC = $$$OK  Try {  set tError = ..%New()  set tError.EventDateTime = $zdatetime($zts,3)  set tStatus = pException.AsStatus()  set tSC = $System.Status.DecomposeStatus(tStatus,.tErrorContent)  set n = ""  Do {  set n= $order(tErrorContent(n))  if n = "" quit  set tErrorContent = $get(tErrorContent)\_$get(tErrorContent(n))  } while n='""  set tError.ErrorDescription = $Get(tErrorContent)  set tSC = tError.%Save()  }  Catch tE {  Set ^FAQError(tError.EventDateTime)= $Get(tStatus)  }  quit tSC  } |

## データの関連を処理する

データの関連を処理する方法としてリレーショナルデータベースと同様、外部キーでも表現できますが、オブジェクト参照やリレーションシップのメソッドを利用して適切な関連を構築することができます。

## その他

### 計算フィールド

Cachéでは物理的な値を持つのではなく、何等かの計算により導きだせる項目を定義することができます。

このような項目を計算フィールドと呼びます

例えば、人間の属性として年齢という項目が考えられますが、年齢を物理項目とした場合の１つの問題というか手間は、誕生日が来るたびに更新する必要がある点です。

個々人の誕生日に合わせて更新処理を設計、実装する必要があります。

しかし、年齢は現在の日付と誕生日から一意に計算できます。

該当プロパティを計算フィールドとして宣言し、実行時にどんな処理を実装するかを定義（メソッド）することで動的に値を取得するようにします。

こうすることで物理的更新を行わなくてすみます。

もちろん、計算フィールドはクエリーの実行時に動的に呼ばれるという構造上、パフォーマンスという観点からは注意深く適用を検討する必要がありますが、実用性とパフォーマンスへの影響を勘案しながら適用することでシステム実装の柔軟性を増すことができます。

例えば一般的にデータを加工する処理としてETL(Extract Trasnform Loading)と呼ばれる単純な変換を繰り返し実行する方法が広く使われていますが、計算フィールドを適切に使うことにより、ETLの必要性をかなり削減できるのではないかと思います。

ETLは情報システムの保守性を複雑化させる１つの大きな要因となっていると言われています。

### クエリー実行

CachéObjectScript言語を使ってクエリーを実行する方法が2種類用意されています。

* 静的実行

埋め込みSQL文を使って、クエリを実行できます。

クエリーが変化しない場合にはこの方法が簡便でしかもパフォーマンス上も優れています。

* 動的実行

%SQL.Statementオブジェクトを利用したクエリーの実行です。

実行時にクエリーコマンドの内容を変更できるなど柔軟な対応が可能です。

性能面では埋め込みSQL文に劣ります。

### ファイルI/O

シーケンシャルファイルを読み込んだり、書き込んだりする処理は頻繁に利用されます。

ファイルI/Oの方法はいくつかありますが、％Streamクラスを使う方法をサンプルとして用意しています。

PM.ActivityクラスのExportXMLメソッドの中で%StreamFileCharacterクラスを使用したファイルの書き込み処理を実装しています。

# Acrtive Analytics

CachéはDWHのような分析用の別のデータベースを作ることなく、いまある

オンライン上のデータを利用して分析を行える仕組み（DeepSee）を用意しています。

## モデル作成

プロジェクトをデータソースとしたモデル（キューブ）を作ってみましょう。

DeepSeeのアーキテクトを使ってキューブの定義を行います。

### 前準備

まずUSERネームスペースでDeepSeeを利用するためにターミナルでログイン後、以下のコマンドを実行します。

USER>zn "%SYS"

%SYS>Do EnableDeepSee^%SYS.cspServer("/csp/user/")

### キューブの作成

アーキテクトのスクリーン上の新規ボタンを押します。

新しい定義を作成というタイトルのダイアログボックスが表示されます。

以下を入力します。

キューブ名　ProjectCube

ソースクラス　参照ボタンを押して、PM.Projectを選択します。

キューブのクラス名　PM.ProjectCube

### メジャーの定義

左ペイン上にあるソースクラスのActualAmont, ActualManHours, AnticipatedManHours,OrderAmountをそれぞれ中ペインのメジャーにドラッグ&ドロップします。

### ディメンジョンの定義

左ペイン上にあるソースクラスのCustomerの左にある黒い▼をクリックします。

Nameを中ペインのディメンジョンにドラッグ&ドロップします。

作成した中ペインのNameディメンジョンをクリックします。

右ペインに表示される名前をNameからCustomerNameに変更します。

同様にNameをディメンジョンにドラッグ&ドロップして名前をProectNameに変更します。

ProjectManagerのNameも同様にディメンジョンにドラッグ&ドロップして名前をPMNameに変更します。

### 詳細リストの定義

中ペインの詳細リストをクリックして選択します。

中ペインの要素を追加というラベルをクリックします。

キューブに要素を追加というタイトルのダイアログボックスが表示されます。

新しい要素名を入力　デフォルトの名前New\_listing1を入力します。

詳細リストのラジオボタンがチェックされていることを確認します。

OKボタンを押します。

### 計算メンバーの定義

プロジェクトのProfitというメンバーを追加しましょう。

中ペインの計算メンバーをクリックして選択します。

要素を追加ボタンをクリックします。

キューブに要素を追加というタイトルのダイアログボックスが表示されます。

新しい要素名を入力　Profit

計算メンバー（メジャー）というラジオボタンが選択されていることを確認します。

OKボタンを押します。

メジャーにProfitが表示されるのでそれをクリックします。

右ペインの表現の所に以下を入力します。

%source.OrderAmount - %source.ActualAmount

保存ボタンを押します。

コンパイルボタンを押します。

コンパイル結果にエラーがないことを確認します。

構築ボタンを押します。

## データ探索

アーキテクトで作成したデータモデルを使って、データを探索するためにはDeepSeeのアナライザを使用します。

管理ポータル>DeepSee>アナライザをクリックします。

箱の形をしたアイコンをクリックします。

ファインダダイアログというタイトルのダイアログボックスが表示されます。

表示されているProjectCubeをクリックします。

左ペインのメジャーのOrderAmountをクリックして、ドラッグ&ドロップで右ペインのメジャーの所にドロップします。

左ペインのディメンジョンのProectNameをクリックしてドラッグ&ドロップで右ペインの行の所にドロップします。

受注金額トップ5だけを表示したい場合、以下の操作を行います。

行の所の真ん中のボタン（カーソルを持っていくとテーブル内の行オプションを設定と表示される）を押します。

軸のオプションというタイトルのダイアログボックスが表示されます。

メンバーでソートのチェックボックスをチェックして、その下のリストボックスからOrderAmountを選択します。

最初のｎメンバーを返すもチェックして、カウントのテキストボックスに5を入力します。

OKボタンを押します。

後で今定義したクエリーを再実行可能なようにクエリーに名前を付けて保存することが可能です。

名前を付けて保存という名前のボタンを押します。

ピボットを保存というタイトルのダイアログボックスが表示されます。

フォルダー名にPMを入力します。

ピボット名にTopSales5と入力します。

OKボタンを押します。

## ダッシュボード作成

保存したピボットテーブルを使ってダッシュボードを作成してみましょう。

管理ポータル>DeepSee>ユーザーポータルをクリックします。

メニューから新規ダッシュボードを選びます。

ダッシュボードを作成というタイトルのダイアログボックスが表示されます。

フォルダーにPMを選択します。

ダッシュボード名にBigProjectTop5を入力します。

タイトルに案件トップ５を入力します。

OKボタンを押します。

メニューから新規ウィジェットを追加を選択します。

ウィジェット・ウィザードというタイトルのダイアログボックスが表示されます。

ピボットとグラフをクリックして円グラフをクリックします。

データソースの所で検索ボタン（虫めがねのアイコン）をクリックしてPM>TopSales5を選択します。

OKボタンを押します。

円グラフが表示されるのを確認します。

# パフォーマンス、スケーラビリティ

機能上問題ないソフトウェアでも運用の段階では、想定した性能（応答時間、スループット）

が出ないということが往々にして起こり得ります。

一般的には性能問題は、システム開発と別物ととらえられているケースが多いですが、実際

には開発の段階で性能をコントロールすべきです。

性能問題を取り扱う時の格言の１つに

測定できないものを管理することはできない。

というのがあります。

つまり開発の段階で色々な指標を測定できる仕組みを組み込んでおかなければ、実際に性能問題が起こった時に対処が難しいということになります。

## 性能を管理するために行うべきこと

アプリケーション性能を反映する数量化できる指標を導入します。

それらの指標を捕えて、分析する方法を学びます。

開発サイクルの中でアプリケーションの性能をプロアクティブに管理していきます。

### アプリケーション指標

アプリケーション性能をコントロールするための最初のステップです。

アプリケーションの仕事量、速度を反映します。

* アカウントスクリーンを開く時間
* レポートを作成する時間
* 1分/1秒当たりのトランザクション数

時間を主とした指標が十分でない理由

活動時間が短すぎる（ミリ秒）

　ストップウォッチで計測するのが現実的でない

　コードの中に$ZHを入れることで収集できる

メモリー（キャッシュ）の状況で変化する（再現不能）

　グローバルバッファーが吐き出さすことで軽減されるのでは？

　グローバルを吐き出すのは非現実の状況を作り出す

同じサーバーの他プロセスの影響

　隔離した状態でテストを実行

CPUの問題なのかI/Oの問題なのか切り分けが難しい

　他の指標が必要

### システムレベル指標

CPU使用率

メモリー使用量

ディスク・レーテンシー、キューイング

Cachéレベル指標

ルーチンコマンド(CPU)

メモリー（グローバル参照）

non DB I/O, File, Network

DB I/O(CACHE.DAT CACHE.WIJ)

GLOREF

グローバルに対する１つのアクセス（get, set, kill）

大まかに言うとI/Oに変換される（DBファイルとメモリー）

ルーチンコマンド

　１つのCOS命令

　大まかにいうとCPUロードを反映

実行時間

　性能の全体的な計測

　外部要因により大きく変化する

管理するということは

　データを捕えて

　分析し

　何が起こっているかを理解し

　改善する

ということ

重要な指標を捕えるためのツール

GLOSTAT システム全体

PERFMON ルーチン毎、グローバル毎分析

%SYS.MONLBL コマンド行毎の分析

%SYS.PTools SQLレベルの統計情報収集

コードの中で指標を捕える

全ての操作のためにUnitTestで以下のことを記録

時間

　$ZH

グローバル参照、ルーチンコマンド

　%SYS.ProcessQuery

さらに詳細な指標には%Monitor.Processを使用

SQLアプリケーション

　%SYS.Ptoolsがテーブルにデータを維持する

DeepSeeを使って分析

全てのウェブサービス呼び出しの際に以下を記録

　メソッド名

　開始時間

　終了時間

　ユーザー

　例外

DeepSeeで最も遅いメソッドを探すなど

全てのウェブサービス呼出しの際に以下を記録

メソッド名

$ZH

グローバル参照

ルーチンコマンド

ビルド毎にUnitTestを実行

　毎回テストに同じデータを取得し、デグレードがないかをチェック

　問題あれば性能問題として開発部門に差し戻し

役に立つユーティリティ

DeepSee + ログテーブル

＄ZHorolog

%SYS.ProcessQuery

%Monitor.Process

%SYS.PTools

インデックスアナライザー

コードとUnitTestにそれらのツールの呼出しを追加する

DeepSeeで分析

改善

負の結果が出た場合にはそこで対処する

開発サイクルの一部として性能を管理していく

データベースサイズを2倍にしてテスト

現実的なデータ量にする必要はない

　100を200にするので十分

UnitTestを実行しグローバル参照、ルーチンコマンドを見張る

　指標が2倍になってる場合には警告

　（テーブル全検索か適切なインデックスがない）

　指数級数的増加　赤信号　アルゴリズムの見直しが必要

大規模データベースをエミュレーションする

テストDBは、超大規模サイズの断片

　100～200MBでGBをエミュレートする

DBサイズの断片に基づいてグローバルバッファーを減らす

100MBデータベース用には5MBなど）

サンプル実装やUnitTestの相対的な性能劣化を見張る

　テストや操作が重大な減少を示すものは大抵ディスク読み込み

　を行っている。

　大きなデータベースでも遅くなる

グローバルバッファーのリセット

　テスト毎にメモリーの状態を一定に保つ

　データベースをディスマウントすると全てのバッファーをクリアにする

アップグレード

　様々な改善

　高速コンパイラ

　新しい高速なローカル変数

## よくあるパフォーマンス問題

### データロード時間

データロード時間で問題となる典型的なものは、テスト的に例えば1万件のデータを投入した時

の所要時間をもとに本番データのロード時間を単純なデータ容量の比率で見積もってしまうケースです。

テストデータ件数が例えば10万件で、その処理時間1分と仮定します。

本番データは100万件なので、データ件数が100倍として、100分だろうと見積もったとします。

しかし、通常はこのように予想通りにいくとは限りません。

データの初期投入の時点ではデータベースキャッシュにまだ余裕があるため、ロード処理がメモリー上で完結します。

しかし、データベースのサイズがデータベースキャッシュサイズを超えてくるあたりからキャッシュ上のデータを一部退避しなければ追加データの処理を継続できない可能性が高まります。一部データを退避するためにディスクI/Oが発生するために処理遅延が発生し始めます。

遅延の度合いは処理パターンおよびキャッシュ量とデータベースのサイズ等いくつかの因子に影響されます。

特にインデックスデータ生成のようなランダムデータを主体とした書き込み処理が多い場合にこの影響が顕著です。

### インデックス構築

このような状況が発生した際の対処法として、データのロード時にはインデックスを生成せずにあとでまとめてインデックスを生成する手法があります。

Cachéにはインデックス生成のようなデータベース書き込みの負荷が高い処理のI/O負荷を軽減する$SORTBEGINと$SORTENDという機能があります。

これはデータベースにランダム書き込みを行う前にできるだけメモリー上でデータを並べ替えてデータベースへの書き込みを連続化するテクニックです。

### ジャーナル、トランザクション

Cachéは、一般的なトランサクションをサポートしたDBMSと同様に、データベースの更新とともにジャーナルを書き込むことでトランザクションのACID属性を担保しています。

しかしデータの初期投入などの比較的データの復旧が容易な状況では、トランサクションおよびジャーナルの書き込みを無効にすることでデータロード処理時間の改善が可能です。

特にビットマップインデックス、ビットスライスインデックスは、ジャーナルI/O負荷が高いため、これらのインデックスを多用している場合には、この方法を検討することをお勧めします。

### クエリーパフォーマンス

クエリパフォーマンスを改善する方法はいろいろな方法があります。

以下の資料に関連情報がまとまっています。

（Caché SQL最新情報）

<http://www.intersystems.co.jp/support/symposia.html>

特に新たにシステム環境を構築する場合（テスト環境、本番環境を問わず）、テーブルチューニングは重要です。

新規にシステムを構築する場合には、ある程度のデータを投入した後に全てのテーブルに対して

テーブルチューニングを実施するようにしてください。　通常最初に一回だけ行えばそれ以降は再度行う必要はありません。

これを行わない場合には、本来のクエリー性能を発揮できません。

### パラレルクエリー

2015.1よりマルチコアシステム上でのクエリ処理の性能向上のため、パラレルクエリー機能が追加されました。

比較的処理分割の容易なクエリーは、この機能により利用するシステムによっては大幅な性能改善が期待できます。

### ローカル変数使用

2012.2よりローカル変数の容量制限が大幅に改善されました。

従来容量の関係でローカル変数の代わりにグローバル変数、CacheTemp変数または

プロセスプライベートグローバル変数を使っていた処理をローカル変数に置き換えること

で劇的に性能改善するケースがあります。

ローカル変数へのアクセスはグローバル変数へのアクセスに比べて圧倒的に高速です。

しかしシステムの物理的なメモリーの容量には必ず限界がありますので、そのシステム

のメモリー資源を使い果たさないように注意して利用する必要があります。

## スケーラビリティ

### スケールアップ　or　スケールアウト

昨今プロセッサーに関する技術革新は目覚ましいものがあり、マルチコアシステムが

普及してきました。

結果として1つのシステム（物理的な筐体）で処理できるデータ量が劇的に向上してきました。

その結果、一時期スケーラビリティを上げていく方法として複数のサーバーを一体化して

運用するスケールアウトが主流でしたが、サーバの性能を上げていくスケールアップ手法

が再評価されつつあります。（さらに一台の物理サーバーを複数の仮想システムとして運用するケースも普及）

インターシステムズでも世の中の変化に対応すべく、従来はECPを使ったスケールアウト手法

を推奨してきましたが、昨今は単一サーバーによるスケールアップ手法を推奨しています。

### ECP

上記のようにスケールアップ手法が見直されつつある現状がありますが、当然限界や制約も

ありますので、その場合にはECPによるスケールアウト手法も検討の余地があります。

（私見ですが、それほどのスケールを求められる案件は、海外では可能性がありますが、

現状日本市場では、あまり考えにくいと思います。）

）

しかし、ECPを使用したスケールアウト手法には、様々な避けるべき落とし穴がありますので、

利用の際には注意して下さい。

#### データベース更新

バッチ処理による大量データ更新のような処理はでき得る限りECPクライアント（アプリケーションサーバー）上で実施するのではなくECPサーバー上で直接実行することを推奨します。

データ更新処理に関してはECPによるネットワークキャッシュ効果は非常に限定されますし、インタラクティブな処理では、人の入力時間やシンキングタイムなどのコンピュータ処理よりずっと遅い処理時間が介在することで、ECPに関する処理遅延の全体への影響を軽減することができますが、インタラクティブでない処理では、ECP部分の遅延が直接全体に大きく影響します。

ローカルなメモリー処理とネットワークを介した処理では簡単に何百倍、何千倍の処理時間の差が発生してしまいます。

#### データベース大量読み込み

同様にECPネットワークキャッシュ効果があるとは言え、ECPを経由したデータベース読み込みは、ローカルなデータベース読み込みに比較し、何十倍も遅いです。

従って、インタラクティブでない比較的長い時間を要する処理は、ECPサーバー側で処理することを推奨します。

#### ロングストリング

8Kブロックデータベースの場合、データ長が4000文字を少し超えるグローバルノードをロングストリング・ノードと呼んでおり、通常のノードとは異なったデータ構造を持っています。

ロングストリングノードはECPのキャッシュの対象にはなりません。

従ってECPクライアントからアクセス毎にECPサーバーからネットワーク経由で取得します。

従ってロングストリングノードはECPサーバー上でアクセスするか、ECPクライアント上のローカルデータベースとして保持する方法を推奨します。

ロングストリングデータになる可能性のあるデータは、バイナリーストリーム（画像など）やキャラクターストリーム（文書など）、長い文字列情報、ビットスライス、またはビットマップインデックスなどがあります。

#### $DATA

ECPクライアントからECPサーバー上のデータにアクセスする際にそのデータの存在確認を行うために$DATA関数を使用するケースがありますが、この場合$DATAの対象となるグローバルが存在しない場合、存在確認のためにキャッシュとデータベースの両方を確認しなければならないためECPキャッシュの効果を得ることができません。

従って特に頻繁に行われる処理内では、$DATAによる存在確認は行わないことを推奨します。

## その他

以下の問題が原因で性能に影響することがあります。

### Windows Large Page問題

Windows上で比較的大きなメモリーを確保しようとすると失敗する

詳細は以下参照のこと

<http://faq.intersystems.co.jp/csp/faq/result.CSP?DocNo=258>

### ロックエスカレーション

大量レコードの一括更新などの際に行ロックがテーブルロックに遷移することにより問題を引き起こすことがある。

詳細は以下参照のこと

<http://faq.intersystems.co.jp/csp/faq/result.CSP?DocNo=244>

# 補足資料

## サンプルデータ

この文書の理解を深めるためにこの文書で使用しているモデルを実際に実装したサンプルを添付しています。

### サンプルファイルの構成

PM.inc

このサンプルプロジェクトで使用する共通のリテラル値を定義したファイル

PM.Setup.cls

テストデータを自動生成するための処理を記述したクラス

PopulateBasics()　メソッド

このプロジェクトで使用するクラスの基本的なデータを自動生成するメソッド

PopulateRelations()　メソッド

データ間のリレーションを設定するメソッド

PopulateTransactions()　メソッド

アクティビティを自動生成するメソッド

PM.Utility.cls

テスト支援クラス

ImportYubinData()メソッド

郵便データをロードするメソッド

LoadPersonImage()メソッド

人の写真データを読み込んでデータベースに登録する処理

PM.PopulateUtilsクラス

自動データ生成を支援するクラス

PM.YubinData.cls

日本郵便の全国郵便データを取り込むためのクラス

PM.Activity.cls

アクテイビティを定義するクラス

PM.Address.cls

住所用クラス

PM.Customer.cls

顧客クラス

PM.Errorクラス

Error用共通クラス

StoreErrorInformation()メソッド

エラー情報を永続化する処理

PM.Managerクラス

マネージャ（管理職）クラス

PM.Memberクラス

メンバー（プロジェクトメンバー）クラス

PM.Organization　クラス

組織クラス

PM.Partyクラス

人、組織の基底クラス

PM.Personクラス

人クラス

PM.Phaseクラス

プロジェクトフェーズクラス

PM.Projectクラス

プロジェクトクラス

PM.ActivityEntryPage.cls

PM.ActivityEntryTemplate.cls

Zen Mojo アクテイビティ入力ページ

PM.ActivityQueryPage.cls

PM.ActivityQueryTemplate.cls

Zen Mojo アクテイビティ検索ページ

ActivityReport.cls

Zen Reportサンプル

### セットアップ方法

solution.zip

PM.XML プロジェクト管理サンプル一式

TextMessage.zip Zen Form日本語化ツール一式

JDate.zip UnitTestサンプル一式

PM-TopSales-pivot.xml DeepSeeダッシュボードサンプル

#### プロジェクト管理サンプル

PM.XMLをスタジオからロードします。

スタジオ>ツール>ローカルからインポート（L）

郵便データをロードします。

コマンド実行例（ファイルをc:\tempに置いたという前提）

USER>Set ^%SYS("sql","sys","week ISO8601") = 1

USER>do ##class(PM.YubinData).Import(,"c:\temp\ken\_all.csv")

データ生成処理を実行します。

USER>do ##class(PM.SetUp).PopulateBasics()

USER>do ##class(PM.SetUp).Relations()

USER>do ##class(PM.SetUp).Transactions()

### Zen Formの日本語化

手順は、TextMessage.zip内のreadmeを参照してください。

### UnitTestサンプル

#### jdate.xmlをインポート

スタジオ>ツール>ローカルからインポート（L）

jdatetest.xmlを適当なディレクトリを作成してそこに置く

c:\UnitTests\JDate

以下のコマンドを発行

USER>Set ^UnitTestRoot = "c:\UnitTests"

次に以下のメソッドを実行

USER>do ##class(%UnitTest.Manager).DebugLoadTestSuite("JDate")

#### テスト実行

続いて以下のメソッドを実行

do ##class(%UnitTest.Manager).DebugRunTestCase("JDate")