ExCella Coreリファレンスガイド

[1. ExCella Coreとは 3](#_Toc232580168)

[1.1. 利用イメージ 3](#_Toc232580169)

[1.2. ExCella Coreフレームワーク 3](#_Toc232580170)

[1.2.1. コントローラ 4](#_Toc232580171)

[1.2.2. パーサ 5](#_Toc232580172)

[1.2.2.1. SheetParser 5](#_Toc232580173)

[1.2.2.2. TagParser 5](#_Toc232580174)

[1.2.3. エクスポータ 7](#_Toc232580175)

[1.2.3.1. BookExporter 7](#_Toc232580176)

[1.2.3.2. SheetExporter 7](#_Toc232580177)

[1.2.4. 解析結果の利用 8](#_Toc232580178)

[2. チュートリアル 9](#_Toc232580179)

[2.1. 勤務表の取り込み 9](#_Toc232580180)

[2.1.1. ヘッダの読み込み 9](#_Toc232580181)

[2.1.1.1. ListParser 9](#_Toc232580182)

[2.1.1.2. MapParser 10](#_Toc232580183)

[2.1.2. 明細の読み込み 11](#_Toc232580184)

[2.1.2.1. ArraysParser 11](#_Toc232580185)

[2.1.2.2. MapsParser 12](#_Toc232580186)

[2.1.2.3. ObjectsParser 13](#_Toc232580187)

[3. 標準TagParser仕様 15](#_Toc232580188)

[3.1. TagParser共通情報 15](#_Toc232580189)

[3.2. ListParser 15](#_Toc232580190)

[3.3. MapParser 15](#_Toc232580191)

[3.4. ArraysParser 16](#_Toc232580192)

[3.5. MapsParser 16](#_Toc232580193)

[3.6. ObjectsParser 16](#_Toc232580194)

[4. Tips 18](#_Toc232580195)

[4.1. エラーファイルの出力 18](#_Toc232580196)

[5. 技術情報 19](#_Toc232580197)

[5.1. 処理フロー/カスタマイズポイント 19](#_Toc232580198)

[5.2. カスタムTagParserの使用 19](#_Toc232580199)

[5.2.1. カスタムTagParserの作成 19](#_Toc232580200)

[5.2.2. タグとの関連付け 20](#_Toc232580201)

[5.3. カスタムSheetParseListenerの使用 21](#_Toc232580202)

[5.3.1. カスタムSheetParseListenerの作成 21](#_Toc232580203)

[5.3.2. カスタムSheetParseListenerの追加 22](#_Toc232580204)

[5.4. カスタムBookExporterの使用 22](#_Toc232580205)

[5.4.1. カスタムBookExporterの作成 22](#_Toc232580206)

[5.4.2. カスタムBookExporterの追加 23](#_Toc232580207)

[5.5. カスタムSheetExporterの使用 24](#_Toc232580208)

[5.5.1. カスタムSheetExporterの作成 24](#_Toc232580209)

[5.5.2. カスタムSheetExporterの追加 25](#_Toc232580210)

[5.6. ユーティリティクラス 25](#_Toc232580211)

[5.6.1. PoiUtil 25](#_Toc232580212)

[5.6.2. TagUtil 25](#_Toc232580213)

# ExCella Coreとは

ExCella CoreはJavaでExcelファイルを操作するためのフレームワークです。

ExCella Coreを利用することで、Excel上のデータを簡単にJavaから利用できるようになります。

本ソフトウェアはApache License, Version 2.0にて公開しています。

## 利用イメージ

1. ExcelのデータをO/Rマッピングツールを通してRDBMSにインポート

ExCella Core

Excel

ファイル

Javaオブジェクト

へ変換

O/Rマッピングツール

RDBMS

1. Excelのデータを置換して帳票を作成

ExCella Core

Excel

ファイル

Excelのデータ

を置換

置換後のデータをファイル出力

Excel

ファイル

## ExCella Coreフレームワーク

ExCella Coreフレームワークは大きく、以下のように構成されています。

コンソール出力

ファイル出力

Excel出力

(標準)

Excel ⇒ Javaパーサ

(標準)

ExCella Core

Excel

ファイル

出力結果

1. コントローラ

③エクスポータ

②パーサ

① コントローラ … Excelファイルのデータ解析から出力までの処理を

制御する部分。BookControllerクラスのこと。

② パーサ … Excelファイルのデータの解析処理を行う部分。

SheetParserとTagParserがある。

* SheetParser … Excelシートのデータを解析し、TagParserを呼び出す。
* TagParser … Excelシート内に記述されたタグ毎にデータを解析する。≪拡張可能≫

③ エクスポータ … 解析結果の出力処理を行う部分。

BookExporterとSheetExporterがある。

* BookExporter … ブック全体の解析結果の出力処理を行う。≪拡張可能≫
* SheetExporter … シート単位で解析結果の出力処理を行う。≪拡張可能≫
* 標準で、Excel ⇒ Java変換のTagParserと各種Exporterが用意されていますが、

TagParserとExporterを拡張、差し替える事で様々な用途への応用が可能です。

### コントローラ

Excelファイルのデータ解析から出力までの大まかな処理の流れを説明します。

より詳細な情報は、[5.1. 処理フロー/カスタマイズポイント](#_標準パーサ仕様)を参照してください。

1. コントローラがExcelファイルのシート毎にSheetParserを呼び出す。
2. SheetParserがシート内のタグを検査し、タグに応じたTagParserを呼び出す。
3. TagParserが解析処理を実行する。
4. コントローラがエクスポータを呼び出す。
5. エクスポータが解析結果の出力処理を実行する。

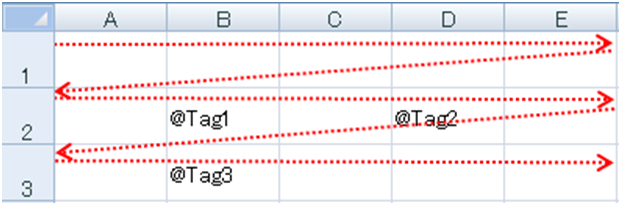
### パーサ

パーサはExcelファイルのデータの解析を行う処理クラスです。SheetParserとTagParserがあります。

#### SheetParser

SheetParserはExcelシートのデータを解析し、TagParserを呼び出すクラスです。

ブックに含まれるシートを先頭から順に、シート内では下図の矢印方向にタグ(※)を検査します。



上記の例では、@Tag1 → @Tag2 → @Tag3の順でタグに対応したTagParserの処理を呼び出し、データを解析します。

##### タグとは

タグはTagParserを起動するためにExcelファイル中に埋め込む定義で、

タグ種別(赤字部分)と、パラメータ(**青字**部分)に分けられ、タグ種別によって実行されるTagParserが決まります。

パラメータ部分は実行時にTagParserに渡され、パラメータによってTagParserの動作を制御します。

なお、タグ種別には任意の文字列を設定することができます。

例）

@List{**DataRowFrom=1,DataRowTo=4**}

#### TagParser

TagParserはExcelファイル中のデータをタグ単位で解析するための処理クラスです。

Excelファイル中に記述されたタグによって呼び出され、タグ種別と関連付けて使用します。

##### 標準TagParser

ExCella Coreでは標準TagParserとしてExcel ⇒ Java変換用の下記のTagParserが用意されており、

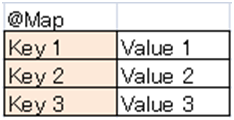
標準TagParserとExcel上のタグを組み合わせることでJavaへの変換を実現しています。

* ListParser … 列方向のセルの値をjava.util.Listに変換します。



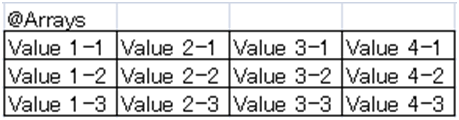
Value 1 ～ Value 3のリストを作成

* MapParser … 列方向のセルの値の組み合わせ(Key, Value)を、java.util.Mapに変換します。



キー：Key 1 ～ 3、値：キーに対応するValueのマップを作成

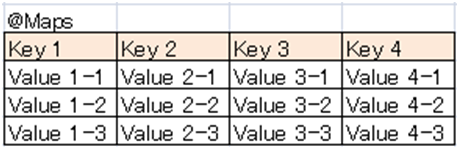
* ArraysParser … java.util.List<Object[]>へと変換します。(Object[]は一行分のデータ)



Value 1-1 ～ Value 4-1を一つの配列とし、

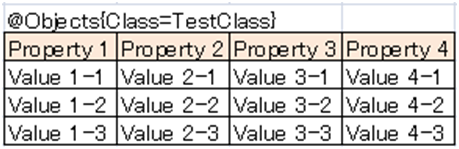
3行分の配列データを持つリストを作成

* MapsParser … java.util.List<Map>へと変換します。(Mapは一行分のデータ)



キー：Key 1 ～ 4、値：キーに対応するValue 1-1 ～Value 4-1を一つのマップとし、3行分のマップデータを持つリストを作成

* ObjectsParser … java.util.List<Object>へと変換します。(Objectは1行分のデータ)



Property 1 ～ 4のプロパティを持つクラスTestClassのオブジェクトに、対応するValue 1-1 ～4-1のプロパティ値をセットしたものを一つのデータとし、3行分のオブジェクトデータを持つリストを作成

##### カスタムTagParser

上記の標準TagParserで対応できない場合は、独自のカスタムTagParserを追加することができます。

詳細は[5.2. カスタムTagParserの使用](#_カスタムTagParserの使用)を参照してください。

### エクスポータ

エクスポータはパーサを利用して解析した結果を出力するための処理クラスです。

BookExporterとSheetExporterがあります。

#### BookExporter

ブック単位で任意の出力処理を差し込むためのインターフェイスです。

##### 標準BookExporter

標準で、下記のBookExporterが用意されています。

* ConsoleExporter … コンソールへの出力
* TextFileExporter … ファイルへの出力
* WorkbookExporter … ワークブックへの出力

##### カスタムBookExporter

上記の標準BookExporterで対応できない場合は、独自のカスタムBookExporterを追加することができます。

詳細は[5.4. カスタムBookExporterの使用](#_カスタムBookExporterの使用)を参照してください。

#### SheetExporter

シート単位で任意の出力処理を差し込むためのインターフェイスです。

シート単位で出力処理を切り替えたい場合に使用します。

##### 標準Exporter

標準で、下記のSheetExporterが用意されています。

* ConsoleExporter … コンソールへの出力
* TextFileExporter … ファイルへの出力

##### カスタムSheetExporter

上記の標準SheetExporterで対応できない場合は、独自のカスタムSheetExporterを追加することができます。

詳細は[5.5. カスタムSheetExporterの使用](#_カスタムSheetExporterの使用)を参照してください。

### 解析結果の利用

解析結果は解析処理後にBookControllerからBookDataとして取得でき、以降は自由に利用することができます。

BookDataはシート数分のSheetDataを含み、SheetDataはシート内のタグの処理結果をMapで保持します。

Mapのキーはデフォルトではタグ名、[ResultKeyパラメータ](#_TagParser共通情報)を指定した場合は指定値になります。

Mapの値にはタグの処理結果が入ります。

BookData





|  |
| --- |
| BookController controller = new BookController( args[0]);  ：  controller.parseBook();    // 結果の取得  BookData bookData = controller.getBookData();  // シートデータリストの取得  Collection<SheetData> sheetDatas = bookData.getSheetDatas();  for ( SheetData sheetData : sheetDatas) {  // 任意の処理  } |

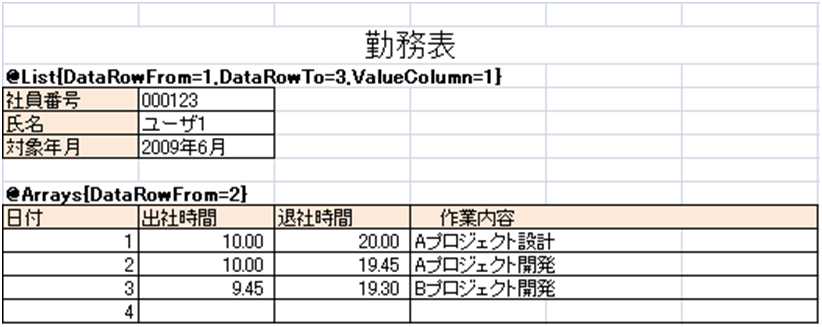
# チュートリアル

## 勤務表の取り込み

それではExCella Coreを利用して下記の様なExcelの勤務表を取り込む場合の手順を見ていきます。

* ここで紹介するプログラムおよびテンプレートはサンプル(org.bbreak.excella.core.samples. WorkFactReportReader)に含まれています。

なお、説明上、タグ行は表示状態で記載していますが、Excel上でタグ行を非表示にしても処理は可能です。



ヘッダ部分

明細部分

### ヘッダの読み込み

社員番号～対象年月部分までを読み込む方法について説明します。

このように列にデータが並んでいる場合は

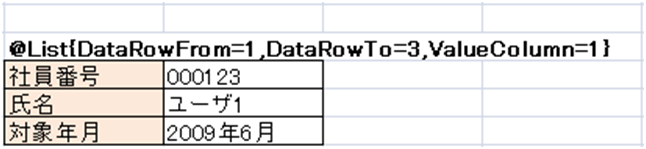
* ListParser
* MapParser

の2種類の標準TagParserが利用できます。

#### ListParser

下記のようにExcelファイルのヘッダ部分にタグ(@List)を埋め込みます。

タグ中で使用しているパラメータの詳細は、[3. 標準TagParser仕様](#_標準パーサ仕様_2)を参照してください。



読み込みたい範囲は、タグのセルから見て、

1行目～3行目(DataRowFrom=1,DataRowTo=3)

1列目(ValueColumn=1)として処理

“@List”タグにListParserを対応付けて追加します。

また、結果をコンソールに出力するために、ConsoleExporterを追加します。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // 標準TagParserの追加  controller.addTagParser( new ListParser( "@List"));  // 結果をコンソールに出力  controller.addBookExporter( new ConsoleExporter());  // 解析の実行  parser.parseBook(); |

* + **実行結果**

@List [000123, ユーザ1, Mon Jun 01 00:00:00 JST 2009]

ListにDataRowFrom～DataRowToまでのデータが追加されていることが確認できます。

#### MapParser

MapParserを使用すると、キーと値の組み合わせでデータを取得できます。

また、データの順序はキーが追加された順序を保持します。

下記のようにExcelファイルのヘッダ部分にタグ(@Map)を埋め込みます。



読み込みたい範囲は、タグのセルから見て、

1行目～3行目(DataRowFrom=1,DataRowTo=3)

キーは0列目：デフォルトのため未指定

値は1列目：デフォルトのため未指定

として処理

“@Map”タグにMapParserを対応付けて追加します。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // 標準TagParserの追加  controller.addTagParser( new MapParser( "@Map"));  // 結果をコンソールに出力  controller.addBookExporter( new ConsoleExporter());  // 解析の実行  controller.parseBook(); |

* + **実行結果**

@Map {社員番号=000123, 氏名=ユーザ1, 対象年月=Mon Jun 01 00:00:00 JST 2009 }

MapにDataRowFrom～DataRowToまでのデータ（キー、値）が追加されていることが確認できます。

### 明細の読み込み

続いて明細部分の読み込みを行います。

明細部分のような一連のデータを扱う場合には

* ArraysParser
* MapsParser
* ObjectsParser

の3種類の標準TagParserが利用できます。

#### ArraysParser

下記のようにExcelファイルのヘッダ部分にタグ(@Arrays)を埋め込みます。



読み込みたい範囲は、タグのセルから見て、

2行目以降(DataRowFrom=2)

“@Arrays”タグにArraysParserを対応付けて追加します。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // 標準TagParserの追加  controller.addTagParser( new ArraysParser( "@Arrays"));  // 結果をコンソールに出力  controller.addBookExporter( new ConsoleExporter());  // 解析の実行  controller.parseBook(); |

* + **実行結果(4行分のリスト)**

@Arrays [[Ljava.lang.Object;@a86d12, [Ljava.lang.Object;@190d8e1, [Ljava.lang.Object;@8eae04, [Ljava.lang.Object;@142c778]

4行分のリスト。各要素には下記のように値が入る(下記は1行目の場合)

[0]=1.0(Double)

[1]=10.0(Double)

[2]= 20.0(Double)

[3]=Aプロジェクト設計(String)

※ 数値データはすべてDouble型になります。

#### MapsParser

下記のようにExcelファイルのヘッダ部分にタグ(@Maps)を埋め込みます。



タグのセルから見て、

1行目をキー(KeyRow=1)

2行目以降をデータ行として処理

“@Maps”タグにMapsParserを対応付けて追加します。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // 標準TagParserの追加  controller.addTagParser( new MapsParser("@Maps"));  // 結果をコンソールに出力  controller.addBookExporter( new ConsoleExporter());  // 解析の実行  controller.parseBook(); |

* + **実行結果(見やすいように加工してあります)**

@Maps [

{日付=1.0, 出社時間=10.0, 退社時間=20.0, 作業内容=Aプロジェクト設計},

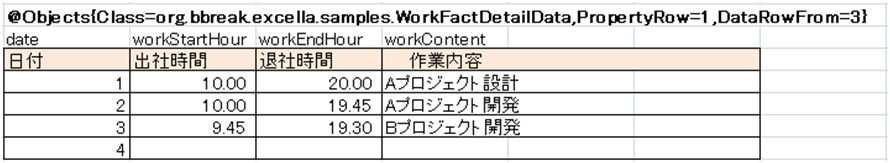
{日付=2.0, 出社時間=10.0, 退社時間=19.45, 作業内容=Aプロジェクト開発},

{日付=3.0, 出社時間=9.45, 退社時間=19.3, 作業内容=Bプロジェクト開発},

{日付=4.0, 出社時間=null, 退社時間=null, 作業内容=null}]

#### ObjectsParser

下記のようにExcelファイルのヘッダ部分にタグ(@Objects)を埋め込みます。



1行目をプロパティ(PropertyRow=1)、

3行目以降をデータ(DataRowFrom=3)

として、”org.bbreak.excella.samples.WorkFactDetailData”オブジェクトに値を設定

“@Objects”タグにObjectsParserを対応付けて追加します。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // 標準TagParserの追加  controller.addTagParser( new ObjectsParser("@Objects"));  // 結果をコンソールに出力  controller.addBookExporter( new ConsoleExporter());  // 解析の実行  controller.parseBook(); |

* + **実行結果(見やすいように加工してあります)**

@Objects [

[date=Mon Jan 01 00:00:00 JST 1900,workStartHour=10.0,workEndHour=20.0,workContent=Aプロジェクト設計],

[date=Tue Jan 02 00:00:00 JST 1900,workStartHour=10.0,workEndHour=19.45,workContent=Aプロジェクト開発],

[date=Wed Jan 03 00:00:00 JST 1900,workStartHour=9.45,workEndHour=19.3,workContent=Bプロジェクト開発],

[date=Thu Jan 04 00:00:00 JST 1900,workStartHour=null,workEndHour=null,workContent=null]]

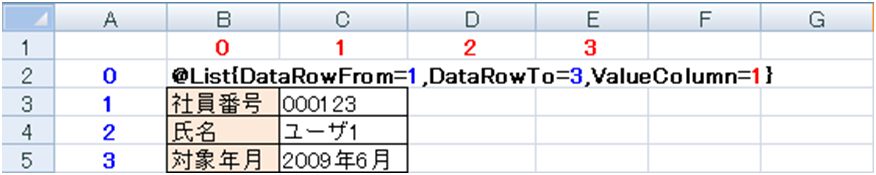
# 標準TagParser仕様

## TagParser共通情報

全ての標準TagParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| DataRowFrom | @List{DataRowFrom=1} | 任意 | タグが定義されたセルから見て何行目からデータ行が開始するか。  未指定の場合はそれぞれのタグのデフォルト値を適用する。  ≪デフォルト値≫  ・@List … 1  ・@Map … 1  ・@Arrays … 1  ・@Maps … 2  ・@Objects … 2 |
| DataRowTo | @List{DataRowTo=1} | 任意 | タグが定義されたセルから見て何行目までをデータ行として処理を行うか。  未指定の場合は、シートの最終行（Apache POIのSheet.getLastRowNum()の取得値）までをデータ行とみなす |
| ResultKey | @List{ResultKey=UserList} | 任意 | 処理結果に対応するキー。  未指定の場合はタグ種別が使用される。 |

* パラメータで指定する行・列の数値はタグが定義されたセルを基準（0, 0）とし、そのセルから見たときの相対値で表します。



タグが定義されたセルが基準(0, 0)となる

## ListParser

ListParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| ValueColumn | @List{ValueColumn=1} | 任意 | タグが定義されたセルからみて何列目を値として使用するか。  未指定の場合は0 |

## 

## MapParser

MapParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| Key | @Map{Key=XXX} | 任意 | キーに固定文字列を使用する場合に指定。 |
| Value | @Map{Value=YYY} | 任意 | 値に固定文字列を使用する場合に指定。 |
| KeyColumn | @Map{KeyColumn=1} | 任意 | タグが定義されたセルからみて何列目をキーとして使用するか  未指定の場合は0 |
| ValueColumn | @Map{ValueColumn=2} | 任意 | タグが定義されたセルからみて何列目を値として使用するか  未指定の場合は1 |
| KeyCell | @Map{KeyCell=1:1} | 任意 | タグが定義されたセルから見て何行、何列目のセルをキーとして使用するか。 |
| ValueCell | @Map{ValueCell=1:2} | 任意 | タグが定義されたセルから見て何行、何列目のセルを値として使用するか。 |

※ Key, KeyColumn, KeyCellの2つ以上が定義されていた場合は不正な定義としてParseExceptionがThrowされます

※ Value, ValueColumn, ValueCellの2つ以上が定義されていた場合は不正な定義としてParseExceptionがThrowされます

## ArraysParser

ArraysParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| DataColumnFrom | @Arrays{DataColumnFrom=1} | 任意 | セルから見て何列目からをデータとして使用するか。  未指定の場合は1 |
| DataColumnTo | @Arrays{DataColumnTo=4} | 任意 | セルから見て何列目までをデータとして使用するか。  未指定の場合は最終列まで。 |

## 

## MapsParser

MapsParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| KeyRow | @Maps{KeyRow=1} | 任意 | タグが定義されたセルから見て何行目をキーとするか  未指定の場合は1 |

## 

## ObjectsParser

ObjectsParserが理解するパラメータ情報です。

|  |  |  |  |
| --- | --- | --- | --- |
| パラメータ | 定義例 | 必須/任意 | 説明 |
| Class | @Objects{Class=org.\*\*\*} | 必須 | Listに詰めるオブジェクトのクラス。 |
| PropertyRow | @Objects{PropertyRow=1} | 任意 | タグが定義されたセルからみて何行目をプロパティ行として使用するか  未指定の場合は1 |

* プロパティ名の先頭を"-"にするとコメントとみなされ処理対象外になります。

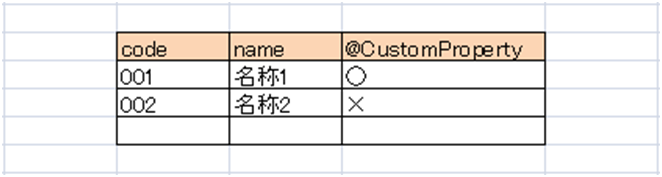
ObjectsParserでは単純にオブジェクトに対してプロパティを設定するだけでなく、

特定のプロパティ(カスタムプロパティ)に対して任意の処理を差し込む事ができます。

カスタムプロパティも通常のタグと同様にプロパティ行(PropertyRowで指定した行)のプロパティ名を

任意のタグで記述し、タグに対応するプロパティパーサを追加するとパーサの処理が実行されます。

**Excel定義**



**カスタムプロパティパーサの追加**

|  |
| --- |
| BookController controller = new BookController( args[0]);  // Objectsパーサの生成  ObjectsParser objectsParser = new ObjectsParser( "@Objects");  // カスタムプロパティパーサの追加  objectsParser.addPropertyParser( new CustomPropertyParser( "@CusotmProperty"));  // パーサの追加  controller.addTagParser( objectsParser);  // 解析の実行  controller.parseBook(); |

# Tips

## エラーファイルの出力

パース時に何らかのエラーが発生した際に、デバッグ用のエラーハンドラを設定しておくと、

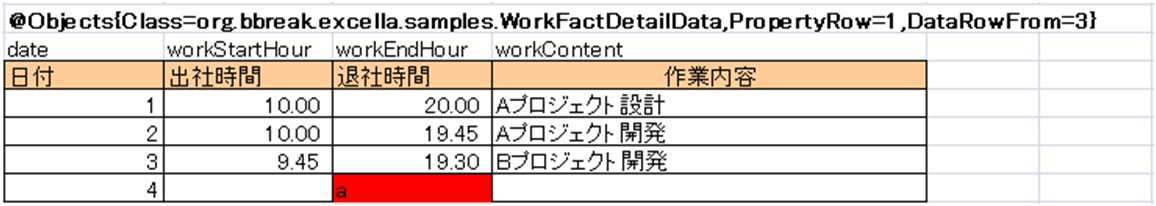
エラー情報の追加されたExcelファイルが実行ディレクトリ直下に出力されます。

エラーとなったセルが特定可能な場合はエラーのセルの背景色が赤色になっていますので、

こちらのファイルを参照すると原因が特定できる可能性があります。

|  |
| --- |
| BookController controller = new BookController( args[0]);  // デバッグ用エラーハンドラの設定(エラー時はエラーファイルを作成)  controller.setErrorHandler( new DebugErrorHandler()); |

例）エラーファイル



数値への変換に失敗したためエラー。

背景色が赤色で表示される。

※Ver1.4からセルコメントとして詳細なエラー内容が表示されるようになりました。(xlsのみ)

# 技術情報

本章ではExCella Coreをカスタマイズする開発者向けの技術情報を紹介します。

## 処理フロー/カスタマイズポイント

BookController

SheetParseListener

SheetParser

TagParser

BookExporter

SheetExporter

new()

setErrorHandler()

addTagParser()

addSheetParseListener()

parseBook()

preParse()

parseSheet()

postParse()

setup()

export()

tearDown()

setup()

export()

tearDown()

シートごとに

並び順で処理

初期化

パース

エクスポート

カスタマイズポイント

カスタムTagParserの追加

カスタマイズポイント

リスナの追加でイベント時に処理追加

カスタマイズポイント

カスタムBookExporterの設定

カスタマイズポイント

カスタムSheetExporterの設定

parse()

addSheetExporter ()

addBookExporter ()

## カスタムTagParserの使用

独自の解析処理を実現する場合には、TagParserを継承したクラスを作成し、任意のタグと関連付けて使用します。

### カスタムTagParserの作成

例として、解析結果をList<String>で返すCustomTagParserを作成してみます。

TagParserを継承し、オーバーライドしたparseメソッドに解析処理を記述し、戻り値の型を

List<String>型にします。ここで返却したデータが解析結果データとしてSheetDataに格納されます。

|  |
| --- |
| public class CustomTagParser extends TagParser<List<String>> {  /\*\*  \* コンストラクタ  \* @param tag タグ  \*/  public CustomTagParser( String tag) {  super( tag);  }  /\*\*  \* パース処理  \* @param sheet 対象シート  \* @param tagCell タグが定義されたセル  \* @param data BookControllerのparseBook(), parseSheet()メソッド、  \* SheetParserのparseSheetメソッドで引数を渡した場合に  \* TagParserまで引き継がれる処理データ  \* @return 解析結果  \* @throws ParseException パース例外  \*/  @Override  public List<String> parse( Sheet sheet, Cell tagCell, Object data)  throws ParseException {  List<String> result = new ArrayList<String>();  // 解析処理を記述  ：  ：  // 解析結果を返す  return result;  }  } |

### タグとの関連付け

上記で作成したCustomTagParserを“@Custom”タグに関連付けます。

全てのシートでCustomTagParserを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // @CustomタグにCustomTagParserを関連付ける  controller.addTagParser( new CustomTagParser( "@Custom"));  ：  ：  controller.parseBook(); |

Sheet1のみでCustomTagParserを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // Sheet1の@CustomタグにCustomTagParserを関連付ける  controller.addTagParser( "Sheet1", new CustomTagParser( "@Custom"));  ：  ：  controller.parseBook(); |

## カスタムSheetParseListenerの使用

解析処理の実行前後に任意の処理を実行する場合には、SheetParseListenerを実装したクラスを作成し、BookControllerに追加して使用します。

### カスタムSheetParseListenerの作成

例として、解析処理の実行前と実行後にコンソールに文字列を出力するCustomParseListenerを作成します。

SheetParseListenerを実装し、preParse()、postParse()にそれぞれコンソールに文字列を出力する処理を記述します。

postParse()では、引数で渡されるsheetDataを使用して、解析結果データに対して処理を行うことができます。

|  |
| --- |
| public class CustomSheetParseListener implements SheetParseListener {  /\*\*  \* 解析処理前に実行する処理  \* @param sheet 対象シート  \* @param sheetParser シートパーサ  \* @throws ParseException パース例外  \*/  public void preParse( Sheet sheet, SheetParser sheetParser) throws ParseException {  System.out.println( "preParseの実行");  }  /\*\*  \* 解析処理後に実行する処理  \* @param sheet 対象シート  \* @param sheetParser シートパーサ  \* @param sheetData解析結果データ  \* @throws ParseException パース例外  \*/  public void postParse( Sheet sheet, SheetParser sheetParser, SheetData sheetData) throws ParseException {  System.out.println( "postParseの実行");  }  } |

### カスタムSheetParseListenerの追加

上記で作成したCustomSheettParseListenerをBookControllerに追加します。

全てのシートでCustomSheetParseListenerを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // CustomSheetParseListenerをブックに追加  controller.addSheetParseListener ( new CustomSheetParseListener ()**)**;  ：  ：  controller.parseBook(); |

Sheet1のみでCustomSheetParseListenerを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // CustomSheetParseListenerをブックに追加  controller.addSheetParseListener ( "Sheet1", new CustomSheetParseListener ()**)**;  ：  ：  controller.parseBook(); |

## カスタムBookExporterの使用

ブック単位で独自の出力処理を実現する場合には、BookExporterを実装したクラスを作成し、BookControllerに追加して使用します。

### カスタムBookExporterの作成

例として、コンソールに文字列を表示するCustomBookExporterを作成してみます。

BookExporterを実装し、setup()メソッド、export()メソッド、tearDown()メソッドにそれぞれ処理を記述します。

|  |
| --- |
| public class CustomBookExporter implements BookExporter {  /\*\*  \* 初期化処理  \*/  public void setup() {  System.out.println( "初期化処理の実行");  }  /\*\*  \* 出力処理  \* @param book ワークブック  \* @param bookdata ブック単位の解析結果データ  \* @throws ExportException 出力処理例外  \*/  public void export( Workbook book, BookData bookdata)  throws ExportException {  // コンソールにsheetdataの内容を表示  Collection<SheetData> sheetDatas = bookdata.getSheetDatas();  for ( SheetData sheetdata : sheetDatas) {  System.out.println( sheetdata.toString());  }  }  /\*\*  \* 終了処理  \*/  public void tearDown() {  System.out.println( "終了処理の実行");  }  } |

### カスタムBookExporterの追加

上記で作成したCustomBookExporterをBookControllerに追加します。

CustomBookExporterを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // Exporterをブックに追加  controller.addBookExporter( new CustomBookExporter()**)**;  ：  ：  controller.parseBook(); |

## カスタムSheetExporterの使用

シート単位で独自の出力処理を実現する場合には、SheetExporterを実装したクラスを作成し、BookControllerに追加して使用します。

### カスタムSheetExporterの作成

例として、コンソールに文字列を表示するCustomSheetExporterを作成してみます。

SheetExporterを実装し、setup()メソッド、export()メソッド、tearDown()メソッドにそれぞれ処理を記述します。

|  |
| --- |
| public class CustomSheetExporter implements SheetExporter {  /\*\*  \* 初期化処理  \*/  public void setup() {  System.out.println( "初期化処理の実行");  }  /\*\*  \* 出力処理  \* @param sheet シート  \* @param sheetdata シート単位の解析結果データ  \* @throws ExportException 出力処理例外  \*/  public void export( Sheet sheet, SheetData sheetdata) throws ExportException {  // コンソールにsheetdataの内容を表示  System.out.println( sheetdata.toString());  }  /\*\*  \* 終了処理  \*/  public void tearDown() {  System.out.println( "終了処理の実行");  } |

### カスタムSheetExporterの追加

上記で作成したCustomSheetExporterをBookControllerに追加します。

Sheet1でCustomSheetExporterを使用する場合

|  |
| --- |
| BookController controller = new BookController( args[0]);  // Sheet1の解析結果を処理するExporterをシートに追加  controller.addSheetExporter( "Sheet1", new CustomSheetExporter()**)**;  ：  ：  controller.parseBook(); |

## ユーティリティクラス

カスタマイズに利用できるユーティリティクラスを紹介します。ユーティリティクラスはorg.bbreak.excella.core.util下にあります。

### PoiUtil

POI操作のユーティリティクラスです。

セルの値をセルのタイプに応じた型で取得するメソッドやセルに値をセットするメソッド等、POI操作関連のメソッドが用意されています。

より詳細な情報は、ExCella CoreのJavadocを参照してください。

### TagUtil

タグ定義操作のユーティリティクラスです。

タグ定義からパラメータ部分をマップ型で取得するメソッドやタグ定義からパラメータ部分を除いた文字列を取得するメソッド等、タグ操作関連のメソッドが用意されています。

より詳細な情報は、ExCella CoreのJavadocを参照してください。