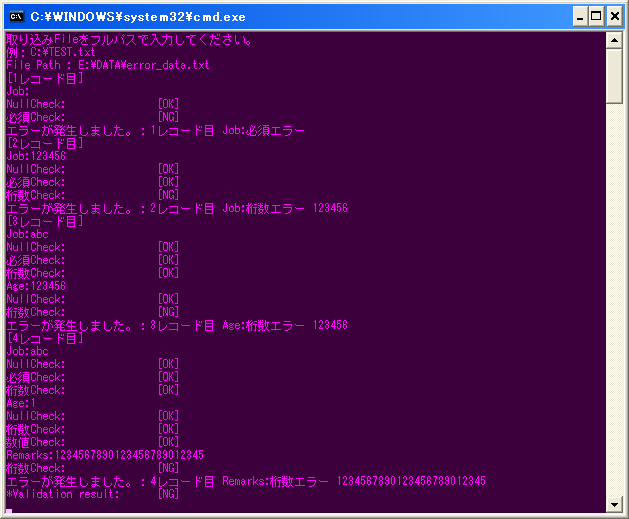
概要

何らかの値をデータベースへ登録することを前提とし、テーブルの項目に合わせた検証を行う処理を行います。  
コンソールより、テーブル項目の順番にカンマ区切りで定義されたファイルを読み込み、各項目ごとに必須、桁数、数値チェックなどの検証を行います。  


データ構成

Job、Age、Remarksの3項目を持つテーブルを想定します。

public class MyData

{

public string Job { get; set; }

public string Age { get; set; }

public string Remarks { get; set; }

public static MyData SetData(string[] lineData)

{

MyData data = new MyData();

data.Job = lineData[0];

data.Age = lineData[1];

data.Remarks = lineData[2];

return data;

}

}

検証はValidateメソッドにて行えるようインタフェースを用意し、項目ごとに実装します。  
検証結果がNGの場合はメッセージと共に例外を投げます。

interface IValidation

{

void Validate();

}

項目：Job

必須項目であり、5桁までの値を許します。

class Job : IValidation

{

private readonly string \_s;

public Job(string s)

{

this.\_s = s;

}

public void Validate()

{

Console.WriteLine("Job:" + \_s);

if (!ValidationTools.NullCheck(\_s)) {

throw new Exception("Job:Nullエラー " + \_s);

};

if (!ValidationTools.RequiredCheck(\_s)) {

throw new Exception("Job:必須エラー " + \_s);

};

if (!ValidationTools.DigitsCheck(\_s, 5)) {

throw new Exception("Job:桁数エラー " + \_s);

};

}

}

項目：Age

数値項目であり、3桁までの値を許します。

class Age : IValidation

{

private readonly string \_s;

public Age(string s)

{

this.\_s = s;

}

public void Validate()

{

Console.WriteLine("Age:" + \_s);

if (!ValidationTools.NullCheck(\_s))

{

throw new Exception("Age:Nullエラー " + \_s);

};

if (!ValidationTools.DigitsCheck(\_s, 3))

{

throw new Exception("Age:桁数エラー " + \_s);

};

if (!ValidationTools.NumberCheck(\_s))

{

throw new Exception("Age:数値エラー " + \_s);

};

}

}

項目：Remarks

20桁までの値を許します。

class Remarks : IValidation

{

private readonly string \_s;

public Remarks(string s)

{

this.\_s = s;

}

public void Validate()

{

Console.WriteLine("Remarks:" + \_s);

if (!ValidationTools.DigitsCheck(\_s, 20))

{

throw new Exception("Remarks:桁数エラー " + \_s);

};

}

}

検証内容

様々な検証処理はValidationToolsとしてまとめておきます。  
検証結果はNGの場合はfalse、OKの場合はtrueを返します。

static class ValidationTools

{

public static bool NullCheck(string s)

{

if (s == null)

{

Console.WriteLine("NullCheck: [NG]");

return false;

}

Console.WriteLine("NullCheck: [OK]");

return true;

}

public static bool RequiredCheck(string s)

{

if (s.Length == 0)

{

Console.WriteLine("必須Check: [NG]");

return false;

}

Console.WriteLine("必須Check: [OK]");

return true;

}

public static bool DigitsCheck(string s,int digit)

{

if (s.Length > digit)

{

Console.WriteLine("桁数Check: [NG]");

return false;

}

Console.WriteLine("桁数Check: [OK]");

return true;

}

public static bool NumberCheck(string s)

{

int o;

if (!int.TryParse(s,out o))

{

Console.WriteLine("数値Check: [NG]");

return false;

}

Console.WriteLine("数値Check: [OK]");

return true;

}

}

検証を行うクラス

検証する項目、実効順序は検証用インタフェースのListとして保持し、Validateメソッドをまとめて実行するクラスを用意します。

class Validator

{

private readonly List<IValidation> validationList = new List<IValidation>();

public List<IValidation> ValidationList

{

get

{

return validationList;

}

}

public void Validate()

{

foreach (IValidation item in validationList)

{

item.Validate();

}

}

}

検証処理の制御

検証を行う項目および実行順序を設定します。

static Validator CreateValidator(MyData data)

{

Validator validator = new Validator();

validator.ValidationList.Add(new Job(data.Job));

validator.ValidationList.Add(new Age(data.Age));

validator.ValidationList.Add(new Remarks(data.Remarks));

return validator;

}

Main処理

コンソールよりファイルパスを入力し、ファイルを読み込みます。  
読み込んだファイルの内容をカンマで分割し、各項目に割り当てて検証を行い問題があれば[NG]を、なければ[OK]を画面に表示します。

class Program

{

static void Main(string[] args)

{

Console.WriteLine("取り込みFileをフルパスで入力してください。");

Console.WriteLine("例：C:\\TEST.txt");

Console.Write("File Path : ");

string filePath = Console.ReadLine();

if (File.Exists(filePath))

{

string line = string.Empty;

using (StreamReader sr = new StreamReader(filePath, Encoding.GetEncoding("Shift-JIS")))

{

int i = 0;

bool result = true;

while ((line = sr.ReadLine()) != null)

{

string[] lineData = line.Split(',');

if (lineData.Count() != 3)

{

Console.WriteLine("フォーマットが異なります");

throw new Exception();

}

MyData data = MyData.SetData(lineData);

Console.WriteLine("[" + ++i + "レコード目]");

Validator validator = CreateValidator(data);

try

{

validator.Validate();

}

catch (Exception ex)

{

Console.WriteLine("エラーが発生しました。：" + i + "レコード目 " + ex.Message);

result = false;

}

}

if (result)

{

Console.WriteLine("\*Validation result: [OK]");

}

else

{

Console.WriteLine("\*Validation result: [NG]");

}

}

}

else

{

Console.WriteLine("Fileが存在しません。");

}

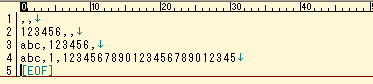
Console.ReadKey();

}

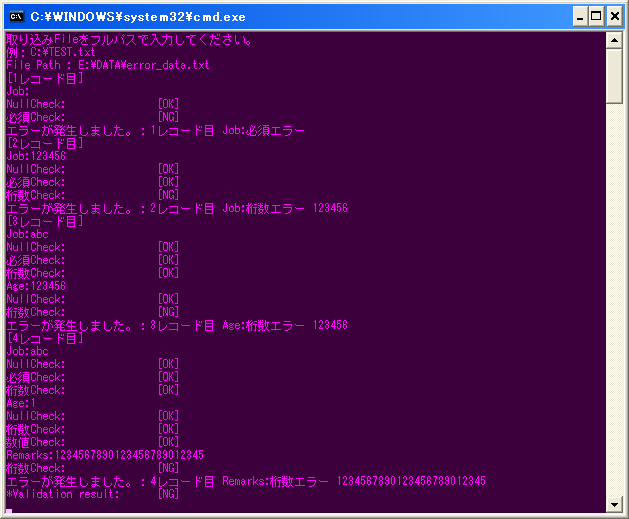
}

実行結果

テストデータ



結果



参考

[](http://www.amazon.co.jp/exec/obidos/ASIN/4873115175/hatena-blog-22/)

[**プログラミングC# 第6版**](http://www.amazon.co.jp/exec/obidos/ASIN/4873115175/hatena-blog-22/)