記事
Toshihiko Minamoto · 2020年10月5日 15m read

Open Exchange

CachéとCosFakerを使ったテスト駆動開発の簡単な紹介

CachéとCosFakerを使ったテスト駆動開発の簡単な紹介

読了**目安時間**: 6分

皆さん、こんにちは。

私がTDDに初めて出会ったのは約9年前のことです。すぐに夢中になってしまいました。
最近は非常に人気が出てきているようですが、残念ながら多くの企業ではあまり使われていないようです。 また、主に初心者の方ではありますが、一体それがなんであるのか、どのように使うのかといったことさえも知らない開発者もたくさんいます。

概要

この記事は、%UnitTestでTDDを使用する方法を紹介することを目標としています。
ワークフローを示し、私の最初のプロジェクトであったcosFaker
の使用方法を説明します。こ
れはCachéを使って作成したものであり、最近になってOpenExchangeにアップロードしたものです。

では、ベルトを締めて出発しましょう。

TDDとは？

テスト駆動開発（TDD）は、自動テストが失敗した場合に、開発者に新しいコードの書き方のみを示すプログラミング実践として定義できます。
このメリットに関する記事、講義、講演などは数多く存在しますが、どれもが正しい内容です。
コードはテスト済みで生成されること、過度なエンジニアリングを避けるために定義された要件に、システムが実際に適合していることを確認できること、継続的にフィードバックを得ることが挙げられます。

では、TDDを使用しない理由は何でしょうか。 TDDにはどのような問題があるのでしょうか。
答えは単純です。そう、コストです！ とにかくコストがかかります！
TDDではより多くの行のコードを記述する必要があるため、その処理には時間がかかります。 しかし、TDDを使用すると、製品を作成するための最終コストは現時点で発生し、後で追加コストをかける必要がありません。
常にテストを実行すれば、早期にエラーを検出できるため、修正にかかるコストが削減されるのです。
というわけで、私からのアドバイスは、ただ実行に移しましょう！

セットアップ

InterSystems
は、%UnitTestの使用方法
に関するドキュメントとチュートリアルを用意しています。こちらからお読みください。

開発にはvscodeを使用します。 この方法で、テスト用に別のフォルダを作成し、 UnitTestRootにプロジェクトコードパスを追加して、テストを実行する際に、テストサブフォルダの名前を渡します。
そして必ず、修飾子loadudlを渡します。

```
Set ^UnitTestRoot = "~/code"

  Do ##class(%UnitTest.Manager).RunTest("myPack","/loadudl")
```

手順

おそらく、「レッド ➡グリーン➡リファクタリング」という有名なTDDサイクルを耳にしたことがあるでしょう。
失敗するテストを書き、合格する単純なプロダクションコードを書き、そしてそのプロダクションコードをリファクタリングするというサイクルです。
では、実際に手を動かして、計算を行うクラスとそれをテストする別のクラスを作成することにしましょう。
後者のクラスは、%UnitTest.TestCaseを拡張します。
では、整数の2乗を返すClassMethodを作成しましょう。

```
Class Production.Math

{


ClassMethod Square(pValue As %Integer) As %Integer

{

}


}
```

そして、2を渡すとどうなるかテストします。 4を返すはずです。

```
Class TDD.Math Extends %UnitTest.TestCase

{


Method TestSquare()

{

    Do $$$AssertEquals(##class(Production.Math).Square(2), 4)

}


}
```

次のコードを実行します。

```
Do ##class(%UnitTest.Manager).RunTest("TDD","/loadudl")
```

テストは失敗します。

レッド！ 次のステップは、これをグリーンにすることです。
グリーンにするために、Squareメソッドの実行結果として4を返すようにしましょう。

```
Class Production.Math

{


ClassMethod Square(pValue As %Integer) As %Integer

{

  Quit 4

}


}
```

そしてテストを再実行します。

1つのシナリオでしか機能しないのですから、このソリューションでは、おそらくあまり嬉しくないのではないでしょうか。 わかりました！ では次のステップに進みましょう。
別のシナリオを作成することにします。今度は負の数を渡してみます。

```
Class TDD.Math Extends %UnitTest.TestCase

{
```

```
Method TestSquare()

{

    Do $$$AssertEquals(##class(Production.Math).Square(2), 4)

}


Method TestSquareNegativeNumber()

{

    Do $$$AssertEquals(##class(Production.Math).Square(-3), 9)

}


}
```

テストを実行します。

また失敗してしまうので、プロダクションコードをリファクタリングしましょう。

```
Class Production.Math

{


ClassMethod Square(pValue As %Integer) As %Integer

{

  Quit pValue * pValue

}


}
```

そして、テストを再実行します。

これですべてがうまく機能するようになりました... これが凝縮版のTDDサイクルです。

なぜこの手順に従う必要があるのか、と思っていることでしょう。
なぜテストを失敗させる必要があるのか、と。
私はプロダクションコードを記述したチームで作業したことがありますが、テストを作成したのはその後でした。
それでも、この小さな一歩に従う方を好むのには、次の理由があります。
ボブおじさん（Robert C.
Martin）は、コードを書いた後でテストを書くことは「TDD」ではなく「時間の無駄」だと呼んだのです。
もう少し述べれば、テストが失敗し、次に合格することから、テストをテストしていることになります。
このテストはコードに変わりなく、誤りが含まれることもあります。
そして、それをテストする方法こそ、失敗して合格する必要のある場合に失敗と合格を保証することになります。

つまり、「テストをテストした」ということになります。

cosFaker

適切なテストを作成するには、最初にテストデータを生成する必要があります。
これを行う方法の1つが、データのダンプを生成して、テストに使うという方法です。
別の方法には、cosFaker
を使用して、必要なときに偽のデータを簡
単に作り出す手があります。https://openexchange.intersystems.com/package/CosFaker

xml ファイルをダウンロードするだけです。その後、管理ポータル -> システムエクスプローラ -> Classes -> Importに移動します。 インポートする xml ファイルを選択するか、そのファイルを Studio にドラッグします。
また、ターミナルを使用してインポートすることもできます。

```
Do $system.OBJ.Load("yourpath/cosFaker.vX.X.X.xml","ck")
```

ローカリゼーション

cosFakerは、デフォルトのCSPアプリケーションフォルダにロケールファイルを追加します。
現時点では、英語とブラジルポルトガル語（私の母国語）の2言語しかありません。
データの言語は、Cachéの構成に応じて選択されます。
cosFakerのローカリゼーションは進行中のプロセスです。ご協力いただける方は、ぜひ、自身のロケール向けにローカライズされるプロバイダを作成してプルリクエストを提出してください。
cosFakerを使用すれば、ランダムな語、段落、電話番号、名前、住所、メール、価格、製品名、日付、16進色コードなどを生成することができます。

すべてのメソッドは、クラスでサブジェクトごとにグループ化されます。つまり、Latitudeを生成するには、AddressクラスのLatitudeメソッドを呼び出します。

```
 Write ##class(cosFaker.Address).Latitude()

-37.6806
```

また、テスト用のJsonを生成することもできます。

```
Write ##class(cosFaker.JSON).GetDataJSONFromJSON("{ip:'ipv4',created_at:'date.backward 40',login:'username', text: 'words 3'}")

{
    "created_at":"2019-03-08",
    "ip":"95.226.124.187",
    "login":"john46",
    "text":"temporibus fugit deserunt"
}
```

以下は、**cosFaker**クラスとメソッドの全リストです。

- **cosFaker.Address**
  - StreetSuffix
  - StreetPrefix
  - PostCode

        - StreetName
        - Latitude
            - *出力:* -54.7274
        - Longitude
            - *出力:* -43.9504
        - Capital(Location = “ ”)
        - State(FullName = 0)
        - City(State = “ ”)
        - Country(Abrev = 0)
        - SecondaryAddress
        - BuildingNumber
- **cosFaker.App**
    - FunctionName(Group= “ ”, Separator = “ ”)
    - AppAction(Group= “ ”)
    - AppType
- **cosFaker.Coffee**
    - BlendName
        - *出力:* Cascara Cake
    - Variety
        - *出力:* Mundo Novo
    - Notes
        - *出力:* crisp, slick, nutella, potato defect!, red apple
    - Origin
        - *出力:* Rulindo, Rwanda
- **cosFaker.Color**
    - Hexadecimal
        - *出力:* #A50BD7
    - RGB
        - *出力:* 189,180,195
    - Name
- **cosFaker.Commerce**
    - ProductName
    - Product
    - PromotionCode
    - Color
    - Department
    - Price(Min = 0, Max = 1000, Dec = 2, Symbol = “ ”)
        - *出力:* 556.88
    - CNPJ(Pretty = 1)
        - CNPJはブラジルの法人用税務登記番号です
        - *出力:* 44.383.315/0001-30
- **cosFaker.Company**
    - Name
    - Profession
    - Industry
- **cosFaker.Dates**
    - Forward(Days = 365, Format = 3)
    - Backward(Days = 365, Format = 3)
- **cosFaker.DragonBall**
    - Character
        - *出力:* Gogeta
- **cosFaker.File**
    - Extension
        - *出力:* txt
    - MimeType
        - *出力:* application/font-woff
    - Filename(Dir = “ ”, Name = “ ”, Ext = “ ”, DirectorySeparator = “ / ”)
        - *出力:* repellat.architecto.aut/aliquid.gif
- **cosFaker.Finance**

    - Amount(Min = 0, Max = 10000, Dec = 2, Separator=“ , ” , Symbol = “ ”)
        - *出力:* 3949,18
    - CreditCard(Type = “ ”)
        - *出力:* 3476-581511-6349
    - BitcoinAddress(Min = 24, Max = 34)
        - *出力:* 1WoR6fYvsE8gNXkBkeXvNqGECPUZ
- **cosFaker.Game**
    - MortalKombat
        - *出力:* Raiden
    - StreetFighter
        - *出力:* Akuma
    - Card(Abrev = 0)
        - *出力:* 5 of Diamonds
- **cosFaker.Internet**
    - UserName(FirstName = “ ” , LastName = “ ”)
    - Email(FirstName = “ ” , LastName = “ ” , Provider = “ ”)
    - Protocol
        - *出力:* http
    - DomainWord
    - DomainName
    - Url
    - Avatar(Size = “ ”)
        - *出力:* http://www.avatarpro.biz/avatar?s=150
    - Slug(Words = “ ” , Glue = “ ”)
    - IPV4
        - *出力:* 226.7.213.228
    - IPV6
        - *出力:* 0532:0b70:35f6:00fd:041f:5655:74c8:83fe
    - MAC
        - *出力:* 73:B0:82:D0:BC:70
- **cosFaker.JSON**
    - GetDataOBJFromJSON(Json = “ ” // JSON template string to create data)
        - *パラメーターの例*: "{dates:'5 date'}"
        - *出力*: {"dates":["2019-02-19","2019-12-21","2018-07-02","2017-05-25","2016-08-14"]}
- **cosFaker.Job**
    - Title
    - Field
    - Skills
- **cosFaker.Lorem**
    - Word
    - Words(Num = “ ”)
    - Sentence(WordCount = “ ” , Min = 3, Max = 10)
        - *出力:* Sapiente et accusamus reiciendis iure qui est.
    - Sentences(SentenceCount = “ ” , Separator = “ ”)
    - Paragraph(SentenceCount = “ ”)
    - Paragraphs(ParagraphCount = “ ” , Separator = “ ”)
    - Lines(LineCount = “ ”)
    - Text(Times = 1)
    - Hipster(ParagraphCount = “ ” , Separator = “ ”)
- **cosFaker.Name**
    - FirstName(Gender = “ ”)
    - LastName
    - FullName(Gender = “ ”)
    - Suffix
- **cosFaker.Person**
    - cpf(Pretty = 1)
        - CPFはブラジルの社会保障番号です
        - *出力:* 469.655.208-09
- **cosFaker.Phone**

        - PhoneNumber(Area = 1)
            - *出力:* (36) 9560-9757
        - CellPhone(Area = 1)
            - *出力:* (77) 94497-9538
        - AreaCode
            - *出力:* 17
- **cosFaker.Pokemon**
    - Pokemon(EvolvesFrom = “ ”)
        - *出力:* Kingdra
- **cosFaker.StarWars**
    - Characters
        - *出力:* Darth Vader
    - Droids
        - *出力:* C-3PO
    - Planets
        - *出力:* Takodana
    - Quotes
        - *出力:* Only at the end do you realize the power of the Dark Side.
    - Species
        - *出力:* Hutt
    - Vehicles
        - *出力:* ATT Battle Tank
    - WookieWords
        - *出力:* nng
    - WookieSentence(SentenceCount = “ ”)
        - *出力:* ruh ga ru hnn-rowr mumwa ru ru mumwa.
- **cosFaker.UFC**
    - Category
        - *出力:* Middleweight
    - Fighter(Category = “ ”, Country = “ ”, WithISOCountry = 0)
        - *出力:* Dmitry Poberezhets
    - Featherweight(Country = “ ”)
        - *出力:* Yair Rodriguez
    - Middleweight(Country = “ ”)
        - *出力:* Elias Theodorou
    - Welterweight(Country = “ ”)
        - *出力:* Charlie Ward
    - Lightweight(Country = “ ”)
        - *出力:* Tae Hyun Bang
    - Bantamweight(Country = “ ”)
        - *出力:* Alejandro Pérez
    - Flyweight(Country = “ ”)
        - *出力:* Ben Nguyen
    - Heavyweight(Country = “ ”)
        - *出力:* Francis Ngannou
    - LightHeavyweight(Country = “ ”)
        - *出力:* Paul Craig
    - Nickname(Fighter = “ ”)
        - *出力:* Abacus

ユーザー名を返すメソッドでユーザーのクラスを作成してみましょう。ユーザー名はFirstNameとLastNameを連結したものになります。

```
Class Production.User Extends %RegisteredObject

{
```

```
Property FirstName As %String;

Property LastName As %String;

Method Username() As %String
{
}

}
```

```
Class TDD.User Extends %UnitTest.TestCase
{

Method TestUsername()
{
  Set firstName = ##class(cosFaker.Name).FirstName(),
    lastName = ##class(cosFaker.Name).LastName(),
    user = ##class(Production.User).%New(),
    user.FirstName = firstName,
    user.LastName = lastName

  Do $$$AssertEquals(user.Username(), firstName _ "." _ lastName)
}

}
```

リファクタリング:

```
Class Production.User Extends %RegisteredObject
{

Property FirstName As %String;

Property LastName As %String;
```

```
Method Username() As %String

{

  Quit ..FirstName _ "." _ ..LastName

}


}
```

アカウントの有効期限を追加して、検証することにします。

```
Class Production.User Extends %RegisteredObject

{


Property FirstName As %String;


Property LastName As %String;


Property AccountExpires As %Date;


Method Username() As %String

{

  Quit ..FirstName _ "." _ ..LastName

}


Method Expired() As %Boolean

{

}


}
```

```
Class TDD.User Extends %UnitTest.TestCase

{


Method TestUsername()

{
```

```
  Set firstName = ##class(cosFaker.Name).FirstName(),
    lastName = ##class(cosFaker.Name).LastName(),
    user = ##class(Production.User).%New(),
    user.FirstName = firstName,
    user.LastName = lastName
    Do $$$AssertEquals(user.Username(), firstName _ "." _ lastName)
}

Method TestWhenIsNotExpired() As %Status
{
  Set user = ##class(Production.User).%New(),
    user.AccountExpires = ##class(cosFaker.Dates).Forward(40)
  Do $$$AssertNotTrue(user.Expired())
}

}
```

リファクタリング:

```
Method Expired() As %Boolean
{
  Quit ($system.SQL.DATEDIFF("dd", ..AccountExpires, +$Horolog) > 0)
}
```

では、アカウントの有効期限が切れた場合をテストしてみましょう。

```
Method TestWhenIsExpired() As %Status
{
  Set user = ##class(Production.User).%New(),
    user.AccountExpires = ##class(cosFaker.Dates).Backward(40)
  Do $$$AssertTrue(user.Expired())
}
```

すべてがグリーンです。

これらはあまり大したことのない例かもしれませんが、このようにすることで、コードだけでなくクラスの設計も単純にすることができます。

まとめ

この記事では、テスト駆動開発と%UnitTestクラスの使用方法について少しだけ学習しました。
また、cosFakerとテスト用の偽のデータの生成方法についても説明しました。

テストとTDDについては、これらの実践をレガシーコードで使用する方法、統合テスト、受け入れテスト駆動開発など、ほかにも学習することがたくさんあります。
詳細については、次の2冊が私のイチオシです。

『Test Driven Development Teste e design no mundo real com Ruby』Mauricio Aniche著 -
これについては英語版が出版されているかわかりません。 Java、C#、Ruby、およびPHP版があります。
あまりの素晴らしさに感銘を受けた一冊です。

そしてもちろん、Kent Beckの『Test Driven Development by Example』。

コメントやご質問はお気軽にどうぞ。
これで、おしまいです。

#テスト #Caché #InterSystems IRIS
InterSystems Open Exchangeで関連アプリケーションを確認してください

ソースURL:
https://jp.community.intersystems.com/post/cach%C3%A9%E3%81%A8cosfaker%E3%82%92%E4%BD%BF%E3%81%A3%E3%81%9F%E3%83%86%E3%82%B9%E3%83%88%E9%A7%86%E5%8B%95%E9%96%8B%E7%99%BA%E3%81%AE%E7%B0%A1%E5%8D%98%E3%81%AA%E7%B4%B9%E4%BB%8B