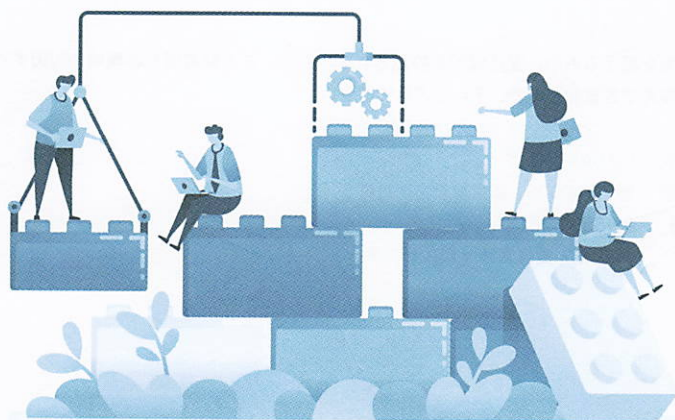


ドメイン駆動 設計入門

ボトムアップでわかる! ドメイン駆動設計の基本

成瀬 允宣 著



SE
SHOEISHA

本書内容に関するお問い合わせについて

このたびは翔泳社の書籍をお買い上げいただき、誠にありがとうございます。

弊社では、読者の皆様からのお問い合わせに適切に対応させていただくため、以下のガイドラインへのご協力をお願い致しております。

下記項目をお読みいただき、手順に従ってお問い合わせください。

ご質問される前に

弊社 Web サイトの「正誤表」をご参照ください。これまでに判明した正誤や追加情報を掲載しています。

正誤表 <https://www.shoeisha.co.jp/book/errata/>

ご質問方法

弊社 Web サイトの「刊行物 Q&A」をご利用ください。

刊行物 Q&A <https://www.shoeisha.co.jp/book/qa/>

インターネットをご利用でない場合は、FAXまたは郵便にて、下記翔泳社愛読者サービスセンターまでお問い合わせください。電話でのご質問は、お受けしていません。

回答について

回答は、ご質問いただいた手段によってご返事申し上げます。ご質問の内容によっては、回答に数日ないしはそれ以上の期間を要する場合があります。

ご質問に際してのご注意

本書の対象を越えるもの、記述個所を特定されないもの、また読者固有の環境に起因するご質問等にはお答えできませんので、予めご了承ください。

郵便物送付先および FAX 番号

送付先住所 〒160-0006 東京都新宿区舟町5

FAX 番号 03-5362-3818

宛先 (株)翔泳社 愛読者サービスセンター

※本書に記載された URL 等は予告なく変更される場合があります。

※本書の対象に関する詳細はvページをご参照ください。

※本書の出版にあたっては正確な記述につとめましたが、著者や出版社などのいずれも、本書の内容に対してなんらかの保証をするものではなく、内容やサンプルに基づくいかなる運用結果に関してもいっさいの責任を負いません。

※本書に掲載されているサンプルプログラムやスクリプト、および実行結果を記した画面イメージなどは、特定の設定に基づいた環境にて再現される一例です。

※本書に記載されている会社名、製品名はそれぞれ各社の商標および登録商標です。

※本書の内容は、2020年1月執筆時点のものです。

開発者にとってドメイン駆動設計を学ぶことはその後のエンジニアリングに対するスタンスを変えうるほどの大きな学びです。

ドメイン駆動設計のコンセプトは単純です。ビジネスの問題を解決するためにビジネスの理解を進め、ビジネスの表現をする。ビジネスとコードを結びつけて継続的かつ反復的な改良を施せるように枠組みを作ることにより、ソフトウェアをより役立つものにしようというものです。いわば、ソフトウェアシステムにおいて当たり前のことを行おうとしています。

しかし、いざドメイン駆動設計を学ぼうとした初学者の多くは、その難解さに打ちのめされてしまいます。なぜ打ちのめされてしまうのでしょうか。その原因は多種多様な専門用語の難しさにあるのではないかと考えます。

ドメイン駆動設計の解説はユビキタス言語という用語の解説から始まります。ページを少しめくってみればドメインエキスパートという言葉が筆頭に、境界付けられたコンテキスト、エンティティ、値オブジェクトといった多くの専門用語が現れます。これらの用語は当然のことながら、初学者にとっては初めて見聞きするものです。専門用語の中には多くの予備知識を前提とするものが存在し、ほとんど恐怖と似たような感情を思い起こさせ、理解を妨げてしまっています

人が恐怖する原因をご存知でしょうか。

人は知らないことに恐怖します。たとえ風に揺れるカーテンも、正体を知らなければ幽霊になり替わります。まさにドメイン駆動設計の専門用語は幽霊で、恐怖を与えるものに違いありません。恐怖は人を鈍らせます。恐怖は人を遠ざけます。

幽霊が怖い理由はもちろん、その正体がわからないからです。怖さを和らげるためにすべきことは単純です。幽霊の正体を確かめて、それがカーテンであることを知ってしまえばよいのです。

ドメイン駆動設計を学習するうえで登場する用語は大別すると2種類に分かれます。ソフトウェアにとって重要な概念を抽出するためのモデリングと概念を実装に落とし込むためのパターンです。モデリングに関しては言葉による説明を理解し実践して学ぶほかありませんが、パターンに関しては詳細なサンプルをもって説明できます。開発者にとって理解しやすいのはもちろん後者です。

そこで本書はモデリングについてはいったん棚上げにして、具体的なコードをベースにパターンを集中的に解説し、全体を通して最終的なコードを提示します。どこかへ行こうとすると、目的地がわからないということはとても不安を覚えます。本書が提示するコードは明確な形をもった目的地としてあなたの道しるべになるものです。

本書はパターンを解説することに重きを置いています。しかし、これはパターンを実践することがすなわちドメイン駆動設計の実践であり、そのすべてであるという主張をしているわけではありません。本書が目的としているのは、ドメイン駆動設計という巨大な試練に立ち向かうための準備です。ある知識を理解しようとするとき、また別の知識が手掛かりとなることは多く、なるほど知識は連鎖するものです。ドメイン駆動設計という大きなテーマと対峙する準備として必要なことは、あらかじめ理解の領地を広げておくことです。

本書から得られるパターンに関しての理解はモデリングを含めたドメイン駆動設計の全貌を理解するのに役に立つものです。これからドメイン駆動設計の世界を旅するあなたの武器となるでしょう。

よりよいソフトウェアを開発するために、開発者として一段と高みへ上るために、あなたがドメイン駆動設計を学ぶことを選んだのは間違いではありません。

2020年1月吉日

成瀬允宣

Acknowledgments

謝辞

本書を執筆するにあたり、多くの方々に助けをいただき、支えていただきました。

末安 章花氏の熱烈な後押しがなければ本書は始まりませんでした。

大平 道介氏が背中を押さなければここまでたどり着けませんでした。

林 宏勝氏の丁寧な査読は多くの言葉に磨きをかけました。

森 怜峰氏の鋭いコメントはその度に大幅な加筆につながりました。

松岡 幸一郎氏の芯の通った考えは筆者の考察をより強固にしました。

加藤 潤一氏の忌憚ない意見は考察の機会を与えてくれました。

増田 亨氏の言葉は筆者に勇気を与えてくれました。

貴重な時間を惜しみなく費やし、支えてくれた彼等に感謝を。

そして私を励まし続けてくれた妻に感謝を。

Introduction

本書の対象読者

本書は主にオブジェクト指向プログラミング言語を用いているソフトウェア開発者向けに書かれていますが、これはドメイン駆動設計を実践するにはオブジェクト指向プログラミング言語が必須であるということを意味しません。オブジェクト指向以外のパラダイムをもつ言語であっても、本書のエッセンスをソフトウェア開発に役立てることは可能です。

本書を読み進めるには一般的なオブジェクト指向プログラミングの基礎知識が必要です。具体的に求められるレベルとしては、クラスとインスタンスについての理解が必要です。他にはインターフェース（抽象型）機能をふんだんに利用しているので、インターフェースを用いたポリモーフィズムの処理の流れは理解しておく読み進める助けになります。

本書で取り扱うサンプルコードのプログラミング言語はC#を採用しています。C#は一般的なオブジェクト指向プログラミング言語としての機能を網羅しており、また記述方法にあまり癖がない言語と考えているからです。たとえC#についてまったく知識がなかったとしても、オブジェクト指向プログラミング言語を利用している開発者であれば読み換えることはそう難しいことでないでしょう。

いずれにせよ深い知識は必要なく、コードを書くことを生業とする読者であれば読み進めることは可能でしょう。

本書ではいくつか C# 特有の構文が利用されています。いずれも冗長さを排除するための記述ですが、事前にここで解説をしておきます。

| readonly

フィールド（インスタンス変数）に再代入される可能性があるか否かというのは貴重な情報です。再代入が不可能であれば、その値が変化したときのことを考慮せずに済みます。

C# ではインスタンス変数に `readonly` 修飾子を付けることで再代入を不可能にできます。

リスト 1 : `readonly` 修飾子を用いたフィールドの定義

```
class MyClass
{
    private readonly string invariant; // 再代入不可能
    private string variant; // 再代入可能
    (...略...)
}
```

なお、`readonly` で修飾されたインスタンス変数はコンストラクタ以外で代入を行うことができません。

リスト 2 : `readonly` への代入

```
class MyClass
{
    private readonly string value;

    public MyClass(string value) {
        this.value = value; // OK
    }

    public void ChangeValue(string value) {
        this.value = value; // コンパイルエラー
    }
}
```

プロパティ

プロパティはフィールドに外部からアクセスするための機能です。リスト 3 のように定義すると内部でフィールドが自動実装され、アクセスできるようになります。

リスト 3：プロパティによる定義

```
class MyClass
{
    public string Property { get; private set; }
}
```

プロパティは値を取得するためのゲッターと値を設定するためのセッターを定義できます。ゲッターとセッターはそれぞれ異なるアクセス修飾子を設定できます。ゲッターのみを定義した場合は読み取り専用のプロパティになり、コンストラクタでのみ代入が可能になります。

またゲッターやセッターはメソッドのようにふるまいをもたせることも可能です。

リスト 4：ふるまいをもったプロパティ

```
class MyClass
{
    private string property;

    public string Property
    {
        get { return property; }
        set { property = value; }
    }
}
```

using 句

ファイルなどのリソースを保持するオブジェクトは、オブジェクトが不要になったときにリソースを解放するために、終了処理を実行する必要があります。C# ではそういった終了処理を確実に呼び出すために using 句を利用します。

using 句はオブジェクトのスコープを明示し、スコープから抜ける際にそのオブジェクトの終了処理を呼び出します。

リスト 5 : using 句を用いたリソースの取得

```
using(var connection = new SqlConnection(connectionString))
{
    (...略...)
}
```

using 句に指定されるオブジェクト (リスト 5 でいうところの SqlConnection) は IDisposable インターフェースを実装する必要があります。スコープを抜ける際に IDisposable インターフェースの Dispose メソッドが呼び出されます。

Introduction

本書のサンプルの動作環境とサンプルプログラムについて

本書の第 14 章のサンプルは表 1 の環境で、問題なく動作することを確認しています。

項目	内容
OS	Windows 10 Home
IDE	Visual Studio 16.4

表 1 : 実行環境

付属データのご案内

付属データは、以下のサイトからダウンロードできます。

- ・付属データのダウンロードサイト

URL <https://github.com/nrslib/itddd>

注意

付属データに関する権利は著者および株式会社翔泳社が所有しています。許可なく配布したり、Web サイトに転載したりすることはできません。

付属データの提供は予告なく終了することがあります。予めご了承ください。

会員特典データのご案内

会員特典データは、以下のサイトからダウンロードして入手いただけます。

・会員特典データのダウンロードサイト

URL <https://www.shoeisha.co.jp/book/present/9784798150727>

注意

会員特典データをダウンロードするには、SHOEISHA iD（翔泳社が運営する無料の会員制度）への会員登録が必要です。詳しくは、Webサイトをご覧ください。

会員特典データに関する権利は著者および株式会社翔泳社が所有しています。許可なく配布したり、Webサイトに転載したりすることはできません。

会員特典データの提供は予告なく終了することがあります。予めご了承ください。

免責事項

付属データおよび会員特典データの記載内容は、2020年1月現在の法令等に基づいています。

付属データおよび会員特典データに記載されたURL等は予告なく変更される場合があります。

付属データおよび会員特典データの提供にあたっては正確な記述につとめました。が、著者や出版社などのいずれも、その内容に対して何らかの保証をするものではなく、内容やサンプルに基づくいかなる運用結果に関してもいっさいの責任を負いません。

付属データおよび会員特典データに記載されている会社名、製品名はそれぞれ各社の商標および登録商標です。

著作権等について

付属データおよび会員特典データの著作権は、著者および株式会社翔泳社が所有しています。個人で使用する以外に利用することはできません。許可なくネットワークを通じて配布を行うこともできません。個人的に使用する場合は、ソースコードの改変や流用は自由です。商用利用に関しては、株式会社翔泳社へご一報ください。

2020年1月

株式会社翔泳社 編集部

はじめに	iii
謝辞	v
本書の対象読者	v
取り扱う C# 特有の構文	vi
本書のサンプルの動作環境とサンプルプログラムについて	viii

Chapter

1

ドメイン駆動設計とは

001

1.1 ドメイン駆動設計とは何か	002
1.2 ドメインの知識に焦点をあてた設計手法	003
1.2.1 ドメインモデルとは何か	004
1.2.2 知識をコードで表現するドメインオブジェクト	006
1.3 本書解説事項と目指すゴール	007
COLUMN ドメイン駆動設計の実践を難しくするもの	009
1.4 本書で解説するパターンについて	010
1.4.1 知識を表現するパターン	011
1.4.2 アプリケーションを実現するためのパターン	011
1.4.3 知識を表現する、より発展的なパターン	012
COLUMN なぜいま、ドメイン駆動設計か	014

Chapter

2

システム固有の値を表現する「値オブジェクト」

015

2.1 値オブジェクトとは	016
2.2 値の性質と値オブジェクトの実装	019
2.2.1 不変である	019
COLUMN 不変のメリット	021
2.2.2 交換が可能である	022
2.2.3 等価性によって比較される	023
2.3 値オブジェクトにする基準	028
2.4 ふるまいをもった値オブジェクト	033
2.4.1 定義されないからこそわかること	035
2.5 値オブジェクトを採用するモチベーション	036
2.5.1 表現力を増す	037
2.5.2 不正な値を存在させない	039

Chapter

3

ライフサイクルのあるオブジェクト「エンティティ」 047

2.5.3 誤った代入を防ぐ	040
2.5.4 ロジックの散在を防ぐ	043
2.6 まとめ	046
3.1 エンティティとは	048
3.2 エンティティの性質について	049
3.2.1 可変である	049
COLUMN セーフティネットとしての確認	052
3.2.2 同じ属性であっても区別される	052
3.2.3 同一性をもつ	055
3.3 エンティティの判断基準としてのライフサイクルと連続性	058
3.4 値オブジェクトとエンティティのどちらにもなりうるモデル	059
3.5 ドメインオブジェクトを定義するメリット	059
3.5.1 コードのドキュメント性が高まる	060
3.5.2 ドメインにおける変更をコードに伝えやすくする	062
3.6 まとめ	063

Chapter

4

不自然さを解決する「ドメインサービス」 065

4.1 サービスが指し示すもの	066
4.2 ドメインサービスとは	066
4.2.1 不自然なふるまいを確認する	067
4.2.2 不自然さを解決するオブジェクト	069
4.3 ドメインサービスの濫用が行き着く先	070
4.3.1 可能な限りドメインサービスを避ける	072
4.4 エンティティや値オブジェクトと共に ユースケースを組み立てる	073
4.4.1 ユーザエンティティの確認	073
4.4.2 ユーザ作成処理の実装	075
COLUMN ドメインサービスの基準	078
4.5 物流システムに見るドメインサービスの例	079
4.5.1 物流拠点のふるまいとして定義する	079
4.5.2 輸送ドメインサービスを定義する	081
COLUMN ドメインサービスの命名規則	082
4.6 まとめ	083

5.1	リポジトリとは	086
	COLUMN リポジトリはドメインオブジェクトを際立たせる	087
5.2	リポジトリの責務	087
5.3	リポジトリのインターフェース	092
	COLUMN nullの是非とOption型	094
5.4	SQLを利用したリポジトリを作成する	094
5.5	テストによる確認	098
5.5.1	テストに必要な作業を確認する	099
5.5.2	祈り信者のテスト理論	099
5.5.3	祈りを捨てよう	100
5.6	テスト用のリポジトリを作成する	100
5.7	オブジェクトリレーショナルマッパーを用いた リポジトリを作成する	104
5.8	リポジトリに定義されるふるまい	108
5.8.1	永続化に関するふるまい	108
5.8.2	再構築に関するふるまい	109
5.9	まとめ	111

6.1	アプリケーションサービスとは	114
	COLUMN アプリケーションサービスという名前	114
6.2	ユースケースを組み立てる	115
6.2.1	ドメインオブジェクトから準備する	115
6.2.2	ユーザ登録処理を作成する	119
6.2.3	ユーザ情報取得処理を作成する	120
	COLUMN 煩わしさを減らすために	127
6.2.4	ユーザ情報更新処理を作成する	127
	COLUMN エラーかそれとも例外か	133
6.2.5	退会処理を作成する	134
6.3	ドメインのルールの流れ	135
6.4	アプリケーションサービスと凝集度	144
6.4.1	凝集度が低いアプリケーションサービス	147
6.5	アプリケーションサービスのインターフェース	153
6.6	サービスとは何か	155

6.6.1 サービスは状態をもたない	156
6.7 まとめ	157

Chapter 7 柔軟性をもたらす依存関係のコントロール 159

7.1 技術要素への依存がもたらすもの	160
7.2 依存とは	161
7.3 依存関係逆転の原則とは	165
7.3.1 抽象に依存せよ	166
7.3.2 主導権を抽象に	167
7.4 依存関係をコントロールする	168
7.4.1 Service Locator パターン	170
7.4.2 IoC Container パターン	175
7.5 まとめ	178

Chapter 8 ソフトウェアシステムを組み立てる 179

8.1 ソフトウェアに求められるユーザーインターフェース	180
COLUMN ソフトウェアとアプリケーションの使い分け	180
8.2 コマンドラインインターフェースに組み込んでみよう	181
COLUMN シングルトンパターンと誤解	182
8.2.1 メインの処理を実装する	183
8.3 MVC フレームワークに組み込んでみよう	185
8.3.1 依存関係を設定する	186
8.3.2 コントローラを実装する	192
COLUMN コントローラの責務	195
8.4 ユニットテストを書こう	196
8.4.1 ユーザ登録処理のユニットテスト	196
8.5 まとめ	203
COLUMN 本当に稀な怪談話	204

Chapter 9 複雑な生成処理を行う「ファクトリ」 205

9.1 ファクトリの目的	206
9.2 採番処理をファクトリに実装した例の確認	207
COLUMN ファクトリの存在に気づかせる	213

9.2.1 自動採番機能の活用	214
9.2.2 リポジトリに採番用メソッドを用意する	216
9.3 ファクトリとして機能するメソッド	218
9.4 複雑な生成処理をカプセル化しよう	220
COLUMN ドメイン設計を完成させるために必要な要素	221
9.5 まとめ	221

Chapter 10

データの整合性を保つ 223

10.1 整合性とは	224
10.2 致命的な不具合を確認する	225
10.3 ユニークキー制約による防衛	228
10.3.1 ユニークキー制約を重複確認の主体としたときの 問題点	228
10.3.2 ユニークキー制約との付き合い方	230
10.4 トランザクションによる防衛	231
10.4.1 トランザクションを取り扱うパターン	232
10.4.2 トランザクションスコープを利用したパターン	235
10.4.3 AOPを利用したパターン	237
10.4.4 ユニットオブワークを利用したパターン	239
COLUMN 結局どれを使うべきか	250
10.4.5 トランザクションが引き起こすロックについて	250
10.5 まとめ	250

Chapter 11

アプリケーションを1から組み立てる 251

11.1 アプリケーションを組み立てるフロー	252
11.2 題材とする機能	252
11.2.1 サークル機能の分析	253
11.3 サークルの知識やルールをオブジェクトとして準備する	253
11.4 ユースケースを組み立てる	258
11.4.1 言葉との齟齬が引き起こす事態	262
11.4.2 漏れ出したルールがもたらすもの	263
11.5 まとめ	266

12.1 集約とは	268
12.1.1 集約の基本的構造	268
COLUMN 集約を保持するコレクションを図に表すか	272
12.1.2 オブジェクトの操作に関する基本的な原則	273
12.1.3 内部データを隠蔽するために	276
COLUMN よりきめ細やかなアクセス修飾子 (Scala)	280
12.2 集約をどう区切るか	280
12.2.1 IDによるコンポジション	285
COLUMN IDのゲッターに対する是非	288
12.3 集約の大きさと操作の単位	289
COLUMN 結果整合性	290
12.4 言葉との齟齬を消す	291
12.5 まとめ	292

13.1 仕様とは	294
13.1.1 複雑な評価処理を確認する	294
13.1.2 「仕様」による解決	297
13.1.3 リポジトリの使用を避ける	299
13.2 仕様とリポジトリを組み合わせる	302
13.2.1 お勧めサークルに見る複雑な検索処理	302
13.2.2 仕様による解決法	304
13.2.3 仕様とリポジトリが織りなすパフォーマンス問題	307
13.2.4 複雑なクエリは「リードモデル」で	309
COLUMN 遅延実行による最適化	314
13.3 まとめ	316

14.1 アーキテクチャの役目	318
14.1.1 アンチパターン：利口なUI	318
14.1.2 ドメイン駆動設計がアーキテクチャに求めること	321
14.2 アーキテクチャの解説	322
14.2.1 レイヤードアーキテクチャとは	322

14.2.2	ヘキサゴナルアーキテクチャとは	334
14.2.3	クリーンアーキテクチャとは	338
14.3	まとめ	342

Chapter 15

ドメイン駆動設計のとびらを開こう 343

15.1	軽量DDDに陥らないために	344
	COLUMN パターンの濫用とパターンを捨てる時	344
15.2	ドメインエキスパートとモデリングをする	345
15.2.1	本当に解決すべきものを見つけよう	347
15.2.2	ドメインとコードを結びつけるモデル	348
15.3	ユビキタス言語	349
15.3.1	深い洞察を得るために	352
15.3.2	ユビキタス言語がコードの表現として使われる	353
	COLUMN ユビキタス言語と日本語の問題	354
15.4	境界付けられたコンテキスト	355
15.5	コンテキストマップ	360
15.5.1	テストがチームの架け橋に	362
15.6	ボトムアップドメイン駆動設計	363
15.7	まとめ	364

Appendix 付録

ソリューション構成 365

A.1	ソフトウェア開発の最初の一步	366
	COLUMN C#特有のプロジェクト管理用語	366
A.1.1	ドメインレイヤーのパッケージ構成	367
A.1.2	アプリケーションレイヤーのパッケージ構成	369
A.1.3	インフラストラクチャレイヤーのパッケージ構成	370
A.2	ソリューション構成	370
A.2.1	すべてを別のプロジェクトにする	371
A.2.2	アプリケーションとドメインだけ 同じプロジェクトにする	372
A.2.3	言語機能が与える影響	373
A.3	まとめ	373

参考文献	374
------	-----

INDEX	374
-------	-----