ボトムアップで分かる！ドメイン駆動設計の基本 Chap.13

～複雑な条件を表現する「仕様」～

2022/03/10

本日のまとめ

- オブジェクトの評価はそれ単体で知識（ドメイン）になりうる
  - 仕様は評価の条件や手順をモデルにしたドメインオブジェクト
- オブジェクトの評価をオブジェクト自身にさせることが常に正しいとは限らない
  - 仕様のような外部のオブジェクトに評価させる手法が良いケースもある
- 特定の条件に合致したデータを取得したい場合は、仕様をリポジトリに引き渡してフィルタリングする
  - ドメインは維持されるが、パフォーマンス問題のリスクが付随する
- 読み取り操作においてはドメインという考え方を棚上げして、利用者の利便性を重視した形で実装する考えも必要

13.1 仕様とは

- 定義
  - あるオブジェクトがある評価基準に達しているかを判定するオブジェクト
- 背景
  - オブジェクトの評価が単純であれば、エンティティのメソッドとして定義してサービスから呼び出せばいい
  - しかし評価が複雑な場合はドメインのルールがぼやけたり、流出してしまう
- 解決策
  - 仕様という外部のオブジェクトに評価させることで、ドメインの意図を明確にし、ルールの流出を防ぐ

13.1.1 複雑な評価処理を確認する（簡単な例）

- 今まで扱ってきたサークルの評価メソッド

```
public class Circle
{
  ...

  public bool IsFull()
  {
    return CountMembers() >= 30;
  }
}
```

- これなら問題ないけど、もう少し複雑な場合はどうか

13.1.1 複雑な評価処理を確認する（複雑な例）

- サークルの新ルール（人数上限が所属しているユーザーのタイプによって変動）
  - ユーザーにはプレミアムユーザーと呼ばれるタイプが存在する
  - サークルに所属するユーザーの最大数はサークルのオーナーとなるユーザーを含めて 30 名まで
  - プレミアムユーザーが 10 名以上所属しているサークルはメンバーの最大数が 50 名に引き上げられる
- Circle は UserId のコレクションを保持しているだけなので、プレミアムユーザーの人数は知らない
  - ユーザーのリポジトリに問い合わせる必要がある

リポジトリを保持するアプリケーションサービス上で判定する

```
public class CircleApplicationService {
  private readonly ICircleRepository circleRepository;
  private readonly IUserRepository userRepository;

  ...

  public void Join(CircleJoinCommand command) {
    var circleId = new CircleId(command.CircleId);
    var circle = circleRepository.Find(circleId);

    // サークルに所属しているプレミアムユーザーの人数により上限が変わる
    var users = userRepository.Find(circle.Members);
    var premiumUserNumber = users.Count(user => user.IsPremium);
    var circleUpperLimit = premiumUserNumber < 10 ? 30 : 50;
    if (circle.CountMembers() >= circleUpperLimit) {
      throw new CircleFullException(circleId);
    }

    ...
  }
}
```

アプリケーションサービス上で判定した結果

- 本来サークルが満員かどうかの確認はドメインのルール
  - サービスにドメインのルールに基づくロジックを記述することは避けなければならない
  - 放置するとドメインオブジェクトは何も語らず、ドメインの重要なルールはサービスのあちこちで記述されるようになる

本来の思想に立ち戻ろうとする

- ドメインのルールはドメインオブジェクトに定義するべき
  - Circle クラスの IsFull メソッドとして定義することを考える
    - 今度は Circle クラスがユーザー情報として識別子しか保持していないことが問題となる
    - ユーザーの識別子からユーザー情報を取得するためには IsFull メソッドがリポジトリを受け取る必要がある

エンティティがリポジトリを受け取る例

```
public class Circle {
  // プレミアムユーザーの人数を探したいが保持しているのはUserIdのコレクションだけ
  public List<UserId> Members { get; private set; }

  ...

  // ユーザーのリポジトリを受け取る？
  public bool IsFull(IUserRepository userRepository) {
    var users = userRepository.Find(Members);
    var premiumUserNumber = users.Count(user => user.IsPremium);
    var circleUpperLimit = premiumUserNumber < 10 ? 30 : 50;
    return CountMembers() >= circleUpperLimit;
  }
}
```

エンティティがリポジトリを受けとった結果

- 良くない例となった
  - リポジトリはドメイン由来のものではないため、Circle はドメインモデルの表現に徹していない
    - エンティティや値オブジェクトがドメインモデルの表現に専念するためには、リポジトリを操作することを可能な限り避ける必要がある

13.1.2 「仕様」による解決

- エンティティや値オブジェクトにリポジトリを操作させないための解決策は「仕様」と呼ばれるオブジェクト
  - サークルが満員かどうかを評価する処理を仕様として切り出してみる

サークルが満員化どうかを評価する仕様

```
public class CircleFullSpecification {
  private readonly IUserRepository userRepository;

  public CircleFullSpecification(IUserRepository userRepository) {
    this.userRepository = userRepository;
  }

  public bool IsSatisfiedBy(Circle circle) {
    var users = userRepository.Find(circle.Members);
    var premiumUserNumber = users.Count(user => user.IsPremium);
    var circleUpperLimit = premiumUserNumber < 10 ? 30 : 50;
    return circle.CountMembers() >= circleUpperLimit;
  }
}
```

- 仕様はオブジェクトの評価のみを行う
  - 複雑な評価手順をオブジェクトから切り出すことで、その趣旨が明確になる
- 仕様を利用したときのサークルメンバー追加処理は次となる
  - 複雑な評価手順はカプセル化され、コードの意図は明確となっている

仕様を利用する

```
puclic class CircleApplicationService {
  private readonly ICircleRepository circleRepository;
  private readonly IUserRepository userRepository;
}

...

public void Join(CircleJoinCommand command) {
  var circleId = new CircleId(command.CircleId);
  var circle = circleRepository.Find(circleId);

  var circleFullSpecification = new CircleFullSpecification(userRepository);
  if (circleFullSpecification.IsSatisfiedBy(circle)) {
    throw new CircleFullException(circleId);
  }

  ...
}
```

趣旨が見えづらいオブジェクト

- オブジェクトの評価処理を安直にオブジェクト自身に実装すると、オブジェクトの趣旨はぼやける

```
public class Circle {
  public bool IsFull();
  public bool IsPopular();
  public bool IsAnniversary(DateTime today);
  public bool IsRecruiting();
  public bool IsLocked();
  public bool Join(User user);
}
```

- 仕様のように外部のオブジェクトとして切り出すことで扱いやすくなることもある

13.1.3 リポジトリの使用を避ける

- 仕様はれっきとしたドメインオブジェクトであり、その内部で入出力を行う（リポジトリを使用する）ことを避ける考えもある
  - その場合はファーストクラスコレクションを利用する選択肢がある
    - List といった汎用的な集合オブジェクトを利用するのではなく、特化した集合オブジェクトを用意するパターン
  - たとえばサークルのメンバー群を表現するファーストクラスコレクションは以下

余談）ファーストクラスコレクションとは

- 配列をも全てクラスを作ってラップする手法
  - コレクションもドメインオブジェクトと捉える
  - **List<Customer>** ではなく、独自に定義した **CustomerList クラス** として扱う
- メリット
  - 業務ルールの散在を防ぐ
    - 配列に対する業務ルール(全ての Product の Price を合計するなど)を 1 箇所にまとめられる
  - 内部状態の変更を防ぐ
    - 値オブジェクトと同様に内部の値を変更する場合は、配列を別オブジェクトとして返す
- 参考）ファーストクラスコレクション(コレクションクラス)とは何か

サークルに所属するメンバーを表すコレクションクラス

```
public class CircleMembers {
  private readonly User owener;
  private readonly List<User> membersl;

  public CircleMembers(Circle id, User owner, List<User> members) {
    Id = id;
    this.owner = owner;
    this.members = members;
  }

  public CircleId Id {get;}
  public int CountMembers() {
    return members.Count() + 1;
  }

  public int CountPremiumMembers(bool containsOwner = true) {
    var premiumUserNumber = members.Count(member => member.IsPremium);
    if (containsOwner) {
      return premiumUserNumber + (owner.IsPremium ? 1 : 0);
    } else {
      return premiumUserNumber;
    }
  }
}
```

- CircleMembers は汎用的な List とは異なり、サークルの識別子と所属するメンバーをすべて保持している
- また独自の計算処理をメソッドとして定義できる
- この CircleMembers を利用した仕様は以下となる

CircleMembers を利用した仕様

```
public class CircleMembersFullSpecification {
  public bool IsSatisfiedBy(CircleMembers members) {
    var premiumUserNumber = members.CountPremiumMembers(false);
    var circleUpperLimit = premiumUserNumber < 10 ? 30 : 50;
    return members.CountMembers() >= circleUpperLimit;
  }
}
```

- ファーストクラスコレクションを利用する場合はアプリケーションサービスでデータの詰替処理が必要となる

ファーストクラスコレクションに詰め替える

```
var owner = userRepository.Find(circle.Owner);
var members = userRepository.Find(circle.members);
var circleMembers = new CircleMembers(circle.Id, owner, members);
var circleFullSpec = new CircleMembersFullSpecification();
if (circleFullSpec.IsSatisfiedBy(circleMembers)) {
  ...
}
```

- 入出力をドメインオブジェクトから可能な限り排除することは重要
  - ファーストクラスコレクションを利用した解決法はその方針を支える手立てになる

13.2 仕様とリポジトリを組み合わせる

- 仕様は単独で取り扱う以外にもリポジトリと併用する手法が存在する
  - リポジトリに仕様を引き渡して、仕様に合致するオブジェクトを検索する
- 業務ルールを含む検索処理をリポジトリのメソッドとして定義してしまうと、業務ルールがドメイン由来のないリポジトリの実装クラスに記述されてしまう
  - 重要なルールを仕様オブジェクトとして定義しリポジトリに引き渡せば、ルールがリポジトリの実装クラスに漏れ出すことを防げる

13.2.1 お勧めサークルに見る複雑な検索処理

- お勧めサークル検索機能を開発することを考える
- 次の２つの条件にしたがったサークルをお勧めサークルとする
  - 直近１ヶ月以内に結成されたサークルである
  - 所属メンバー数が１０名以上である
- さて、どこにお勧めサークルにの検索処理を実装すべきか？

検索を担うリポジトリに定義してみる

- 引数に日付を渡すとお勧めサークルを提案してくれる FindRecommended メソッドを実装する

```
public interface ICircleRepository {

  ...

  public List<Circle> FindRecommended(DateTime now);
}
```

- アプリケーションサービスは FindRecommended メソッドを利用してユーザーにお勧めサークルを提案する

お勧めサークルを探し出すアプリケーションサービスの処理

```
public class CircleApplicationService {
  private readonly DateTime now;

  ...

  public CircleGetRecommendResult GetRecommend(CIrcleGetRecommendRequest request) {
    // リポジトリに依頼するだけ
    var recommendCircles = circleRepository.FindRecommended(now);

    return new CircleGetRecommendResult(recommendCircles);
  }
}
```

- 処理は正しく動作するが、お勧めサークルを導き出す条件がリポジトリの実装クラスに依存している
  - お勧めサークルの条件は重要なドメインルール
  - インフラストラクチャのオブジェクトであるリポジトリの実装クラスに左右されることは推奨されない

13.2.2 仕様による解決法

- お勧めサークルかどうかを判断する処理を仕様として定義する

```
public class CircleRecommendSpecification {
  private readonly DateTime executeDateTime;

  public CircleRecommendSpecification(DateTime executeDateTime) {
    this.executeDateTime = executeDateTime;
  }

  public bool IsSatisfiedBy(Circle circle) {
    if (circle.CountMembers() < 10) {
      return false;
    }
    return circle.Created > executeDateTime.AddMonths(-1);
  }
}
```

仕様を利用しお勧めサークルを検索する

```
public class CircleApplicationService {
  private readonly ICircleRepository circleRepository;
  private readonly DateTime now;

  ...

  public CircleGetRecommendResult GetRecommend(CircleGetRecommendRequest request) {
    var recommendCircleSpec = new CircleRecommendSpecification(now);

    var circles = circleRepository.FindAll();
    var recommendCircles = circles
      .Where(recommendCircleSpec.IsSatisfiedBy)
      .Take(10)
      .ToList();

    return new CircleGetRecommendResult(recommendCircles);
  }
}
```

- お勧めサークルの条件をリポジトリに記述する必要はなくなった

リポジトリに仕様を引き渡す場合

- 直接的に仕様のメソッドをスクリプト上で呼び出す以外にも、リポジトリに仕様を引き渡してメソッドを呼び出させることにより、対象となるオブジェクトを抽出させる手法もある
- この手法を採用する場合は、仕様のインターフェースを用意する

```
public interface ISpecification<T> {
  public bool IsSatisfiedBy(T value);
}

public class CircleRecommendSpecification : ISpecification<Circle> {
  ...
}
```

- リポジトリはこのインターフェースを受け取り、結果となるセットを返却する

```
public interface ICircleRepository {
  ...

  public List<Circle> Find(ISpecification<Circle> specification);
}
```

- 仕様をインターフェースにすることで、リポジトリには仕様ごとにメソッドを追加定義する必要がなくなる
  - ISpecification<Circle> を実装した新たな仕様を定義すれば、ICircleRepository に引き渡しての検索が可能

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

- 仕様をリポジトリに引き渡す手法はルールをオブジェクトに表現しつつ、拡張性を高める有効な手段だが、パフォーマンスにデメリットがある
  - DB からテーブルのデータを全件取得している
  - インスタンスを生成してひとつひとつ仕様に合致するか検査している

13.2.4 複雑なクエリは「リードモデル」で

- 特定の条件に合致したオブジェクトのみを検索したいという要求はあるし、パフォーマンスに関する要求も高くなりがち
- この場合は仕様やリポジトリにといったパターンを扱わないことも視野に入る

サークル一覧を取得する処理（問題がある例）

```
public class CircleApplicationService {
  public CIrcleGetSummariesResult GetSummaries(CircleGetSummariesCommand command) {
    // 全件取得して
    var all = circleRepository.FindAll();
    // その後にページング
    var circles = all
      .Skip((command.Page - 1) * command.Size)
      .Take(command.Size);

    var summaries = new List<CircleSummaryDate>();
    foreach(var circle in circles) {
      // サークルのオーナーを改めて検索
      var owner = userRepository.Find(circle.Owner);
      summaries.Add(new CircleSummaryData(circle.Id.Value, owner.Name.Value));
    }
    return new CircleGetSummariesResult(summaries);
  }
  ...
}
```

サークルのサマリー一覧を取得する処理の問題

- サークル集約を全件取得している
  - 不要なインスタンスは捨てられる
- サークルに所属するユーザーの検索処理が繰り返しになっている
  - 本来であれば JOIN 句などで 1 回で済むクエリが大量発行される
- ドメインレイヤーにあるべき知識の流出を防ぐにはこのコードが正解であるはずだが、それを理由にシステム利用者の利便性を無視していいのか？

筆者の考え

”そもそもシステムは何のために存在するのでしょう。
それは間違いなく利用者の問題を解決するためです。
システムはその利用者に対して友好的である必要があります。
もしもシステムの利用者に対する態度が友好的でなくなれば、そのシステムはやがて使われなくなっていくでしょう。
ドメインの防衛を理由に、利用者に対して不便を強いるのは正しい道ではありません。
ドメインの表現を守り、領域を保護することは大切なことですが、アプリケーションの領域はプレゼンテーション（ひいてはシステムの利用者）を強く意識する必要があります。
複雑な読み取り動作においては局所的にドメインオブジェクトから離れることもあります。”

最適化のために直接クエリを実行する

```
public class CircleQueryService {
  ...

  public CircleGetSummariesResult GetSummaries(CircleGetSummariesCommand command) {
    var connection = provider.Connection;
    using (var sqlCommand = connection.CreateCommand()) {
      /* circleId と ownerName の取得に最適化したクエリの組み立て */
      /* クエリのパラメータに値を代入 */
      /* クエリ実行 */
      var summaries = new List<CircleSummaryData>();
      while (reader.Read()) {
        /* クエリ結果から circleId と ownerName を取得 */
        var summary = new CircleSummaryData(circleId, ownerName);
        summaries.Add(summary);
      }
      return new CircleGetSummariesResult(summaries);
    }
  }
}
```

上記コードのクエリの組み立て箇所

```
sqlCommand.CommandText = @"
        SELECT
          circles.id as circleId,
          users.name as ownerName
        FROM circles
        LEFT OUTER JOIN users
        ON circles.ownerId = users.id
        ORDER BY circles.id
        OFFSET @skip ROWS
        FETCH NEXT @size ROWS ONLY
      ";
```

読み取りと書き込みの性質に応じた最適化

- 読み取り（クエリ）で要求されるデータは複雑だが、その動作自体は単純でロジックと呼べるものはほとんどない
- 反対に書き込み（コマンド）はドメインとしての成約が多く存在する
- このことから、コマンドにおいてはドメインを隔離するためにドメインオブジェクトなどを積極的に利用し、クエリにおいてはある程度その制約を緩和することがある
  - この考えは CQRS (Command Query Responsibility Segregation) という考えに基づくもの
    - FYI）CQRS 実践入門 [ドメイン駆動設計]

13.3 まとめ

- オブジェクトの評価はそれ単体で知識（ドメイン）になりうる
  - 仕様は評価の条件や手順をモデルにしたドメインオブジェクト
- オブジェクトの評価をオブジェクト自身にさせることが常に正しいとは限らない
  - 仕様のような外部のオブジェクトに評価させる手法が良いケースもある
- 特定の条件に合致したデータを取得したい場合は、仕様をリポジトリに引き渡してフィルタリングする
  - ドメインは維持されるが、パフォーマンス問題のリスクが付随する
- 読み取り操作においてはドメインという考え方を棚上げして、利用者の利便性を重視した形で実装する考えも必要

余談）思ったこと

- DynamoDB などではユースケースからテーブル設計をしたり、コストやレイテンシーを中心に考慮することが多いので、クエリ側に仕様を反映した設計になることが多いと感じる
- ランキング画面など、画面に対応したテーブルを用意し、事前にデータを処理してから保存することもある
- RDB の場合は柔軟なクエリが発行できるので、そこまで気にしなくても開発をスタートできるのかも
- 何が最適な解なのか？
  - とりあえずはドメインをきれいに保ちつつ、コストやパフォーマンス問題にぶつかったらリファクタリング？
  - みなさんの意見を伺いたいです