

# 全問正解率 3%: RubyKaigiで出題したやりがちな 危険コード5選

Kaigi on Rails 2025, 2025.09.26 (Fri.)@JP TOWER Hall & Conference Hall Red

Hubble, Inc Backend Tech Lead

Yuta Nakashima



# Partner with RubyStackNews<sup>I</sup>

Independent Ruby & Rails publication for senior developers

## Why RubyStackNews?

- Focused on Ruby and Ruby on Rails
- Long-form articles based on real conference talks
- Audience of senior developers and tech leads
- Readers from the US, Europe, and Asia

RubyStackNews turns conference talks and real-world experience into practical, production-focused technical articles.

## Partnerships & Sponsorships

- Article sponsorships
- Inline placements inside articles
- Sidebar visibility

[View partnership details](#)

[partner-with-rubystacknews](#)

## 自己紹介

---



Yuta Nakashima

Hubble, Inc Backend Tech Lead

エンジニア歴6年、Rails歴4年半

HubbleにCTO以来の一人目バックエンドエンジニアとして入

趣味: メタルラウド系のライブでよくモッシュしてます

AI契約業務・管理クラウド



契約書の作成、社内のやりとり、  
検討過程や合意文書を一元管  
理。



# 契約業務の オールインワンプラットフォーム

発生・起案

審査・交渉

押印申請・捺印

保管・管理

更新

**Hubble**

あなたの案件・相談

- ① 進行中・相談中
- ② 未着手
- ③ 完了

全て

- 契約審査
- 法律相談

あなたの案件・相談

タイトル	ステータス	期日	法務担当	依頼部署
株式会社秋葉原テック商事との取引の件	未着手			営業第一部
取引基本契約書	未着手			営業第一部
個別契約書	未着手			営業第一部
新規卸売業者との取引の件	完了	2024/08/20	上村 雄太	営業第一部
株式会社品川テクノロジーとのOEMの件	進行中	2024/09/27	上村 雄太	営業第一部
研究所増設計画に関する法務相談	相談中	2024/10/20	上村 雄太	総務部
中国における新規事業立ち上げPJ	未着手	2024/11/30	山下 俊	新規事業部
下議法改正対応関連	完了	2024/08/30	山下 俊	経営企画部

売買基本契約書 株式会社品川テクノロジー\_20240201

1 契約成立

2 契約成立

3 契約成立

4 契約成立

5 契約成立

6 契約成立

7 契約成立

8 契約成立

9 契約成立

10 契約成立

11 契約成立

12 契約成立

13 契約成立

14 契約成立

15 契約成立

16 契約成立

17 契約成立

18 契約成立

19 契約成立

20 契約成立

21 契約成立

22 契約成立

23 契約成立

24 契約成立

25 契約成立

26 契約成立

27 契約成立

28 契約成立

29 契約成立

30 契約成立

31 契約成立

32 契約成立

33 契約成立

34 契約成立

35 契約成立

36 契約成立

37 契約成立

38 契約成立

39 契約成立

40 契約成立

41 契約成立

42 契約成立

43 契約成立

44 契約成立

45 契約成立

46 契約成立

47 契約成立

48 契約成立

49 契約成立

50 契約成立

51 契約成立

52 契約成立

53 契約成立

54 契約成立

55 契約成立

56 契約成立

57 契約成立

58 契約成立

59 契約成立

60 契約成立

61 契約成立

62 契約成立

63 契約成立

64 契約成立

65 契約成立

66 契約成立

67 契約成立

68 契約成立

69 契約成立

70 契約成立

71 契約成立

72 契約成立

73 契約成立

74 契約成立

75 契約成立

76 契約成立

77 契約成立

78 契約成立

79 契約成立

80 契約成立

81 契約成立

82 契約成立

83 契約成立

84 契約成立

85 契約成立

86 契約成立

87 契約成立

88 契約成立

89 契約成立

90 契約成立

91 契約成立

92 契約成立

93 契約成立

94 契約成立

95 契約成立

96 契約成立

97 契約成立

98 契約成立

99 契約成立

100 契約成立

表示 お気に入り 更新・解約履歴 CSVダウンロード

一週間以内に更新されたドキュメント 一週間以内に受け付けたドキュメント 未締結のドキュメント 今月通知期限をむかえる契約

ドキュメント	ドキュメント名	契約相手方	契約開始日	契約終了日	更新/解約通知日	契約状況	契約更新状況
1	売買基本契約書	株式会社品川テクノロジー	2024年04月01日	2025年06月30日	2025年03月30日	契約中	
2	業務委託契約書	渋谷コンサルティング	2024年01月01日	2025年06月30日	2025年05月30日	契約中	
3	共同研究開発契約書	株式会社セントラル	2023年06月01日	2024年08月31日	2024年07月31日	終了	
4	共同研究開発契約書	株式会社セントラル	2023年06月01日	2024年08月31日	2024年07月31日	終了	
5	業務委託契約書	青研株式会社	2024年03月01日	2025年03月31日	2025年02月28日	契約中	判断待ち
6	業務委託契約書	青研株式会社	2024年03月01日	2025年03月31日	2025年02月28日	契約中	判断待ち
7	秘密保持契約書	株式会社大田重信機械工業	2024年03月01日	2025年03月31日	2025年02月28日	契約中	終了予定
8	業務委託契約書	池上自研製薬所	2024年02月01日	2025年01月31日	2024年12月31日	終了	
9	業務委託契約書	株式会社ZOTEC	2024年03月01日	2025年10月31日	2025年09月30日	契約中	
10	秘密保持契約書	ハムス株式会社	2024年02月01日	2025年03月31日	2025年02月28日	契約中	更新予定
11	秘密保持契約書	プライムクロイツ合同会社	2024年03月01日	2025年03月31日	2025年02月28日	契約中	判断待ち
12	秘密保持契約書	宇瀬物産株式会社	2024年01月01日	2025年12月31日	2025年11月30日	契約中	
13	秘密保持契約書	キノマタ合同会社	2024年01月01日	2025年12月31日	2025年11月30日	契約中	



# 前提



RubyKaigi 2025

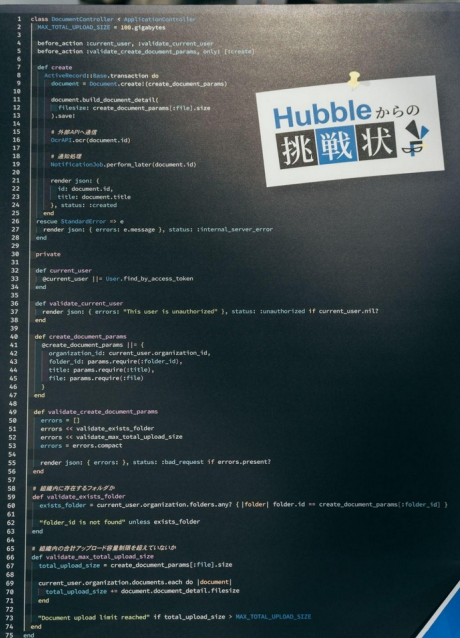
初参加

初スポンサー

初ブース出展

# 前提

ブースにてコード問題を出題  
全5問で正解数に応じて豪華景品を配布



```
1 class DocumentController < ApplicationController
2   include ActionController::MimeResponds
3   before_action :current_user, :validate_current_user
4   before_action :validate_create_document_params, unless: [:create]
5
6   def create
7     respond_to do |format|
8       format.html {
9         document = Document.create(create_document_params)
10        document.build_document_detail(
11          filename: create_document_params[:file].size
12        ).save!
13      }
14      # 外部API連携
15      format.json { render json: document, id: }
16    end
17    # 通知送信
18    NotificationJob.perform_later(document.id)
19  end
20
21  def show
22    render json: {
23      id: document.id,
24      title: document.title,
25    }, status: :created
26  end
27  rescue StandardError => e
28    render json: { errors: e.message }, status: :internal_server_error
29  end
30
31  private
32
33  def current_user
34    @current_user ||= User.find_by_access_token
35  end
36
37  def validate_current_user
38    render json: { errors: "This user is unauthorized" }, status: :unauthorized if current_user.nil?
39  end
40
41  def create_document_params
42    create_document_params = {
43      organization_id: current_user.organization_id,
44      folder_id: params.require(:folder_id),
45      title: params.require(:title),
46      file: params.require(:file)
47    }
48  end
49
50  def validate_create_document_params
51    errors = []
52    errors << validate_exists_folder
53    errors << validate_max_total_upload_size
54    errors << errors.compact
55    render json: { errors: }, status: :bad_request if errors.present?
56  end
57
58  # 組織内に存在するフォルダか
59  def validate_exists_folder
60    exists_folder = current_user.organization.folders.any? { |folder| folder.id == create_document_params[:folder_id] }
61    "folder_id is not found" unless exists_folder
62  end
63
64  # 組織内の合計アップロード容量制限を越えていないか
65  def validate_max_total_upload_size
66    total_upload_size = create_document_params[:file].size
67    current_user.organization.documents.each do |document|
68      total_upload_size += document.document_detail.filesize
69    end
70    "document upload limit reached" if total_upload_size > MAX_TOTAL_UPLOAD_SIZE
71  end
72
73  end
74 end
```

## 前提

---

結果、3日間で100名程度の方が参加

5問全問正解が **3名**

3問以上正解が**約 15名**

回答しづらい環境ではありましたが、想像以上に

難問だった可能性...🤔



# 前提

---

もしかして **Kaigi on Rails** で内容・解答を話せば、  
役立つのでは...?

→ **この発表**になります!

# 本題

---

今回話すことは所謂、**教科書的な話**

- Transaction の範囲
- 非同期処理発火のタイミング
- エラーハンドリング
- SQL 側の責務とRuby側の責務の切り分け、テーブル設計

実際の**プロダクトコード**になると気づかずに書いてしまっていることや、レビューでも気づかないことが往々にしてある

# 出題形式

---

## 問題文 + 問題のあるコード

### (問題文)

- バグ要因、パフォーマンス、セキュリティ観点で何箇所か直したほうが良いところがあるので、どう修正すべき(カogeをtransaction外に出す等)を個数と行数を含めて箇条書きで回答してください。
- 問題の都合上、ModelバリデーションではなくController側でバリデーションしていること、Controllerにビジネスロジックを書いていること、外部APIコールのレスポンス制御部分は対象外です。

```

class DocumentsController < ApplicationController
  MAX_TOTAL_UPLOAD_SIZE = 100.gigabytes

  before_action :current_user, :validate_current_user
  before_action :validate_create_document_params, only: [:create]

  def create
    ActiveRecord::Base.transaction do
      document = Document.create!(create_document_params)

      document.build_document_detail(
        filesize: create_document_params[:file].size
      ).save!

      # 外部APIへ通信
      OcrAPI.ocr(document.id)

      # 通知処理
      NotificationJob.perform_later(document.id)

      render json: {
        id: document.id,
        title: document.title
      }, status: :created
    end
  rescue StandardError => e
    render json: { errors: e.message }, status: :internal_server_error
  end

  private

  def current_user
    @current_user ||= User.find_by_access_token
  end

  def validate_current_user
    render json: { errors: "This user is unauthorized" }, status: :unauthorized if current_user.nil?
  end

  def create_document_params
    @create_document_params ||= {
      organization_id: current_user.organization_id,
      folder_id: params.require(:folder_id),
      title: params.require(:title),
      file: params.require(:file)
    }
  end
end

```

```

  def validate_create_document_params
    errors = []
    errors << validate_exists_folder
    errors << validate_max_total_upload_size
    errors = errors.compact

    render json: { errors: }, status: :bad_request if errors.present?
  end

  # 組織内に存在するフォルダか
  def validate_exists_folder
    exists_folder = current_user.organization.folders.any? { |folder| folder.id == create_document_params[:folder_id] }

    "folder_id is not found" unless exists_folder
  end

  # 組織内の合計アップロード容量制限を超えていないか
  def validate_max_total_upload_size
    total_upload_size = create_document_params[:file].size

    current_user.organization.documents.each do |document|
      total_upload_size += document.document_detail.filesize
    end

    "Document upload limit reached" if total_upload_size > MAX_TOTAL_UPLOAD_SIZE
  end
end

```

# Hubbleを模した

# ドキュメントアップロードAPIが題

# 材



```
1 class DocumentsController < ApplicationController
2   MAX_TOTAL_UPLOAD_SIZE = 100.gigabytes
3
4   before_action :current_user, :validate_current_user
5   before_action :validate_create_document_params, only: [:create]
6
7   def create
8     ActiveRecord::Base.transaction do
9       document = Document.create!(create_document_params)
10
11       document.build_document_detail(
12         filesize: create_document_params[:file].size
13       ).save!
14
15       # 外部APIへ通信
16       OcrAPI.ocr(document.id)
17
18       # 通知処理
19       NotificationJob.perform_later(document.id)
20
21       render json: {
22         id: document.id,
23         title: document.title
24       }, status: :created
25     end
26   rescue StandardError => e
27     render json: { errors: e.message }, status: :internal_server_error
28   end
```

```
1 class DocumentsController < ApplicationController
2   MAX_TOTAL_UPLOAD_SIZE = 100.gigabytes
3
4   before_action :current_user, :validate_current_user
5   before_action :validate_create_document_params, only: [:create]
6
7   def create
8     ActiveRecord::Base.transaction do
9       document = Document.create!(create_document_params)
10
11       document.build_document_detail(
12         filesize: create_document_params[:file].size
13       ).save!
14
15       # 外部APIへ通信
16       OcrAPI.ocr(document.id)
17
18       # 通知処理
19       NotificationJob.perform_later(document.id)
20
21       render json: {
22         id: document.id,
23         title: document.title
24       }, status: :created
25     end
26   rescue StandardError => e
27     render json: { errors: e.message }, status: :internal_server_error
28   end
```

```
30 private
31
32 def current_user
33   @current_user ||= User.find_by_access_token
34 end
35
36 def validate_current_user
37   render json: { errors: "This user is unauthorized" }, status: :unauthorized if current_user.nil?
38 end
```

- アクセストークンからユーザを取得
- ユーザが取得できなければ、401エラー

問題なさそう!

```
1 class DocumentsController < ApplicationController
2   MAX_TOTAL_UPLOAD_SIZE = 100.gigabytes
3
4   before_action :current_user, :validate_current_user
5   before_action :validate_create_document_params, only: [:create]
6
7   def create
8     ActiveRecord::Base.transaction do
9       document = Document.create!(create_document_params)
10
11       document.build_document_detail(
12         filesize: create_document_params[:file].size
13       ).save!
14
15       # 外部APIへ通信
16       OcrAPI.ocr(document.id)
17
18       # 通知処理
19       NotificationJob.perform_later(document.id)
20
21       render json: {
22         id: document.id,
23         title: document.title
24       }, status: :created
25     end
26   rescue StandardError => e
27     render json: { errors: e.message }, status: :internal_server_error
28   end
```



```
49 def validate_create_document_params
50   errors = []
51   errors << validate_exists_folder
52   errors << validate_max_total_upload_size
53   errors = errors.compact
54
55   render json: { errors: }, status: :bad_request if errors.present?
56 end
```

- バリデーションエラーの配列を作成し、エラーを格納
- 空配列でなければ、400エラー

問題なさそう!

```
49 def validate_create_document_params
50   errors = []
51   errors << validate_exists_folder
52   errors << validate_max_total_upload_size
53   errors = errors.compact
54
55   render json: { errors: }, status: :bad_request if errors.present?
56 end
```

```
58 # 組織内に存在するフォルダか
59 def validate_exists_folder
60   exists_folder = current_user.organization.folders.any? { |folder| folder.id == create_document_params[:folder_id] }
61
62   "folder_id is not found" unless exists_folder
63 end
```

- ユーザの組織からフォルダを全て取ってきて `folder_id` と一致するかを `any?` で評価



## 問題点1: SQLで全件取得

```
58 # 組織内に存在するフォルダか
59 def validate_exists_folder
60   exists_folder = current_user.organization.folders.any? { |folder| folder.id == create_document_params[:folder_id] }
61
62   "folder_id is not found" unless exists_folder
63 end
```

- **大量のオブジェクト生成** N個のフォルダがある場合、N個の ActiveRecord オブジェクトがメモリ上に生成される
- **メモリリーク** 不要なデータが大量にメモリに保持されGCの負荷が増大
- **ネットワーク帯域の無駄遣い** DBサーバとAPIサーバ間の不要な通信量増大 (**コネクションプール** を占有してAPIにも影響する可能性)
- **DBサーバのCPU負荷**: 不要なデータ(id以外のカラム)をシリアルライズして送信する処理コスト
- **レスポンス時間の劣化**: フォルダ数に比例してレスポンスが遅くな(最悪計算量  $O(N)$ )
- **スケーラビリティの欠如** データ量増加に対して線形的にパフォーマンスが悪化エンタープライズだと、数万数十万は全然有り得る話)



## 解決方法 1: exists?でSQLで存在確認

```
58 # 組織内に存在するフォルダか
59 def validate_exists_folder
60   exists_folder = current_user.organization.folders.exists?(id: create_document_params[:folder_id])
61
62   "folder_id is not found" unless exists_folder
63 end
```

- **ActiveRecordからBooleanに:** exists?の返り値がTrueClass or FalseClassの1オブジェクトだけになるので大幅削減
- **ネットワーク帯域を圧迫しない** DBサーバとAPIサーバ間の不要な通信量が最小
- **DBサーバのCPU負荷が最小:** 不要なカラムのデータを取得しない
  - 発行クエリ
  - 今回だと約 `SELECT 1 AS one FROM folders WHERE folders.id = 1 LIMIT 1`
- **レスポンス時間とスケーラビリティ** フォルダ数に関係なくインデックス(今回であればidでprimary key)があれば  $O(\log N)$

```
def validate_create_document_params
  errors = []
  errors << validate_exists_folder
  errors << validate_max_total_upload_size
  errors = errors.compact

  render json: { errors: }, status: :bad_request if errors.present?
end
```

```
65 # 組織内の合計アップロード容量制限を超えていないか
66 def validate_max_total_upload_size
67   total_upload_size = create_document_params[:file].size
68
69   current_user.organization.documents.each do |document|
70     total_upload_size += document.document_detail.filesize
71   end
72
73   "Document upload limit reached" if total_upload_size > MAX_TOTAL_UPLOAD_SIZE
74 end
```

- ユーザの組織からドキュメントを全て取ってきてる
- ドキュメントのリレーション先のドキュメント詳細を取得してる



## 問題点2: SQLクエリではなくRuby側で計算

```
65 # 組織内の合計アップロード容量制限を超えていないか
66 def validate_max_total_upload_size
67   total_upload_size = create_document_params[:file].size
68
69   current_user.organization.documents.each do |document|
70     total_upload_size += document.document_detail.filesize
71   end
72
73   "Document upload limit reached" if total_upload_size > MAX_TOTAL_UPLOAD_SIZE
74 end
```

- **1 + N回のクエリ実行** 1,000件のドキュメントがあれば1001回のクエリが発行される（初回のdocuments取得 + 各documentごとのdocument\_detail取得）
- **コネクションプール枯渇** 大量の小さなクエリでデータベース接続が長時間占有され、他のリクエストが接続待ちになる可能性 (Railsは1リクエスト1コネクション)
- **DBサーバCPU負荷:** 同じような小さなクエリを大量に処理する非効率性と **クエリパースング** の繰り返し
  - クエリパースング(字句解析)はSQL実行のおよそ40%を占める
  - 同一クエリでもパラメータが違えば別クエリとして扱われる
  - キャッシュが効かずパースングが繰り返される
- **ラウンドトリップ回数増大** アプリケーションサーバとDBサーバ間の通信がN回発生し、ネットワークレイテンシーに累積(一般に同一リージョン通信は1.5ms - 2.0ms、クエリ実行自体は0.1ms - 2.0ms)



## 解決方法2: sumでSQLで計算

```
65 # 組織内の合計アップロード容量制限を超えていないか
66 def validate_max_total_upload_size
67   total_upload_size = create_document_params[:file].size
68
69   total_upload_size += current_user.organization.documents
70   .joins(:document_detail)
71   .sum("document_details.filesize")
72
73   "Document upload limit reached" if total_upload_size > MAX_TOTAL_UPLOAD_SIZE
74 end
```

- **単一クエリ実行** 1,001回 → 1回のクエリで処理完了
- **SQL集約関数活用**: データベースエンジンの最適化されたSUM処理を利用
- **サーバCPU負荷軽減**: Ruby側での繰り返し処理が不要
- **ネットワーク通信最小化** 1回の通信で処理完了

(別解) organizationテーブルにtotal\_filesizeカラムを持つ

- **ドキュメントとの分離** ドキュメント数が増えても影響なし、計算量organizationの $O(\log N)$

(別解) KVSキャッシュ + 更新時検証

- **EC在庫管理等** によくあるパターン

```
7 def create
8   ActiveRecord::Base.transaction do
9     document = Document.create!(create_document_params)
10
11     document.build_document_detail(
12       filesize: create_document_params[:file].size
13     ).save!
14
15     # 外部APIへ通信
16     OcrAPI.ocr(document.id)
17
18     # 通知処理
19     NotificationJob.perform_later(document.id)
20
21     render json: {
22       id: document.id,
23       title: document.title
24     }, status: :created
25   end
26 rescue StandardError => e
27   render json: { errors: e.message }, status: :internal_server_error
28 end
```

```
7 def create
8   ActiveRecord::Base.transaction do
9     document = Document.create!(create_document_params)
10
11     document.build_document_detail(
12       filesize: create_document_params[:file].size
13     ).save!
14
15     # 外部APIへ通信
16     OcrAPI.ocr(document.id)
17
18     # 通知処理
19     NotificationJob.perform_later(document.id)
20
21     render json: {
22       id: document.id,
23       title: document.title
24     }, status: :created
25   end
26   rescue StandardError => e
27     render json: { errors: e.message }, status: :internal_server_error
28   end
29
```

## 問題点3: トランザクション内での外部API実行

```
7 def create
8   ActiveRecord::Base.transaction do
9     # 外部APIへ通信
10    OcrAPI.ocr(document.id)
11  end
12  rescue StandardError => e
13    render json: { errors: e.message }, status: :internal_server_error
14  end
```

- **参照ロック長時間保持** 外部キー制約により参照先テーブル (organization) の行が外部APIのresponseが終わるまで共有ロック状態
- **排他ロック待ち発生** 同一Organization行の更新・削除操作が全てブロックされる
- **問題切り分けが難化** ロジックが問題なのか、外部API障害なのかの切り分けが難しくなる

## 解決方法3: 外部API実行をトランザクション外に

```
7  def create
8    document = ActiveRecord::Base.transaction do
9      Document.create!(create_document_params)
10    end
11    # 外部APIへ通信
12    OcrAPI.ocr(document.id)
13  rescue StandardError => e
14    render json: { errors: e.message }, status: :internal_server_error
15  end
```

- 通常のTransactionと同じロック長時間参照先テーブルをロックしない
- 問題切り分けが容易: ロジックが問題なのか、外部API障害なのかの切り分けがわかりやすい



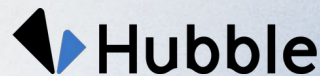
```
7 def create
8   ActiveRecord::Base.transaction do
9     document = Document.create!(create_document_params)
10
11     document.build_document_detail(
12       filesize: create_document_params[:file].size
13     ).save!
14
15     # 外部APIへ通信
16     OcrAPI.ocr(document.id)
17
18     # 通知処理
19     NotificationJob.perform_later(document.id)
20
21     render json: {
22       id: document.id,
23       title: document.title
24     }, status: :created
25   end
26 rescue StandardError => e
27   render json: { errors: e.message }, status: :internal_server_error
28 end
```

## 問題点4: トランザクション内での非同期処理発火

```
7 def create
8   ActiveRecord::Base.transaction do
9     # 通知処理
10    NotificationJob.perform_later(document.id)
11  end
12  rescue StandardError => e
13    render json: { errors: e.message }, status: :internal_server_error
14  end
```

- **Race Condition発生:** Job実行タイミングによってはコミット前のデータにアクセス  
例外発生
- **ロールバック時の不整合** トランザクションロールバック後も非同期ジョブがキューに  
存在して不要実行
- **不整合データ生成:** ロールバックしているのに他モデルのデータを作ってしまう可能性
- **エラー状態の不明確性:** トランザクション失敗とジョブ失敗の区別が困難

## 解決方法 4: 非同期処理発火をトランザクション外に



```
7 def create
8   document = ActiveRecord::Base.transaction do
9     Document.create!(create_document_params)
10  end
11  # 通知処理
12  NotificationJob.perform_later(document.id)
13 rescue StandardError => e
14   render json: { errors: e.message }, status: :internal_server_error
15 end
```

- **DBの整合性:** 未コミットデータアクセス問題の解消
- **システム安定性の改善:** Race Conditionやデッドロックの回避
- **エラー処理の明確化:** 問題の切り分けとデバッグ効率化
- **開発・保守性向上** テスト容易性とコードの責務分離

### 3.8 ジョブのトランザクション整合性

Solid Queueは、デフォルトではメインアプリケーションとは別のデータベースを利用します。これにより、トランザクションの整合性に関する問題が回避され、トランザクションがコミットされた場合にのみジョブがエンキューされるようになります。

ただし、Solid Queueをアプリと同一のデータベースで利用する場合は、Active Jobの `enqueue_after_transaction_commit` オプションでトランザクションの整合性を有効にできます。このオプションは、ジョブごとに有効にすることも、以下のように `ApplicationJob` ですべてのジョブに対して有効にすることも可能です。

```
class ApplicationJob < ActiveJob::Base
  self.enqueue_after_transaction_commit = true
end
```

また、Solid Queueジョブ用のデータベースコネクションを別途設定することで、トランザクションの整合性の問題を回避しながら、アプリと同一のデータベースを利用するようにSolid Queueを構成することも可能です。

14:10 ~ 14:25



Hall Red

非同期jobをtransaction内で呼ぶなよ！絶対に呼ぶなよ！

Yuto Urushima 

より詳細なことは明日の4:10からの発表で！



```
7 def create
8   ActiveRecord::Base.transaction do
9     document = Document.create!(create_document_params)
10
11     document.build_document_detail(
12       filesize: create_document_params[:file].size
13     ).save!
14
15     # 外部APIへ通信
16     OcrAPI.ocr(document.id)
17
18     # 通知処理
19     NotificationJob.perform_later(document.id)
20
21     render json: {
22       id: document.id,
23       title: document.title
24     }, status: :created
25   end
26 rescue StandardError => e
27   render json: { errors: e.message }, status: :internal_server_error
28 end
29
```

```
7 def create
8   ActiveRecord::Base.transaction do
9     document = Document.create!(create_document_params)
10
11     document.build_document_detail(
12       filesize: create_document_params[:file].size
13     ).save!
14
15     # 外部APIへ通信
16     OcrAPI.ocr(document.id)
17
18     # 通知処理
19     NotificationJob.perform_later(document.id)
20
21     render json: {
22       id: document.id,
23       title: document.title
24     }, status: :created
25   end
26   rescue StandardError => e
27     render json: { errors: e.message }, status: :internal_server_error
28   end
29
```

## 問題点5: StandardErrorで全例外を処理

```
26 rescue StandardError => e
27   render json: { errors: e.message }, status: :internal_server_error
28 end
29
```

- **内部実装詳細の漏洩** : 内部エラー時にDB構造、テーブル名、カラム名、ファイルパス、環境変数等が露出
- **SQLインジェクション** : DBエラーで内部クエリ構造が露出
- **HTTPステータスコードの固定化** 全てのエラーが500番になり適切でない(この場合だとバリデーションエラー500になる)
- **ユーザビリティ低下** 技術的なエラーメッセージでユーザーが混乱(ユーザーが次に何をすべきかわからない)
- **テスト品質低下** エラー条件のテストが不十分になる

## 解決方法5: エラー分岐、ステータスコード整理

```
26 < rescue ActiveRecord::RecordInvalid => e
27   render json: { errors: e.record.errors.full_messages }, status: :bad_request
28 < rescue OcrAPI::TimeoutError => e
29   Rails.logger.error("OCR timeout: #{e.message} backtrace: #{e.backtrace}")
30 <   render json: {
31     error: "外部サービスとの処理がタイムアウトしました。しばらく時間をおいて再試行してください"
32   }, status: :gateway_timeout
33 < rescue OcrAPI::InvalidResponse, OcrAPI::ServiceError => e
34   Rails.logger.error("OCR service error: #{e.message} backtrace: #{e.backtrace}")
35 <   render json: {
36     error: "外部サービスとの通信でエラーが発生しました"
37   }, status: :bad_gateway
38 < rescue StandardError => e
39   Rails.logger.error("Unexpected error: #{e.message} backtrace: #{e.backtrace}")
40 <   render json: {
41     error: "予期しないエラーが発生しました"
42   }, status: :internal_server_error
43 end
```

- 内部情報漏洩防止: 技術的詳細を隠蔽し、攻撃者に有用な情報を提供しない
- クライアント側エラー判別しやすいバリデーションエラー(400) と外部サービスエラー(500/504) の明確な区別
- 具体的なエラーケース検証 各例外 (TimeoutError、ServiceError等) を個別にテスト可能

# まとめ

- 意外とちゃんと **意識**しないといけないポイントが多い
  - Transactionの範囲
  - 非同期処理発火タイミング
  - エラーハンドリング
  - SQL側の責務とRuby側の責務の切り分け、テーブル設計
- 知識的に知ってても **プロダクトコード** になると**意識**せずに書き
- 特に最近のAIによる**Vibe Coding**では**意識**していききたいところ



## 最後に

---

Hubbleでは様々な**技術的課題** (特にパフォーマンスチューニング)に  
取り組んでいます!

ドライブシステムの複雑な親子関係の**Ruby**計算をSQLに置き換え  
アプリケーションの権限管理を**QL**でのBit演算で計算  
AIエージェントの他機能への横展開and more...

そういったことに興味のある方はぜひ **ブース**まで!