TRAINING

下流工程担当向け

Claude Code 活用研修

Day3 壊れたときに効く動き

2026年9月17日（木）

株式会社インフォメーション・ディベロプメント 御中

Givery

本資料の二次転載の禁止

本資料の著作権は株式会社ギブリーに帰属します。受講者ご本人の
学習目的に限り、社内での閲覧と保管を認めます。

第三者への配布、SNSや社外サイトへの掲載、複製物の販売など、
研修の範囲を超える二次利用はお断りします。

演習で使うソースコードとログはすべて研修向けに作成したもので、
実在する取引や個人の情報は含みません。

この資料は 2026年9月17日 の開催に合わせて作成しています。

講師紹介

人的資本インテリジェンス部門講師・
生成AIエバンジェリスト

安田 光喜

Mitsuki Yasuda

プロフィール

公立はこだて未来大学 大学院（メディアデザイン領域）を卒業し、街づくり会社での複合商業ビルの立ち上げにおける情報インフラ整備やデザイン全般業務の責任として関与。その後デザイン事務所を立ち上げ、飲食店を中心としたデザイン業務や美術館展示のアートコンテンツ開発などを行う。また、地域ブランディングにも注力し、市主催の企画運営業務を行なった。2018年10月『小中学生向けのプログラミング教育』に注目し、函館市で初めての小中学生向けプログラミングスクール『D-SCHOOL北海道』を立ち上げて運営。その後北海道のドラッグストア『サッポロドラッグストアー』からスクールの買収を受け、教育を専門とするグループ会社『株式会社シーラクンス』にて技術開発責任者として運営。スクール設計・運営、20以上の自治体連携（イベントや出張教室運営）、大人向けプログラミングスクールメンターなどさまざまな『次世代IT教育』に取り組む。Dify公式資料の日本語翻訳や、SecondMeアンバサダーなど生成AI最新トレンドに注力。

得意領域

- 生成AIをはじめとした世界最先端トレンド
- 生成AIを用いた新規事業の開発、運営、補助
- アプリケーション開発/
  自動化プログラム環境構築
- 教育（大学講師、研修、スクール運営など）

実績

- 生成AI研修
- 大手企業、外資、教育機関
  （小中高大）でのITトレンド講演
- 自治体と連携した地域教育実績
- 上場企業社員対象のリスキリング支援

メッセージ

みなさんがこれから生成AIをパートナーに
新しい時代を生きていけるように、
3日間全力でサポートしていきます！
なにかご不明点などありましたら、
講師陣になんでも気軽にご質問ください。
#ClaudeCode #Codex #Cursor #Antigravity
#全国統一生成AI活用技能試験S1
#生成AIパスポート　#G検定

3日間の流れと現在地

Day3 の主題は、動いたあとに起きること

Day1 書く

期間検索の追加
`findByDateRange`

検索用エンドポイント
`/api/orders/date-range`

一覧画面の作成
`static/dashboard.html`

明細つきの返却形
`OrderItemDTO`

遷移チェックの追加
`validateStatusTransition`

Day2 確かめる

異常系テストの追加
`OrderServiceImplTest`

仕込みバグ3件の修正
`BuggyOrderCalculator`

長いクラスの整理
`LegacyReportService`

Day3 直す・見えるようにする

無限再帰の特定
D3-1 StackOverflowError

ログからの切り分け
D3-2 性能劣化ログ

監視の追加
D3-3 ヘルスチェック

レビューの裏取り
D3-4 チェックリスト

Day1 と Day2 で作ったものが、そのまま今日の題材になります。

本日の流れ

長い演習は前半に2本

区分	時間
振り返り	[10min]
D3-1 StackOverflowError の原因特定	[35min]
D3-2 性能劣化ログの分析	[35min]
休憩	[10min]
D3-3 ヘルスチェックと処理時間ログ	[30min]
D3-4 セキュリティと規約のレビュー	[20min]
発展枠	[10min]
まとめ	[15min]
質疑応答・アンケート	[15min]
合計	**[180min]**

全員が終わってから次へ進みます。演習の途中で止まってもかまいません。

本日のゴール

合格ラインは、直したことではなく説明できること

1. **証拠を渡して原因を絞る**
   エラー文とログをそのまま渡し、当たりを付けてから自分で確かめます。
   D3-1・D3-2

2. **壊れる前に気づく仕組み**
   調べる道具を1本足して、外から状態を確認できるようにします。
   D3-3

3. **レビュー結果の裏取り**
   AI が挙げた指摘をチェックリストと突き合わせ、採否を自分で決めます。
   D3-4

どれも、今日その場で動かして確かめるところまでやります。

Day2 までに手を入れた場所

今日さわるのは、動くコードの外側

controller	Day1	OrderController に date-range を追加
	今日	HealthController を新規で作成
service	Day1	findByDateRange・validateStatusTransition
	Day2	BuggyOrderCalculator の3件・LegacyReportService の整理
repository・dto	Day1	期間検索のクエリと OrderItemDTO
static・config	Day1	static/dashboard.html を新規で作成
	今日	処理時間ログの設定を config に追加
test・exercises	Day2	OrderServiceImplTest に異常系を追加
	今日	exercises/day3 の2ファイル

今日さわる3か所は、どれも Day1 と Day2 で開いたファイルの隣にあります。

場面ごとの3操作の使い分け

選び方の基準は、壊れる範囲の広さ

場面	使う操作
質問だけしたい	ターミナルでそのまま聞く
1ファイルだけ小さく直したい	@ でファイルを指定して頼む
複数ファイルにまたがる・消える処理がある	**Planモード（Shift+Tab）で計画を先に出させる**
文脈が伸びて精度が落ちた	/clear でリセット、/compact で要約
レビューだけしたい	/review で観点をそろえて指摘を出す

Planモードに入ったつもりで入っていないことが、いちばん多いつまずきです。
いま入っているモードは、入力欄の下に出ています。

スライドと教材サイトの分担

手順は教材サイト、このデッキは地図

このスライド

- その日の狙いと全体像
- 図解と実測した数字
- クイズと答え合わせ
- 演習の前後の確認

教材サイトの D3-N カード

- さわるファイルの場所
- 手順とプロンプト例
- OK基準と確認の仕方
- 詰まったときの言い換え

どの演習でも同じ4ステップ

仮説を書く ▶ Claude Code に渡す ▶ 差分を読む ▶ 動かして確かめる

Claude Code　VS Code

※ 各ロゴは各社の商標です。

D3-1

StackOverflowError の原因特定

深夜のバッチが落ちて、朝には数百行のスタックトレースだけが
残っている場面を扱います。トレースは長さではなく形を見ます。
つまずきの大半は、上から順に全部読もうとすることです。

所要 [35min]

原因の当たりの付け方

AI で当たりを付け、確認だけを自分でやる進め方

現場の場面	今日の扱い
数百行のトレースだけが残る 読む順を決めないまま上から追う	**トレースとコードを一緒に渡す** 型と反復行が手がかりになります
当たりを付ける前に手が止まる どこを見ればいいか分からない	**当たり付けは AI、確認は自分** 指摘は仮説として受け取ります
直したつもりで直っていない 原因の行を確かめていない	**該当コードで裏づけてから採用** 差分の1か所を目で確かめます

見える成功は1つです。バッチが最後まで走り、成功3件・失敗2件で終わります。
題材は exercises/day3/CrashingOrderProcessor.java です。
Spring Boot 本体とは独立して動く、単体実行のクラスです。

呼び出しスタックと StackOverflowError

戻り先を積む領域が溢れたときの実行時エラー

正常な再帰

スタック領域の上限

上限に届かないまま終わる

retryCount = 0 → 戻る

retryCount = 1

retryCount = 2

retryCount = 3

呼び出しごとに残り回数が減り、0で戻ります。

止まらない再帰

スタック領域の上限

同じ値のまま積み続ける

retryCount = 3

retryCount = 3

retryCount = 3

retryCount = 3

残り回数が減らず、戻り先が積まれ続けます。

メソッド呼び出しの戻り先を積む領域が溢れたときに出る実行時エラーです。

出所 Java SE 17 API仕様 java.lang.StackOverflowError
https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/StackOverflowError.html

1031行のトレースの読む順

読むのは先頭の型と、反復している行

ここに実画面を差し込む:
CrashingOrderProcessor を実行した
ときの stderr 全1031行
先頭の型・JDK 内部フレーム・反復行が
同時に写る位置で撮る

1. **先頭は java.lang.StackOverflowError**
   メッセージは付きません
2. **続く JDK 内部フレームは犯人ではない**
   スタックが尽きた瞬間の出力処理です
3. **com.example.order が出る最初の行**
   ここまで下ってから読み始めます
4. **同じ行番号の反復を探す**
   CrashingOrderProcessor.java:80 が続きます
5. **main は出てこない**
   1024フレームで打ち切られるため、
   下から読む方法は使えません

反復する1行の中身

80行目の引数が減らないまま渡り続ける状態

```
76          processOrder(order);
77      } catch (Exception e) {
78          if (retryCount > 0) {
79              System.out.println("  リトライ残り: " + retryCount + "回");
80              processWithRetry(order, retryCount);          ◀ 残り回数が減っていない
```

仕様

リトライは最大3回で打ち切ります。
上限を超えたら例外を投げます。

実際の出力

「リトライ残り: 3回」だけが並び続けます。
残り回数が3のまま呼び直されています。

直し方はここでは出しません。変わるのは引数の1トークンだけです。

クイズ

クイズ 原因にたどり着く行

Q. 1031行のトレースのうち、原因の特定にいちばん効くのはどの行か

- **A** 先頭の java.lang.StackOverflowError の行
- **B** java.base/java.io.PrintStream.println の行
- **C** CrashingOrderProcessor.processWithRetry が繰り返される行
- **D** トレース末尾の main の行

正解は次のページで確認します。まず自分で考えてみてください。

解説

解説 反復する行が原因

正解はC。反復している行が原因を名指し

C **正解 CrashingOrderProcessor.processWithRetry が繰り返される行**
CrashingOrderProcessor.java:80 が積み上がり、増えるフレームを名指しします。

A 先頭の java.lang.StackOverflowError の行
型は分かりますが、どこで起きたかは分かりません。

B java.base/java.io.PrintStream.println の行
スタックが尽きた瞬間に走っていた出力処理です。

D トレース末尾の main の行
1024フレームで打ち切られるため表示されません。

長いトレースは上から順に読みません。自分のパッケージが出る最初の行と、
同じ行番号の反復を先に探します。

ワーク

演習 D3-1 無限再帰の修正

[35min]

再帰を止めて、バッチを最後まで通す状態

さわる

読む
exercises/day3/CrashingOrderProcessor.java

書く
同ファイルの再帰呼び出し1行だけ

できたら

StackOverflowError が出ない
最終行が「成功: 3件, 失敗: 2件」
リトライ残りの表示が各受注で3回だけ

流れ

1. 実行してエラーを再現する
2. 基底条件が成立するかを1文で書く
3. トレースとファイルを Claude Code へ渡す
4. 差分に残り回数の減算があるかで採否を決める

詳細は教材サイトの D3-1 カードにあります。

補足

D3-1 でつまずきやすい箇所

実行が止まらなくなっても、中断して大丈夫

起きること	対処
クラスが見つからない	クラスファイルの出力先を固定してから実行します。 教材サイトのコマンドをそのまま使ってください。
実行が止まらない	ターミナルで中断して構いません。 H2 も再起動で初期データに戻ります。
減算以外の書き換え提案	採否の基準は「残り回数が減っているか」の 1点だけです。ほかの整形は採らなくて構いません。
発展でリトライ上限を変えた	次の演習に進む前に3へ戻します。 戻し忘れると、あとの確認結果がずれます。

解説

D3-1 の答え合わせ

バッチが最後まで走り、成功3件・失敗2件

ここに実画面を差し込む:
修正後の実行結果のターミナル画面
最終行「成功: 3件, 失敗: 2件」と
各受注のリトライ残り 3回/2回/1回が
写ること

- **StackOverflowError が出ない**
  バッチ処理終了まで到達しています
- **100万円以下の3件は成功**
  BATCH-001・002・003 が成功件数に入ります
- **100万円超の2件は失敗**
  BATCH-004・005 がリトライ上限に到達します
- **最終行が「成功: 3件, 失敗: 2件」**
  失敗2件はリトライ上限による結果です

再帰ではなくループで書けば、スタックは消費しません。
どちらがこのバッチに向くかは、持ち帰りの論点にしてください。

補足

長いトレースと短いトレース

トレースの型で、読む向きが変化

長いトレース

1031行あり、下端は1024フレームで打ち切り
下から読む方法が使えません
手がかりは反復している行です

反復していた行

`CrashingOrderProcessor.java:80`

読む向き: 上から

短いトレース

application-slow.log の末尾3行
型は OutOfMemoryError: Java heap space
下から読むと自分のサービスの行に着きます

原因に近い行

`OrderServiceImpl.java:40`

読む向き: 下から

40行目は受注を全件読み出す行です。右のトレースは、次の D3-2 で扱います。

D3-2

[35min]

性能劣化ログの分析

「最近この一覧が重い」という曖昧な報告から、
ログだけで原因を切り分けます。コードは1行も直しません。
直し方を言える状態が、この演習の合格ラインです。
つまずきの大半は、遅い原因を1つに決め打ちすることです。

ログだけでの切り分け

コードを直さずに、原因の名指しと対策2方向

報告されること

「最近、受注一覧が重い」

条件も再現手順も書かれていません。
手元にあるのはログ1本だけです。

手元にあるログ

```
09:01 GET /api/orders 200 OK (156ms)
11:00 GET /api/orders 200 OK (3350ms)
11:25 GET /api/orders 200 OK (9200ms)
11:31 GET /api/orders OutOfMemoryError
```

今日の到達点

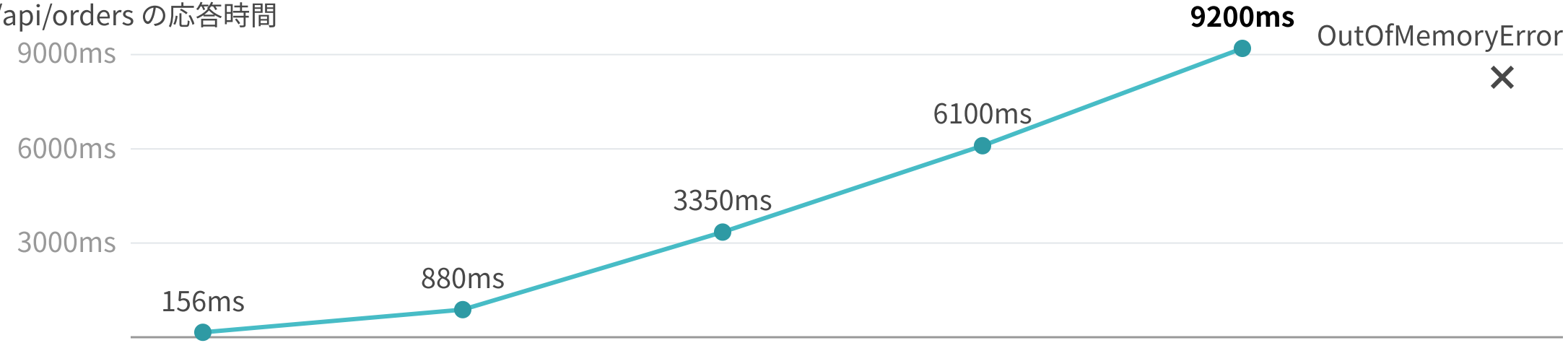
1. **原因の名指し** — 根拠になるログ行を1つ以上あげます
2. **対策は2方向** — 減らすものが違う2つを並べます
3. **効き先の区別** — 応答時間とメモリのどちらに効くか

題材は exercises/day3/application-slow.log です。コードは1行も直しません。

応答時間とヒープの推移

156ms から 9200ms まで、59倍の劣化

GET /api/orders の応答時間



時刻	09:01	10:00	11:00	11:15	11:25	11:31
SQL 本数	1	1	1	**6**	**0**	
GC 停止				450ms	1200ms	2300ms
ヒープ使用率				78%	92%	**96%**

縦軸は等間隔のままです。156ms は、いちばん下でほぼ潰れて見えます。
GC 停止とヒープ使用率は、その時刻の前後に記録された値です。

クイズ

クイズ 最も遅いリクエストの SQL 本数

Q. 応答に9200ms かかったリクエストは、SQL を何本発行しているか

- **A** 0本 一覧の SELECT も出ていない
- **B** 1本 一覧の SELECT だけ
- **C** 6本 一覧1本と明細5本
- **D** 11本 一覧1本と明細10本

正解は次のページで確認します。まず自分で考えてみてください。

解説

解説 SQL が0本の最遅リクエスト

正解はA。いちばん遅いのに SQL は0本

A **正解 0本 一覧の SELECT も出ていない**
11:25 のリクエストには、SQL の DEBUG ログが1行もありません。

B 1本 一覧の SELECT だけ
1本で済んでいたのは 09:01・10:00・11:00 のリクエストです。

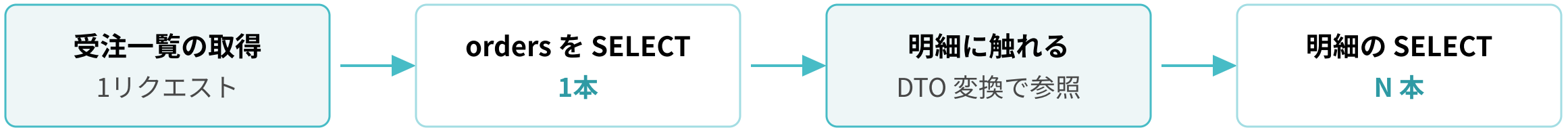
C 6本 一覧1本と明細5本
6本は 11:15 のリクエストです。ここが N+1 の観測点です。

D 11本 一覧1本と明細10本
明細の SELECT が10本並ぶ箇所は、このログにありません。

SQL が1本だけの 10:00 と 11:00 で、応答は既に880ms と3350ms です。
いちばん遅い 11:25 は、SQL を1本も発行していません。
N+1 だけを犯人にすると、この2つを説明できません。

N+1 が起きる仕組み

一覧1本のあとに明細が N 本続いて、発行は合計 1+N 本



受注が10件なら11本、100件なら101本、1000件なら1001本です。 **合計 1+N 本**

```
11:15:33.100 GET /api/orders - 受注一覧取得
11:15:38.900 select o1_0.id,... from orders o1_0
11:15:38.901 select i1_0.id,... from order_items i1_0 where i1_0.order_id=?
11:15:38.920 select i1_0.id,... from order_items i1_0 where i1_0.order_id=?
同じ SELECT があと3本続き、この1リクエストで6本になります
11:15:39.200 応答完了 200 OK (6100ms)
```

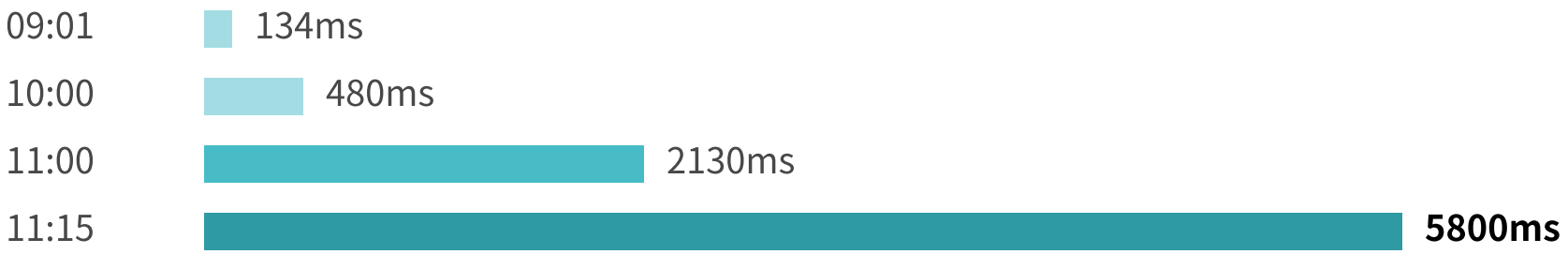
引き金は3つです。明細側の遅延ロード、受注側が明細を持つ関連、DTO 変換で明細に触ること。

6100ms の内訳

SQL に使った時間は65ms、残りは到達前の5800ms

SQL に到達する前	**5800ms**	受付ログから最初の SQL ログまで
SQL の発行	**65ms**	一覧1本と明細5本を合わせた時間
SQL のあと	**235ms**	DTO 変換とレスポンスの組み立て

受付から最初の SQL までの間隔



N+1 は起きていますが、6100ms の大半はその手前で使われています。

劣化要因の切り分け

要因は3つあり、どれも同じログの中に証拠

要因	証拠になるログ	打ち手
データ量の増加と全件ロード	OutOfMemoryError が OrderServiceImpl.java:40 で発生 バッチは 25件5431ms から 120件45677ms へ	ページネーション 読む件数を区切る
N+1	11:15 台に一覧1本と明細5本の SELECT 発行そのものは65ms	JOIN FETCH または @EntityGraph
GC 停止とヒープ枯渇	ヒープ使用率は 78% から 92%、96% へ GC 停止は 450ms から 1200ms、2300ms へ Full GC の停止は1.2秒と2.3秒	保持するデータ量 そのものを減らす

1つに決め打ちはできません。3つが重なって、この日の遅さになっています。

対策の効き方

N+1 対策で減るのはSQL の本数、ロード量は同じ

対策の効き先	メモリに効く	メモリには効かない
応答時間に効く	**ページネーション** 1回に読む件数そのものが減ります 全件ロードの前提が変わります	**JOIN FETCH / @EntityGraph** SQL の本数は1本に減ります 1回に読む量は変わりません
応答時間には効かない	このログでは該当なし	このログでは該当なし

軸の外 spring.jpa.open-in-view の無効化

どちらにも直接は効きません。ビューでの遅延ロードが効かなくなります。

ワーク

演習 D3-2 ログからの原因特定

[35min]

原因を名指しし、対策2方向の効き先まで言える状態

さわる

読む
exercises/day3/application-slow.log

書く　　分析メモ1本（任意）

できたら

N+1 の根拠を1つ以上あげた
OutOfMemoryError と時系列を結びつけた
対策を2方向あげた
どちらに効くかを分けた
open-in-view に触れた

流れ

1. 自分の目でログを1回追う
2. 分析だけと添えて Claude Code に渡す
3. JOIN FETCH 案と @EntityGraph 案を比較
4. 自分のメモと突き合わせて差を見る

コードは変更しません。この演習で作るのは
分析の言葉だけです。

詳細は教材サイトの D3-2 カードにあります。

解説

D3-2 の答え合わせ

自分のメモと AI の回答で、見落としの突き合わせ

ここに実画面を差し込む:
ログを渡したあとの分析回答
ターミナルの Claude Code 画面
N+1 と全件ロードの指摘が
写ること

- **N+1 の根拠**
  11:15 台に一覧1本と明細5本
- **メモリ不足との接続**
  11:31 に OrderServiceImpl.java:40
- **対策は2方向**
  JOIN FETCH と @EntityGraph
- **効き先の区別**
  応答時間とメモリを分けて言えたか
- **open-in-view**
  警告が出ている行に触れたか

ここではコードを直しません。直すのはこの先の工程です。
バッチの劣化も同じ全件ロードを踏んでいるのかは、持ち帰りの論点です。

休憩と後半の流れ

後半は、壊れる前に気づくための2本

休憩
[10min]

後半の流れ

D3-3 ヘルスチェックと処理時間ログ	[30min]
D3-4 セキュリティと規約のレビュー	[20min]
発展枠	[10min]
まとめ	[15min]

VS Code とターミナルは開いたままでかまいません。

D3-3

[30min]

ヘルスチェックと処理時間ログ

ここまでは、壊れてから調べる話でした。
ここからは、壊れる前に見える形にする話です。
つまずきの大半は、後始末の1行を忘れることです。

壊れる前に見える形

異常に早く気づける状態を、自分の手で用意

いま困ること

同時アクセスでログが混ざる
どのリクエストのエラーか追えません

稼働確認で業務 API を叩く
そのたびにデータを読みに行きます

今日足すもの

リクエストIDと処理時間の記録
同じ処理のログ行を束ねられます

軽い専用のヘルスチェック
業務 API を叩かずに状態を見られます

見える成功は1つです。GET /api/health が JSON を返します。
業務のコードには手を入れません。足すのはこの2本だけです。

リクエスト単位で束ねるログ

行の先頭にIDを付け、処理単位でログを回収

IDが無い状態

```
INFO OrderController : GET /api/orders
INFO CustomerController: GET /api/customers
INFO OrderController : GET /api/orders
WARN OrderServiceImpl : slow response
```

IDを付けた状態

```
[a1f3c2] INFO OrderController : GET /api/orders
[b7d904] INFO CustomerController: GET /api/customers
[c5e118] INFO OrderController : GET /api/orders
[b7d904] WARN OrderServiceImpl : slow response
```

到着

IDを採番して
置き場に入れます

処理

ログ行の先頭に
IDが付きます

完了

経過ミリ秒を記録し、
置き場を空にします

しきい値を超えた応答だけ警告に上げると、あとから機械的に拾えます。

補足

MDC の後始末

消し忘れると、前のIDが次のリクエストに混入

IDの置き場は MDC と呼びます。スレッド単位で持ち、スレッドは使い回されます。

消したとき	**リクエストA** ID a1f3 を入れ、完了時に消す	**リクエストB** ID b7d9 が入る
消し忘れたとき	**リクエストA** ID a1f3 を入れ、消さずに終わる	**リクエストB** ID a1f3 のまま記録される

完了時の処理が呼ばれない場合

開始の処理が正常に通ったときだけ、
完了の処理が呼ばれます。

別スレッドに渡した場合

IDは引き継がれません。渡す先に
明示的に持たせる必要があります。

持ち帰りは1点です。消す処理を入れたかを、自分で気づけるかどうか。

/api/health が返すもの

先に決めるのは、返す JSON の形

status	アプリ全体の状態
application	アプリ名
version	バージョン
database	データベースの状態
├ status	接続の可否
└ orderCount	受注の件数
memory	メモリの状況
├ used	使用量
├ max	上限
└ usagePercent	使用率

database.status

接続確認は軽いクエリ1本です。
つながらなければ DOWN を返します。

database.orderCount

リポジトリの count を使います。
配布データの受注は10件です。

memory

実行環境の値から算出します。
使用量・上限・使用率の3つです。

項目を先に決めておくと、受け取る画面側の実装も同時に書けます。

実演

実演 ヘルスチェックの応答

これから作るものの完成形を先に確認

ここに実画面を差し込む:
/api/health のレスポンス
上部に status が UP と出ること
database.orderCount が 10 のこと
memory の使用量と割合も写ること

画面: ヘルスチェックの応答

ここに実画面を差し込む:
同じリクエストIDと処理時間が
付いた起動ログの画面
終了行にミリ秒が写ること

画面: 起動ログ

ここは手を止めて、画面だけ見てください。作るのはこのあとの演習です。

補足

差分で必ず見る2点

見るのは、依存の受け取り方と例外処理の2点

見る点	採用できる形	採用できない形
依存の受け取り方	コンストラクタで受け取る形 テスト時に差し替えられます	フィールドに直接注入する形 配布物の規約で禁止しています
接続確認の例外処理	失敗したら DOWN を返す形 状態が外から分かります	catch の中が空の形 失敗が握りつぶされます

あわせて、既存の CORS 設定を消していないかも見ます。
AI が禁止パターンで書いてくる前提で、自分で見つけて直すところまでが演習です。

ワーク

演習 D3-3 監視の追加

[30min]

処理時間ログとヘルスチェックを1本ずつ追加

さわる

新規

config/RequestLoggingInterceptor.java
controller/HealthController.java

追記

config/WebConfig.java

できたら

同じ処理のログ行に同じIDが並ぶ
終了行に処理時間のミリ秒が出る
状態は UP、件数は10と一致

流れ

1. 返す JSON の形を先に自分で決める
2. Planモードで計画を確認してから承認する
3. 差分で後始末と依存の受け取り方を見る
4. 起動して /api/health を叩く

詳細は教材サイトの D3-3 カードにあります。

解説

D3-3 の答え合わせ

ログが同じIDで束ねられ、件数は10件と一致

ここに実画面を差し込む:
リクエストIDと処理時間が付いた
アプリ起動ログの画面
終了行のミリ秒が写ること

- **同じIDでログが束ねられている**
  1リクエストぶんだけを絞り込めます
- **終了行に処理時間が出ている**
  ミリ秒まで記録されています
- **ヘルスチェックが JSON を返す**
  200 で応答し、状態は UP になります
- **件数が配布データと一致**
  受注は10件です

ダッシュボードの育ち方

発展として、Day1 で作ったダッシュボードにこの表示欄を足せます。

一覧の表 → 明細欄 → ステータス変更 → **ヘルス表示**

D3-4

セキュリティと規約のレビュー

Claude Code は存在しない問題を挙げ、本当のリスクを
見落とすことがあります。だから物差しを先に持ちます。
指摘の実物は、演習に入る前に見せます。

所要 [20min]

AI レビューの外し方

AI を下書き役に使い、最終判断は自分で持つ型

実在しない問題を挙げる

配布物に無いコードを前提にした指摘が
混じることがあります。

見分け方

チェックリストに該当項目がない
指摘は、いったん保留にします。

本当のリスクを見落とす

認証なしの公開や全許可の設定を
挙げないことがあります。

見分け方

AI が拾わなければ自分で足します。
チェックリスト側が根拠になります。

※ 各ロゴは各社の商標です。

Claude Code の指摘は下書き

見える成功は、指摘が3件以上出て、そのうち1件を直せている状態です。

指摘と物差しの突き合わせ

紐づく物差しが無い指摘は、いったん保留

Claude Code の指摘

分類

Critical　Warning　Info

各指摘にそろえる3点

- 該当箇所
- 理由
- 修正方針

指摘は仮説です。物差しに当ててから採否を決めます。

docs/セキュリティチェックリスト.md

- SQL のパラメータ化
- 入力の検証
- 個人情報をログや例外に出さない
- 例外の詳細をレスポンスに出さない
- DTO と Entity の分離
- H2 コンソールの本番無効化
- CORS を必要最小限に

docs/コーディング規約.md

- 命名の統一
- 層の責務の分離
- フィールドインジェクション禁止
- 生 SQL の文字列連結禁止
- 空 catch 禁止と標準出力の print 禁止

顧客情報の露出と入力の丸受け

認証なしで住所・電話・メールが返る状態

CustomerController の一覧

```
@GetMapping
public List<Customer> findAll() {
  return repository.findAll();
}
```

CustomerController の登録

```
@PostMapping
public Customer create(
    @RequestBody Customer c) {
  return repository.save(c);
```

Customer が持つ項目

id | name | **address** | **phone** | **email**

認証なしで1回叩くだけで、配布データの
顧客5件の住所・電話・メールが
そのまま返ります。

起きること

id を含めて投げると、既存のレコードを
書き換えうる形になっています。
入力の検証も走っていません。
受け口は入力用のクラスに分けます。

CORS 全許可の設定

認証が無いので、全許可がそのまま全公開

```
@Override
public void addCorsMappings(CorsRegistry registry) {
   registry.addMapping("/api/**")
        .allowedOrigins("*")
        .allowedMethods("GET", "POST", "PUT", "DELETE");
```

1. この API には認証がないので、全許可がそのまま全公開になります。
2. 該当するチェック項目は「許可オリジンをワイルドカードにしない」です。
3. 直すなら、許可オリジンを設定ファイルから注入する形にします。

配布物では、この状態を意図的に残してあります。

クイズ

クイズ 個人情報が返る API

Q. 認証なしに叩くと、顧客の住所・電話・メールがそのまま返るのはどれか

A GET /api/orders

B GET /api/orders/{id}

C POST /api/customers

D GET /api/customers

正解は次のページで確認します。まず自分で考えてみてください。

解説

解説 エンティティ直返しの影響

正解はD。エンティティをそのまま返す一覧

D **正解 GET /api/customers**
Customer をそのまま返すので、address・phone・email が全部含まれます。

A GET /api/orders
受注の一覧は DTO を通すので、この形になりません。

B GET /api/orders/{id}
同じく DTO 経由で、顧客の連絡先は含まれません。

C POST /api/customers
入力を丸ごと受ける別の問題で、情報が返る話ではありません。

該当するチェック項目は「顧客情報を返す API は必要な項目だけの DTO に絞る」です。
受注側と同じように、返す項目を選んでから外に出します。

ワーク

演習 D3-4 セキュリティと規約の点検

[20min]

AI の指摘を物差しで裏取りした一覧

さわる

読む

OrderServiceImpl・OrderController
WebConfig・CustomerController
チェックリストと規約の2ファイル

書く

レビュー一覧（ファイル化は任意）

できたら

Critical・Warning・Info で分類されている
該当箇所・理由・修正方針がそろっている
CORS の全許可を拾えている
紐づいた1件を直した

流れ

1. **渡す前に自分で3件メモする**
   気になる箇所を先に言語化します
2. **3観点で分類させる**
   セキュリティ・規約・テスト容易性
3. **チェックリストの項目に紐づける**
   紐づかない指摘は保留にします
4. **紐づいた1件を直す**
   修正方針までそろったものを選びます

詳細は教材サイトの D3-4 カードにあります。

解説

D3-4 の答え合わせ

3種類に仕分けてから、直す1件を選ぶ順番

自分だけが見つけた
チェックリストを根拠に足す

AI だけが挙げた
実コードで裏づけて判断する

両方が挙げた
優先して直す1件にする

ここに実画面を差し込む:
/review の出力画面
Critical・Warning・Info の分類が
写る位置で撮る

- **3段階に分類されている**
  Critical・Warning・Info が付いています
- **指摘に3点がそろっている**
  該当箇所・理由・修正方針が書かれています
- **CORS の全許可を拾えている**
  チェック項目に紐づけられます
- **紐づいた1件を直した**
  直した理由を自分の言葉で言えます

直したあとに同じ依頼を投げ直すと、その指摘が消えるかを確かめられます。

発展

発展枠の3本

[10min]

全員向けではなく、早く終わった方の吸収先

番号	内容	後続への影響
D3-2+	**キャンセル判定の抜けの修正** 配布状態で1件だけ赤いテストが残っています。 赤を証拠として渡し、緑になるまで直します。	影響なし
D3-3+	**SQL 発行件数のしきい値監視** 1リクエストで出た SQL の本数を数え、しきい値を 超えたら警告を出します。しきい値は定数にします。	影響なし
D3-4+	**規約ファイルの有無による出力差** 同じ依頼を規約ファイルのあり・なしで投げ、 出てくるコードの差を確かめます。	影響なし 確認後に名前を戻す

3本とも独立しています。気になったものだけを選んで構いません。

発展

規約ファイルありなしの出力差

同じ依頼でも、規約ファイルの有無で出力が変化

規約ファイルあり

ここに実画面を差し込む:
規約ファイルがある状態で
生成させた同一メソッドの画面

- ステータスは定数を使う
- 参照専用の指定が付く
- ロガー経由でログを出す

規約ファイルなし

ここに実画面を差し込む:
規約ファイルを無効にして
生成させた同一メソッドの画面

- ステータスを文字列で直書き
- 参照専用の指定がない
- 標準出力への print が混じる

出力は毎回ばらつくので、差が出なければ数回繰り返してください。
チームに配ると、全員の前提を同じにそろえられます。
確認が終わったら、規約ファイルの名前を元に戻します。

実演

Recommender と skill-creator の実演

薦められる設定と自作スキルを、画面で確認

ここに実画面を差し込む:
claude-code-setup の Recommender が
このプロジェクトへの推奨を出した画面

1 Recommender

公式の claude-code-setup の機能です。
いまのプロジェクトに合う設定を
一覧で出します。

ここに実画面を差し込む:
skill-creator で作ったスキルを
呼び出したところの画面

2 skill-creator

手順をまとめたスキルを作り、
その場で呼び出せる状態まで
画面で確かめます。

ここは操作せず、画面を見るだけで大丈夫です。

まとめ

[15min]

現場に戻ってからの最初の一歩

3日間で触った場所を1枚で確認します。
持ち帰るのは、感想ではなく作業です。
明日ひとりでできることと、今週チームでできることに分けます。

振り返り | D3-1 | D3-2 | D3-3 | D3-4 | まとめ

3日間で触った場所

3日間で足したのは、動くコードと確かめる仕組み

controller	Day1	OrderController に date-range を追加
	Day3	HealthController を新規で作成
service	Day1	findByDateRange・validateStatusTransition
	Day3	キャンセル判定の抜けを修正
repository・dto	Day1	OrderItemDTO と期間検索のクエリ
static・config	Day1	static/dashboard.html を新規で作成
	Day3	RequestLoggingInterceptor を config に追加
test・exercises	Day2	OrderServiceImplTest の異常系と BuggyOrderCalculator の3件
	Day3	CrashingOrderProcessor の再帰を1行だけ修正

コンパイルが通ったではなく、画面とテストで確かめてきました。

Claude Code に渡す情報の線引き

線引きは、再現したものか本物かの一点

渡してよいもの	渡してはいけないもの
ダミーデータで再現したログとトレース	本番ログそのもの
自分の担当プロジェクトのソース	顧客の氏名・住所・電話・メール
公開されている仕様やエラーの型	認証情報や接続文字列
社内で共有してよい規約ファイル	外部提供が禁止されている設計書

> 今日扱ったログもトレースも、すべてダミーデータです。
> 判断に迷ったら、投入する前に社内のルールを確認します。

明日の一歩に、承認も新しい環境も不要

1. **プロジェクト直下の規約ファイル1枚**
   自分の担当プロジェクトの直下に置くと、依頼のたびに読み込まれます。
   根拠 D3-4+

2. **先に見るのはトレースの型と反復行**
   エラーが出たら、上から全部読まずに繰り返している行を探します。
   根拠 D3-1

3. **応答時間と SQL 本数の並記**
   遅いと言われたら、まずこの2つを時刻順に並べて出します。
   根拠 D3-2

詰まったときの言い換えは、教材サイトに残っています。

今週チームでできること

チームに効くのは、物差しと計測の2本

1

レビュー観点のチェックリスト1枚

呼び出せる形で置くと、レビュー前に全員が同じ物差しを見られます。

今日つくったもの docs/セキュリティチェックリスト.md と docs/コーディング規約.md

根拠 D3-4

2

ヘルスチェックと処理時間ログの常設

先に計測を持つと、遅いという報告を数字で受け取れます。

今日つくったもの HealthController と RequestLoggingInterceptor

根拠 D3-3

どちらも、今日その場で作ったものと同じ形です。

質問

質疑応答とアンケート

[15min]

答えきれない質問は、後日まとめて回答

手元に残るもの

配布 ZIP の演習ファイルと docs の2ファイルは、研修後もそのまま使えます。

教材サイト

`idunder02.give-app.net`

アンケート

チャットで案内するフォームから、この場でご回答をお願いします。

株式会社ギブリー (Givery, Inc.)
〒150-0036 東京都渋谷区南平台町15-13 帝都渋谷ビル8F