DAY 3　2026年9月17日（木）

壊れたときに効く動き

壊れたコードと遅くなったログを前にして、原因を絞ってから直す動きを一通り通します。最後はヘルスチェックとレビューで、3日かけて書いたコードを運用に耐える形へ寄せます。

この日の演習

演習は上から順に進みます。全員が終わるのを待ってから次へ進むので、詰まったら手を挙げてください。早く終わった方には各演習に発展課題があります。

D3-1 無限再帰の原因特定と修正 [35min]

ねらい

実行して出たスタックトレースを証拠にして、止まらない再帰の1行を突き止めて直します。

さわるファイル

書く	exercises/day3/CrashingOrderProcessor.java	再帰の引数1行だけ直す
読む	exercises/day3/exercise1-デバッグ.md	手順と模範プロンプト

手順

1. **考える** — processWithRetry を開き、この再帰がどこで止まる想定なのかを1行メモします。retryCount が呼び出しごとにどう変わるかも書き添えます。
2. **実行** — VSCode で配布フォルダ（mvnw.cmd がある階層）を開き、ターミナルで cd exercises\day3 のあと javac -d . CrashingOrderProcessor.java と java -cp . com.example.order.debug.CrashingOrderProcessor を打ちます。リトライ残り: 3回 が5000行以上流れたあと、*** StackOverflowError が発生しました *** に続けて java.lang.StackOverflowError から始まるスタックトレースが出ます。
3. **書く** — claude を起動し、@CrashingOrderProcessor.java 実行すると StackOverflowError が出ます。原因はどこですか、とだけ送ります。返答を見てから リトライは最大3回で打ち切る仕様です を足し、java.lang.StackOverflowError の行と processWithRetry が並ぶ部分を20行ほど指示文の末尾に貼ります。
4. **答え合わせ** — 承認前に出る差分で、processWithRetry(order, retryCount - 1); が入っているか、変更行が1行だけかを見てから承認します。もう一度コンパイルして実行し、確認できたら cd ..\.. でプロジェクト直下に戻ります。

できたら

- [ ] === バッチ処理終了 === が出て、その次の最終行が 成功: 3件, 失敗: 2件 になる
- [ ] BATCH-004 と BATCH-005 で リトライ残り: 3回 / 2回 / 1回 の3行が出て打ち切られる
- [ ] 承認前の差分で、変更が processWithRetry の再帰1行だけに収まっている

考えること

> リトライを再帰で書くのとループで書くのとでは、このバッチにはどちらが向くか。

AI の出方

> 原因を当てても、修正案が for ループへの全面書き換えになったり、try の位置ごと動かしてきたりします。差分に retryCount - 1 が入っていなければ、再帰に渡す引数を1つ減らす形だけで直して、と言い直してください。

D3-1+ maxRetries を 3 から 0 に変えて保存し、javac -d . CrashingOrderProcessor.java を打ち直してから java -cp . com.example.order.debug.CrashingOrderProcessor を実行します。コンパイルし直さないと古いクラスファイルが動いて表示が変わりません。リトライ表示が1行も出ないまま失敗2件に入るかを、予想を先に書いてから確かめると、境界の効き方を自分の言葉で説明しやすくなります。確認したら 3 に戻し、同じくコンパイルからやり直します。

くわしく（背景・詰まったときの対処）

配布フォルダの exercises\day3 に、受注バッチを模したクラスが1本だけ置いてあります。Spring Boot 本体からは切り離した単体実行用で、package com.example.order.debug を宣言しています。金額が100万円を超える BATCH-004 と BATCH-005 は外部連携で必ず失敗する作りです。ここでリトライが止まらなくなります。

StackOverflowError は、メソッド呼び出しが深くなりすぎて呼び出しスタックが溢れたときに出る実行時エラーです。このクラスの main はそれを受け止め、見出しを1行出したあと e.printStackTrace() でスタックトレースを標準エラーへ流します。at com.example.order.debug.CrashingOrderProcessor.processWithRetry の行が1000行あまり並び、最後にヒントが2行出ます。JVM が保持するフレームは既定で1024までなので、実際の再帰回数はこれより多いです。Claude Code へ渡す証拠は、標準出力に流れるリトライ表示ではなく、この標準エラーのスタックトレースです。想定しているのは、夜中に落ちたバッチの調査で、朝に残っているのがこの出力だけ、という場面です。当たりを付けるところを Claude Code にやらせて、確認は自分の手でやります。

コンパイルは通るのに ClassNotFoundException や NoClassDefFoundError が出るときは、javac の -d . を付け忘れています。クラスファイルは exercises\day3\com\example\order\debug\ の下に出ます。新しいファイルは作りません。クラス名・パッケージ・メソッドの引数はそのままにして、判定に関わる1行だけを直してください。演習が終わったら cd ..\.. でプロジェクト直下、つまり mvnw.cmd がある階層に戻ります。D3-2 以降で打つ .\mvnw.cmd はそこからでないと見つかりません。

D3-2 性能劣化ログの切り分け [35min]

ねらい

だんだん遅くなるログから、N+1 と全件取得を分けて名指しできる状態にします。

さわるファイル

読む	`exercises/day3/application-slow.log`	劣化の根拠を時系列で拾う
読む	`exercises/day3/exercise1-デバッグ.md`	D3-2 の手順と模範プロンプト
新規	`exercises/day3/analysis-slow-log.md`	原因と対策のメモを残す

手順

1. **考える** application-slow.log を VSCode で開き、応答時間が 156ms から 9200ms まで伸びる 5・14・23・35・39 行目と、order_items への SELECT が5行続く 30〜34 行目の行番号をメモに控えます。OutOfMemoryError が出た 48〜49 行目のメソッド名も控えます。
2. **書く** claude を起動し、@application-slow.log を付けて、分析だけ、コードは変更しない、と添えてから劣化の原因と対策を時系列の根拠つきで出させます。
3. **答え合わせ** 自分のメモと返ってきた指摘を並べ、N+1 の解消と全件取得の見直しという2方向に分かれているかを読みます。根拠が曖昧なところは、どのログ行が根拠か、と聞き返します。
4. **書く** exercises\day3\analysis-slow-log.md を作り、根拠にした行番号、対策2方向、それぞれが応答時間とメモリのどちらに効くかを箇条書きで残します。

できたら

- [ ] analysis-slow-log.md に、N+1 の根拠として 30〜34 行目の連続 SELECT が行番号つきで書けている
- [ ] 同じメモに、N+1 の解消は応答時間に、全件取得の見直しはメモリに効くと書き分けられている
- [ ] 48〜49 行目の OutOfMemoryError と findAll の全件ロードをつないだ1文がメモにある

考えること

> 対策を1つしか入れられないとしたら、N+1 の解消とページネーションのどちらを先に打つか。

AI の出方

分析だけを頼んでも、途中からコードを直しにかかることがあります。指示文の冒頭に分析だけと置き、それでも編集の許可を求めてきたら断ってください。根拠が曖昧な回答には、どのログ行が根拠か、と聞き返してください。ログ行を示せない指摘は落とせます。

D3-2+ プロジェクト直下（mvnw.cmd がある階層）で作業します。src\main\java\com\example\order\service\OrderServiceImpl.java の cancelOrder を開き、SHIPPED を弾く if の直後に、同じ InvalidOrderStateException を投げる DELIVERED 用の if を1つ足します。メッセージには キャンセルできません を残してください。既存の SHIPPED のテストがこの文言を見ているので、外すと落ちます。.\mvnw.cmd test -Dtest=OrderServiceImplTest を流し、DELIVERED のテストが PASS に変わるところで合否を判定します。API でも見る場合は、Java を書き換えたので起動中なら Ctrl+C で止め、.\mvnw.cmd spring-boot:run を打ち直してから、DELIVERED の id=6 ORD-20260404-006 に PATCH /api/orders/6/cancel を打ちます。応答は、例外を HTTP ステータスへ変換する @RestControllerAdvice が未実装なら 500、D1-5+ で入れていれば 400 です。確認できたら Ctrl+C で止めておきます。8080 を掴んだままだと D3-3 の起動が失敗します。

くわしく（背景・詰まったときの対処）

application-slow.log は本番想定で取った Spring Boot のログです。GET /api/orders の応答が 156ms、880ms、3350ms、6100ms、9200ms と伸び、最後は OutOfMemoryError で止まっています。30〜34 行目では、orders を1回引いた直後に order_items への SELECT が where i1_0.order_id=? のまま5行続きます。これが N+1 です。一覧を1回引いたあとで明細を1件ずつ取りに行くため、件数 N に比例してクエリが N+1 本に膨らみます。

本編ではコードを直しません。最近この画面が重い、という曖昧な報告から、ログだけで原因を切り分けて対策の効きどころまで言える状態が到達点です。N+1 の解消は応答時間に効き、全件取得をやめる方はメモリに効きます。この2つを混ぜずに分けて言えると、後の改善でどちらから手を付けるかを決めやすくなります。OrderServiceImpl を書き換えるのは発展の D3-2+ だけです。

ログには spring.jpa.open-in-view is enabled by default の警告と、GC pause が Young Gen 450ms から Old Gen 2300ms へ伸びてヒープ使用率が 78 パーセントから 96 パーセントへ上がる様子も残っています。ここまで拾えると、遅いという報告が枯渇の手前だったことまで説明できます。メモは exercises\day3\analysis-slow-log.md に置いてください。行番号を書いておくと、後半の改善で見返すときにログを探し直さずに済みます。

休憩

[10min]　ここで一度手を止めます。詰まっている方はこの間に声をかけてください。

D3-3 処理時間ログとヘルスチェック [30min]

ねらい

リクエスト単位の処理時間ログと、稼働状態を1回のリクエストで確かめられる口を足します。

さわるファイル

書く	`src/main/java/com/example/order/config/WebConfig.java`	addInterceptors を追記して登録
新規	`src/main/java/com/example/order/config/RequestLoggingInterceptor.java`	IDと処理時間。logback-spring.xml も新規
新規	`src/main/java/com/example/order/controller/HealthController.java`	GET /api/health。HealthResponse も新規

手順

1. **考える** ログに出す3点、つまりリクエストIDをどこに持たせるか、処理時間をどこからどこまで測るか、MDC に入れた値をいつ消すかを手元に書き出します。あわせて返す JSON の形も決め、status、database.status、database.orderCount、memory.usagePercent の入れ子と型をここで固めます。
2. **書く** claude を起動し、@WebConfig.java を付けて、リクエストIDと処理時間を出す RequestLoggingInterceptor の新規作成と /api/** への登録を依頼します。あわせて src\main\resources\logback-spring.xml を新規作成し、パターンを %d{HH:mm:ss.SSS} [%X{requestId}] %-5level %logger{36} - %msg%n にするところまで頼みます。
3. **実行** プロジェクト直下（mvnw.cmd がある階層）で .\mvnw.cmd spring-boot:run を打ち、VSCode のターミナルを分割して2枚目から curl.exe http://localhost:8080/api/orders を打ちます。同じリクエストのログ行の角括弧に同一のIDが並び、終了行に処理時間のミリ秒が出るのを読みます。
4. **答え合わせ** 決めた JSON の形のまま /api/health を返す HealthController の新規作成を依頼し、差分に @Autowired のフィールドが混じっていないかを見てから承認します。Java が増えたので Ctrl+C で止めて .\mvnw.cmd spring-boot:run を打ち直し、curl.exe -s http://localhost:8080/api/health | jq . で database.orderCount が 10 になるかを確かめます。

できたら

- [ ] curl.exe -s http://localhost:8080/api/health | jq . が JSON を返し、database.orderCount が 10 になる
- [ ] 1リクエストのログ行の角括弧に同じIDが入り、終了行に処理時間のミリ秒が出る
- [ ] 差分の依存が private final と @RequiredArgsConstructor で組まれている

考えること

> DB が落ちているとき、全体を DOWN にするか、database.status だけ DOWN にして status は UP のままにするか。

AI の出方

返す JSON の形と入れ子は毎回ぶれます。先に自分で書いた形をそのまま貼って渡すと揃います。@Autowired のフィールドインジェクションで書いてくることがあるので、そのときはコンストラクタインジェクションに直して、と一言足してください。

D3-3+ 1リクエストで発行された SQL の本数を数えて、10件を超えたら WARN を出します。Hibernate の Statistics を使うと依存を増やさずに済みますが、既定は無効です。配布時点の src\main\resources\application.yml には generate_statistics の指定が無く、このままだと本数が常に 0 で WARN が出ません。spring.jpa.properties.hibernate.generate_statistics: true を足し、設定ファイルなので Ctrl+C で止めて .\mvnw.cmd spring-boot:run を打ち直します。しきい値は SQL_COUNT_THRESHOLD という定数に切り出してください。D1-4 で OrderDTO に明細を含める実装を終えていれば、受注一覧で SQL が 11本になって WARN が出ます。未実装のままだと SQL は1本しか出ないので、しきい値を一時的に 1 へ下げて動作だけ確かめます。

くわしく（背景・詰まったときの対処）

このプロジェクトは Spring Boot Actuator を入れていません。アプリが生きているか、DB に繋がっているかを外から確かめる口が無い状態です。監視に GET /api/orders を使うと業務データを毎回読みに行くので、軽い専用の口を別に置きます。返す中身は status、application、version、database.status、database.orderCount、memory.used、memory.max、memory.usagePercent です。orderCount は配布データの受注件数と同じ 10 になります。

処理時間ログの方は、config パッケージの WebConfig に addInterceptors を足して登録します。HandlerInterceptor は、コントローラの処理の前後に共通処理を差し込む仕組みで、入り口の preHandle と出口の afterCompletion を使います。リクエストIDは MDC に入れます。MDC はログ出力ライブラリがスレッド単位で値を持つ置き場で、消し忘れると次のリクエストに前のIDが残ります。afterCompletion の最後に消す行が入っているかを、差分で必ず見てください。MDC に入れただけではログ行にIDは出ません。既

定のログパターンに %X{requestId} が無いためで、logback-spring.xml を新しく作ってパターンを差し替えるところまでが1組です。

依存の受け取り方は2通りあります。private final のフィールドをコンストラクタで受ける形が規約で推奨、@Autowired をフィールドに付ける形が規約で禁止です。差分に @Autowired の行があれば直させてください。新しく作るのは RequestLoggingInterceptor、logback-spring.xml、HealthController、応答を組み立てる src\main\java\com\example\order\dto\HealthResponse.java の4本で、これに WebConfig への addInterceptors 追記が加わります。レスポンスは Map で組まず、record か DTO の専用型にします。JdbcTemplate の Bean が見つからないと言われたら、EntityManager のネイティブクエリで SELECT 1 を実行する形に直させます。jq が入っていない環境では、| jq . を外して curl.exe http://localhost:8080/api/health だけで中身を読めます。細かい打鍵と模範プロンプトは exercises\day3\exercise2-監視とセキュリティ.md にあります。

この口ができたら、Day1 で作った src\main\resources\static\dashboard.html から /api/health を呼び、自分の画面の上に UP と受注件数を並べられます。この画面ファイルはアプリを起動したまま書き換えられ、保存してブラウザを再読み込みすると新しい中身に差し替わります。表示を見て気に入らないところを claude に直させ、また再読み込みして見る、という往復をそのまま続けられます。止めて起動し直すのは Java か設定ファイルを書き換えたときだけです。

新規4本と追記1本を30分で通します。時間内に両方が届かないときは、先にヘルスチェックを動かしてください。処理時間ログは logback-spring.xml までを1組として、残った時間で足します。

D3-4 セキュリティと規約のレビュー [20min]

ねらい

生成したコードを2つのチェックリストに照らし、指摘を絞り込んでから1件だけ直します。

さわるファイル

読む	docs/セキュリティチェックリスト.md	指摘を突き合わせる基準
読む	docs/コーディング規約.md	禁止パターンの基準
書く	src/main/java/com/example/order/config/WebConfig.java	CORS の全許可を絞る

手順

1. **考える** — OrderServiceImpl と OrderController をざっと読み、気になった点を3つメモします。例外ハンドラが無いこと、changeStatus が Map で入力を受けていることあたりが候補です。
2. **書く** — claude を起動し、@OrderServiceImpl.java と @OrderController.java と @WebConfig.java を付けて、セキュリティ・規約・テスト容易性の3観点で Critical と Warning と Info に分けた指摘を依頼します。添付コードに実在する箇所だけ、と添えます。
3. **答え合わせ** — 返ってきた指摘を2つのチェックリストの項目へ1件ずつ紐づけます。紐づかないものには、どのファイルの何行か、と聞き返し、示せなければ落とします。
4. **実行** — @WebConfig.java を付けて allowedOrigins を http://localhost:8080 だけに絞る形に直して と依頼し、差分を読んで承認します。Java を変えたので Ctrl+C で止めてから .\mvnw.cmd spring-boot:run を打ち直し、curl.exe -i -H "Origin: http://example.com" http://localhost:8080/api/orders が 403 と Invalid CORS request を返すのを見ます。

できたら

- [ ] Critical と Warning と Info に分けた指摘が3件以上、該当メソッド名つきで並んでいる
- [ ] WebConfig の allowedOrigins の全許可が指摘に入っている
- [ ] 絞ったあと、Origin: http://example.com を付けた要求が 403 と Invalid CORS request で弾かれる

考えること

> Claude Code が挙げた指摘のうち、チェックリストに対応する項目が無いものをどう扱うか。

AI の出方

実在しない問題を Critical で挙げることもあれば、CORS の全許可を素通りさせることもあります。指摘ごとに、どのファイルの何行か、と聞き返して、コードに無ければ落としてください。

D3-4+ CLAUDE.md を CLAUDE.md.off に改名します。CLAUDE.md は claude の起動時に読み込まれるため、改名しただけでは起動中のセッションには効きません。claude をいったん終了して起動し直してから同じ依頼をもう一度投げ、ステータス定数の使われ方や @Transactional の付き方がどう変わるかを見ます。確認したら名前を戻し、もう一度起動し直します。

くわしく（背景・詰まったときの対処）

レビューの基準は 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 の2つです。前者は SQL のパラメータ化、入力値の検証、個人情報をログや例外メッセージに出さない、例外の詳細を API レスポンスに含めない、DTO と Entity の分離、H2 コンソールの本番無効化、機密情報をログ・設定に残さない、CORS を必要最小限に、という並びです。後者には禁止パターンとして、フィールドインジェクション、生 SQL の文字列連結、空 catch、System.out.println が挙がっています。手順の細かい打鍵と模範プロンプトは exercises\day3\exercise2-監視とセキュリティ.md にあります。

CustomerController の findAll は Customer エンティティをそのまま返しています。email と phone と address が丸ごとレスポンスに出る形です。createCustomer は @RequestBody Customer でエンティティを直接受けるので、利用者に決めさせたくない項目まで外から書き換えられます。WebConfig の addCorsMappings は allowedOrigins が全許可。3つとも配布時点の実コードにあり、探せば見つかります。

/review は直近で変更したファイルを規約に照らして見るコマンドです。観点を細かく指定したいときは、@ でファイルを渡して自然言語で頼む方が狙いどおりになります。指摘一覧を残すなら exercises\day3\review-result.md あたりです。Origin を付けない素の curl.exe は絞る前も後も 200 を返すので、それは設定を壊していない確認として読みます。絞る前に Origin: http://example.com を付けると 200 と Access-Control-Allow-Origin: * が返り、絞ったあとは Spring が同じ要求を 403 と本文 Invalid CORS request で弾きます。ヘッダだけ消えた 200 にはなりません。直す1件は CORS が手軽ですが、例外ハンドラを新規に作る方を選んでもかまいません。ちなみに例外ハンドラを入れると、存在しない ID への問い合わせが 500 から 404 に変わるので、直した効果がそのまま画面に出ます。

到達チェック

この日の終わりに、自分で確かめてください。全部埋まっていなくても構いません。

- [ ] バッチが 成功: 3件, 失敗: 2件 で終わり、リトライ残り: の表示が3行で打ち切られている
- [ ] analysis-slow-log.md に根拠の行番号、対策2方向、それぞれの効きどころが書けている
- [ ] curl.exe -s http://localhost:8080/api/health | jq . が JSON を返し、orderCount が 10 になっている
- [ ] 1リクエストのログ行の角括弧に同じIDが入り、終了行に処理時間のミリ秒が出ている
- [ ] CORS を絞ったあと、Origin: http://example.com を付けた要求が 403 と Invalid CORS request で弾かれる
- [ ] Claude Code が挙げた指摘のうち、ファイル名と行を示せたものだけを残し、示せなかったものを落とした

つまずいたとき

よく出る詰まりどころです。当てはまるものがなければ、その場で声をかけてください。

生成されたコードが正しいのか、自分で判断できる自信がありません

判断の基準を自分の外に置いてください。D3-1 は最終行の集計、D3-3 は curl.exe が返す JSON、D3-4 は docs の2つのチェックリストが基準です。読んで意味が取れない行が出たら、その行だけを指して、ここが何をしているか1行で説明して、と claude に聞き返してください。

Planモードに切り替えるのを忘れます

入力欄で Shift+Tab を押すたびにモードが切り替わります。表示が plan mode になるまで押してください。忘れて実装が始まっても、ファイルを書き換える前に許可を聞かれるので、そこで断れば止まります。複数ファイルに広がる依頼や新規クラスの生成の前だけ意識すれば十分で、1行の修正まで計画を挟む必要はありません。

javac は通るのに ClassNotFoundException で起動できません

-d . を付け忘れると、クラスファイルがソースと同じ場所に出て、パッケージ階層をたどる起動指定とずれます。javac -d . CrashingOrderProcessor.java からやり直し、java -cp . com.example.order.debug.CrashingOrderProcessor で起動してください。

curl を打っても結果が出ません

PowerShell では curl が別のコマンドの別名になっていることがあります。.exe を付けて curl.exe と打ってください。jq が入っていない環境では、| jq . を外して curl.exe http://localhost:8080/api/health だけで中身を読めます。

ログ行にリクエストIDが出ません

MDC に入れただけでは出ません。既定のログパターンに %X{requestId} が無いためで、src\main\resources\logback-spring.xml を新しく作り、パターンを %d{HH:mm:ss.SSS} [%X{requestId}] %-5level %logger{36} - %msg%n にしてから起動し直してください。

/api/health が 404 になります

Java のクラスは起動し直すまで反映されません。起動中のターミナルで Ctrl+C を押し、.\mvnw.cmd spring-boot:run を打ち直してください。それでも 404 なら、クラスの @RequestMapping とメソッドの @GetMapping を合わせたパスが /api/health になっているかを差分で確認してください。

dashboard.html を直しても表示が変わりません

アプリは止めなくて構いません。src\main\resources\static\ に置いた HTML・CSS・JavaScript は保存した時点で配信対象に入るので、まずブラウザを Ctrl+F5 で読み込み直してください。それでも古いままなら、保存先が src\main\resources\static\ になっているか、開いている URL が http://localhost:8080/dashboard.html かを見てください。