이요한_중급프로젝트 회고

개발 리포트 작성 항목

1. 프로젝트 개요

프로젝트의 목적

- 개인 맞춤형 뉴스 스크랩과 소셜 기능을 제공하는 스마트 뉴스 플랫폼인 Monew 서비스
- Monew는 다양한 뉴스 API 및 RSS로부터 뉴스를 수집하고, 사용자 관심사 기반으로 선별하여 제공하는 맞춤형 뉴스 스크랩 서비스입니다. 사용자는 관심사에 따라 뉴스를 구독하고, 기사에 댓글을 남기거나 좋아요를 누르며 소통할 수 있습니다. 사용자 활동을 MongoDB에 저장하여 조회 성능을 최적화하며, 기사 데이터는 정기적으로 AWS S3에 백업·복구됩니다.

2. 담당한 작업

> *프로젝트 내에서 본인이 맡은 역할과 기여한 부분을 구체적으로 기술해 주세요.*

- Spring Batch 기반 뉴스 기사 수집 및 알림 자동화
- AWS S3를 통한 기사 백업 및 복구 프로세스 구현
- 알림 자동 삭제 배치

3. 기술적 성과

> *프로젝트에서 사용한 기술 스택과 구현한 주요 기능을 설명해 주세요.*

✅ 기술 스택

분야	기술 스택
백엔드	Spring Boot, Spring Batch, Spring Data JPA, QueryDSL
데이터베이스	PostgreSQL, MongoDB
인프라	AWS S3, AWS ECS, GitHub Actions, ECR, MongoDB Atlas, RDS, CodeDeploy
테스트	JUnit 5, Mockito

✅ 구현한 주요 기능

1️⃣ 뉴스 기사 수집 후 S3백업 및 DB 저장 기능

- 전체적인 흐름 : 기사 수집 ➡ 필터링 ➡ S3백업 ➡ 객체 변환(Article + ArticleInterest) ➡ DB 저장
- 기사 수집 : 여러 사이트에서의 기사 수집을 Spring Batch의 Parallel Steps방식을 통해 병렬 처리
- 필터링 : DB저장된 것 포함 각 기사가 sourceUrl과 겹치지 않게하고 특정 키워드에 맞는 기사로 필터링
- S3 백업 : 엔티티로 변환 후 DB에 저장하는 chunk작업 전 FlatFileWriter를 사용해 S3에 백업
- 객체 변환 chunk 프로세싱

2️⃣복구 프로세스

- 클라이언트가 요청한 기간 내 기사 복구 로직
- 복구할 기사
  - 기간 내 논리삭제된 기사
  - S3에는 백업된 자료이지만 DB에는 저장 시 누락된 기사
- 전체적인 흐름 : DB에서 필터링에 필요한 자료들 수집 ➡ 논리 삭제된 기사 복구 ➡ 특정 기간 내 S3에 저장된 기사를 읽어와 필터링 후(sourceUrl비교를 통해) 유실된 데이터 백업

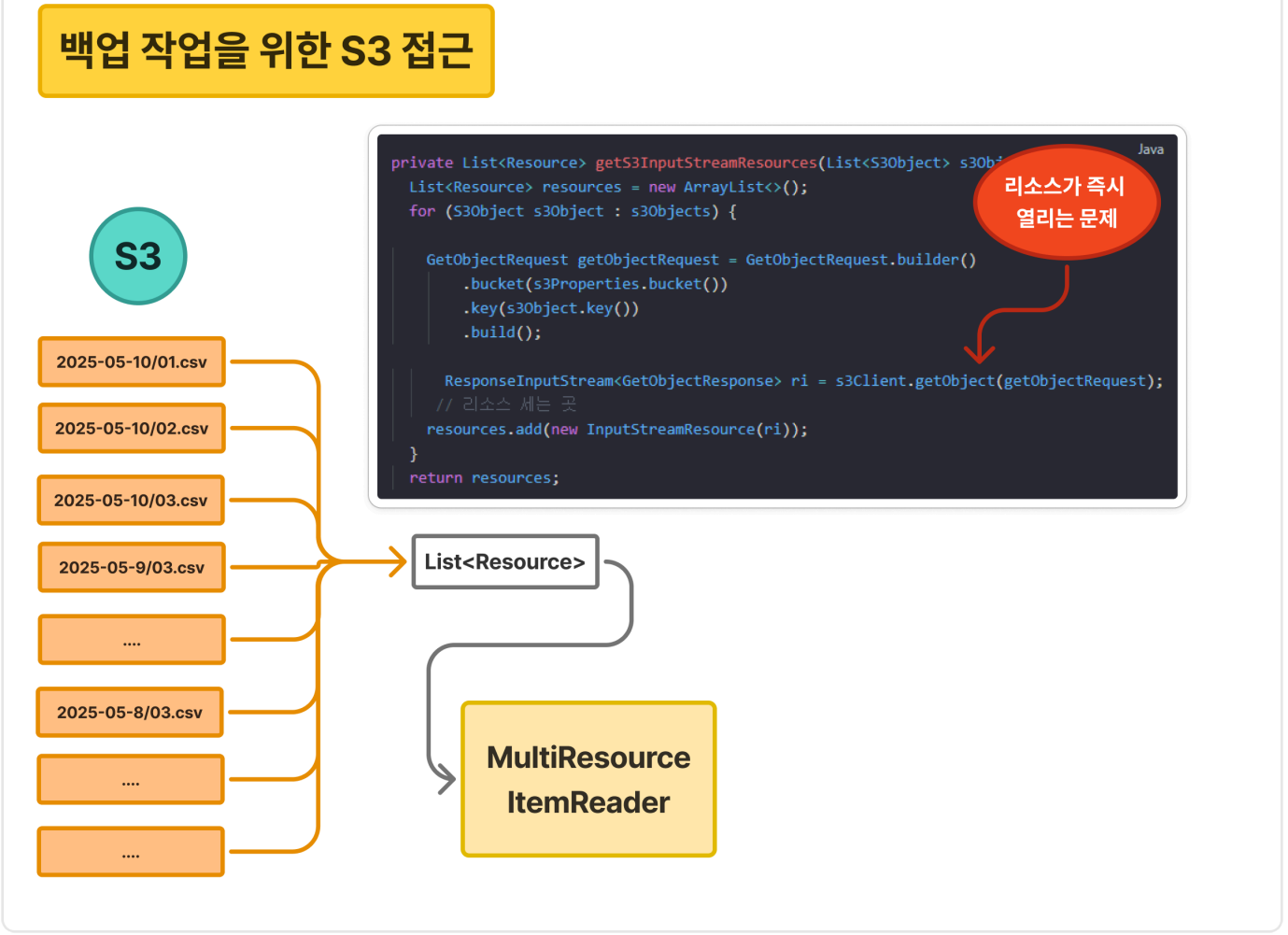
4. 문제점 및 해결 과정

> ⚠ ❌**문제점 1 - Resource 누수 오류 : 커넥션 풀 마르는 현상**

Spring batch를 활용한 백업을 하며 S3에 저장된 여러개의 CSV파일을 읽는 작업을 수행 시도하던 중,
Timeout 오류가 발생했습니다. 에러 로그를 확인해보니 커넥션 풀의 문제가 있음을 발견했습니다.

✅ 원인 분석

Section 23



S3객체를 연결하는 문제가 발생하는 코드는 다음과 같았습니다.

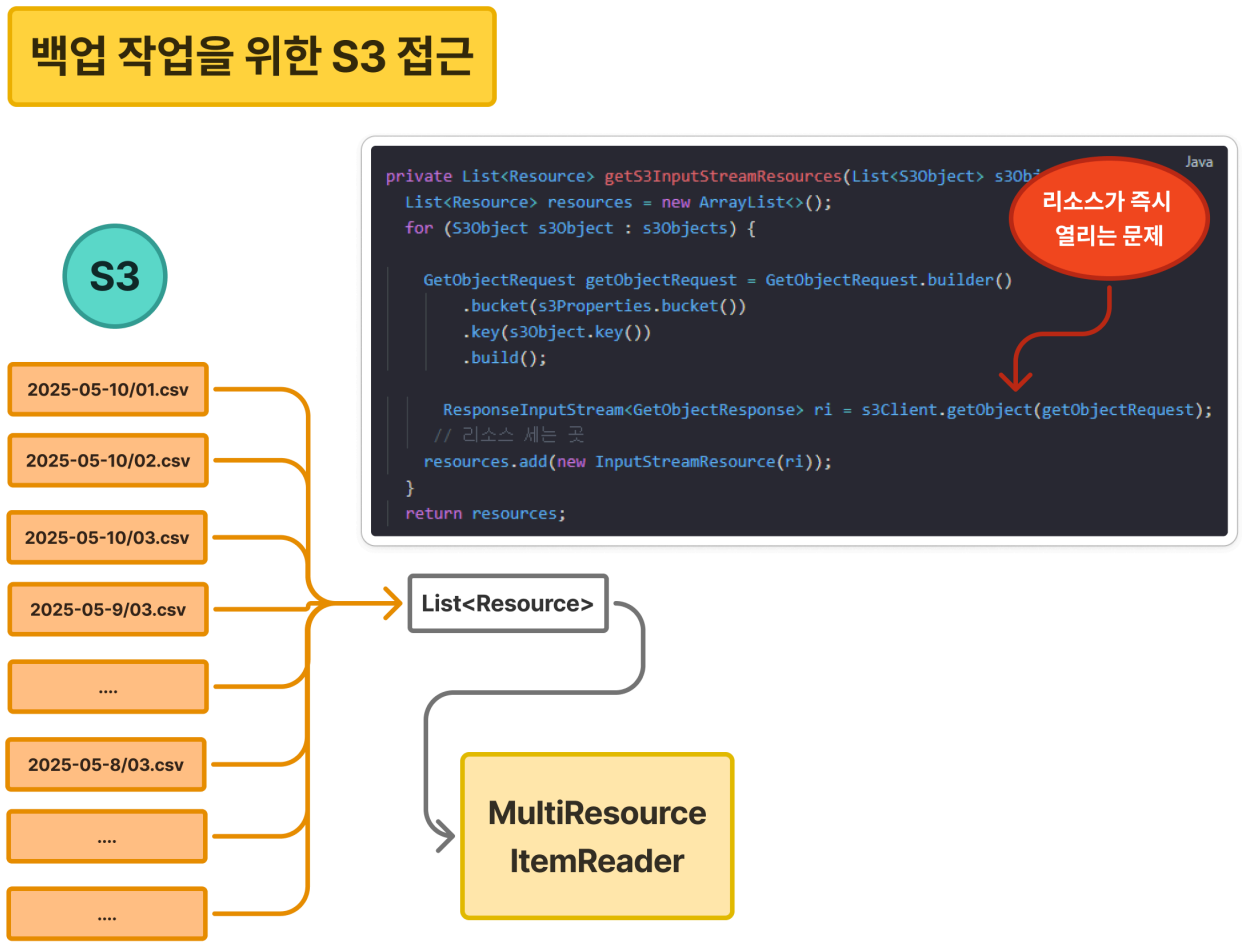
```java
private List<Resource> getS3InputStreamResources(List<S3Object> s3Objects) {
  List<Resource> resources = new ArrayList<>();
  for (S3Object s3Object : s3Objects) {
    GetObjectRequest getObjectRequest = GetObjectRequest.builder()
        .bucket(s3Properties.bucket())
        .key(s3Object.key())
        .build();
    ResponseInputStream<GetObjectResponse> ri = s3Client.getObject(getObjectRequest);
    resources.add(new InputStreamResource(ri));
  }
  return resources;
```

JAVA

이 코드는 `s3Client.getObject()` 호출 시점에 **즉시 S3와 연결을 열고 InputStream을 생성**합니다. 그리고 이 열린 스트림을 `InputStreamResource`로 감싸 리스트에 담아 `MultiResourceReader`에 전달하는 구조입니다.
문제는 `MultiResourceReader`가 파일을 처리하기도 전에, S3가 열린다는 것입니다. 100개의 s3Object객체라면 반복문에서 100개의 커넥션이 한꺼번에 열리는 구조로, 실행 도중 커넥션 풀이 말라버려 Timeout예외가 발생한 것입니다.
정리하자면, Spring bathc의 구현체가 여러 s3Obejct들을 다룰 때는 순차적으로 읽으며 리소스를 관리하지만 이 구현체에 Resource객체가 도달하기 전에 커넥션이 전부 열려 문제가 발생한 것입니다.

✅ 해결 - 지연 로딩 방식으로 Resouces저장

Section 23



JAVA

```java
private final ApplicationContext resourceLoader;

    private List<Resource> getS3InputStreamResources(List<S3Object> s3Objects) {
      List<Resource> resources = new ArrayList<>();
      for (S3Object s3Object : s3Objects) {
        String location = "s3://" + s3Properties.bucket() + "/" + s3Object.key();
        Resource resource = resourceLoader.getResource(location);
        resources.add(resource);
      }
      return resources;
```

결국 이 문제를 해결하기 위해서는 MultiResourceReader에 도달하기 전에 커넥션이 열리지 않도록 하는 것입니다. 이에 대한 해결 방법으로, 위의 코드와 같은 방법을 택했습니다.
이 방법은 Spring의 `ApplicationContext`를 통해 S3 리소스를 **지연 로딩 방식으로 처리**하는 것입니다.
이 방식으로 변경함으로써 Resource생성 시점에 커넥션이 맺어지지 않고, `MultiResourceItemReader`가 실제 `getInputStream()`을 호출할 때 S3 **커넥션이 순차적으로 열렸으며, 파일 읽기 종료 후 커넥션이 즉시 반환 처리가**되었습니다.
결과적으로 커넥션 풀 고갈 문제가 말끔히 해결되었고, **성공적으로 수십-수백개의 S3 객체를 처리하는 대용량 배치 작업이 안정적으로 수행**되었습니다.

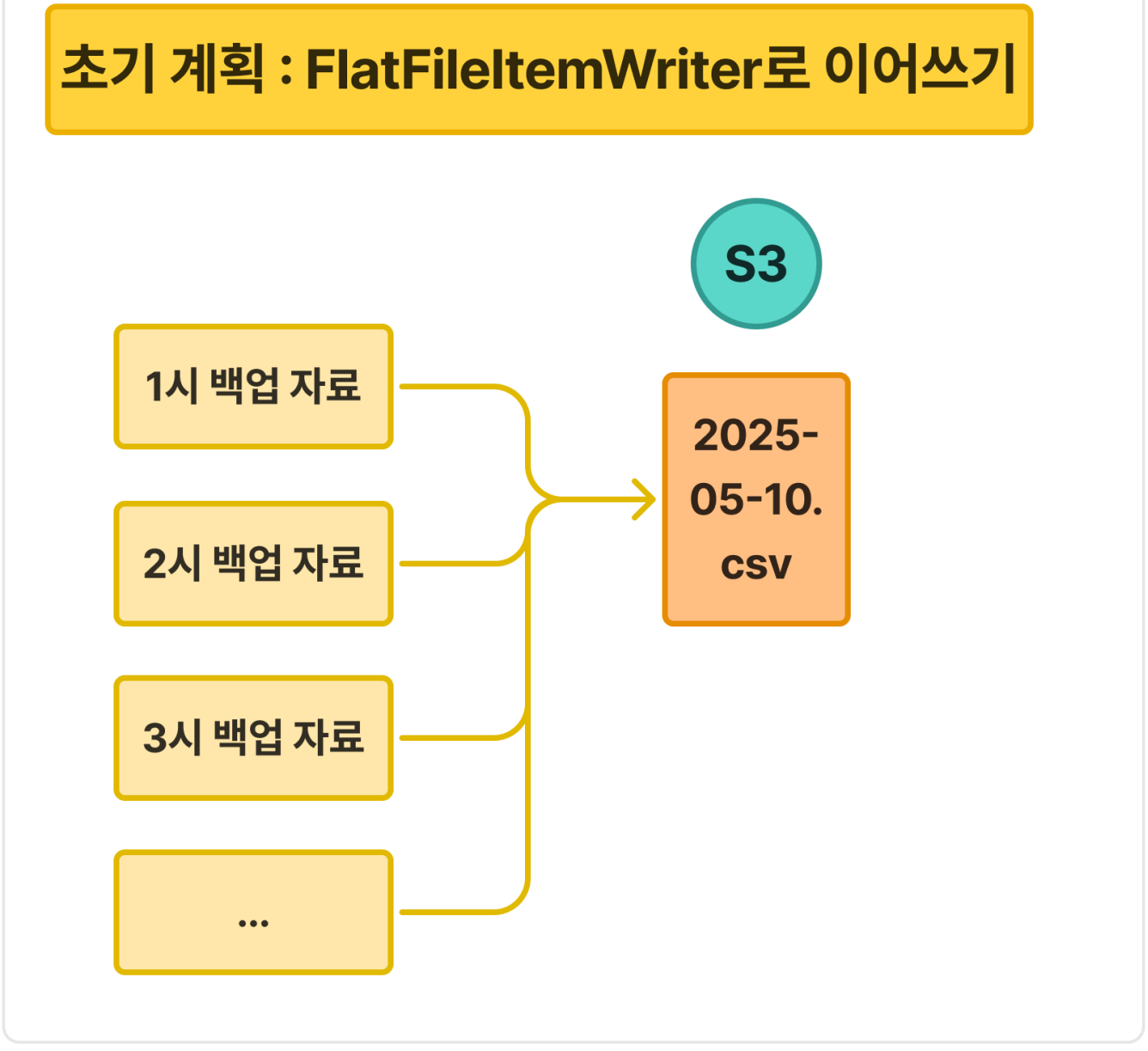
⚠ ❌문제점 2 - S3와 FlatFileWriter의 호환 문제

✅요구 사항 및 초기 설계 문제

- 요구사항 : 기사 데이터를 S3에 날짜별 기사를 백업
- 초기 설계
  - 하루 24번 수집되는 기사 데이터를 하나의 파일에만 저장하도록하여 추후 복구 시 S3 객체 접근 을 최소화하려고 시도

✅문제 : FlatFileWriter의 S3 직접 호환 불가

Section 21



- FlatFileWriter의 .append(true) 옵션을 써서 매 시간 기사 데이터를 S3의 하나의 파일에 저장 시도
- Error happens

```
java.lang.UnsupportedOperationException:
Amazon S3 resource can not be resolved to java.io.File.objects.Use getInputStream()
```
JAVA

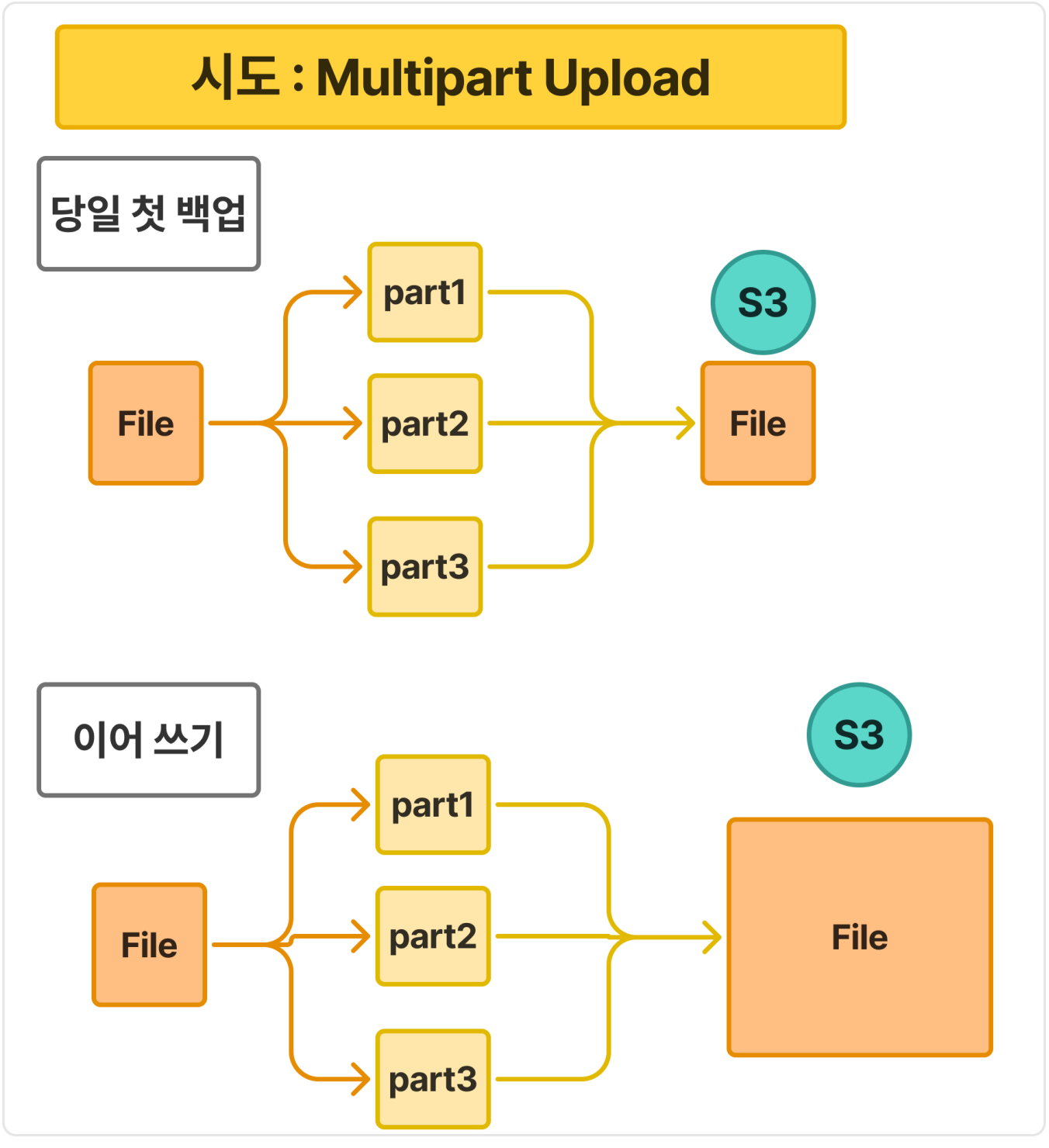
✅ 원인

- FlatFileItemWriter는 내부적으로 getFile() 메서드를 실행시켜 File 객체를 직접 참조하려한다.
- 하지만, S3Resource는 getFile()이 구현되지 않았다.
- 결국 FlatFileItemWriter 는 쓰기 대상이 S3인 경우 호환되지 않는 문제 발생

✅ 실패한 시도

시도. MultiPart Upload 방식 구현 시도

Section 20

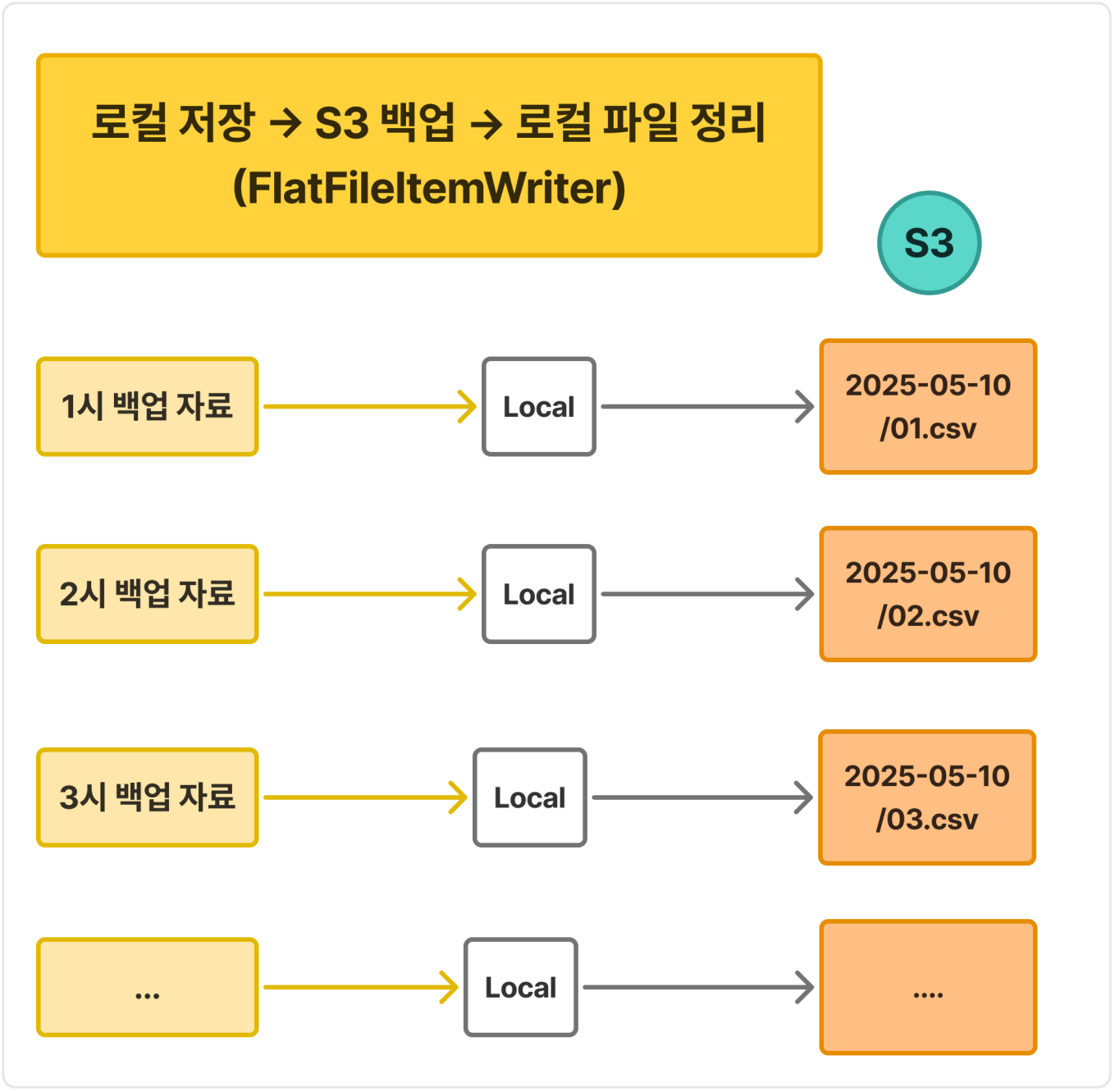


- S3에서 제공하는 Multipart Upload API를 사용하여 하루 하나의 파일에 매 시간 단위로 내용을 이 어붙이는 방식 구현을 시도
- 장점
  - 단일 파일 관리
  - 이어쓰기 및 부분 재전송 지원
- 한계
  - 러닝커브 : 이해하고 쓰기에는 학습곡선이 존재
  - 배치 작업 자체가 러닝커브가 있는 작업인데 MultiPart방법을 또 학습하기에는 시간 부족

✅ 해결

https://github.com/4monument/sb1-monew-team04/pull/127

Section 22



> *로컬 저장 후 S3 전송 + MultiResourceItemReader**

- **흐름**
  - 매 시간 기사 수집 후 FlatFileItemWriter 로 로컬 CSV 파일 생성
  - 완료된 파일을 S3에 업로드
  - 복구가 필요한 경우 해당 날짜 폴더 내의 모든 파일을 조회
  - MultiResourceItemReader 를 통해 여러 개의 CSV 파일을 하나의 Step에서 순차적으로 처리
- **장점**
  - 낮은 학습 곡선 : Spring-Batch 기본 구현체 그대로 사용
- **단점**
  - S3객체 수 증가

> *파일 수 증가라는 단점보다, 배치 구조 단순화 및 개발 일정 준수라는 장점이 더 크다고 판단했습니다,*

5. 협업 및 피드백

> *팀원들과의 협업 과정에서 느낀 점, 배운 점, 그리고 피드백을 기록해 주세요.*

협업

이번 프로젝트에서 가장 인상 깊었던 점은 **소통 중심의 협업 문화**였습니다.
이전까지의 경험에서는 간단한 소통 후 주로 정해진 역할을 맡아 각자 작업하늠 방식이었습니다. 그래서 IT분야에서 협업은 '말보다는 코드로 증명하는 것'이라는 생각을 막연히 갖고 있었습니다.
하지만 이번 팀에서는 **대화와 피드백을 중심으로 한 협업**이 활발하게 이루어졌습니다. 매일 시간이 오래 걸리더라도 확실히 피드백을 하고 기능 구현에 들어가며, 개발 도중에도 끊임없이 의견을 교환했고, 이를 통해 각자가 구현한 작업이 잘 맞물릴 수 있었습니다.
물론, 이런 방식으로 인해 개발 시간이 조금 줄어들 수 있었지만, 의도와는 다른 방향으로 개발되는 경우의 수를 줄일 수 있었다는 점에서 '오히려 효율적인 방식이구나' 라는 생각을 하게되었습니다.

피드백

아쉬운점 1 : 시간 부족

- 이번 프로젝트를 하면서 중간에 개인적인 외부 일정들이 생기다보니 프로젝트 중간부터 몰입을 못하게 되어서 아쉬웠습니다.
- 이로 인해, 개선할만한 포인트들을 인지하면서도 못하고 있는 점과 마지막에 팀원들과의 소통에 소홀히하게 됐던점이 아쉬웠습니다.

아쉬운점 2 : 문서화 부족

- RoadMap을 통해 이슈들의 진행 계획과 상황을 문서화하기로 정했지만 저의 게으름 때문에 중반부터 소홀히 하게 되었습니다.

6. 코드 품질 및 최적화

> *프로젝트 중 코드의 가독성과 유지보수성을 어떻게 고려했는지, 성능 최적화를 위한 작업을 설명해 주세요.*

1 일급 컬렉션과 비슷한 객체 사용

JAVA

```
@Slf4j
@Component
public class InterestContainer {

  private final List<Interest> interests = new ArrayList<>();
  private final Set<String> sourceUrlFilterSet = ConcurrentHashMap.newKeySet();

  (필터링 기능)
  public ArticleApiDto filter(ArticleApiDto articleApiDto) {
      if (isContainKeywords(articleApiDto.summary()) && ~~~ )
    return articleApiDto;
  }
  ...
```

```
  (관련된 Interest 매칭 후 변환 기능)
  public ArticleWithInterestList toArticleWithRelevantInterests(ArticleApiDto
articleApiDto) {
```

- `InterestContainer`는 단순 컬렉션 모음이 아니라, 필터링, 매핑, 관련 데이터 추출 로직을 캡슐화한 객체입니다.
- DB로부터 `Interest` 목록과 관련 URL을 가져와 이 객체로 초기화하며, 데이터와 그에 수반되는 **도메인 로직을 함께 관리**할 수 있도록 설계했습니다.
- 물론, 하나의 컬렉션만 포함하지는 않기 때문에 일급컬렉션은 아니지만, 일급 컬렉션의 철학인 불변성 유지, 책임 부여, 캡슐화를 따라 설계했습니다.
- 효과 : 응집도, 안정성, 유지보수성 상승 + 책임 부여

2 컨텍스트 저장 ➡ 싱글톤 패턴 적용

https://github.com/4monument/sb1-monew-team04/pull/171#issue-3040309101

초기 - 단순 batch Context사용

- 기사관련 배치작업을 할 때 Component가 아닌 pojo객체로 ExecutionContext에 저장해서 다른 Step과 공유를 했었습니다.
- 하지만 batch Context를 이용하는 것에는 몇가지 단점이 있습니다
  1. 직렬화 비용 : JSON ↔ String 변환 비용 大
  2. 타입 안정성 떨어지는 문제 : 직렬화를 통한다면 null 값 오류가 나올 수도 있고, Context에 집어넣거나 가져올 시 문자열로 직접 작성해야한다는 단점이 있습니다.

JAVA
```
  @Value("#{jobExecutionContext['hankyungArticleDtos']}")`
```

이후 - 스레드 안정적인 Singleton 객체

- 배치에서 Singleton을 사용하면 데이터가 오염될 위험이 있습니다 Cuz 동시성 이슈
- 하지만 스레드 안정적인 객체를 사용한다면 이 문제는 피할 수 있고, 성능적으로 이점을 얻을 수 있습니다.
- 필터링 및 객체 변환 시 도움을 주는 `InterestContainer` 객체는 일부 조심만 하면 동시성 이슈가 크게 없을 거라고 생각되어 Singleton 객체를 사용했습니다.
- 이를 위해 일부 변하는 부분만 Concurrent 자료구조를 만들어 SIngleton 객체를 공유하고 Job이 끝나면 bean을 정리하는 방식을 사용했습니다.

JAVA
```
@Slf4j
@Component
public class InterestContainer {

  private List<Interest> interests = new ArrayList<>();
  private final Set<String> keywords = ConcurrentHashMap.newKeySet();
  private final Set<String> sourceUrlFilterSet = ConcurrentHashMap.newKeySet();

  public void register(List<Interest> interests, List<String> sourceUrls) {
    clearBean();
    this.interests = interests;
```

```
    this.interests.stream()
```

3 결합도 줄이기

JAVA

```
[결합도 줄이기 전]
public static ArticleInterestJdbc create(Article article, Interest interest) {
  UUID id = UUID.randomUUID();
  return new ArticleInterestJdbc(
      id,
      article.getArticleId,
      interest.getInterestId,
      Instant.now()
  );
}


[결합도 줄인 후]
public static ArticleInterestJdbc create(UUID articleId, UUID interestId) {
  UUID id = UUID.randomUUID();
  return new ArticleInterestJdbc(
      id,
      articleId,
      interestId,
      Instant.now()
  );
}
```

- 초기 : Article, Interest 객체를 그대로 넘겨 그 객체들의 일부 필드만 활용했습니다.
- 문제점 : 불필요한 정보와 결합도 증가 문제
- 개선 : 객체, 자료구조를 파라미터로 통째로 넘기는 스탬프 결합도에서 자료 결합도로 바꿈으로써 결합도를 낮추고 리팩토링 및 확장 용이한 코드로 개선해봤습니다.

4 무리한 inline 지양, 의도 명시로 가독성 확보

과거

- inline variable을 최대한 활용해 코드를 줄이는 것을 선호했습니다.
- 하지만 유지보수성과 명시성을 고려하여 코드 스타일을 바꾸려고 노력했습니다.

이번 프로젝트

- 메서드 분리와 메서드명을 통해 코드가 조금 길어지더라도 의도를 명확히 하려고 노력했습니다.
- 또한 의도를 명확히 드러내는 변수를 만들기 위해 inline variable 사용을 지양했습니다.
- 가령, 아래 코드에서 .key() 부분에 String.format 코드를 바로 넣어도 되지만, 이렇게 분리함으로써 가독성 및 의도파악을 명시하고 유지보수성을 높이는 식으로 코드를 짰습니다.

JAVA

```
LocalDateTime now = LocalDateTime.now();
String s3Path = String.format("%s/%s.csv", now.toLocalDate(), now.getHour());
```

```
PutObjectRequest putObjectRequest = PutObjectRequest.builder()
    .key(s3Path)
```

5 JpaWriter ➡ JdbcWriter

https://github.com/4monument/sb1-monew-team04/pull/148#issue-3028470958

JAVA

```
[JdbcWriter]
@Bean
@StepScope
public JdbcBatchItemWriter<Article> articleJdbcItemWriter() {

  String articleInsertSql =
      "INSERT INTO articles (id, source, source_url, title, publish_date, summary,
deleted) "
          + "VALUES (?, ?, ?, ?, ?, ?, ?)";

  return new JdbcBatchItemWriterBuilder<Article>()
```

💢 JpaWrtier의 문제

- 초기에는 JpaItemWriter를 사용해 데이터를 저장하는 로직을 짰습니다.
- 하지만, 이는 bulk연산이 안돼 대규모 처리를 위해서는 적합하지 않다고 생각했습니다.
- 또한, 타 팀의 중간 발표에서 발견된 락관련 문제도 있던 것으로 발견됐습니다.

✅ JdbcWriter로 개선

- JpaWriter의 단점을 보완하고자 처음에는 배치가 제고하는 구현체들을 커스텀하려고 했으나 시간이 많이 부족해서 간단히 JdbcWriter로 개선했습니다.
- Jdbc는 bulk연산을 제공하기 때문에 처리 시간이 눈에 띄게 줄어드는 효과가 있었습니다.

6 배치에서 step끼리 자원 공유 시 PromotionListener

- 배치에서 step끼리 자원을 공유하는 방법에는 3가지가 있습니다.
  1. jobContext에 저장하기 ⬅ 기존에 쓰던 방식
  2. Singleton 객체 사용하기
  3. stepContext에 저장 후 PromotionListener 사용하기 ✅ 최적

⭐1 ➡ 3 으로 바꾼 이유

- JobContext에 바로 저장하지 말아야 되는 이유
  - 유실 위험 : Step이 중간에 실패하면 데이터가 유실될 위험이 있다.
  - 결합도 : Step이 해당 Job에 강하게 결합되어 재사용하기 어려워진다.
- 3번 방식은 스프링 공식 레퍼런스에서 추천하는 것으로 코드가 길어질 수는 있지만 job과의 결합도를 낮추고 안전한 방식입니다.

7. 향후 개선 사항 및 제안

개선할 수 있는 부분

1. **AOP 활용 : 수많은 배치 관련 빈들을 모니터링하는 로직 구현** ⭐ ⭐ ⭐
   - 구현된 배치 관련 빈들이 batch 디렉토리와 내부 config 디렉토리에도 많이 구현되어 있습니다.
   - 이런 빈들을 세세히 모니터링하기 위해 액츄에이터를 활용해 모니터링하려 했으나 비즈니스 로직이 더럽혀지는 문제가 발생해서 그만뒀습니다.
   - 하지만, 팀 프로젝트가 끝나고 생각해보니 그러한 **횡단 관심사**는 예전에 배웠던 **AOP를 활용**하면 해결할 수 있을거라는 생각이 들었습니다.
   - 만약, 시간이 남는다면 가장 해보고 싶은 개선 부분입니다.
2. **알림 생성 위한 DB조회 시 Chunk 프로세싱**
   - 알림 생성을 위해 DB에서 필요한 정보들을 가져오는 부분에서 Chunk단위로 작업을 할 수 있는 부분들이 있었습니다.
3. **백업 로직 시 방식 변경**
   - 현재 : 백업 시 로컬에 백업 후 FlatFileItemWriter를 사용해 S3에 백업
   - 개선 : 로컬 백업 없이 Multiupload 방식 구현
4. **알림 생성을 위해 가져온 DTO부분에서 결합도 낮추는 설계**
   - 현재 : `unReadInterest~~` 부분의 객체 전체를 파라미터로 전달 후 처리
   - 개선 : 필요한 데이터만 파라미터로 넘겨 결합도를 낮추기 (스탬프 결합도 ➡ 자료 결합도)
5. **JdbcWriter 개선 포인트1.** `CompositeItemWriter` 사용해서 로직을 더 깔끔하게 구성
6. **JdbcWriter 개선 포인트2.** 커스텀하기 : 시간이 많이 소요되는 작업
7. **직렬화 오류 체크 관련 로직 추가**
8. **Chunk 부분 예외 설계**