전태구

Software Engineer

010-4138-2383
devistiem@gmail.com
Github
Linked In

오늘의집

Trouble Shooting

지속적인 메모리 사용량증가 트러블슈팅

문제 상황

파드가 죽어 버렸다.
Java 앱 배포 후 시간이 지날수록 메모리가 증가되는 현상이 발생한다. 컨테이너 정보를 확인해 보니 `reason: OOMKilled`의 상태로 종료되었다.
쿠버네티스에서는 노드에 최대한 많은 Pod를 넣기 위해 요청(request)리소스와 제한(limit) 리소스를 설정할 수 있다. 리소스의 종류에는 CPU coremillis, Memory 등이 있다.

CPU의 경우 limit까지 사용한다면, CPU Throttling이 발생하고, 처리가 지연되지만, 메모리의 경우는 파드가 죽어버린다. 이때 파드 로그에는 OOMKilled만 남는다. 내가 겪었던 OOMKilled의 상황은 파드가 죽고 다시 뜨더라도 OOMKilled -> CrashLoopBack으로 이어졌다.

원인 분석

1. 점검 내역에 배포 연관 사항이 많아 원복을 할 수 없어서, 메모리 제한을 올려놓고 확인했다.
2. 먼저 k9s에서는 파드의 메모리는 약 2.5기가로 잡혔다. 평소에 1기가 정도 사용하던 것에 비하면, 2.5배 이상 사용하는 것이다. 이를 좀 더 정확히 확인하기 위해 컨테이너 shell로 접속하여 프로세스 메모리 정보를 확인했다.

```
$ pmap -x 1 | sort -k3 -nr | head -n 20
total             3888188  2604495  2604495       0
0000000083e00000  2480704  2348588  2348588       0  rw-p    [ anon ]
0000ffff78601000    28564    28500    28500       0  rw-p    [ anon ]
...
```

* 주소 0000000083e00000에서 2.2GB가량 사용중이다.

3. 프로세스 자체에서도 많이 잡고 있어서 추가적으로 `jcmd 1 VM.native_memory summary`로 확인을 때도 Heap 사용량이 상당히 높아 보였다. 확신을 가지고 정확히 어떤 객체가 메모리를 점유하는지 객체별 메모리 사용 상황을 확인했다.

```
$ jmap -histo 1 | sort -k3 -nr | head -n 20
Total         398315      2557083624
...
*:           125002         841397512  org.cache2k.core.Entry
*:           24342          541913616  *.*.*.MonthlyRevisedRanker
...
```

* MonthlyRevisedRanker 객체가 상당량 할당되게 보였고, 지속적으로 증가되고 있었다.
로컬 캐시는 Cache2k를 사용하기 때문에, 캐시 엔트리도 같이 증가된게 보인다.

전태구

Software Engineer

010-4138-2383
devistiem@gmail.com
Github
Linked In

오늘의집

Trouble Shooting

지속적인 메모리 사용량증가 트러블슈팅

4. MonthlyRevisedRanker 객체로 범위가 좁혀지면서, 해당 객체가 왜 사라지지 않는지, 관련 커밋과 비교하며 모두 보았다.

```
@Cacheable("monthly_revise_rankers")
public List<MonthlyReviseRanker> getMonthlyRevisedRankers(Integer offset) {
    // offset 값을 이용해 레디스 캐시에서 엔터티를 조회하고 VO로 바꾸어 로컬 캐시 하는 로직
    ...
}
```

* 문제의 코드. 파라미터로 조회 조건을 걸고 동시에 결과를 파라미터로 캐시한다.

5. 내부 로직은 문제가 없었다. 분명 QA 진행에서도 비슷한 이슈는 나오지 않았었다. 이 로직에서는 캐시 키가 변경되지 않는 한 있는 값을 사용한다. Kibana에서 해당 조회 API 전체 Access 로그(요청 당 1회 로깅) 요청 정보를 보았다.

   각 요청에 대한 offset 값은 다르게 들어왔고, 계속 다른 캐시 키로 저장될 수밖에 없었다. 상황을 확인해 보니 해당 랭킹 페이지는 최상위 100개만 보이는 스펙이였고, 클라이언트에서도 실제로 100개까지만 노출되었다.

   문제는 클라이언트에서 스크롤을 내리는 경우 API를 계속해서 호출했고, 이때 스크롤을 내릴때 마다 offset 값을 증가시켜서 요청했다. 이미 랭킹 정보 100개가 랜더링 된 클라이언트는 응답 랭킹 목록을 보여주지 않았기 때문에, QA 단계에서도 발견할 수 없었던 것이었다.

해결 과정

클라이언트를 수정하려면 앱 심사를 다시 받아야 해서, 임시방편으로 API의 응답 값을 수정했다. 먼저 offset 값이 범위를 초과하는 값으로 들어오면 앞단에서 빈 배열을 응답하도록 처리했다.

```
// 페이징 범위가 아닌경우 빈배열 반환
if (isNotPageRange(offset) {
    return ofEmptyRankers();
}

List<MonthlyReviseRanker> rankers = rankerCacheService.getMonthlyRevisedRankers(offset);
...
```

코드 추가 후, 메모리 사용량은 증가되지 않았고, 서비스는 다시 안정화되었다.

회고

실제로 대응한 게 저 if 문 세 줄 코드라서 스스로가 부끄러워졌다. 응답 값이 비어있는지만 확인했지 기획서의 랭킹범위 대로 페이지 오프셋까지 검증하는 건 생각하지 못했다.

조금 이해가 안 가는 부분이 랭킹 페이지의 끝이 이미 보이는 화면에서 스크롤을 계속 내려 offset 값을 계속 증가시켰는지가 신기할 따름이었다.
직접 테스트해봤을 때도, 스크롤을 내려야지만 값이 올라갔다. 그 말은 즉 끝이 이미 보인 스크롤을 계속 내렸다는 의미였다.

* 추측하기론 iOS의 튕기는? 스크롤이 아닌가한다.

전태구

Software Engineer

010-4138-2383
devistiem@gmail.com
Github
Linked In

오늘의집

Personal Project

Opan API 스팩 컨버터: openapi-resource-converter

OpenAPI 스펙으로 출력되는 openapi.json 파일을 다양한 자원으로 변환할 수 있는 Typescript 라이브러리

프로젝트 시작 동기

- 회사에서 Postman 컬렉션 파일을 수동으로 만들어서 관리하고 있었습니다.
  굉장히 불편했던 부분이 API 하나 만들면 포스트맨 컬렉션에 HTTP 요청을 만들고, URL, 요청값을 모두 수동으로 기입하고 이걸 내보내기로 뽑은 JSON을 버져닝 하더라고요.
  이게 한개의 프로젝트를 여러명에서 수정하면, 그 컬렉션 버전 관리에 대한 충돌까지도 병합해야해서 생산성 손실이 이만저만이 아니였습니다.
  회사에서는 이미 하고 있는 일도있고, 제 마음대로 만들수 없어서 개인프로젝트로 시작 했습니다.

프로젝트 소개

- 개인 프로젝트로 만들어서, 파이프라인으로 시연해서 사용하는 것을 보여주고 실내 일부 팀내에서 공식적으로 사용할 수 있게 되었습니다.
  현재도 구성원들의 업무시간을 **일일 약 3시간 (20분 * 10명) 절약**하고 있습니다.
- 프로젝트를 시작하고 어떻게하면 더 효율성있게 사용할 수 있을까 생각해보다 빌드 단계에서 만들어진 openapi.json 파일을 Postman Collection 파일로 변환후,
  호스팅하면 자동화 할수 있을것 같아 파이프라인 내에서도 가볍게 설치할수있는 Typescript 로 만들었습니다.
- 커뮤니티에 소개한다고 프로젝트 README를 영어로 작성했습니다. 추후 한글로 변경 예정입니다.

개선이 필요한 점

- 현재는 설정을 읽고 설정대로 변환을 해야하기 때문에 스크립트가 필요 하지만 추후에는, 설정파일 yaml 읽어서 처리할수 있는 Command line Tool로 만들려고 하고있습니다.

엔터티 프린터: entity-printer

POJO 객체를 간단한조작으로 직관적인 시각화를 지원하는 JAVA 라이브러리

프로젝트 시작 동기

- 로컬 및 개발 환경에서, 테스트시 toString() 및 Gson으로 출력해도 눈에 한번에 들어오지 않아, 좀 더 익숙한 형태로 볼 없을까 생각해보다 시작하게 되었습니다.

프로젝트 소개

- 개인 프로젝트에서도 사용중이고, 회사에서도 백오피스 개발시 의존성만 임시로 추가해서 자주 사용합니다. POJO 객체, Map 모두 지원하며 컬렉션도 사용가능 합니다.
  기본적으로 매우 간결한 사용 방법이 특징이에요.

```
List<Coordinate> coordinates = geometryService.getCoordinates();
String result = printer.drawCollection(coordinates, Coordinate.class);
```

전태구

Software Engineer

010-4138-2383
devistiem@gmail.com
Github
Linked In

Personal Project

```
System.out.println(result)
/** 출력 결과
* +--------------+---------------+
* | x (double)     y (double)    |
* +--------------+---------------+
* | 21.239832931   293.29391239  |
* +--------------+---------------+
* | 732.2182192    706.293213243 |
* +--------------+---------------+
*/
```

- 프로젝트 README 내 에는 다양한 옵션을 활용하여 사용하는 사례가 설명 되어 있습니다.

오프셋 페이징 유틸: offset-paginator

Offset Paging 을 쉽게 쓸 수 있는 Java 라이브러리

프로젝트 시작 동기

- 개인 프로젝트를 하며, 페이징 로직이 필요할 때 마다 만들수 없고, 페이징 종류, 사용하는 데이터베이스 모두 다를때가 있어 이를 지원하기 위해 만들었습니다.

프로젝트 소개

- 페이징을 위한 인터페이스는 간결하게 만들었습니다.

```
PaginatedObject result = new OffsetPaginator(boardList.size(), currentPage)
                .elastic()
                .move(isPre, isNext)
                .build()
                .paginate();
```

- HTML 과 기본적인 부트스트랩 css 코드도 제공이 됩니다.
  호스팅하면 자동화 할수 있을것 같아 파이프라인 내에서도 가볍게 설치할수있는 Typescript 로 만들었습니다.
- 프로젝트 README 내 에는 이해하기 쉽도록 GIF 이미지와 설명을 같이 넣었습니다.

개선이 필요한 점

- 현재는 OffsetPaginator 객체 하나로 만들어서 사용중인데, 각 페이징 방식(elastic, fixed) 별로 페이지네이터를 따로 만들고 build() 메소드와 paginate() 메소드 없이 static 메소드로 직관적인 인터페이스 개선이 필요해 보입니다.

전태구

Software Engineer

010-4138-2383
devistiem@gmail.com
Github
Linked In

오늘의집

Skill

Language

보유기술	• Java 17, Typescript, Kotlin(배우는중)
응용지식	• 1.8 (Lambda, Stream, Optional)에 대한 내부 동작인 ReferencePipeline의 operation에대한 이해와 활용 • Java Compile Time 내 Annotation Processor 의 동작의 이해와 활용 • 그 외 Reflection 활용, GC(JVM) 내 Reference Reachability 에 따른 객체 수명주기 이해 • Typescript 및 Node 이벤트 루프에 대한 이해

Framework

보유기술	• Spring Boot, Spring Data JPA/Hibernate, Spring MVC, Vue 3, Nuxt, React(하)
응용지식	• Spring core 의 IoC, DI, AOP에 대한 동작 이해 • Spring Boot 의 자동설정 원리 와 활용 • MVC 구조와 동작원리 • Spring Data JPA/Hibernate 내부 동작원리(Persistence context)

Database

보유기술	• MySQL, DynamoDB, Redis, Rabbit MQ
응용지식	• ANSI SQL • 기본키 B+TREE, index BTREE 구조 이해 및 활용 • Index Node 페이징 이해 • MVCC & Isolation 의 이해

Network

응용지식	• OSI 7 Layer(Physics, Switching, Routing, Port Forwarding, Application), NAT • TCP/IP 3handshake • TLS (4 handshake), RESTful(Application Architecture) • HTTP Protocol

Infrastructure

함께 사용하는 기술	• Git, Jira • Docker, Kubernetes, Helm • AWS ECR, EKS, S3, CF, API Gateway 등등 • ElasticSearch(OpenSearch), Grafana, Prometheus