주문 프로세스 성능 측정

1. 들어가며

프로젝트 런칭 후 운영하던 과정에서 한 가지 이슈가 발생했습니다. 키오스크에서 상품을 주문하는데 시간이 너무 오래 걸려서 이용이 불편하다는 내용이었습니다. 그래서 긴급하게 키오스크와 연결하여 상품 주문을 테스트했더니, 정말 주문하는데 시간이 오래 걸리는 것을 확인할 수 있었습니다. 그래서 개발팀에서 원인을 찾기위해 프로세스를 분석하고 원인을 찾아 해결하고 패치하는 과정을 거쳐 정상화 되었습니다.
하지만 패치하기까지 2~3일 정도 소요되었고, 패치 이후 "그럭저럭 안 느리네" 과 같은 불확실한 부분이 마음에 걸려 방법을 고민해보았습니다.

'디버깅 시간을 줄이고**,** 코드의 성능을 더 정확하게 분석할 수는 없을까**?'**
'패치 이전과 패치 이후를 비교했을 때 성능이 어느정도 개선된 걸까**?'**

위와 같은 문제점들을 해결하기 위해, 향후 이런 문제들이 발생하지 않게 하기위해서 어떤 일들을 할 수 있는지 고민한 결과, 몇가지 방법을 찾을 수 있었고, 프로젝트에 적용했습니다.

2. 정확한 성능 측정

처음 이슈가 접수되었을 때, 저는 복잡하고 긴 프로세스 로직 중 어떤 곳에서 시간이 많이 소요되는지 전혀 알 수가 없어서 한참 고민했습니다. 일반적으로 생각하면 특정 쿼리의 수행시간이 지나치게 길어 그럴 수 있다는 의심은 했지만 어떠한 지표도 없고 확실하지 않았습니다. 그래서 우선적으로 고려한 것이 정확한 성능 측정입니다.

```
@RequestMapping(value = "insertSpOrderKi.jspx", method = { RequestMethod.GET,
RequestMethod.POST })
public ModelAndView insertSpOrderKi(HttpServletRequest request,
HttpServletResponse response) throws Exception {
  StopWatch stopWatch = new StopWatch();
  stopWatch.start();

  StopWatch dataCollectionStopWatch = new StopWatch();
  dataCollectionStopWatch.start();
```

먼저, 주문 메소드 전체 수행시간을 체크하는 타이머를 지정합니다. 그리고 클라이언트에서 넘어온 주문정보들을 변환하고 수집하는 성능을 측정하는 타이머를 별도로 설정합니다.

```
dataCollectionStopWatch.stop();
LOGGER.info("---------------------- insertSpOrderKi data collection time.
-----------------");
LOGGER.info(dataCollectionStopWatch.prettyPrint());

StopWatch dbProcessStopWatch = new StopWatch();
dbProcessStopWatch.start();
```

변환 및 수집 작업이 끝나는 부분에서 data collection(변환 및 수집) 타이머를 종료시키고 예쁘게(?) 출력한 뒤 DB I/O 작업 성능을 측정하는 타이머를 설정합니다.

```
dbProcessStopWatch.stop();
LOGGER.info("---------------------- insertSpOrderKi DB I/O time
-----------------");
LOGGER.info(dbProcessStopWatch.prettyPrint());

stopWatch.stop();
LOGGER.info("---------------------- insertSpOrderKi response time
-----------------");
LOGGER.info(stopWatch.prettyPrint());
```

이후 DB I/O 타이머와 메소드 첫부분에서 선언한 전체 타이머를 종료시킨 뒤 마찬가지로 출력합니다.

데이터를 수집·변환하는 과정과 DB에 I/O 하는 과정으로 나누어 각각의 성능을 측정하고 이를 합친 전체 성능을 측정하여 메소드의 어떤 작업에서 시간이 얼마나 소요되는지 알 수 있습니다.

[10-30 16:27:59][INFO](SpOrderController.java:3932) - ---------------------- insertSpOrderKi data collection time. -----------------
[10-30 16:27:59][INFO](SpOrderController.java:3933) - StopWatch '': running time (millis) = 129
---
ms % Task name

00129 100%

[10-30 16:28:05][INFO](SpOrderController.java:4148) - ---------------------- insertSpOrderKi DB I/O time -----------------
[10-30 16:28:05][INFO](SpOrderController.java:4149) - StopWatch '': running time (millis) = 6208
---
ms % Task name

06208 100%

[10-30 16:28:05][INFO](SpOrderController.java:4221) - ---------- insertSpOrderKi response time ----------
[10-30 16:28:05][INFO](SpOrderController.java:4222) - StopWatch '': running time (millis) = 6338
---
ms % Task name

06338 100%

실제 로그에 기록된 성능측정 로그입니다. 살펴보면 data collecting 과정은 0.129초, DB I/O 과정은 0.6208초 소요되었음을 알 수 있습니다.
이렇게 프로세스를 나누어서 타이머를 기록하게되면 어느 부분을 개선해야하는지 쉽게 파악할 수 있습니다.
여태까지의 내용은 하나의 메소드에 대한 성능측정입니다.
현재 프로젝트에 구현되어있지는 않지만, Spring AOP 기능을 활용하여 프로젝트 전체에 걸쳐 메소드별로 성능을 측정하고 일정시간 이상 걸리는 메소드가 발견될 시 슬랙, 텔레그램, 이메일등의 수단으로 알림을 보내는 기능으로 확장할 수도 있습니다.