# 개요

## 목적

본 문서는 Nginx, Tomcat, Oracle으로 구성된 시스템에서 HTTP 502 Bad Gateway 알림이 대량 발생한 시나리오 상황에 대해, 장애를 재현하기 위한 과정, 발생한 장애에 대한 Root-Cause 분석, 긴급 조치 및 영구 조치를 포함한 패치(Hot-Fix 제시)에 대한 내용을 기술하는 것을 목적으로 한다.

## 범위

본 문서는 장애 재현 및 패치 제시에 관한 문서로서 다음과 같은 내용을 포함한다.

* Docker 기반 시나리오 시스템 구성 방법
* JMeter 도구를 활용한 스레드 고갈 현상 재현
* 장애가 발생한 레이어 식별
* 패치(Hot-Fix) 제시

## 과제 수행 환경

|  |  |
| --- | --- |
| 구분 | 종류 |
| CPU | Inter® Pentium® CPU G4560 3.50GHz |
| RAM | 8.00 GB |
| OS | Windows 10 Pro |
| 저장공간 | 128GB SSD |

# 시나리오 구성

## Docker Image download

Docker 컨테이너를 생성하기 위해 다음과 같은 명령 구문을 실행하여 Nginx, Tomcat, Oracle에 대한 각각의 이미지를 Download 하고, 각 컨테이너를 연결하기 위한 네트워크를 생성한다.

|  |  |
| --- | --- |
| 분류 | 명령 구문 |
| Nginx | docker pull nginx:1.24 |
| Tomcat | docker pull tomcat:8.5.93-jdk8 |
| Oracle | docker pull truevoly/oracle-12c |
| Network | docker network create **testNetwork** |

## Docker Container 생성

Download가 완료된 이미지들을 기반으로 각각의 컨테이너를 생성한 후 실행한다.

|  |  |
| --- | --- |
| 컨테이너 명 | 명령 구문 |
| testNginx | docker run -d -p 80:80 --name testNginx --network testNetwork nginx:1.24 |
| testTomcat | docker run -d -p 8080:8080 --name testTomcat --network testNetwork --pids-limit=200 tomcat:8.5.93-jdk8 |
| testOracle | docker run -d -p 1521:1521 --name testOracle --network testNetwork truevoly/oracle-12c ./entrypoint.sh |

## Docker Container 세부 설정

### Nginx가 받은 Http 요청을 Tomcat server으로 보내기 위한 설정 추가

|  |
| --- |
| testNginx:/etc/nginx/conf.d/default.conf |
| server {  listen 80;  server\_name testTomcat;  location / {  proxy\_pass http://testTomcat:8080;  proxy\_set\_header Host $host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  proxy\_set\_header X-Forwarded-Proto $scheme;  proxy\_connect\_timeout 5s;  proxy\_read\_timeout 5;  proxy\_send\_timeout 5s;  }  } |

|  |
| --- |
| testNginx:/etc/nginx/conf.d/default.conf |
| upstream tomcat\_backend {  server testTomcat:8080 max\_fails=1 fail\_timeout=1s;  } |

### Tomcat server에 maxThread=200 설정 추가 및 Executor 추가

|  |
| --- |
| testTomcat:/usr/local/tomcat/conf/server.xml |
| <Executor name="tomcatExecutor"  namePrefix="catalina-exec-"  maxThreads="200"  minSpareThreads="50"  maxIdleTime="60000"  threadPriority="5" />  <Connector port="8080" protocol="HTTP/1.1"  connectionTimeout="20000"  redirectPort="8443"  maxThreads="200"  acceptCount="100"  maxConnections="200"  maxParameterCount="1000"  executor="tomcatExecutor"/> |

### Tomcat server이 Oracle12c DB와 연결하기 위한 구문을 추가하고, maxActive=50 설정

|  |
| --- |
| testTomcat:/usr/local/tomcat/conf/context.xml |
| <Resource name="jdbc/OracleDS"  auth="Container"  type="javax.sql.DataSource"  driverClassName="oracle.jdbc.OracleDriver"  url="jdbc:oracle:thin:@testOracle:1521:xe"  username="scott"  password="tiger"  maxTotal="50"  maxIdle="10"  maxWaitMillis="10000" /> |

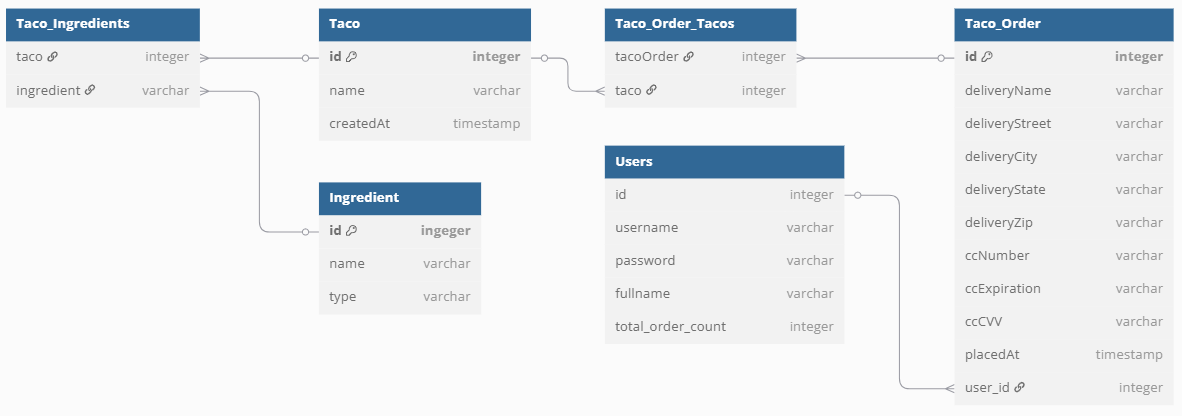
### Oracle12c DB 기본 계정 설정 및 기존 테이블 제거

|  |
| --- |
| testOracle:sqlplus sys as sysdba |
| Enter password : oracle  alter user scott account unlock;  grant connect, resource, dbat to scott;  conn scott;  Enter password:tiger  ERROR:  ORA-28001: the password has expired  Changing password for scott  New password: tiger  Retype new password: tiger  Password changed  Connected.  BEGIN  FOR t IN (SELECT table\_name FROM user\_tables) LOOP  EXECUTE IMMEDIATE 'DROP TABLE ' || t.table\_name || ' CASCADE CONSTRAINTS';  END LOOP;  END; / |

# 장애 재현을 위한 소프트웨어 구현

Tomcat 8.5.93 서버에서 실행될 소프트웨어를 Spring Boot 기반으로 구현한다. 12개의 기본 계정과 10개의 기본 재료가 포함되어 있으며, 기능이 실행될 때마다 랜덤한 유저가, 랜덤한 타코를 랜덤한 개수로 주문하는 것을 주 기능으로 한다. 각 타코에는 여러 개의 재료가 들어갈 수 있으며, 한 사람이 한 번의 주문에서 여러 개의 타코를 주문할 수 있다.

## DB Schema



*(상세 SQL은 첨부파일의 schema.sql 참고)*

## Spring Boot 소프트웨어 구현

소프트웨어를 구현하기 위해 아래의 파일들을 Maven 기반 Spring Boot 어플리케이션으로 구현한다. 빌드를 통해 생성된 war 파일은 testTomcat 컨테이너의 /usr/local/tomcat/ webapps 폴더 내에 복사한다. *(상세 소스 코드는 첨부파일 testProject.zip 참고)*

|  |  |
| --- | --- |
| 파일명 | 목적 |
| RandomOrderController.java | 랜덤 주문을 실행하기 위한 컨트롤러 클래스 |
| Ingredient.java | Ingredient(재료) 모델 클래스 |
| Order.java | 주문 모델 클래스 |
| Taco.java | 타코 모델 클래스 |
| Users.java | 사용자 모델 클래스 |
| IngredientRepository.java | DB와 통신하기 위한 Ingredient JPA Repository 클래스 |
| OrderRepository.java | DB와 통신하기 위한 Order JPA Repository 클래스 |
| TacoRepository.java | DB와 통신하기 위한 Taco JPA Repository 클래스 |
| UsersRepository.java | DB와 통신하기 위한 User JPA Repository 클래스 |
| Literals.java | 상수를 저장하기 위한 클래스 |
| RandomUtils.java | 랜덤 주문에 필요한 난수 생성 기능 구현 클래스 |
| ServletInitializer.java | 어플리케이션을 외장 Tomcat에서 실행하기 위한 설정 정보 전달 |
| TestProjectApplication.java | 어플리케이션 시작점 |
| application.propeties | 애플리케이션의 DB 연결(JNDI)과 JPA 설정을 정의 |
| schema.sql | 어플리케이션이 시작될 때 실행될 SQL 쿼리 파일 |

|  |  |
| --- | --- |
| 명령어 | 목적 |
| docker cp testProject.war /usr/local/tomcat/webapps/testProject.war | 빌드가 완료된 war 파일 복사 |

## 컨테이너 및 어플리케이션 정상 작동 확인

각 컨테이너가 정상적으로 작동 중인지 순차적으로 확인한다. 정상적으로 작동하는지 확인하는 방법은 다음과 같다.

|  |  |
| --- | --- |
| 컨테이너 | 확인 방법 |
| testOracle | SqlDevelpoer 프로그램을 통해 Oracle에 정상적으로 접속되는지, DB가 정상적으로 생성되었는지 확인 |
| testTomcat | 다음 명령어를 testTomcat 컨테이너에서 실행하여 주문 생성 결과를 JSON으로 반환하는지 확인한다.  curl http://localhost:8080/testProject/order |
| Docker 외부 브라우저에서 아래 URL에 접속해 주문 생성 결과를 JSON으로 반환하는지 확인한다.  http://localhost:8080/testProject/order |
| testNginx | 다음 명령어를 testNginx 컨테이너에서 실행하여 주문 생성 결과를 JSON으로 반환하는지 확인한다.  curl http://localhost/testProject/order |
| Docker 외부 브라우저에서 아래 URL에 접속해 주문 생성 결과를 JSON으로 반환하는지 확인한다.  http://localhost/testProject/order |
| Spring application | spring-data-rest에 의해 각 모델에 대한 Rest API가 정상적으로 생성되었는지 확인하기 위해 다음 URL에 접속하여 각 URL이 정상적으로 작동하는지 확인한다.  http://localhost/testProject |

# JMeter 어플리케이션을 사용한 스레드 고갈 현상 재현

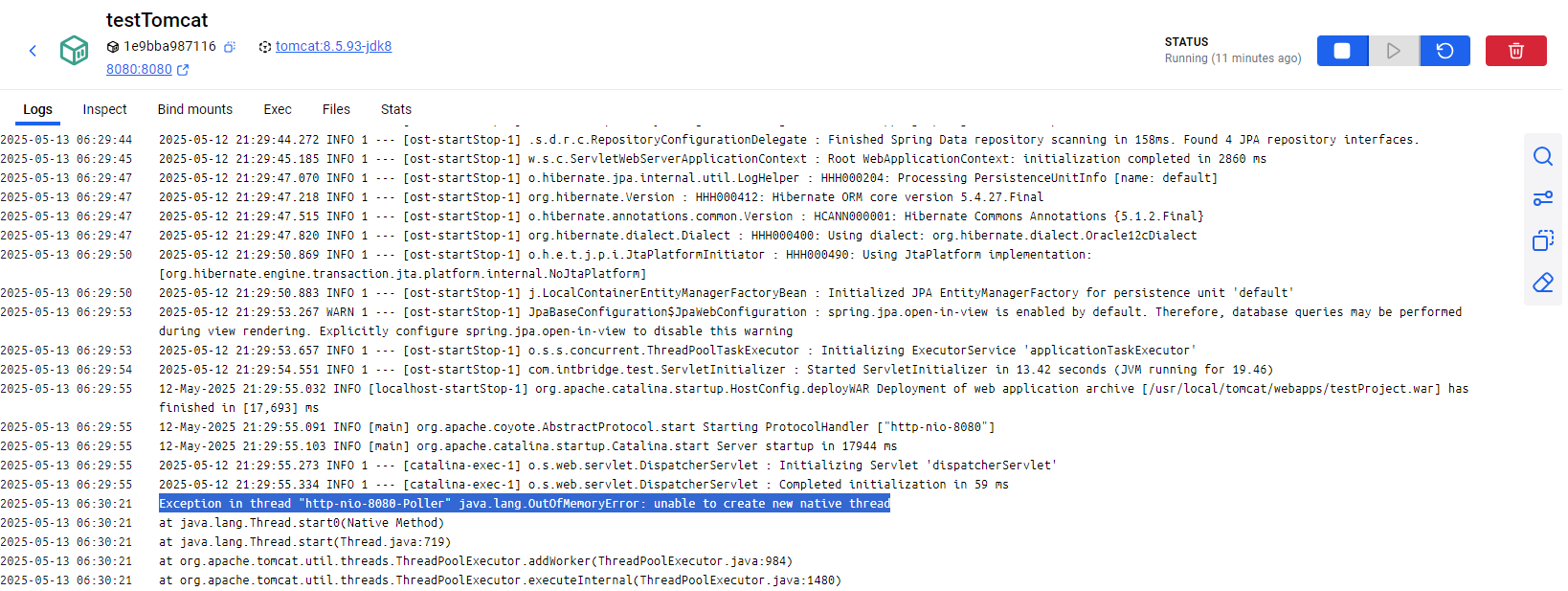
## 테스트 플랜(Test Plan) 작성

JMeter 어플리케이션을 실행 후, 테스트 플랜을 다음과 같이 작성한다. 특정 시간에 음식 주문이 몰리는 것을 표현하기 위해 180초간, 매 초 40명의 사용자가 URL을 통해 각각 타코를 주문하기 위한 Http 요청을 전송한다. 각 Http 요청에 대한 결과를 알 수 있도록 결과들의 트리 보기(View result tree) 리스너를 추가하고, 전체 결과를 한 번에 볼 수 있도록 요약 보고서(Summary report) 리스너를 추가한다.

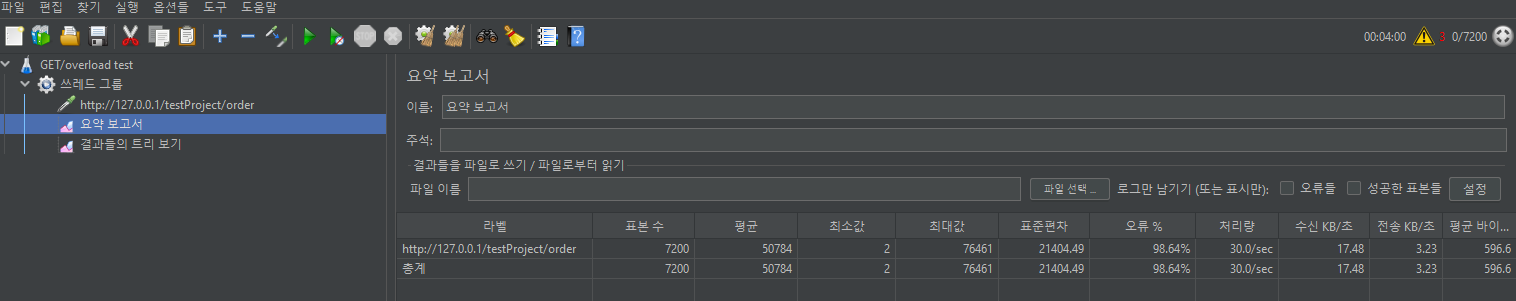
|  |  |
| --- | --- |
| 항목 | 내용 |
| 쓰레드 그룹 (Thread group) | Number of Threads (users) : 7200 |
| Ramp-Up Period (seconds) : 180 |
| Loop Count : 1 |
| Http 요청 (Http Request) | Method: Get  Path: http://localhost:8080/testProject/order |
| 리스너  (Listener) | Summary report |
| View result tree (결과 저장: overload result.csv) |

## 테스트 수행 및 결과

Docker 컨테이너 내에서 생성 가능한 최대 스레드가 200로 설정되어있기 때문에, 테스트 수행 후 약 15초경, java.lang.OutOfMemoryError 오류가 발생했다.

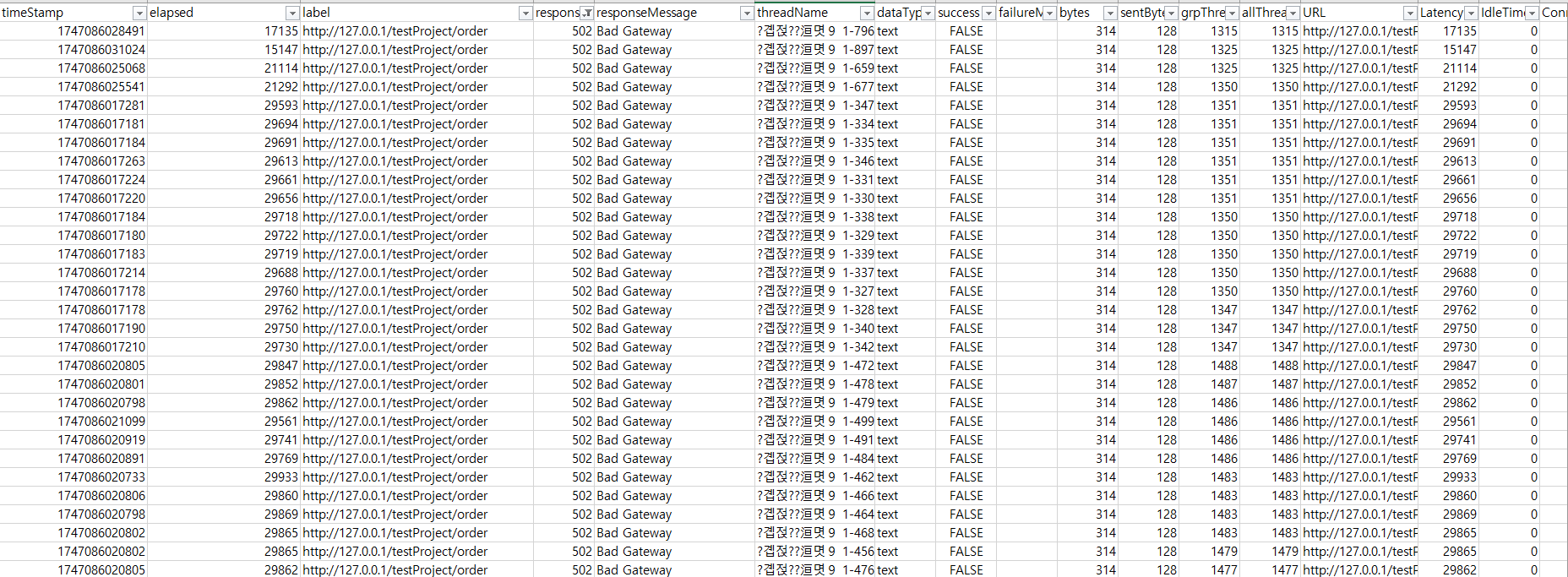


### JMeter Summary report



3분간 총 7200개의 요청을 testTomcat 컨테이너의 Spring application으로 전송하였으며, java. lang.OutOfMemoryError 메모리가 발생한 이후로 모두 에러가 발생한 패킷으로 수신되었다.

### JMeter View result tree (overload result.csv)



7200개의 요청 중 7102개의 요청에서 오류가 발생하였고, 그 중 32개의 요청에서 502 Bad Gateway 오류가 발생하였다. *(첨부파일 overload result.csv 참고)*

# Root-Case 분석

## Jstack을 활용한 데이터 추출

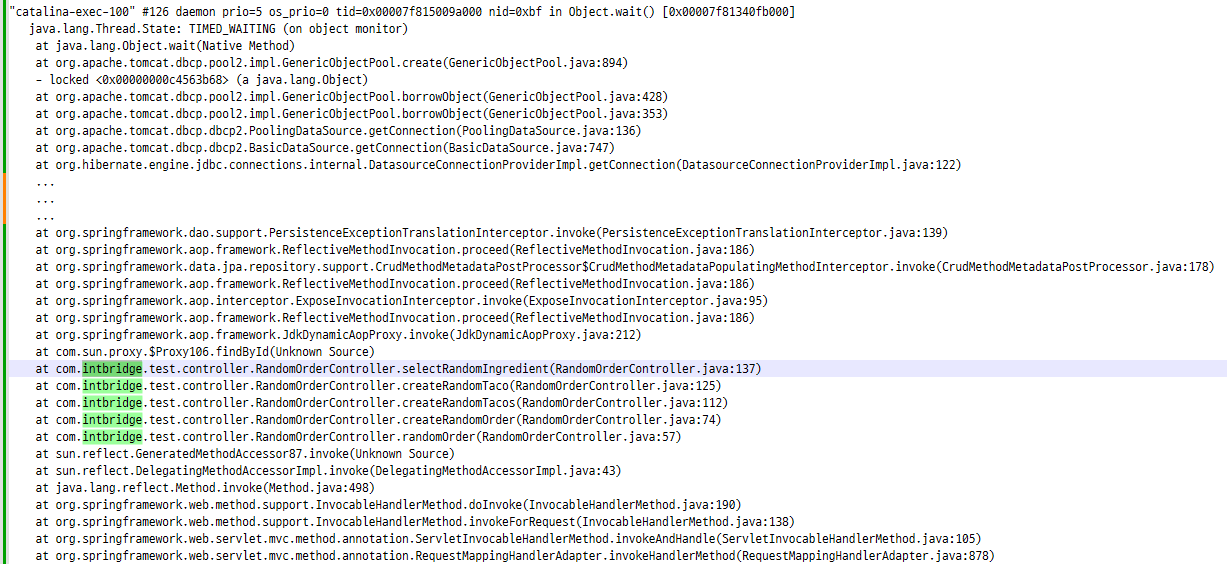
병목 현상이 발생하는 부분을 탐색하기 위해 Java 프로세스의 스레드 상태를 출력하는 도구인 jstack를 testTomcat 컨테이너에서 사용한다. 병목 현상을 탐색하는 것이 목적이기 때문에 테스트에서 사용했던 HTTP 요청을 1초동안, 100번 요청한 후, Tomcat 어플리케이션이 일시적으로 응답하지 않는 상태가 되면, 다음 명령어를 실행한다.

|  |  |
| --- | --- |
| 명령어 | 설명 |
| jstack 1 > test.log | 현재 실행중인 모든 프로세스의 상태를 test.log에 작성한다 |

## 추출된 데이터 분석

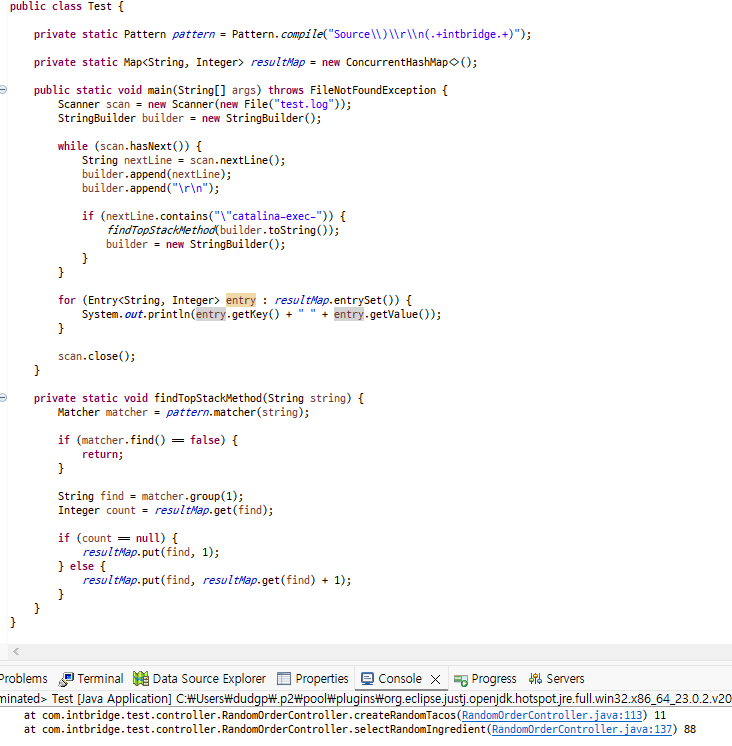
test.log는 각 스레드에 대한 메서드 호출 스택에 대한 정보를 포함하고 있다. 여기서 필요한 정보는 메서드 호출 스택 중, Spring 어플리케이션에서 실행된 메서드 스택, 그 중에서도 가장 상위에 있는 메서드다.

즉, Spring application의 코드들은 com.intbridge.test 패키지에 포함되어 있으므로, test.log의 내용 중 일부에서 com.intbridge.test를 포함하고 있는 메서드 호출 스택 중 가장 위에 있는 메서드의 이름과 코드 라인 수가 필요하다. *(이미지의 포커싱 라인 및 첨부파일 test.log 참고)*



## 추출된 데이터 가공

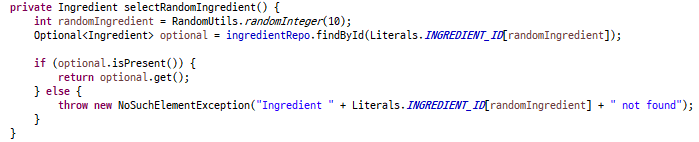
test.log의 내용에서 필요한 부분을 확인했으니, 이제 test.log에 작성된 수십 개의 스레드에서 필요한 내용만을 가져오는 코드를 Java로 작성한다. *(상세 소스 코드는 test.zip 참고)*



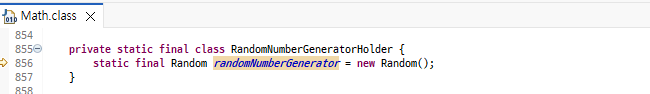
코드 실행 결과, test.log에 기록된 모든 스레드들의 정보에서 Spring 어플리케이션에서 실행된 메서드 스택 중 가장 상위에 있는 메서드는 2개이고, 각각 11번, 88번 호출되었다.

## 원인 분석

주로 많이 호출되었던 selectRandomIngredient 메서드의 코드는 다음과 같다.



단순히 랜덤 타코를 생성할 때 포함할 재료들을 랜덤으로 가져오는 메서드일 뿐이지만, 문제는 RandomUtils.randomInteger(10) 코드에서 발생한다. 난수 발생을 위해 RandomUtils에서는 Math.random() 메서드를 사용하고, 이 메서드 내부에서는 randomNumberGenerator static 인스턴스를 사용한다.



Java 특성상, static 객체는 JVM 내에 단 1개만 생성되기 때문에 비동기로 작동할 수 없고 무조건 동기로 작동한다. 이 때문에 한 스레드가 이 static 객체를 점유하고 있다면, 다른 스레드는 이 객체의 Lock이 풀릴 때까지 대기해야 한다. 병목 현상은 여기서 발생한다.

하나의 static 객체를 모든 스레드가 사용해야 하기 때문에 static 객체의 요청 처리 속도 보다, static 객체를 필요로 하는 스레드의 생성 속도가 더 빠르다면 Http 요청을 처리하기 위한 스레드들은 계속 대기 상태로 쌓이기만 하고, 마침내 제한된 스레드 수가 넘어가면 java.lang.OutOfMemoryError: unable to create new native thread 오류가 발생한다.

# 패치(Hot-Fix) 제시

너무 많은 요청으로 인해 리버스 프록시인 Nginx의 요청에 대한 응답이 너무 늦어서 발생하는 504(Gateway time-out)와 달리 502(Bad gateway)는 Nginx에서 Tomcat으로 요청을 보냈을 때, Tomcat에 문제가 발생하여 이미 종료되어 있고. Tomcat에서 사용하는 8080 포트가 닫혔을 경우 발생한다. 502(Bad gateway) 오류가 다시 발생하지 않도록 하려면 Tomcat이 종료되지 않도록 다음과 같이 조치해야 한다.

## 긴급 조치

### Docker 컨테이너 스레드 수 조절

현재 --pids-limit=200 옵션에 의해 Docker Tomcat 컨테이너의 스레드 제한이 걸려 있는 상황이기 때문에 이 옵션을 조정하여 스레드 수를 늘려서 Tomcat 컨테이너를 재실행한다.

### Tomcat 스레드 수 조절

/usr/local/tomcat/conf/server.xml 파일을의 Connector, Executor 블록을 수정하여 Tomcat 자체적으로 처리할 수 있는 스레드의 수를 낮춰서 컨테이너 자체의 스레드 수 제한에 걸리지 않도록 조절한 후, Tomcat 컨테이너를 재실행한다.

### Nginx 요청 수 조절

Tomcat 어플리케이션이 제한 이상의 스레드를 생성할 수 없도록, 리버스 프록시인 Nginx에서 요청 자체를 조율한다. 이를 위해 Nginx의 etc/nginx/nginx.conf 또는etc/nginx/conf.d/ default.conf 파일의 http server 블록에 limit\_conn, limit\_req 옵션을 추가한 후, Nginx 컨테이너를 재실행한다.

## 영구 조치

### 코드 수정

문제가 발생한 Math.random()은 단일 인스턴스를 사용하므로, static 인스턴스를 사용하지 않는 난수 생성기를 사용하도록 코드를 수정하는 것으로 문제를 해결할 수 있다. 이를 위해 Math 클래스 대신 ThreadLocalRandom 클래스를 사용하여 난수를 생성하도록 코드릴 리팩토링 하는 것으로 문제를 해결한다.

첨부파일 목록

|  |  |
| --- | --- |
| 파일명 | 설명 |
| testProject.zip | Tomcat 컨테이너에서 실행될 Spring boot 기반 어플리케이션 소스 코드 |
| test.zip | test.log 파일에서 필요한 내용만 추출하기 위한 소스 코드 |
| overload test.csv | JMeter 도구 테스트 결과 |
| test.log | jstack 도구 결과 |
| schema.sql | Spring boot 어플리케이션의Database Schema |