Self-project 발표

웹 어플리케이션 캐시 적용

대전 2반 김수용

CONTENTS

01

개요

- TOP SSAFY 팀의 고충

01. 개요

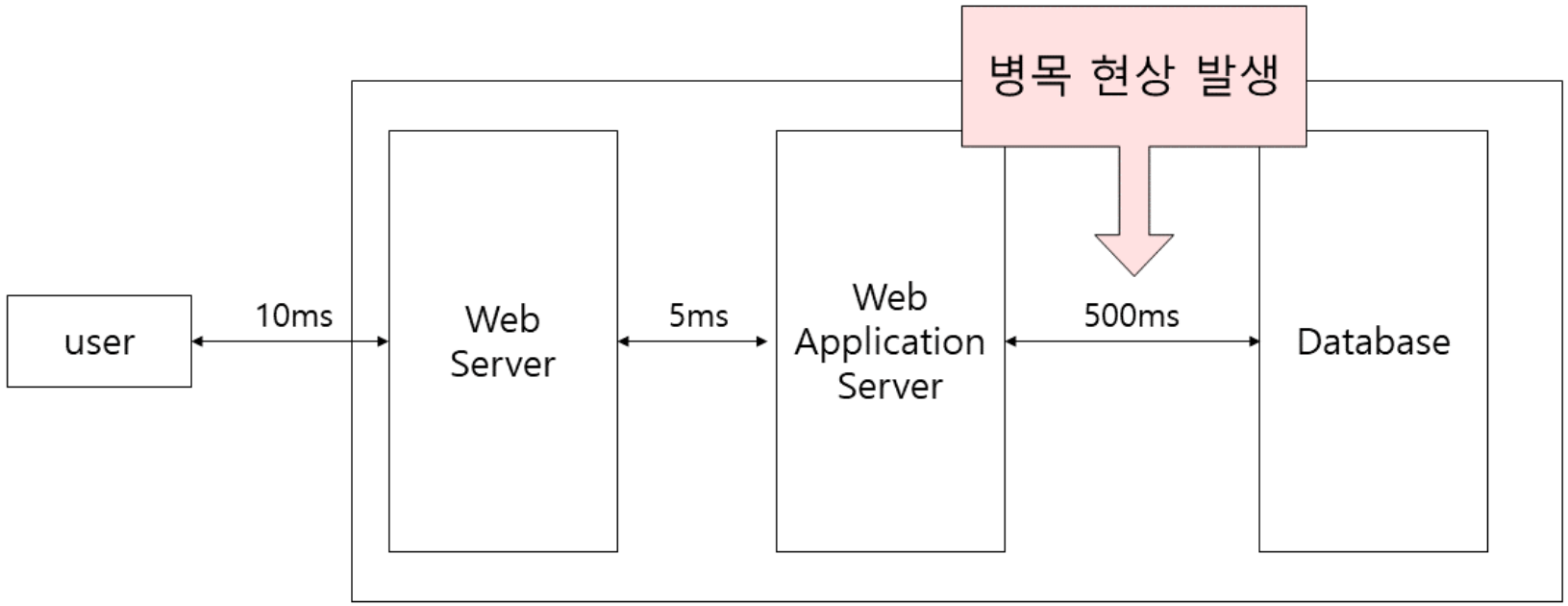
TOP SSAFY 팀의 고충

TOP SSAFY 팀은 고생 끝에 '맛집 추천 서비스'를 성공적으로 구현할 수 있었습니다. 기능 테스트를 꼼꼼하게 진행했기에 소수의 일반 사용자를 대상으로 서비스를 할 때에도 문제는 없었습니다.
그러나 사용자 수가 눈에 띄게 증가함에 따라 웹 어플리케이션에 많은 부하가 걸리게 되었고, 답답한 서비스 응답시간 때문에 고객의 신뢰를 잃을 위기에 처했습니다.

실 서비스 중 발견한 병목현상

01. 개요

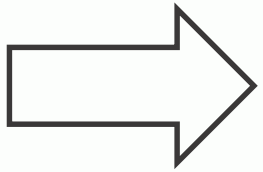
TOP SSAFY 팀의 고충



01. 개요

TOP SSAFY 팀의 고충

개발 중에는 시스템의 비효율성이 큰 문제가 되지 않지만 사용자 요청이 증가함에 따라 웹 어플리케이션 서버에는 많은 부하가 발생



캐시를 통해 해결

02

개념

- 캐시
- Spring의 캐시 추상화
- Redis

Cache?

임시 장소

Cache?

- 서버의 부담을 줄이고, 성능을 높이기 위해 사용되는 기술
- 요청의 처리에 오랜 시간이 걸리는 문제를 해결하기 위해 결과를 미리 저장해두고 가져오는 방법
- 반복적으로 동일한 결과를 반환하는 경우에 용이
- 결과가 매번 달라지는 경우 캐시를 저장하거나 확인하는 작업에 의해 더 성능이 떨어질 수 있음

02. 개념

Spring의 캐시 추상화

- Spring은 AOP 방식으로 메서드에 캐시 서비스를 적용하는 기능을 제공
- 캐시 관련 로직과 핵심 비즈니스 로직을 분리시켜 유지보수에 용이

02. 개념

Redis

- SpringBoot에서 공식 지원하는 Third-Party 캐시 라이브러리
- 추상화된 API와 어노테이션을 제공
- SpringBoot의 Auto Configuration 적용으로 캐시 서버 설정이 간편

03

실습

- 캐시 관련 설정
- 캐시 어노테이션

03. 실습

캐시 관련 설정

1. 의존성 라이브러리 추가

```
<!-- Redis cache -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

03. 실습

캐시 관련 설정

2. Redis 연결 정보 설정

```
#redis cache
spring.cache.type=redis
spring.redis.host=localhost
spring.redis.port=6379
```

03. 실습

캐시 관련 설정

3. @EnableCaching 추가

```
@Configuration
@EnableCaching
public class RedisCacheConfig {
    @Bean
    public CacheManager cacheManager(RedisConnectionFactory connectionFactory) {
        RedisCacheConfiguration redisCacheConfiguration = RedisCacheConfiguration.defaultCacheConfig()
                .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(new StringRedisSerializer()))
                .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(new GenericJackson2JsonRedisSerializer()))
                .entryTtl(Duration.ofMinutes(3L));
        return RedisCacheManager.RedisCacheManagerBuilder.fromConnectionFactory(connectionFactory).cacheDefaults(redisCacheConfiguration).build();
    }
}
```

03. 실습

캐시 어노테이션

@Cacheable

```
@Cacheable(value="notice", key="#noticeNo", cacheManager = "cacheManager")
@Override
public List<NoticeDTO> getNotice(int noticeNo) throws SQLException {
    return sqlSession.getMapper(NoticeMapper.class).getNotice(noticeNo);
}
```

- 캐시를 저장/조회
- 캐시에 데이터가 없다면 메서드를 실행 한 뒤 추가하고,

데이터가 있다면 메서드 실행 없이 반환

- 메서드의 반환값을 통해 동작을 수행한다.

03. 실습

캐시 어노테이션

@CachePut

```
@CachePut(value="notice", key="#noticeNo", cacheManager = "cacheManager")
@Override
public List<NoticeDTO> updateNotice(String noticeNo, String title, String desc, String id) {
    sqlSession.getMapper(NoticeMapper.class).updateNotice(noticeNo, title, desc, id);
    List<NoticeDTO> list = new ArrayList<NoticeDTO>();
    NoticeDTO noticeDTO = new NoticeDTO();
    noticeDTO.setNoticeNo(Integer.parseInt(noticeNo));
    noticeDTO.setTitle(title);
    noticeDTO.setDesc(desc);
    noticeDTO.setId(id);

    list.add(noticeDTO);
    return list;
}
```

- 캐시 저장
- Cacheable과 유사하지만 **항상 메서드를 실행**
- 따라서 주로 Update 시에 활용된다.
- 메서드의 반환값을 통해 동작을 수행한다.

03. 실습

캐시 어노테이션

@CacheEvict

```
@CacheEvict(value="notice", key="#noticeNo", cacheManager = "cacheManager")
@Override
public void deleteNotice(String noticeNo) throws SQLException {
    sqlSession.getMapper(NoticeMapper.class).deleteNotice(noticeNo);
}
```

- 캐시 제거
- **메서드를 실행하며 캐시를 제거**
- beforeInvocation 속성으로 캐시 제거와 메서드 실행의 순서를 정할 수 있다.(default = false)
- 트리거로 동작하므로 **메서드의 반환값이 없어도 동작**한다.

04

마무리

- 주의점
- 고찰

04. 마무리

주의점

Service에서 캐시 서비스를 구현한 이유

```
...Cannot deserialize from Object value (no delegate- or property-based Creator)...
```

오류 발생!

04. 마무리

주의점

Service에서 캐시 서비스를 구현한 이유

```
public class ResponseEntity<T> extends HttpEntity<T> {

    private final Object status;


    /**
     * Create a {@code ResponseEntity} with a status code only.
     * @param status the status code
     */
    public ResponseEntity(HttpStatus status) {
        this(null, null, status);
    }

    /**
     * Create a {@code ResponseEntity} with a body and status code.
     * @param body the entity body
     * @param status the status code
     */
    public ResponseEntity(@Nullable T body, HttpStatus status) {
        this(body, null, status);
    }

    /**
     * Create a {@code ResponseEntity} with headers and a status code.
     * @param headers the entity headers
     * @param status the status code
     */
    public ResponseEntity(MultiValueMap<String, String> headers, HttpStatus status) {
        this(null, headers, status);
    }

    /**
```

ResponseEntity

고찰

Thank you

대전 2반 김수용