**mybatis cache**

2020-10-07

이승진

**학습목표**

mybatis 데이터베이스 캐시 기능을 학습한다.

mybatis SQL 로깅

**목차**

[1. 배경 지식 2](#_Toc84751648)

[1) mybatis cache 2](#_Toc84751649)

[2) mybatis cache 구현 4](#_Toc84751650)

[2. mybatis SQL 로그 5](#_Toc84751651)

[1) application.properties 수정 5](#_Toc84751652)

[2. mybatis cache 구현 6](#_Toc84751653)

[1) DepartmentMapper 수정 6](#_Toc84751654)

[2) Department 수정 7](#_Toc84751655)

[3. 캐시 기능 테스트 8](#_Toc84751656)

[1) 실행 8](#_Toc84751657)

[2) 로그 출력 9](#_Toc84751658)

[4. mybatis 캐시 구현 방법 요약 10](#_Toc84751659)

[1) 캐시 기능 활용 10](#_Toc84751660)

[2) mapper 클래스 수정 10](#_Toc84751661)

[3) DTO 클래스 수정 10](#_Toc84751662)

# 배경 지식

## mybatis cache

정보 시스템에서 병목 지점은 대부분 데이터베이스이다.

데이터베이스의 성능을 향상시키는 것이 정보시스템 성능의 관건이다.

데이터베이스 캐시(cache)가 데이터베이스 성능 향상에 아주 큰 도움이 된다.

### 캐시 실행 절차

mybatis에서 데이터베이스 캐시는 mapper 마다 따로 생성된다.

데이터베이스 캐시는 다음과 같은 절차로 진행된다.

① mapper의 조회 메소드는 @select(...)에 정의된 SELECT SQL 명령을 DB에서 실행한다.

② 조회 결과를 mapper의 캐시에 저장한다. 캐시는 HashMap 형태의 자료구조이다. 조회 메소드 이름과 파라미터 값을 키(key)로, 조회 결과 value로 캐시에 저장한다.

③ mapper의 조회 메소드는 조회 결과를 리턴한다.

④ mapper의 조회 메소드가 또 호출되면, 조회 메소드 이름과 파라미터 값을 키(key)로, 전에 저장해둔 조회 결과를 캐시에서 찾는다.

⑤ 캐시에서 찾았다면, DB에서 다시 조회하지 않고, 캐시에서 찾은 조회 결과를 리턴한다.

⑥ 캐시에서 찾지 못했다면 절차 ① 번으로 넘어가서, DB에서 조회한다.

DB에서 다시 조회하지 않고 메모리에 저장된 데이터를 그냥 리턴하게 되므로 성능 향상 효과가 크다.

### 데이터 수정

예를 들어 홍길동 학생 데이터가 StudentMapper의 캐시에 있을 때,

DB에서 홍길동 학생 레코드가 수정되면,

수정된 홍길동 학생 레코드와, 캐시에 있는 홍길동 학생 데이터가 일치하지 않을 것이다.

이 불일치 문제는 다음과 같은 방법으로 해결된다.

(방법1) mapper에서 insert/update/delete 메소드가 하나라도 실행이 되면, 그 mapper의 캐시를 전부 비운다.

캐시가 모두 비워진 뒤에 호출되는 조회 메소드는 DB에서 조회 결과를 받아오게 된다.

(방법2) 일정 시간 간격으로 mapper의 캐시를 자동으로 비운다.

DB에서 데이터가 수정되는 것을 mybatis가 판단할 때,

mybatis는 오직 mapper의 메소드만 보고 판단한다.

그렇기 때문에, mapper의 메소드를 통하지 않고 DB 데이터가 수정되는 경우에는, DB에서 데이터가 수정되었다는 사실을 mybatis는 알 수 없다.

DB에서 데이터가 수정되는 것을 mybatis가 판단할 때,

mybatis는, mapper의 @select, @insert, @update, @delete 어노테이션 종류만 보고 판단한다.

@select 어노테이션이 붙은 mapper 메소드는 데이터를 수정하지 않는 것으로 판단하고,

@insert, @update, @delete 어노테이션이 붙은 mapper 메소드는 데이터를 수정한다고 판단한다.

그리고 @insert, @update, @delete mapper 메소드가 구체적으로 어떤 테이블의 데이터를 수정하는지와 상관 없이, 오직 그 mapper의 캐시만 비운다.

예를 들어 StudentMapper의 insert 메소드가 호출되면, StudentMapper의 캐시만 비워지고 DepartmentMapper의 캐시는 비워지지 않는다.

만약 StudentMapper의 insert 메소드에서 Department 테이블의 데이터를 수정했다면,

DepartmentMapper의 캐시는 비워지지 않기 때문에 수정되기 전 데이터가 캐시에 들어있게 된다.

그래서 DepartmentMapper의 캐시와 Department 테이블의 데이터가 불일치하는 문제가 발생하게 된다.

따라서 mapper를 구현할 때 다음 규칙을 지켜야 한다.

(규칙1) 테이블 마다 mapper를 따로 구현한다.

(규칙2) 데이터를 수정하는 SQL 문은 그 테이블의 mapper에만 구현한다.

예를 들어 Student 테이블의 데이터를 수정하는 것은, StudentMapper의 메소드에만 구현한다.

그래서 Student 테이블의 데이터가 수정되면 StudentMapper의 캐시가 비워지게 된다.

다른 테이블 mapper에서 Student 테이블을 조회하는 것은 괜찮지만 수정하면 안된다.

StudentMapper의 캐시가 비워지지 않기 때문이다.

위 규칙을 지켜도 여전히 데이터 불일치 문제가 발생할 수 있다.

예를 들어 StudentMapper에서 학생 목록을 조회할 때, Department 테이블을 조인(JOIN)하여, 학과명을 조회한다. 학생 목록 데이터에, 학생 소속 학과명이 들어있다. 이 학과명 중에 "소프트웨어공학과"도 있을 것이다.

DepartmentMapper의 update 메소드가 호출되어, "소프트웨어공학과" 학과명이 "소프트웨어공학전공"으로 변경된 경우에, DepartmentMapper의 캐시는 비워지지만, StudentMapper의 cache는 비워지지 않는다.

그래서 수정하기 전의 "소프트웨어공학과" 학과명이 StudentMapper 캐시의 학생 목록에 남아있어서,

학생 목록을 다시 조회하면, StudentMapper 캐시에 있는 학생 목록이 리턴된다.

이 목록에 "소프트웨어공학과" 라는 학과명이 그대로 들어있을 것이다.

이 캐시 데이터 불일치 문제 때문에, 일정 시간 간격으로 mapper의 캐시가 자동으로 비워지는 기능을 사용해야 한다. 시간을 길게 잡을 수록 DB 조회를 적게 할 테니 성능 향상 효과는 크지만,

데이터 불일치가 발생할 확률도 커진다.

예를 들어 StudentMapper에서 캐시가 자동으로 비워지는 시간을 2초로 설정했다면,

2초가 지나면 StudentMapper의 캐시가 자동으로 비워지고,

그 뒤에 StudentMapper의 조회 메소드는 데이터베이스에서 직접 조회할 것이기 때문에,

데이터 불일치 문제는 최대 2초 동안만 발생한다.

캐시가 비워지는 시간을 1초 정도로 짧게 잡으면 데이터 불일치 문제도 최대 1초간만 발생한다.

1초 정도의 짧은 시간만 보관하는 캐시도 성능 향상 효과가 크다.

예를 들어 수만 명이 동시에 보는 뉴스 사이트를 생각해 보자.

기사 목록 메소드의 조회 결과가 1초 동안 캐시 된다는 것은

그 1초 동안 기사 목록이 DB에서 다시 조회되지 않고 캐시에서 바로 리턴된다는 뜻이다.

예를 들어 동시 접속자가 많은 네이버 뉴스의 첫 페이지가, 1초 동안 평균 1만번 조회된다고 할 때,

1초 시간 간격의 DB 캐시가 있다면,

1초에 1만 번이 아니라 한 번만 DB에서 기사 목록을 조회하면 되고,

9999번은 캐시에서 기사 목록이 리턴될 것이다.

1초 동안 1만번의 조회가 1번으로 줄어들기 때문에 성능 향상 효과가 매우 크다.

동시 접속자 수가 많을 수록 그리고 자주 수정되지 않는 테이블일 수록

데이터베이스 캐시의 성능 개선 효과가 크다.

예컨대 네이버의 카페 게시판, 네이버 블로그, 네이버 뉴스 사이트들의 경우에

데이터 수정은 아주 드물고, 동시 접속자 수는 아주 많다.

카페, 블로그, 뉴스 모두 최신 데이터 위주로 조회된다. 과거 데이터는 엄청 많지만 거의 조회되지 않는다.

뉴스의 경우 조회가 몰리는 데이터가 몇 개 되지 않는다.

위와 같은 사이트들에서 데이터베이스 캐시의 성능 개선 효과는 매우 클 것이다.

## mybatis cache 구현

### @CacheNamespace

mybatis mapper 클래스에 @CacheNamespace 어노테이션만 붙여주면 캐시 기능이 활성화 된다.

따라서 캐쉬 기능 구현은 사소할 정도로 간단하다. 이 기능을 잘 이해하는 것이 관건이다.

mybatis cache의 기능은 다음과 같다.

select 어노테이션으로 정의된 모든 mapper 메소드의 조회 결과가 캐시에 저장된다.

insert/update/delete 어노테이션으로 정의된 메소드가 호출되면 그 mapper의 캐시가 전부 비워진다.

조회 결과는 최대 1024개만 저장된다.

캐시가 꽉 차면 먼저 LRU(Least Recently Used) 항목을 제거하고 새 조회 결과를 저장한다.

LRU = 데이터 사용 시각이 가장 오래된 항목

@Mapper

@CacheNamespace(flushInterval=1000, size=2048)

class DepartmentMapper {

이 어노테이션을 mybatis mapper 클래스에 붙여주면 그 mapper에 캐시 기능이 활성화 된다

flushInterval - 캐시를 비울(flush) 시간 간격을 밀리초 단위로 지정한다. 1000밀리초는 1초이다.

size - 캐시의 크기를 지정한다. 캐시에 저장할 조회 결과의 수 이다.

flushInterval 이나 size 를 꼭 지정해야 하는 것은 아니고, 다음과 같이 생략할 수 있다.

|  |
| --- |
| @Mapper  @CacheNamespace  class DepartmentMapper { |

flushInterval 속성을 지정하지 않으면, 디폴트 값은 0 이다.

이 값이 0 이면 cache가 자동으로 비워지지 않으니, 바람직하지 않다.

flushInterval 값을 지정해 주는 것이 좋다.

size 속성을 지정하지 않으면, 디폴트 값은 1024 이다.

돈 관련 테이블이나, 수강신청 테이블 처럼 잠깐의 데이터 불일치도 허용할 수 없는 테이블에는

cache 기능을 사용하지 말자.

학과명 테이블이 cache를 사용할만한 테이블이다.

학과명은 거의 수정되지 않고, 만약 수정되어도 1초 정도 학과명 불일치는 별 문제 아니다.

### Serializable

DepartmentMapper에 캐시 기능을 활성화 하려면,

Department DTO 클래스가 Serializable 인터페이스를 implements 해야 한다.

캐시에 저장된 데이터를 리턴할 때, 저장된 원본 객체를 리턴하는 것이 아니고,

저장된 객체를 복제해서, 복제된 새 객체를 리턴해야 한다.

이 복제는 deep copy 이어야 한다.

Java 언어에서 deep copy를 손쉽게 구현하는 방법은

serialize/deserialize를 이용하는 것이다.

(1) 원본 객체를 serialize해서 byte[] 배열을 만든다.

(2) byte[] 배열을 deserialize해서 객체를 복구한다.

이렇게 복구된 객체는 원본 객체의 deep copy 이다.

# mybatis SQL 로그

## application.properties 수정

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=test123  server.port=8088  logging.level.net.skhu.mapper.DepartmentMapper=DEBUG |

application.properties 파일을 위와 같이 수정하자.

디버깅할 때,mybatis mapper에서 실행되는 SQL 명령의 로그 기록을 보는 것이 유용하다.

예를 들어 net.skhu.mapper.DepartmentMapper 클래스에서 실행되는 SQL 명령의 로그 기록을 보고 싶으면,

application.properties 파일에서 다음 항목을 추가하면 된다.

|  |
| --- |
| logging.level.net.skhu.mapper.DepartmentMapper=DEBUG |

이 설정은 다음과 같은 구조이다.

|  |
| --- |
| logging.level.패키지명.클래스명=DEBUG |

위와 같이 설정하면, DepartmentMapper에서 실행되는 SQL 명령이

eclipse 콘솔창에 출력된다.

StudentMapper 클래스에서 실행되는 SQL 명령도 보고 싶으면 다음 항목을 추가하면 된다.

|  |
| --- |
| logging.level.net.skhu.mapper.StudentMapper=DEBUG |

# mybatis cache 구현

mybatis1 프로젝트를 수정하자.

## DepartmentMapper 수정

### src/main/java/net/skhu/mapper/DepartmentMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.CacheNamespace;  import org.apache.ibatis.annotations.Delete;  import org.apache.ibatis.annotations.Insert;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Options;  import org.apache.ibatis.annotations.Select;  import org.apache.ibatis.annotations.Update;  import net.skhu.dto.Department;  @Mapper  @CacheNamespace(flushInterval=10000)  public interface DepartmentMapper {  @Select("SELECT \* FROM department")  List<Department> findAll();  @Select("SELECT \* FROM department WHERE id = #{id}")  Department findOne(int id);  @Insert("INSERT department (departmentName) VALUES (#{departmentName})")  @Options(useGeneratedKeys=true, keyProperty="id")  void insert(Department department);  @Update("UPDATE department SET departmentName = #{departmentName} WHERE id = #{id}")  void update(Department department);  @Delete("DELETE FROM department WHERE id = #{id}")  void delete(int id);  } |

DepartmentMapper에 캐시 기능을 활성화 한다.

이 캐시는 10,000 밀리초(10초) 간격으로 자동으로 비워진다.

## Department 수정

### src/main/java/net/skhu/dto/Department.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | package net.skhu.dto;  import java.io.Serializable;  public class Department implements Serializable {  private static final long serialVersionUID = 1L;  int id;  String departmentName;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  } |

클래스에 implements Serializable 만 붙여주면, 이 클래스는 serialize/deserialize 가능하게 된다.

private static final long serialVersionUID = 1L;

이것은 객체 구조의 버전 번호이다.

객체 구조가 변경될 때마다 이 번호를 1씩 올려야 한다.

# 캐시 기능 테스트

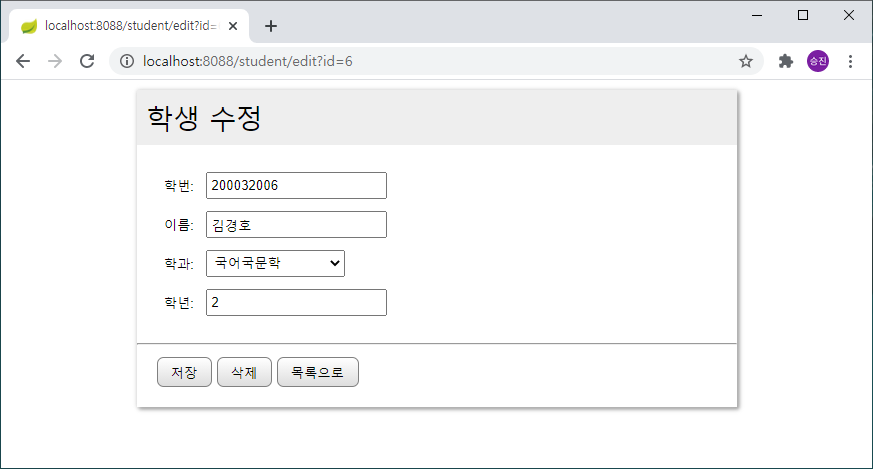
## 실행

mybatis1 프로젝트를 선택하고

Run As Spring Boot App

웹브라우저에서 아래 URL을 실행하자.

http://localhost:8088/student/edit?id=6



위 페이지를 새로 고침할 때마다,

student 테이블에서 김경호 학생 레코드를 조회해야 한고,

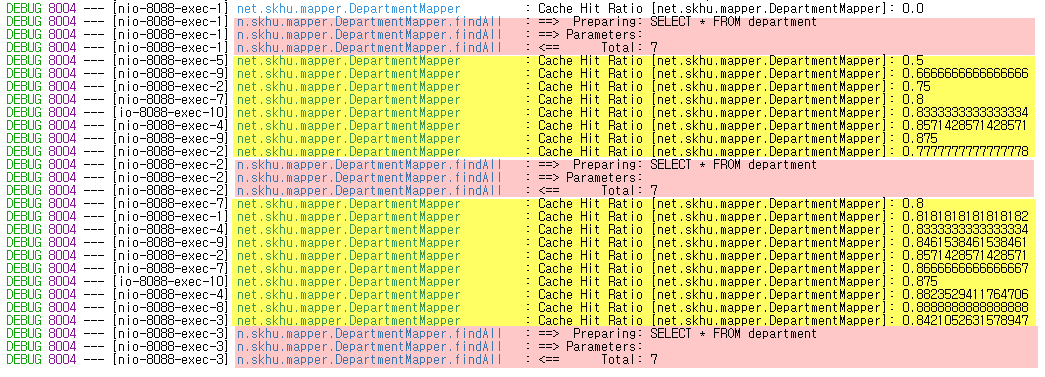
department 테이블에서 학과 목록 전체를 조회해야 한다.

|  |
| --- |
| @GetMapping("edit")  public String edit(Model model, @RequestParam("id") int id) {  Student student = studentMapper.findOne(id);  List<Department> departments = departmentMapper.findAll();  model.addAttribute("student", student);  model.addAttribute("departments", departments);  return "student/edit";  } |

## 로그 출력

http://localhost:8088/student/edit?id=6 URL 웹브라우저 을 1초 간격으로 새로 고침하면,

eclipse 콘솔창에 다음과 같은 로그 메시지 출력을 볼 수 있다.



노란색으로 칠한 부분은,

학과 목록이 DB에서 다시 조회되지 않고, DepartmentMapper 캐시에서 리턴된 경우이다.

분홍색으로 칠한 부분은,

DB에서 학과 목록을 조회한 경우이다.

10초 동안 학과 목록이 캐시에서 리턴되고 있음을 확인할 수 있다.

# mybatis 캐시 구현 방법 요약

## 캐시 기능 활용

### 동시 접속자 수가 많은 경우

동시 접속자 수가 많아서 성능이 문제가 되는 경우.

성능 개선에 mybatis 캐시가 효과적이다.

### 조회가 훨씬 더 많은 테이블

insert/update/delete 보다는 select 조회가 훨씬 더 많은 테이블

(예: 학과 테이블, 학생 테이블, 게시글 테이블)

### mybatis mapper를 통해서만 그 테이블의 데이터가 수정되는 경우

mybatis mapper가 아닌, 다른 경로를 통해서 그 테이블의 데이터가 수정된다면,

데이터 불일치 문제가 발생할 수 있다.

## mapper 클래스 수정

@Mapper

@CacheNamespace(flushInterval=1000)

class DepartmentMapper {

flushInterval은 캐시가 자동으로 비워지는 시간 간격이다. (단위 밀리초)

이 간격이 짧을 수록 데이터 불일치 발생 기간이 짧아진다.

## DTO 클래스 수정

|  |
| --- |
| public class Department implements Serializable {  private static final long serialVersionUID = 1L;  . . . |