第七章 企业项目开发--本地缓存guava cache

1、在实际项目开发中，会使用到很多缓存技术，而且数据库的设计一般也会依赖于有缓存的情况下设计。

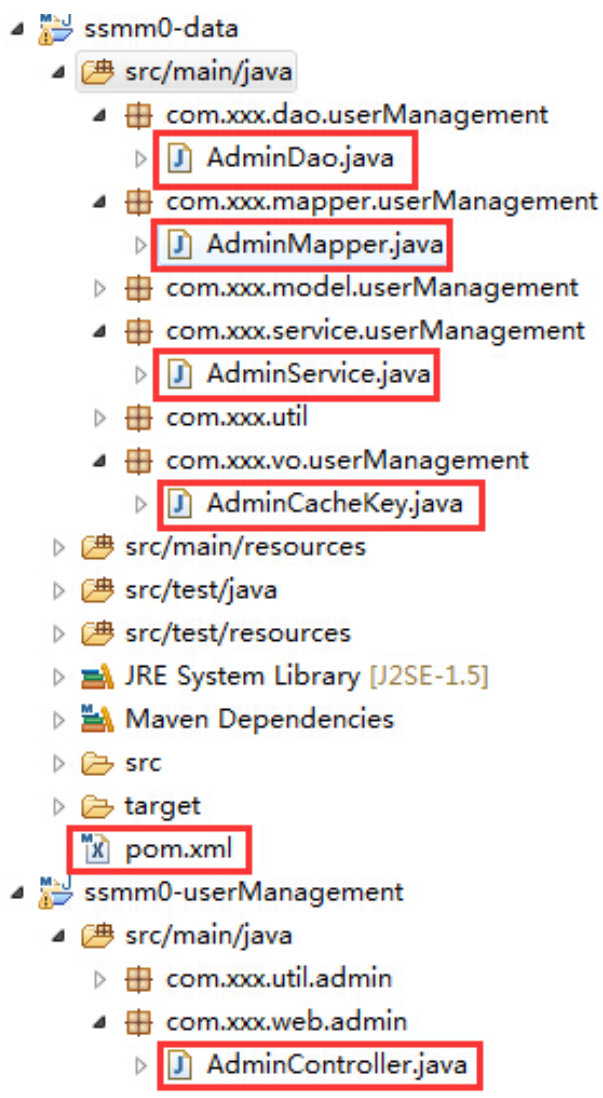
- 常用的缓存分两种：本地缓存和分布式缓存。
- 常用的本地缓存是guava cache，本章主要介绍guava cache在项目中的使用。

关于常用缓存以及每种缓存常用场景的介绍，之后可以去查看我记录的"Java缓存相关"系列博客。链接如下：

《第一章 常用的缓存技术》

2、实际使用

本项目的代码基于第六章的代码进行构建，这里只列出修改过的代码：



2.1、ssmm0-data

pom.xml：

```
        <!-- guava cache -->
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>14.0.1</version>
        </dependency>
```

在pom.xml中引入了guava cache14.0.1的依赖包。

AdminMapper：

```
package com.xxx.mapper.userManagement;

import java.util.List;

import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Param;
import org.apache.ibatis.annotations.Result;
import org.apache.ibatis.annotations.Results;
import org.apache.ibatis.annotations.Select;
```

```
import com.xxx.model.userManagement.Admin;

/**
 * 管理员Mapper
 */
public interface AdminMapper {

    /*************注解**************/
    @Insert("INSERT INTO userinfo(username, password) VALUES(#{username},#{password})")
    public int insertAdmin(Admin admin);

    @Select("SELECT * FROM userinfo WHERE username = #{username} AND password = #{password}")
    @Results(value = {
            @Result(id = true, column = "id", property = "id"),
            @Result(column = "username", property = "username"),
            @Result(column = "password", property = "password") })
    public Admin selectAdmin(@Param("username") String username,
                             @Param("password") String password);

    /**************xml**************/
    /**
     * 条件不定式查询
     * 我们这里使用@Param指定参数，这样的话，在AdminMapper.xml中就不用再使用parameterType属性了；否则得写parameterType属性
     */
    public List<Admin> getAdminByConditions(@Param("username")String username,
                                            @Param("password")String password,
                                            @Param("start")int start,
                                            @Param("limit")int limit);

    /**
     * 返回主键
     */
    public int insertAdminWithBackId(Admin admin);

    /***************guava cache****************/
    @Select("SELECT * FROM userinfo WHERE username = #{username}")
    @Results(value = {
            @Result(id = true, column = "id", property = "id"),
            @Result(column = "username", property = "username"),
            @Result(column = "password", property = "password") })
    public List<Admin> getUserByName(@Param("username") String username);
}
```

将使用到的两个方法：

- public List<Admin> getUserByName(String username)
- public List<Admin> getAdminByConditions(String username, String password, int start, int limit)

AdminDao：

```
package com.xxx.dao.userManagement;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.xxx.mapper.userManagement.AdminMapper;
import com.xxx.model.userManagement.Admin;
```

```java
/**
 * 管理员DAO
 */
@Repository
public class AdminDao {
    @Autowired
    private AdminMapper adminMapper;
    /**************注解*****************/
    public boolean register(Admin admin){
        return adminMapper.insertAdmin(admin)==1?true:false;
    }

    public Admin login(String username ,String password){
        return adminMapper.selectAdmin(username, password);
    }
    /***************xml*****************/
    public List<Admin> findAdmin(String username, String password, int start, int limit){
        return adminMapper.getAdminByConditions(username, password, start, limit);
    }

    public int insertAdminWithBackId(Admin admin){
        return adminMapper.insertAdminWithBackId(admin);
    }
    /*****************guava cache******************/
    public List<Admin> getUserByName(String username){
        return adminMapper.getUserByName(username);
    }
}
```

将使用到的两个方法：

- public List<Admin> getUserByName(String username)
- public List<Admin> findAdmin(String username, String password, int start, int limit)

AdminService：

```java
package com.xxx.service.userManagement;

import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.google.common.cache.CacheBuilder;
import com.google.common.cache.CacheLoader;
import com.google.common.cache.LoadingCache;
import com.xxx.dao.userManagement.AdminDao;
import com.xxx.model.userManagement.Admin;
import com.xxx.vo.userManagement.AdminCacheKey;

/**
 * 管理员service
 */
@Service
public class AdminService {
    @Autowired
    private AdminDao adminDao;

    public boolean register(Admin admin) {
        return adminDao.register(admin);
```

```
    }

    public Admin login(String username, String password) {
        return adminDao.login(username, password);
    }

    /*********** 以下方法是为了测试mybatis中使用xml **********/
    public List<Admin> findAdmin(String username,
                                 String password,
                                 int start,
                                 int limit) {
        return adminDao.findAdmin(username, password, start, limit);
    }

    public Admin insertAdminWithBackId(Admin admin) {
        int record = adminDao.insertAdminWithBackId(admin);
        if (record == 1) {
            return admin;// 这时的admin已经被赋予主键了
        }
        return null;
    }

    /********************** guava cache ************************/
    /***********单条件的查询，key为String***********/
    public List<Admin> findByUsername(String username) {
        List<Admin> adminList = null;
        try {
            adminList = adminListCache.get(username);
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
        return adminList;
    }

    LoadingCache<String, List<Admin>> adminListCache = CacheBuilder.newBuilder()
            .expireAfterWrite(20, TimeUnit.MINUTES)// 缓存20分钟
            .maximumSize(1000)// 最多缓存1000个对象
            .build(new CacheLoader<String, List<Admin>>() {
                public List<Admin> load(String username) throws Exception {
                    return adminDao.getUserByName(username);
                }
            });

    /***********多条件的查询，key为Object（封装了多个条件的VO类）***********/
    public List<Admin> findAdminList(String username,
                                     String password,
                                     int start,
                                     int limit) {
        /*
         * 注意：
         * 如果以一个新建的对象做为key的话，因为每次都是新建一个对象，所以这样的话，实际上每次访问key都是不同的，即每次访问都是重新进行缓存；
         * 但是实际上，我们想要根据对象的属性来判断对象是否相等，只需要根据这些属性重写对象的hashCode与equals方法即可，
         * 所以重写了AdminCacheKey类的hashCode和equals方法，这样，每次访问的话，就会以每个条件是否相等来判断对象（即key）是否相等了，这一块儿的缓存就会起作用了
         */
        AdminCacheKey cacheKey = new AdminCacheKey(username,
                                                   password,
                                                   start,
                                                   limit);
        List<Admin> adminList = null;
        try {
            System.out.println(cacheKey);
            adminList = adminsCache.get(cacheKey);
        } catch (ExecutionException e) {
```

```
            e.printStackTrace();
        }
        return adminList;
    }

    LoadingCache<AdminCacheKey, List<Admin>> adminsCache = CacheBuilder.newBuilder()
            .expireAfterWrite(60, TimeUnit.MINUTES) // 缓存项在给定时间内（60min）没有被写访问（创建或覆盖），则回收
            .maximumSize(100) // 最多缓存100项
            .build(new CacheLoader<AdminCacheKey, List<Admin>>() {
                public List<Admin> load(AdminCacheKey key) throws Exception {
                    return adminDao.findAdmin(key.getUsername(),
                                              key.getPassword(),
                                              key.getStart(),
                                              key.getLimit());
                }
            });

}
```

将使用到的两个方法：

- public List<Admin> findByUsername(String username)
- public List<Admin> findAdminList(String username, String password, int start, int limit)

这一块儿是整个guava cache使用的部分。这里边写出了两种guava cache使用的方式：

- 单查询条件：key为String或Object都可以
- 多查询条件：key为Object，该Object封装了多个查询条件，并通过这些查询条件重写了该Object的hashcode()和equals()

这一部分中guava cache的使用方式，就是实际开发中最常用的方法。

AdminCacheKey：

```
package com.xxx.vo.userManagement;

/**
 * guava cache的key
 */
public class AdminCacheKey {
    private String username;
    private String password;
    private int start;
    private int limit;

    public AdminCacheKey() {
    }

    public AdminCacheKey(String username, String password, int start, int limit) {
        this.username = username;
        this.password = password;
        this.start = start;
        this.limit = limit;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
```

```
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public int getStart() {
        return start;
    }

    public void setStart(int start) {
        this.start = start;
    }

    public int getLimit() {
        return limit;
    }

    public void setLimit(int limit) {
        this.limit = limit;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + limit;
        result = prime * result
                + ((password == null) ? 0 : password.hashCode());
        result = prime * result + start;
        result = prime * result
                + ((username == null) ? 0 : username.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        AdminCacheKey other = (AdminCacheKey) obj;
        if (limit != other.limit)
            return false;
        if (password == null) {
            if (other.password != null)
                return false;
        } else if (!password.equals(other.password))
            return false;
        if (start != other.start)
            return false;
        if (username == null) {
            if (other.username != null)
                return false;
        } else if (!username.equals(other.username))
            return false;
        return true;
    }
}
```

该类是封装了多个查询条件的一个VO类。

2.2、ssmm0-userManagement

AdminController：

```
package com.xxx.web.admin;

import java.util.List;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.servlet.ModelAndView;

import com.xxx.model.userManagement.Admin;
import com.xxx.service.userManagement.AdminService;
import com.xxx.util.admin.AdminCookieUtil;

/**
 * adminController
 */
@Controller
@RequestMapping("/admin")
public class AdminController {

    @Autowired
    private AdminService adminService;

    /**
     * 管理员注册
     */
    @ResponseBody
    @RequestMapping("/register")
    public boolean register(@RequestParam("username") String username,
                            @RequestParam("password") String password){
        Admin admin = new Admin();
        admin.setUsername(username);
        admin.setPassword(password);

        boolean isRegisterSuccess = adminService.register(admin);

        return isRegisterSuccess;
    }

    /**
     * 管理员登录
     */
    @RequestMapping("/login")
    public ModelAndView login(@RequestParam("username") String username,
                              @RequestParam("password") String password,
                              HttpServletResponse response,
                              HttpSession session){


        Admin admin = adminService.login(username, password);
```

```
        ModelAndView modelAndView = new ModelAndView();
        if(admin == null){
            modelAndView.addObject("message", "用户不存在或者密码错误！请重新输入");
            modelAndView.setViewName("error");
        }else{
            modelAndView.addObject("admin", admin);
            modelAndView.setViewName("userinfo");
            /*
             * 这为什么不直接传一个username，而传了一个admin，
             * 是因为在实际开发中，你传过去的信息可能不只是username，还有用户手机号、地址等等
             */
            //使用cookie
            AdminCookieUtil.addLoginCookie(admin, response);
            //使用session
            //session.setAttribute("adminSession", admin);
        }

        return modelAndView;
    }

    /****************************mybatis xml方式解决的问题*****************************/
    /**
     * 根据username或password查找List<Admin>
     */
    @ResponseBody
    @RequestMapping("/findAdmin")
    public List<Admin> findAdmin(@RequestParam(value="username",required=false) String username,
                                 @RequestParam(value="password",required=false) String password,
                                 @RequestParam("start") int start,
                                 @RequestParam("limit") int limit,
                                 HttpServletRequest request,
                                 HttpSession session){
        Admin admin = AdminCookieUtil.getLoginCookie(request);
        //Admin admin = (Admin) session.getAttribute("adminSession");

        if(admin == null){//未登录
            return null;
        }
        System.out.println(admin.toJson());
        List<Admin> adminList = adminService.findAdmin(username, password, start, limit);
        return adminList;
    }

    /**
     * 插入一个用户并返回主键
     * 注意：get请求也会自动装配（即将前台传入的username和password传入admin）
     */
    @ResponseBody
    @RequestMapping("/insert")
    public Admin insertAdminWithBackId(Admin admin){
        return adminService.insertAdminWithBackId(admin);
    }
    /************************guava cache*****************************/
    /**
     * 根据username查找List<Admin>
     */
    @ResponseBody
    @RequestMapping("/findAdminByUsername")
    public List<Admin> findAdminByUserName(@RequestParam(value="username") String username){

        List<Admin> adminList = adminService.findByUsername(username);
        return adminList;
    }

    @ResponseBody
```

```
    @RequestMapping("/findAdminList")
    public List<Admin> findAdminList(@RequestParam(value="username") String username,
                                     @RequestParam(value="password",required=false) String password,
                                     @RequestParam("start") int start,
                                     @RequestParam("limit") int limit){

        List<Admin> adminList = adminService.findAdminList(username, password, start, limit);
        return adminList;
    }
}
```

将使用到的两个方法：

- public List<Admin> findAdminByUserName(String username)
- public List<Admin> findAdminList(String username, String password, int start, int limit)

3、测试

- 单元测试：使用springJunit去测就行
- 整体测试：代码写好之后，注意对代码去做测试的方法，先运行相应的controller的方法，然后对查询出来的部分数据在数据库中直接进行修改，再运行之前的controller对应的方法。出现两种结果：
  - 第二次运行与第一次结果相同：缓存成功
  - 第二次运行与第一次结果不同：缓存不成功

4、总结：

- 常用的几个API：
  - get(Object key)：首先获取value-->若获取不到，先缓存-->再从缓存中去取（以上三步是原子操作），使用该方法优先于使用put
  - getIfPresent(Object key)：获取value，若获取不到，返回null；若获取的到，返回value
  - put(Object key, Object value)：显示的添加缓存key-value
- **guava cache的get(Object key)的value不能为null（这个可以去看源代码的注释）**，看下边的代码例子：

```
    LoadingCache<String, List<Admin>> adminListCache = CacheBuilder.newBuilder()
            .expireAfterWrite(20, TimeUnit.MINUTES)// 缓存20分钟
            .maximumSize(1000)// 最多缓存1000个对象
            .build(new CacheLoader<String, List<Admin>>() {
                public List<Admin> load(String username) throws Exception {
                    //1、下边这样null的话，不抛异常
                    /*List<Admin> admins = adminDao.getUserByName(username);
                    if(admins==null){
                        return null;
                    }
                    return admins;*/
                    //2、但是如果这里查询出来的结果为null的话，也没关系
                    //return adminDao.getUserByName(username);
                    //3、如果这里直接返回null，就会出现
com.google.common.cache.CacheLoader$InvalidCacheLoadException
                    return null;
                }
            });
```

注意：该代码中的三种null情况，只有第三种会抛出异常。前两种不为空的原因是因为，**即使admins没有元素，admins也不会是null，而是[]，这应该是mybatis的功劳？！这个是个问题，以后在读mybatis源码的时候，会仔细研究！！！**但是实际使用中，我们判断一个list是否为空，会使用CollectionUtil.isNotEmpty(list)类似于下边这样，就会抛出异常了。

```
    LoadingCache<String, List<Admin>> adminListCache = CacheBuilder.newBuilder()
            .expireAfterWrite(20, TimeUnit.MINUTES)// 缓存20分钟
            .maximumSize(1000)// 最多缓存1000个对象
            .build(new CacheLoader<String, List<Admin>>() {
```

```
                public List<Admin> load(String username) throws Exception {
                    //1、下边这样null的话，不抛异常
                    List<Admin> admins = adminDao.getUserByName(username);
                    //System.out.println(admins);//如果admins为空，不会返回null，而是返回[]
                    if(CollectionUtils.isEmpty(admins)){
                        System.out.println(admins+"-->");
                        return null;
                    }
                    return admins;
                    //2、但是如果这里查询出来的结果为null的话，也没关系
                    //return adminDao.getUserByName(username);
                    //3、如果这里直接返回null，就会出现
com.google.common.cache.CacheLoader$InvalidCacheLoadException
                    //return null;
                }
            });
```

但是，为了在guava cache的使用中不抛出异常，我们这里直接使用下边这句就好了，由mybatis将[]返回就好了。

```
return adminDao.getUserByName(username);
```