web2 hw4

程序开始运行时，redis已启动

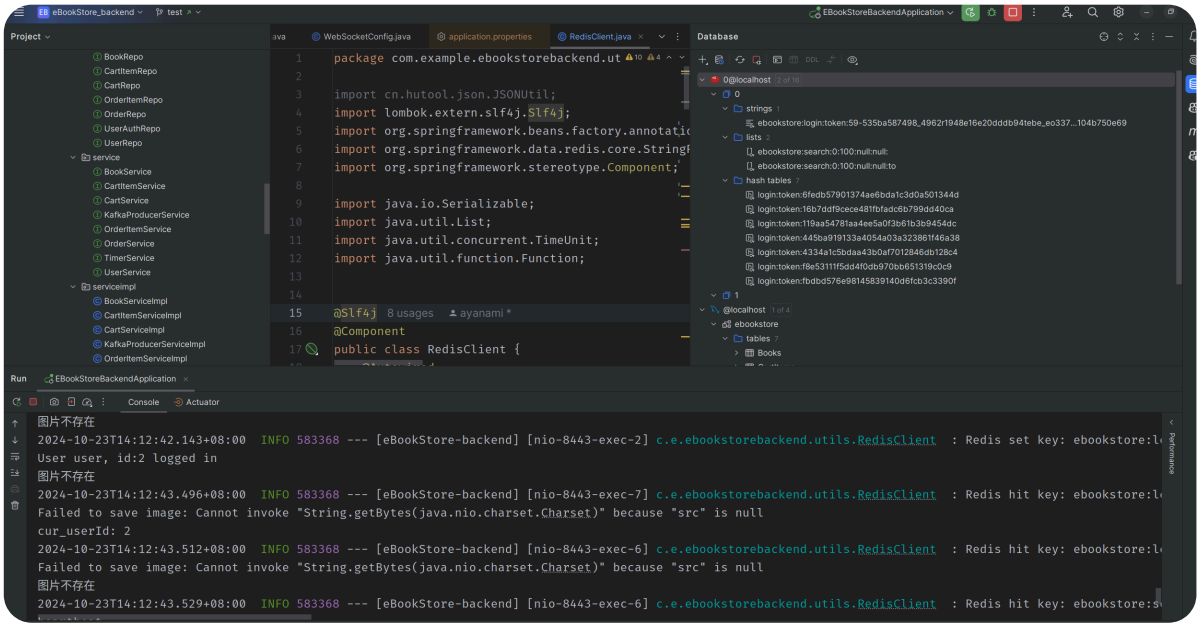
得到日志

首次读写

```
2024-10-23T14:12:42.143+08:00  INFO 583368 --- [eBookStore-backend] [nio-8443-exec-2] c.e.ebookstorebackend.utils.RedisClient  : Redis set key: ebookstore:login:token:59-535ba587498_4962r1948e16e20dddb94tebe_eo337bbfd-0sa87rk369f862607bfcy1-ca363-fb9d1a1ea2a8cbdff47e61fyu104b750e69, value: {"id":2,"role":"user","notes":null,"email":null,"phone":null,"address":null,"firstName":null,"lastName":null,"city":null,"state":null,"balance":0,"status":"normal","userImage":null,"administrator":false,"user":true,"banned":false,"userImagePath":"/images/users/2.jpg","nickname":"user"}
User user, id:2 logged in
图片不存在
2024-10-23T14:12:43.496+08:00  INFO 583368 --- [eBookStore-backend] [nio-8443-exec-7] c.e.ebookstorebackend.utils.RedisClient  : Redis hit key: ebookstore:login:token:59-535ba587498_4962r1948e16e20dddb94tebe_eo337bbfd-0sa87rk369f862607bfcy1-ca363-fb9d1a1ea2a8cbdff47e61fyu104b750e69
cur_userId: 2
2024-10-23T14:12:43.512+08:00  INFO 583368 --- [eBookStore-backend] [nio-8443-exec-6] c.e.ebookstorebackend.utils.RedisClient  : Redis hit key: ebookstore:login:token:59-535ba587498_4962r1948e16e20dddb94tebe_eo337bbfd-0sa87rk369f862607bfcy1-ca363-fb9d1a1ea2a8cbdff47e61fyu104b750e69
2024-10-23T14:12:43.529+08:00  INFO 583368 --- [eBookStore-backend] [nio-8443-exec-6] c.e.ebookstorebackend.utils.RedisClient  : Redis hit key: ebookstore:search:0:100:null:null:
```

后续读写

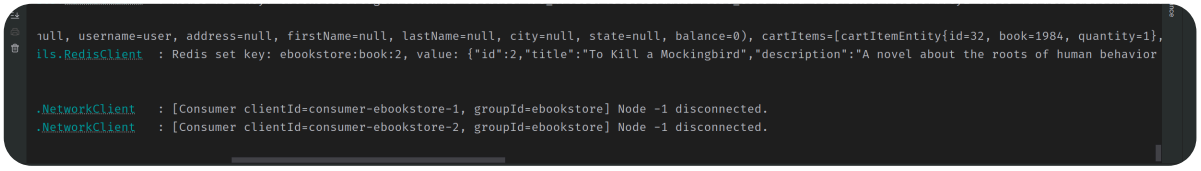
登录并跳转到书籍主页时，redis设置key之后，/user/me等api查询redis缓存的用户信息hit



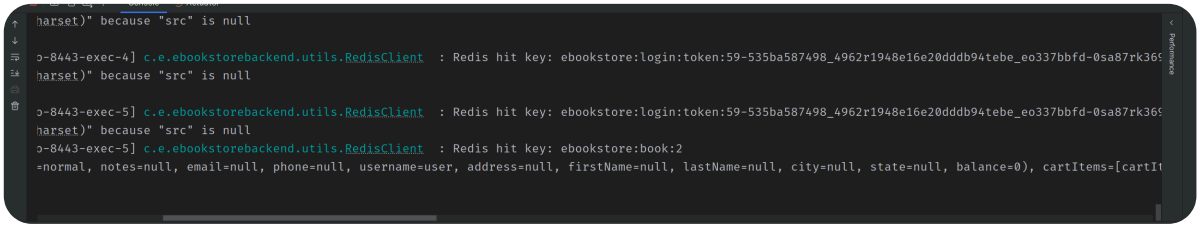
查看到redis中的确有相关数据

关闭redis

之后首次读取，打开id为2的书



刷新前端，后台日志

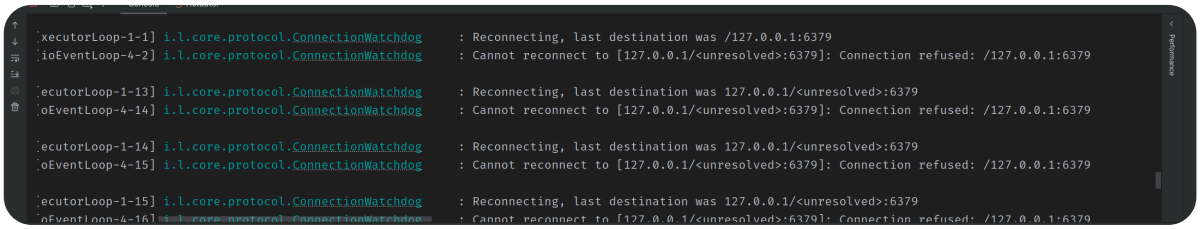


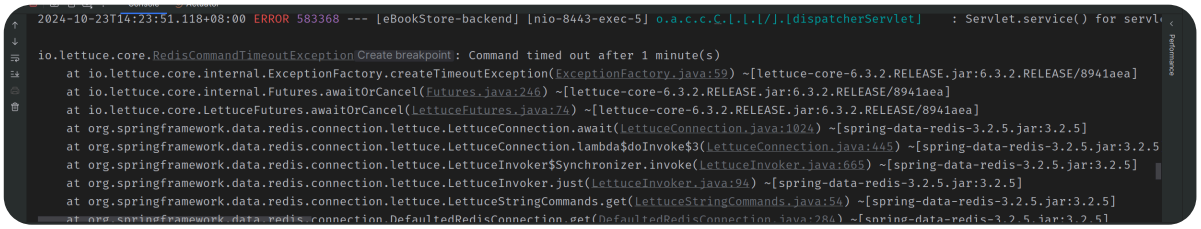
此时，由于前端使用localStorage存储了token，登录hit了redis里面的token

并且查询书籍也hit了先前缓存的数据

之后我使用 systemctl stop redis-server 关闭redis

我的系统在尝试几次重连之后报错redis 连接timeout

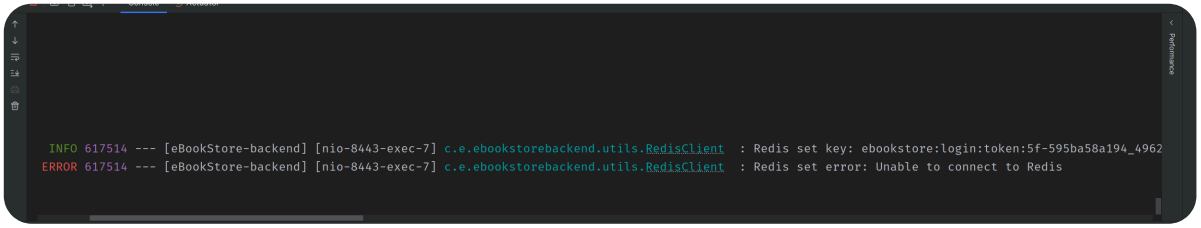




前端无法得到有效的response

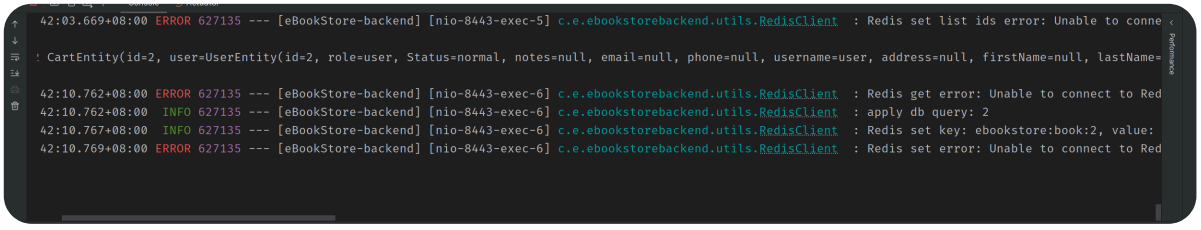
此处是由于Redis的set, get是阻塞操作, 会一直阻塞到超时抛出错误

修改为try-lock之后, 发现还是登录不了, 原因是



redis用于了缓存用户认证的token, 用户登录请求通过后, 后端却缺少比对的信息, 导致拦截器拦截请求

将已有的session之中已经存储的userId也放入拦截器判断后, 能够正常运行(后续将session的数据迁移到ThreadLocal之中)



如图, 虽然无法找到redis, 但最后成功的catch这个connection error, 并查询db

最终redis相关核心代码如下

```java
package com.example.ebookstorebackend.utils;

import cn.hutool.json.JSONUtil;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Component;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;
import java.util.function.Function;
```

```java
@Slf4j
@Component
public class RedisClient {
    @Autowired
    private StringRedisTemplate stringRedisTemplate;
    // ATTENTION: 注意Json序列化库的统一，这里使用的是Jackson，其他库读不到jackson注解
    public void set(String key, Object value, long timeout, TimeUnit unit) {
        try{
            String valueStr = JackSonJsonUtil.toJson(value);
            log.info("Redis set key: {}, value: {}", key, valueStr);
            stringRedisTemplate.opsForValue().set(key, valueStr, timeout,
unit);
        } catch (Exception e) {
            log.error("Redis set error: {}", e.getMessage());
        }

    }
    public <ID extends Serializable>void setListIds(String key, List<ID> ids,
long timeout, TimeUnit unit) {
        // specially optimized for list of ids
        log.info("Redis set list ids({}): {}",key, ids);
        List<String> valueStrList =
ids.stream().map(String::valueOf).toList();
        try{
            stringRedisTemplate.opsForList().leftPushAll(key, valueStrList);
        } catch (Exception e) {
            log.error("Redis set list ids error: {}", e.getMessage());
        }

    }

    public <R> R get(String key, Class<R> type) {
        try {
            String valueStr = stringRedisTemplate.opsForValue().get(key);
            if (valueStr == null || valueStr.equals("null")) {
                return null;
            }
            log.info("Redis hit key: {}", key);
            return JackSonJsonUtil.fromJson(valueStr, type);
        } catch (Exception e) {
            log.error("Redis get error: {}", e.getMessage());
            return null;
        }
    }
    public <ID extends Serializable> List<ID> getListIds(String key, Class<ID>
type) {
        List<String> range;
        try{
            range = stringRedisTemplate.opsForList().range(key, 0, -1);
        } catch (Exception e) {
            log.error("Redis get list ids error: {}", e.getMessage());
            return null;
        }

        if (range == null || range.isEmpty()) {
```

```
            return null;
        }
        if (type == String.class) {
            log.info("Redis hit key: {}", key);
            return (List<ID>) range;
        }
        if (type == Long.class || type == Integer.class) {
            log.info("Redis hit key: {}", key);
            return (List<ID>) range.stream().map(Long::valueOf).toList();
        }
        log.warn("unsupported type: {}", type);
        return null;
    }
    public void delete(String key) {
        stringRedisTemplate.delete(key);
    }
    public boolean tryLock(String key) {
        return stringRedisTemplate.opsForValue().setIfPresent(key, "1",
RedisConstant.LOCK_EXPIRE_TTL, TimeUnit.SECONDS);
    }
    public boolean unlock(String key) {
        return stringRedisTemplate.delete(key);
    }
    public <R, ID extends Serializable> R queryWithPassThrough(String
keyPrefix, ID queryId, Class<R> type, long timeout, TimeUnit unit,
Function<ID, R> dbQuery) {
        R cacheVal = this.get(keyPrefix + queryId, type);
        if (cacheVal != null) {
            return cacheVal;
        }
        log.info("apply db query: {}", queryId);
        R result = dbQuery.apply(queryId);
        if (result != null) {
            this.set(keyPrefix + queryId, result, timeout, unit);
            return result;
        }
        // query non-exist data, set empty object to cache
        this.set(keyPrefix + queryId, "null", timeout, unit);
        return null;
    }

    public <R, ID extends Serializable> R queryHotKey(String keyPrefix, ID
queryId, Class<R> type, long timeout, TimeUnit unit, Function<ID, R> dbQuery)
{
        // 热点key的区别在重建缓存需要拿锁, 避免都去查db导致缓存穿透
        String key = keyPrefix + queryId;
        R cacheVal = get(key, type);
        if (cacheVal != null) {
            return cacheVal;
        }
        // 查db, 拿锁
        try{
            if (tryLock(key)){
                R result = dbQuery.apply(queryId);
                if (result != null) {
                    set(key, result, timeout, unit);
```

```
                    return result;
                }
            }
        } catch (Exception e) {
            throw new RuntimeException(e);
        } finally {
            unlock(key);
        }
        // 未获取到锁, 等待后重试
        try {
            Thread.sleep(100);
            return queryHotKey(keyPrefix, queryId, type, timeout, unit,
dbQuery);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

其中对书籍的CRUD有不同的缓存策略:

读单本书(id查询): redis缓存整本书

范围查询(搜索): redis缓存搜索的参数→得到书的id列表的映射, 再用id列表去db找书

CUD: 写操作不做缓存