Qunar 第四届暨 ITCP 联盟首届 CR 大赛比赛真题

（一）后端【题目】

序号 1

【业务逻辑】：

写压测 case 生成，简单来说就是一个插入数据库的操作，将参数中的 caseList 入库。

【题目】：

```
public boolean batchInsert(List<Case> cazeList) {
    boolean result = false;
    Long start = System.currentTimeMillis();
    try {
        result = caseMapper.batchInsert(cazeList.get(0).getSceneId(), cazeList) ==
cazeList.size();
    } catch (Exception exp) {
        if (exp.getMessage() != null && exp.getMessage().contains("doesn't exist")) {
            caseMapper.createTable(cazeList.get(0).getSceneId());
            caseMapper.addIndex(cazeList.get(0).getSceneId());
            QMonitor.recordOne("db.case.createTable.success");
            result = caseMapper.batchInsert(cazeList.get(0).getSceneId(), cazeList) ==
cazeList.size();
        }
        log.warn(exp.getMessage() != null ? exp.getMessage() : "", exp);
    }
    if (result) {
        log.info("sceneId_{}_groupId_{}： 批量入库（batchInsert）count:{} ， 成功",
cazeList.get(0).getSceneId(), cazeList.get(0).getGroupId(), cazeList.size());
        QMonitor.recordOne("db.case.batchInsert.success", System.currentTimeMillis()
- start);
    } else {
```

```
            log.info("sceneId_{}_groupId_{}： 批量入库（batchInsert）count:{} ， 失败",
cazeList.get(0).getSceneId(), cazeList.get(0).getGroupId(), cazeList.size());
            QMonitor.recordOne("db.case.batchInsert.failed", System.currentTimeMillis() -
start);
        }
        return result;
    }
```

序号 2

【题目】：

```
public class SyncCreateT6OrderService
    protected void doRetry(SyncCreateOrderRequest syncCreateOrderRequest) {
        try {
            createT6OrderFromSyncOrder(syncCreateOrderRequest);
        } catch (Exception e) {
           // 此处省略一些异常处理细节，监控、日志等
            throw e;
        }
    }

    @Transactional(rollbackFor = RuntimeException.class)
    private void createT6OrderFromSyncOrder(SyncCreateOrderRequest
syncCreateOrderRequest) {
        ....
    }
}
```

序号 3

【题目】：

```
public class MqProducer {

      public static final String CONFIG_FILE_PATH = Path.getCurrentPath() +
```

```
        "/../config/mq_producer.config";
        private final DefaultMQProducer producer;
        private static MqProducer instance = null;

        private MqProducer() {
            producer = new DefaultMQProducer();
            producer.init(CONFIG_FILE_PATH); // 路径存在，数据满足预期
            producer.start();
        }

        public static MqProducer getInstance() {
            if (instance == null) {
                synchronized (MqProducer.class) {
                    if (instance == null) {
                        try {
                            instance = new MqProducer();
                        } catch (IOException e) {
                            log.error(" act=MqProducer.getInstance ", e);
                        } catch (MQClientException e) {
                            log.error(" act=MqProducer.getInstance ", e);
                        }
                    }
                }
            }
            return instance;
        }
    }
```

序号 4

【业务逻辑】：

应运营需要，对每日收益最高的几家运营商进行统计，以便更好的跟进代理商的业务。目前采用的收益计算方法是通过系统导出指定排名（当前业务诉求是 Top30）的代理商及其收益信息

代码上下文：

TOP_CAPCITY 指代排名范围，支持 qConfig 配置；

【题目】：

```
try {
        fileName = fileName + System.currentTimeMillis();

        outputStream = response.getOutputStream();

        DownloadScrollSearch downloadScrollSearch = new
DownloadScrollSearch(params);

        List<Profit> result = downloadScrollSearch.getResult();

        if(result.size() > TOP_CAPCITY) {
           // 查询结果大于 TOP 阈值，则只取阈值内的数据
           result = result.sublist(0, TOP_CAPCITY);
        }

        String fullPath = tempFilePath + File.separator + fileName + ".csv";
        FileUtils.download(request, response, fullPath, fileName + ".csv");
        FileUtils.deleteFile(new File(fullPath));

        outputStream.close();
    } catch (Exception e) {
        log.error(e.getMessage());
        return false;
    }

    return true;
```

序号 5

【业务逻辑】：

对于上下文中的代金券列表，取出进行业务处理。

代码上下文：

busListContext 中的 couponList 是通过 dubbo 接口异步获取的 CompletableFuture，利用工具将上下文中的代金券列表取出来进行【业务逻辑】处理。

【题目】：

```
 public void process(BusListContext busListContext) {
 List<CouponInfo> couponList = FutureUtils.getFutureResult("userCouponListFuture",
            busListContext.userCouponListFuture, timeOut, Collections.emptyList());
}
public static <T> T getFutureResult(String futureLogName, CompletableFuture<T> future,
                                    int waitMilliSeconds, T defaultValue) {
    try {
        if (!future.isDone()) {
            log.info(LogConstants.TITLE_Arg1, futureLogName, futureLogName + " not done，wait " + waitMilliSeconds + " ms");
        }
        // max wait xxx ms
        return future.get(waitMilliSeconds, TimeUnit.MILLISECONDS);
    } catch (ExecutionException e) {
        Throwable ex = e;
        if (e.getCause() != null) {
            ex = e.getCause();
        }
        log.warn(LogConstants.TITLE_Arg1, futureLogName, futureLogName + " error, wait " + waitMilliSeconds + " ms", ex);
        return defaultValue;
    } catch (Exception e) {
        QMonitor.recordOne("FutureUtils_getFutureResult_exception");
        log.error("FutureUtils getFutureResult exception,future={}", JsonUtils.toJson(future));
        return defaultValue;
```

```
        }
    }
```

序号 6：

【业务逻辑】：

根据传入的批量红包计划 ids, 查询红包的详细信息返回

【题目】：

```
        // 红包 id 列表
        List<Long> redPlanIdList = req.getRedPlanId();

        List<RedEnvelope4PL> result = Lists.newArrayList();
        Safes.of(redPlanIdList).parallelStream().forEach(redPlanId -> {
            RedPlan plan = NewRedCacheHelper.getRedPlan(redPlanId, logStr);
            if (Objects.isNull(plan)) {
                return;
            }
            List<RedEnvelope4PL> redPlan = getRedEnvelope4PLS(plan, redMetaList,
logStr);
            if (CollectionUtil.isNotEmpty(redPlan)) {
                // 将红包详细信息返回
                result.addAll(redPlan);
            }
        });
```

序号 7：

【题目】：

```
public static void main(String[] args) {
        List<String> testStrArray = new ArrayList<>();
        testStrArray.add("qunar");
        testStrArray.add("ctrip");
        testStrArray.add("elong");
```

```
        for(int i=2; i<testStrArray.size(); i--){
            if(StringUtils.isNotEmpty(testStrArray.get(i))){
                testStrArray.remove(i);
            }
        }
        boolean emp = testStrArray.stream().allMatch(str -> "qunar".equals(str));
        boolean empany = testStrArray.stream().anyMatch(str -> "tujia".equals(str));
        if(emp){
            String q = testStrArray.stream().filter(str ->
"qunar".equals(str)).findFirst().get().substring(0,1);
        }
        if(empany){
            String t = testStrArray.stream().filter(str ->
"tujia".equals(str)).findFirst().get().substring(0,1);
        }
    }
```

序号 8：

【业务逻辑】：

求一个 int 类型正整数的 int 次方取 int 次模（说明问题，同时改正）

【题目】：

```
public static double  pow(int base,int exponent,int mod){
      return Math.pow(base,exponent)%mod;
}
```

序号 9：

【业务逻辑】：

在 tts.2022-10-20-10.log 中有大量的 URL

每行一个，没有空行

对于这些 URL 希望每个都访问一下。并且速度尽量快一些。

不需要纠结这个接口可以重复调用，但是文件名又是写死的。

【题目】：

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

@RequestMapping("/fetch")
@Controller
public class B {
    private final static Executor executor = Executors.newCachedThreadPool();

    @RequestMapping("/url.do")
    @ResponseBody
    public String proceessLog() throws IOException {
        String fileName = "tts.2022-10-20-10.log";
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        final Set<String> visted = new HashSet<>();
        for (String line = reader.readLine(); line != null; line = reader.readLine()) {
            final String url = line.trim();
            executor.execute(new Runnable() {
                @Override
                public void run() {
                    if (!visted.add(url)) {
                        //访问过的 url 不用再次发请求。这里忽略 visted 的内存占用
                        return;
```

```
                }
                sendRequest(url);

            }
        });
    }
    return "OK";
}

private boolean sendRequest(String url) {
    // 访问这个 url, 访问成功返回 true, 否则 fasle
    // 不用考虑这个有什么问题
    return true;
}
}
```

序号 10：

【业务逻辑】：

在 tts.2022-10-20-10.log 中有大量的 URL

每行一个，没有空行

对于这些 URL 希望每个都访问一下。并且速度尽量快一些。

不需要纠结这个接口可以重复调用，但是文件名又是写死的。

【题目】：

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
```

```
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;

@RequestMapping("/fetch")
@Controller
public class B {
    private final static Executor executor = Executors.newCachedThreadPool();

    @RequestMapping("/url.do")
    @ResponseBody
    public String proceessLog() throws IOException {
        String fileName = "tts.2022-10-20-10.log";
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        final Set<String> visted = new HashSet<>();
        for (String line = reader.readLine(); line != null; line = reader.readLine()) {
            final String url = line.trim();
            executor.execute(new Runnable() {
                @Override
                public void run() {
                    if (!visted.add(url)) {
                        //访问过的 url 不用再次发请求。这里忽略 visted 的内存占用
                        return;
                    }
                    sendRequest(url);

                }
            });
        }
        return "OK";
    }

    private boolean sendRequest(String url) {
        // 访问这个 url, 访问成功返回 true, 否则 fasle
```

```
        // 不用考虑这个有什么问题
        return true;
    }
}
```

序号 11：

【业务逻辑】：

在订单表中有一 varchar 字段用于存储序列化的 json 信息，在被序列化的类中有定义的 is 开头的 boolean 类型的属性，通过 FastJson 序列化后存到数据表

当查询订单信息的时候，会对该 JSON 串反序列化，反序列化的工具有可能是 fastjson，也有可能是 gson

其中：JSON.toJSONString 使用的是 fastjson

【题目】：

```
    private Gson gson = new Gson();

    @Data
    public static class OrderExtend{
        private Long snCode;
        private boolean isWmsSend;
    }

    @Test
    public void fastjsonAndGson(){
        OrderExtend orderExtend = new OrderExtend();
        orderExtend.setSnCode(110L);
        orderExtend.setWmsSend(true);

        String extendJsonStr = JSON.toJSONString(orderExtend);
        OrderExtend orderExtendByGson = gson.fromJson(extendJsonStr,
OrderExtend.class);
        System.out.println(orderExtendByGson);
```

```
    }
```

序号 12：

【业务逻辑】：

1、tagger 系统会协助运营人员圈定一批用户进行营销活动的投放，量级在百万到千万量级。

2、 推送平台需要将这些用户信息从数据库中读取出来做一些预处理。UserService 就是来做这个读取并预处理工作的。

【题目】：

```
public class UserService {
    @Autowired
    private UserDAO userDao;
public void preproccessUser(String deviceTableName){
        int totalCount = 10000000; //1000 万行数据
        int pageSize = 1000; //分页读取，每页读取 1000 条记录
        int pageCount = totalCount/pageSize ;// 一共需要读取的页数
        for(int pageIndex =0;pageIndex<pageCount;pageIndex++){
            List<Object> objects = userDao.selectByPage(deviceTableName,pageIndex *
pageCount, pageSize);
            this.preproccess(objects);
        }
    }
    //完成数据预处理
    public void preproccess(List<Object> objectList){
        //预处理
      for(Object obj : objectList) {
            // 每条验证和规范处理
        }
    }
}
```

@DAO

```
interface UserDAO{
    @SQL("select * from :1 limit :2 , :3")
    List<Object> selectByPage(String tableName, int from , int size);
}
```

序号 13：

【业务逻辑】：

模拟下单后商家回调下单结果的一段逻辑

【题目】：

```
package com.quna.business;

public class SupplierCallback {
    public void callBackNotice(String type, String json, Integer a, Integer b) {
        Lock lock = new RedisLock();
        lock.lock();
        processWorkA(a, b);
        try {
            processWorkB(type);
        } finally {
            lock.unlock();
        }
    }

    public void processWorkB(String param, String json) {
        switch (param) {
            case "success":
                System.out.println(json);
                //doSomething
                break;
            case "fail":
                System.out.println("it's fail");
                //doSomething
```

```
                break;
            default:
                System.out.println("default");
        }
    }

    public Integer processWorkA(Integer params1, Integer params2) {
        Boolean flag = false;
        Integer params3 = null;
        return flag ? params1 * params2 : params3;
    }
}
```

序号 14：

【业务逻辑】：

基础数据与携程比价后调用中转点庖丁接口，发现有中转报价缺失时会发 QMQ 消息，当缺失原因为【政策包报价不包含当前中转点过滤】时，需要记录当前航线中转点，并按时间先后补齐缺失中转点城市至最大数量

代码上下文：

1、data 为基础数据推送 QMQ 消息 bean,内有中转航组信息和缺失原因

2、中转航组信息机场码需转城市码

3、获取航线锁，使用 zset 存储缺失中转点至最大数量

4、QMQ 推送消息 QPS 为 2~3

【题目】：

```
        String depAirport1 = data.getDep1();
        String arrAirport1 = data.getArr1();
        String depAirport2 = data.getDep2();
        String arrAirport2 = data.getArr2();
        String flyDate1 = data.getFlyDate1();

        if (!Objects.equals(arrAirport1, depAirport2)) {
```

```
            QMonitor.recordOne("smart_flight_price_miss_mid_different");
        }
        //缺失的中转点
        String missedMidAirport = StringUtils.isBlank(arrAirport1) ? depAirport2 :
arrAirport1;
        if (StringUtils.isBlank(depAirport1) || StringUtils.isBlank(arrAirport2) ||
StringUtils.isBlank(missedMidAirport) || StringUtils.isBlank(flyDate1)) {
            logger.error("缺少必要信息，本次中转点不补充
depAirport1:{},arrAirport2:{},missedMidAirport:{},flyDate1:{}", depAirport1, arrAirport2,
missedMidAirport, flyDate1);
            QMonitor.recordOne("smart_flight_price_miss_param_miss");
            return;
        }

        String lockKey = null;
        boolean tryLock = false;
        try {
            //机场码转城市码
            String depCity1 = InfoCenter.getCityCodeFromAirportCode(depAirport1);
            String missedMidCity =
InfoCenter.getCityCodeFromAirportCode(missedMidAirport);
            String arrCity2 = InfoCenter.getCityCodeFromAirportCode(arrAirport2);

            lockKey = getLockKey(depCity1, arrCity2, flyDate1);
            //尝试获取航线锁
            tryLock = RedisLockUtil.checkLock(transferSedis, lockKey,
QConfigUtils.getConfig("union_price_miss_lock_expire_time", 3000));
            if (!tryLock) {
                logger.info("smart_flight_price_miss_lock_check is locked {},{},{}", depCity1,
arrCity2, flyDate1);
                QMonitor.recordOne("smart_flight_price_miss_lock_check_is_locked");
                return;
            }
```

```
            String redisKey = ctripMissMiddleCityService.buildCacheKey(depCity1,
arrCity2, flyDate1);
            Long sedisSize = transferSedis.zcard(redisKey);
            int selectMaxCount =
QConfigUtils.getConfig(QconfigConstant.SELECT_MAX_COUNT,
QconfigConstant.DEFAULT_SELECT_MAX_COUNT);
            long overSize = sedisSize - selectMaxCount;
            if (overSize >= 0) {
                transferSedis.zremrangeByRank(redisKey, 0L, overSize);
            }
            transferSedis.zadd(redisKey, System.currentTimeMillis(), missedMidCity);
            transferSedis.expire(redisKey,
QConfigUtils.getConfig("union_price_miss_mid_city_expire_time", 60000));
            QMonitor.recordOne("smart_flight_price_miss_put_zset");
        } catch (Exception e) {
            logger.error("获取缺失中转点错误", e);
            QMonitor.recordOne("smart_flight_price_miss_city_add_error");
        } finally {
            if (tryLock && StringUtils.isNotEmpty(lockKey)) {
                //解锁
                RedisLockUtil.unlock(transferSedis, lockKey);
            }
        }
```

序号 15：

【业务逻辑】代码：无

【题目】：

```
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import org.assertj.core.util.Lists;
import java.util.List;
```

```
@Slf4j
public class DemoMain {
    public static void main(String[] args) {
        List<Integer> idList = Lists.newArrayList(1, 2);
        //关闭对应 id 的状态
        idList.parallelStream().forEach(DemoMain::closeStatus);
    }

    /**
     * 关闭对应 id 的状态
     */
    private static void closeStatus(Integer id) {
        //查询状态
        Status status = selectStatusById(id);
        log(id, status);
        //如果当前是开启状态 则变为关闭
        if (status == Status.OPEN) {
            //这里把 id 加进去 表示不同 msg
            status = Status.CLOSE.reason("关闭当前状态" + id);
        }
        log(id, status);
    }

    /**
     * 查询状态
     */
    private static Status selectStatusById(int id) {
        return Status.OPEN;
    }
    public static void log(Integer id, Status status) {
        log.info("id:{} 当前的状态是:{} 变更原因为:{}", id, status.name, status.reason);
    }
```

```
    /**
     * 模拟实际问题的枚举类
     */
    @Getter
    enum Status {
        OPEN(1, "开启"),
        CLOSE(2, "关闭");
        /**
         * 状态
         */
        private int code;
        /**
         * 名称
         */
        private String name;
        /**
         * 状态变更原因
         */
        private String reason;
        Status(int code, String name) {
            this.code = code;
            this.name = name;
        }
        public Status reason(String reason) {
            this.reason = reason;
            return this;
        }
    }
}
```

序号 16：

【业务逻辑】：

本实例为搜索用户筛选品牌信息场景，通过运营配置相关品牌映射关系实现召回相关的品牌商品数据

【题目】：

```
public class KeywordMapBrandTransferAnalyzer extends Analyzer{
    private String configKey;

    @Override
    boolean doAccept(ProcContext context) {
        //从上下文获取接口传入分类品牌信息
        Set<SearchCategory> categorySet = context.getParamPgCate();
        if (CollectionUtils.isEmpty(categorySet)){
            return true;
        }
        //从配置中获取品牌映射关系
        Map<Integer, Set<Integer>> transferMapping =
ConfigCenterValues.getBrandTransferMapping(configKey);
        for (SearchCategory category : categorySet){
            Set<Integer> brands = brandMap.get(category.getBrandId());
            if (category.getBrandId() != 0 && CollectionUtils.isNotEmpty(brands)){
                //如果有配置品牌，则添加进上下文召回
                for (Integer brand : brands){
                    if (brand != category.getBrandId()){
                        SearchCategory mapCategory = new
SearchCategory(category.getCateId(), brand, category.getModelId());
                        categorySet.add(mapCategory);
                    }
                }
            }
        }
        return true;
```

```
    }
}
class SearchCategory {
    Integer cateId;
    Integer brandId;
    Integer modelId;

    public SearchCategory(Integer cateId, Integer brandId, Integer modelId) {
        this.cateId = cateId;
        this.brandId = brandId;
        this.modelId = modelId;
    }

    public Integer getCateId() {
        return cateId;
    }
    public void setCateId(Integer cateId) {
        this.cateId = cateId;
    }
    public Integer getBrandId() {
        return brandId;
    }
    public void setBrandId(Integer brandId) {
        this.brandId = brandId;
    }
    public Integer getModelId() {
        return modelId;
    }
    public void setModelId(Integer modelId) {
        this.modelId = modelId;
    }
}
```

序号 17：

【业务逻辑】：

促销活动查询用户库存

【题目】：

```
public List<PlatformUserStock> queryAllPlatformUserStock(long userId) {
        final List<PlatformUserStock> platformUserDayStocks =
this.platformUserStockService
                .queryUserStock(userId, StoreLimitTypeEnum.EVERY_DAY.getCode());
        final List<PlatformUserStock> platformUserAllStocks =
this.platformUserStockService
                .queryUserStock(userId, StoreLimitTypeEnum.TOTAL_LIMIT.getCode());
        platformUserDayStocks.addAll(platformUserAllStocks);
        return platformUserDayStocks;
    }

    public List<PlatformUserStock> queryUserStock(Long userId, int type) {
        if (userId == null) {
            return Collections.emptyList();
        }
        return platformUserTotalStockMapper.queryUserStock(userId, type);
    }
```

序号 18：

【业务逻辑】：

在 handle 方法中，需要通过多次分页查询取出库中原始数据，并通过多线程方式并行过滤原始数据中符合要求的数据，作为最终结果返回。

【题目】：

```
import com.google.common.collect.Lists;

public class Demo {
```

```
    private final ThreadFactory threadFactory = new
ThreadFactoryBuilder().setNameFormat("demo-pool").build();

    private ExecutorService threadPool = new ThreadPoolExecutor(
            10, 20, 10, TimeUnit.SECONDS,
            new LinkedBlockingQueue<>(100),
            threadFactory,
            (r, executor) -> log.error("...")
    );

    public List<JSONObject> handle() {
        List<JSONObject> result = Lists.newArrayList();
        int pageNum = 1;
        int pageSize = 500;
        JSONObject queryResult = query(pageNum, pageSize);

result.addAll(filter(queryResult.getJSONArray("data").toJavaList(JSONObject.class)));
        Integer total = queryResult.getInteger("total");
        while (pageNum * pageSize < total) {
            pageNum++;
            queryResult = query(pageNum, pageSize);

result.addAll(filter(queryResult.getJSONArray("data").toJavaList(JSONObject.class)));
        }
        return result;
    }

    private List<JSONObject> filter(List<JSONObject> originData) {
        CountDownLatch countDownLatch = new CountDownLatch(originData.size());
        List<JSONObject> result = Lists.newCopyOnWriteArrayList();
        for (JSONObject data : originData) {
            threadPool.submit(() -> {
                try {
```

```
                if (// 满足条件) {
                result.add(data);
            }
        }finaly {
            countDownLatch.countDown();
        }
    });
    try {
        countDownLatch.await();
    } catch (Exception e) {
        log.error("countDownLatch await failed");
        QMonitor.recordOne("demo.countDownLatch_await.failed");
    }
    return result;
}

/**
 * 返回体结构：
 * total：int 型，记录总量
 * data：List<JSONObject>，真正的查询数据
 */
private JSONObject query(int pageNum, int pageSize) {
    // 分页查询
}
}
```

序号 19：

代码逻辑：

originalLineRemark 由“供应商名称&订单号”组成，并且是外部传过来的，现需要把 originalLineRemark 字段拆开，确保拆开后供应商名称是系统存在的

【题目】：

```
    private void matchAndSetSupplierNameAndOrderNo(String originalLineRemark,
ReconLedgerDetail detail) {
        List<String> remarkSplitter = Splitter.on("&").splitToList(originalLineRemark);
        String originalSupplierName = remarkSplitter.get(0);
        List<MPrepaySupplierApiVo> matchedSupplierList =
mPrepaySupplierApi.selectByFuzzySupplierName(originalSupplierName);
        for (MPrepaySupplierApiVo supplierVo : matchedSupplierList) {
            String dbSupplierName = supplierVo.getSupplierName();
            String matchSupplierName = dbSupplierName + "&";
            if (!(originalLineRemark.startsWith(matchSupplierName))) {
                continue;
            }
            String replaceLineRemark =
originalLineRemark.replaceFirst(matchSupplierName, StringUtils.EMPTY);
            detail.setSupplierName(dbSupplierName);
            detail.setOrderNo(replaceLineRemark);
            break;
        }
    }
```

序号 20：

【业务逻辑】：

echo 为需要实现接口，echoimpl 为实现类，在该实现类基础上增加代理类 EchoProxy，预期在 SpringbootApplication 注入 echo 属性，通过其 echo 方法执行代理类和实现类逻辑

【题目】：

springboot 启动类:

```
@Slf4j
@SpringBootApplication
@RestController
@PropertySource(value = {"classpath:/application.properties"})
@EnableAspectJAutoProxy(proxyTargetClass = false)
```

```
public class SpringbootApplication {

    @Autowired
    private EchoImpl echo;

    @RequestMapping("/echo")
    public String echo() {
        echo.echo();
        return "success";
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringbootApplication.class,args);
    }
}
```

api 定义:

```
public interface IEcho {
    void echo();
}
```

实现类定义:

```
@Component
@Slf4j
public class EchoImpl implements IEcho {
    @Override
    public void echo() {
        //do echo logic
        log.info("do origin echo");
    }
}
```

代理类定义:

```
@Slf4j
@Aspect
@Component
public class EchoProxy {

    @Before("execution(* tianqi.springaop.api.IEcho.echo(..))")
    public void preEcho() {
        log.info("do pre echo");
    }
}
```

序号 21：

【业务逻辑】：

机票报价代理商政策批量导入时，会给代理商加锁，只允许一个任务对数据库进行写操作。

【题目】：

```
public static boolean lockImport(String key, String value, int expire) {
        String redisKey = makeKey(key);
        try {
            if (redisClient.setnx(redisKey, value) >= SETNX_SUCCESS) {
                redisClient.expire(redisKey, expire);
                return true;
            }
        } catch (Exception e) {
            QMonitor.recordOne("redis_add_error");
        }
        return false;
    }
```

序号 22：

【业务逻辑】：

在机票首页有一个乘机人填写框，通过填写框可以添加本次搜索的乘机人。在乘机人填写框上有个推荐逻辑，推荐我的联系人列表里的最近有过生单记录的乘机人 2 人，乘机人选取顺序是生单时间最近的 2 人。

代码上下文：

本段代码是过滤掉乘机人证件类型不是身份证的乘机人。

【题目】：

```
private void certsFilter(OmPassengersResponse omPassengersResponse) {
    List<OmPassengersResponse.Passenger> passengers =
omPassengersResponse.getPassengers();
    // 按照订单创建时间排序
    passengers = omPassengersResponse.getPassengers().stream()
                .sorted((a, b) -> a.getOrderTime().compareTo(b.getOrderTime()))
                .collect(Collectors.toList());

    // 无手机号不推荐
    passengers = omPassengersResponse.getPassengers().stream()
            .filter(passenger -> StringUtils.isNotBlank(passenger.getPhoneNum()))
            .collect(Collectors.toList());

    // 无证件过滤,根据姓名和身份证号去重
    passengers = passengers.stream()
                .filter(passenger -> CollectionUtils.isNotEmpty(passenger.getCerts()))
                .collect(Collectors.collectingAndThen(Collectors.toCollection(() -> new
TreeSet<>(Comparator.comparing(compare()))), ArrayList::new));
    if (passengers.size() > configVal.getNum()) {
        passengers = passengers.subList(0, configVal.getNum());
    }
    omPassengersResponse.setPassengers(passengers);
}

/**
```

```
 * 根据姓名和身份证号去重
 */
private Function<OmPassengersResponse.Passenger, String> compare() {
    return p -> p.getName() + p.getCerts().get(0).getNumber();
}
```

序号 23：

【业务逻辑】：

规则校验 充血模式开发，在 entity 中增加一些必要的逻辑代码，例如重写 get 方法等

代码上下文：

规则校验简易代码，RuleParamEntity 为规则校验的入参，runQlExpress 为规则执行代码。

【题目】：

```
class RuleParamEntity implements Serializable {
    private static final long serialVersionUID = -78169511179385889 41L;
    /**
     * 规则编号
     */
    private String ruleNo;
    /**
     * 用户名称
     */
    private String userName;
    /**
     * qunar 手机号
     */
    private String qunarMobile;

    public String getQunarMobile() {
        if (StringUtils.isNotBlank(qunarMobile)) {
            return qunarMobile;
        }
        if (StringUtils.isNotBlank(userName) && StringUtils.isBlank(qunarMobile)) {
```

```
            UserDetail userDetail = UserClient.queryEncryptUserByName(userName);
            qunarMobile = userDetail.getMobile();
        }
        return qunarMobile;
    }
}

    public boolean runQlExpress(ExpressRunner runner, RuleParamEntity request,
TbfRuleConfEntity entity) {
        try {
            QMonitor.recordOne("RuleComponent_ruleChecking_" + entity.getKeyword());
            DefaultContext<String, Object> context = new DefaultContext<String,
Object>();
            Object curInfo = ParseBeanUtils.invokeMethod(request, entity.getKeyword());
            if (curInfo == null) {
                log.info("runQlExpress get curInfo blank ! request : {} entity : {}",
JSONObject.toJSONString(request), JSONObject.toJSONString(entity));
                QMonitor.recordOne("RuleComponent_ruleChecking_" +
entity.getKeyword() + "_NOT_VERIFIED");

entity.setVerificationResult(VerificationResultsEnum.NOT_VERIFIED.getCode());
                return VerificationResultsEnum.NOT_VERIFIED;
            }
            Object res =
runner.execute(entity.getQLExpress(ParseBeanUtils.fieldIsString(request,
entity.getKeyword())), context, null, true, false);
            if (!((Boolean) res).booleanValue()) {
                entity.setVerificationResult(VerificationResultsEnum.DENIED.getCode());
                QMonitor.recordOne("RuleComponent_ruleChecking_" +
entity.getKeyword() + "_DENIED");
                return VerificationResultsEnum.DENIED;
            }
            entity.setVerificationResult(VerificationResultsEnum.PASSED.getCode());
```

```
            QMonitor.recordOne("RuleComponent_ruleChecking_" + entity.getKeyword()
+ "_PASSED");
            return VerificationResultsEnum.PASSED;
        } catch (Exception e) {
            log.error("ruleChecking error request : {} entity : {}",
JSONObject.toJSONString(request), JSONObject.toJSONString(entity), e);
            QMonitor.recordOne("RuleComponent_runQlExpress_Exception");
        }
        entity.setVerificationResult(VerificationResultsEnum.DENIED.getCode());
        return VerificationResultsEnum.DENIED;
    }
```

序号 24：

【业务逻辑】：

更新当前用户的基本信息和地址信息，该操作必须要求用户是登录态；

该方法每周访问用户较多

【题目】：

```
package com.qunar.qboss.demo;

import com.google.common.base.Function;
import com.google.common.collect.Lists;
import org.springframework.util.StringUtils;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import java.util.List;

@RestController
public class DemoController {
```

```
    private final UserService userService;

    public DemoController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping("/update")
    public Object updateUserInfo(@Valid @RequestBody UserUpdateCommand
command, HttpServletRequest request) {
        String openId = getCurrentUser(request).getOpenid();
        if (StringUtils.isEmpty(openId)) {
            return ApiResult.forbidden("用户未登录，不允许进行操作");
        }
        List<Address> addresses = Lists.transform(command.getAddresses(),
                (Function<AddressDTO, Address>) input ->
DTOMapper.INSTANCE.toDTO(input));
        User user = userService.queryUser(command.getOpenId());
        for (Address address : addresses) {
            address.setOpenId(openId);
        }
        DTOMapper.INSTANCE.copyToEntity(command, user);
        user.setAddresses(addresses);
        userService.save(user);
        return ApiResult.success();
    }
}
```

序号 25：

【业务逻辑】：

量房单上都需要处理人，在一些情况下，需要对单子的人员进行变更操作。

代码上下文：

orderDao 和 orderTransferLogDao 是引入的数据库表 dao 的操作。EhrPartUserInfo

userInfo = ehrManager.getUserInfo(userCode);是根据 usercode 查询 EHR 接口获取人员信息。

【题目】：

```
public BizResult < Boolean> changeUser(String orderCode, String userCode) {
    if (StringUtils.isBlank(orderCode) ¦¦
 StringUtils.isBlank(userCode)) {
 return BizResult.error(ErrorCode.PARAM_ERROR.getCode(), "参数不能为空");
}
 // 校验数据
EhrPartUserInfo userInfo = ehrManager.getUserInfo(userCode);         if (Objects.isNull(userInfo))
{
return BizResult.error(ErrorCode.PARAM_ERROR.getCode(), "请输入正确的人员编号");         }
OrderEntity orderInfo = orderDao.getByOrderCode(orderCode);         if
(Objects.isNull(orderInfo)) {
return BizResult.error(ErrorCode.PARAM_ERROR.getCode(), "量房单信息不存在");
}
if (OrderStatusEnums.CANCELED.getCode().equals(orderInfo.getStatus())                ¦¦
OrderStatusEnums.FINISHED.getCode().equals(orderInfo.getStatus())) {
return BizResult.error(ErrorCode.PARAM_ERROR.getCode(), "量房单状态不正确，无法进行人员变更");
}
// 更新数据
this.saveData(orderInfo,userInfo);
return BizResult.success(true);
}

@Transactional(rollbackFor = Exception.class)
public void saveData(OrderEntity orderInfo, EhrPartUserInfo userInfo) {
// 更新量房单表人员信息
UpdateWrapper < OrderEntity> updateWrapper = new UpdateWrapper < >();
updateWrapper.lambda().eq(BaseCommonEntity::getId,
orderInfo.getId())                .eq(BaseCommonEntity::getIsDel,
```

```
YesOrNoEnum.NO.getState())                .set(OrderEntity::getOrderZeCode,
userInfo.getEmpCode());
boolean update = orderDao.update(updateWrapper);
log.info("更新结果：{}", update);
// 插入人员变更记录
        OrderTransferLogEntity orderTransferLogEntity = new OrderTransferLogEntity();
orderTransferLogEntity.setAcceptUserCode(userInfo.getEmpCode())                .setApplyUserCod
e(userInfo.getEmpCode())
.setCreateTime(new Date())
.setUpdateTime(new Date());
orderTransferLogDao.save(orderTransferLogEntity);
}
```

（二）前端【题目】

序号 26：

背景：运行环境 - React Native； TouchableOpacity，等其它依赖已经正常引入；

【题目】：

```
 */
const item = ({title, onPress, styles}) => (
    <TouchableOpacity activeOpacity={1} onPress={onPress}
style={styles.containerStyle>
        <Text style={styles.title}>{title}</Text>
    </TouchableOpacity>
);

export default class ItemComponent extends Component {

    constructor() {
        super();
    }
```

```
    onItemPress = () => {
        console.log('press');
    }

    render() {
        return (
            <View>
              <item title={this.props.title} onPress={this.onItemPress} style={styles}/>
            </View>
        )
    }
}
```

序号 27：

【题目】：下面是一个倒计时 UI 组件的实现，请找出该组件实际被项目使用时，可能存在的问题；(所有外部依赖都已经正常引入，只关心倒计时逻辑本身即可；)

```
*/
import React, { useEffect, useState, useCallback } from "react";
import { View, Text } from "react-native";

const getCountdown = (endDate) => {
    const FormatEndTime = endDate.replace(/\-/g, "/");
    const endtime = new Date(FormatEndTime).getTime();
    const nowtime = new Date().getTime();
    const lefttime = endtime - nowtime,
        lefth = Math.floor(lefttime / (1000  60  60)),
        leftm = Math.floor((lefttime / (1000 * 60)) % 60),
        lefts = Math.floor((lefttime / 1000) % 60);

    return {
        h: lefth >= 10 ? lefth.toString() : "0" + lefth,
        m: leftm >= 10 ? leftm.toString() : "0" + leftm,
```

```
        s: lefts >= 10 ? lefts.toString() : "0" + lefts,
    };
};

function CountDown(props) {
    let timer = null;
    const [time, setTime] = useState({
        h: "00",
        m: "00",
        s: "00",
    });

    const runCountDown = useCallback(() => {
        const {h, m, s} = getCountdown(props.endTime);

        if(h == '00' && m == '00' && s == '00'){
            clearInterval(timer);
        }

        setTime({h, m, s});

    }, [time]);

    useEffect(() => {
        timer = setInterval(() => {
            runCountDown();
        }, 1000);
    }, []);

    const { h, m, s } = time;

    return (
            <View style={{paddingTop: 50}}>
```

```
            <Text>
                剩余时间:{`${h}:${m}:${s}`}
            </Text>
        </View>
    );
}
class demo extends QView {
    render() {
        return <CountDown endTime="2022-12-12 16:23:00"/>;
    }
}
```

序号 28：

【题目】：

```
 */
// 编程语言：JavaScript, 运行环境：React Native；
// 业务场景上下文：火车票占座成功从服务器获取座位号展示出来以提升用户支付意愿，假设接口返回之后数据交由 handleResult 函数处理，数据结构如下：
// let resp = {
//     isSuccessful: false,                          // 请求结果是否成功
//     errorMsg: '服务器繁忙，请稍后再试',                    // 请求失败时用于展示
//     seatList: ['10 车 24 座', '8 车 8 号上铺','9 车 9 号一人软包']     // 用于展示列表的数据
// }
// 答题者可认为已经通过请求拿到了上述数据结构的数据（提示：返回的数据最全集是跟上面一致的，实际的数据可能不完全相同），只需关心下面拿到请求结果后的后续处理逻辑；

function handleResult(resp = {}){
    let {isSuccessful = false, errorMsg = ''} = resp;
    if(!isSuccessful && errorMsg.length > 0){
        showFailReason(errorMsg);
        return;
```

```
    }
    let {seatList = []} = resp;
    showList(seatList);
}
function showList(seatList = []){
    let list = seatList.map(x => `<list>${x}</list>`);
    render(list);
}
function showFailReason(errorMsg){
    console.log('some function to show fail reason:', errorMsg);
}
function render(list){
    console.log('some function to render list:', list);
}
```

序号 29：

代码逻辑：字符串形式的 css 样式在 app 端改成 rem 形式的字符串

【题目】：

```
*/
function stringCssToRem(css) {
    let cssArr = css.split(/;\s|\n/);
    let result = cssArr.reduce((pre, cur) => {
        let curArr = cur.split(/:\s*/);
        let resCur = cur;
        if(curArr[1].includes('px')) {

            let number = curArr[1].match(/\d+/);
            let resNumber = number / 100;
            let value = resNumber + 'rem'
            resCur = `${curArr[0]}:${value}`
        }
        if(pre) {
```

```
            return `${pre};${resCur}`
        } else {
            return `${resCur}`
        }

    }, "")
    return result;
}
/**
```

序号 30：

*** 背景提示 1：**运行环境：ReactNative；代码中的 styles 字段是全局变量，不需要考虑为空的情况。

*** 背景提示 2：**DemoComponent 中会接收的 props 当中有 item 对象，且 item 对象最多可能包含：number, numberUnit, showRedText, img, title, subTitle 这几个 key；

*** 【题目】：**

```
 */
export default class DemoComponent extends Component {
    constructor(props) {
        super(props);
    }

    _appStateChange(nextAppState){
        console.log(nextAppState);
    }

    componentDidMount() {
        Dimensions.addEventListener('change', this._appStateChange.bind(this));
    }

    componentWillUnmount() {
        Dimensions.removeEventListener('change', this._appStateChange.bind(this));
    }
```

```
  render() {
    const item = this.props.item;
    if (!item) {
      return null;
    }
    return (
      <View>
        <View style={styles.cardNumWithPot}>
          <Text style={styles.cardNumber} numberOfLines={1}>
            {item.number}
            <Text style={styles.cardNumberUnit}>{item.numberUnit}</Text>
          </Text>
          {item.showRedText && <RedView />}
        </View>
        {Boolean(item.img) && <Image style={styles.cardIcon} source={{ uri:
item.img }} />}
        <Text
          style={[
            styles.cardLabel,
            { marginTop: Boolean(item.img) ? 4 : 6 }
          ]}
          numberOfLines={1}
        >
          {item.title}
        </Text>
        {!!item.subTitle && <Text style={styles.cardSubTitle} numberOfLines={1}>
          {item.subTitle}
        </Text>}
      </View>
    )
  }
```

```
}
```

序号 31：

```
* 背景：运行环境 React Native； ES6+
*     通过 promise 异步获取数据，过程中需要本地持久化存储的数据，promise 数据返回为 bool 类型，且需要处理错误回调
* 【题目】：
*/
getStatus() {
    return new Promise(async (resolve, reject) => {
      const isShow = await AsyncStorage.getItem('isShow');
      fetchData(isShow,  (res) => {
        if (!res || res.error) {
          throw new Error('error');
        }
        resolve(res.show);
      })
    })
};

// 调用
this.getStatus().then((show) => {
    // 其他处理
    if (show) {
    // xxxxx
    AsyncStorage.setItem('isShow', show);
    }
}).catch((e) => {
    // 业务的异常上报
    console.log(e);
});
```