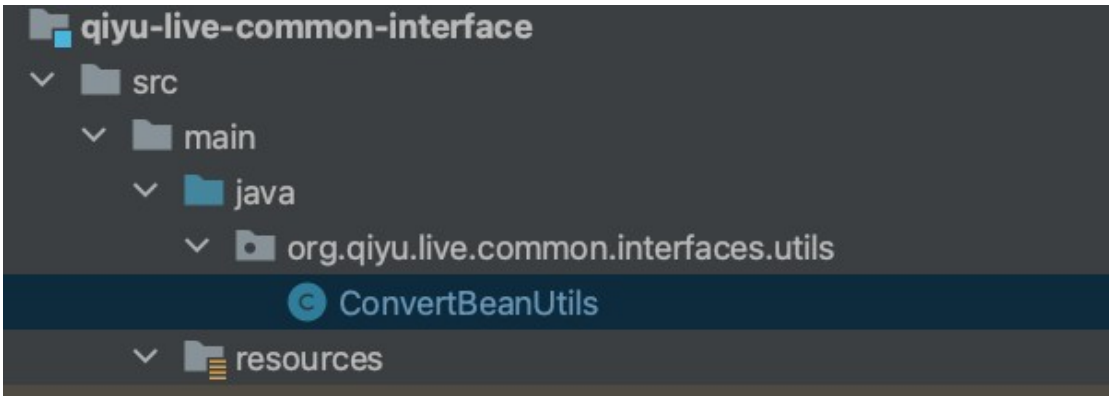
4-11 高并发下用户查询如何提速

代码调整点

1. qiyu-live-api 的 dubbo 配置，统一用 application.yml 进行管理，同时关闭掉 qos 设置

```
XML
dubbo:
  protocol:
    name: dubbo
  registry:
    address: nacos://127.0.0.1:8848?namespace=qiyu-live-
test&&username=qiyu&&password=qiyu
  application:
    name: qiyu-live-api
    qos-enable: false
```

1. common-interface 中的 ConvertBeanUtils 包路径做下调整，用一个 utils 包去存放。



Redis 引入用户中台使用

单用户查询

在 UserService 中使用的时候做些调整：

```
Java
@Resource
```

```
private UserProviderCacheKeyBuilder userProviderCacheKeyBuilder;

private static final UserDTO NOT_EXIST_OBJ = new UserDTO(-1L);

@Override
public UserDTO getByUserId(Long userId) {
    if (userId == null) {
        return null;
    }
    String key =
userProviderCacheKeyBuilder.buildUserInfoKey(userId);
    UserDTO userDTO = redisService.getObj(key, UserDTO.class);
    if (userDTO != null) {
        return
NOT_EXIST_OBJ.getUserId().equals(userDTO.getUserId()) ? null :
userDTO;
    }
    userDTO =
ConvertBeanUtils.convert(userMapper.selectById(userId),
UserDTO.class);
    if (userDTO != null) {
        redisService.setObj(key, userDTO, 30, TimeUnit.MINUTES);
    } else {
        //防止缓存穿透，预防一些恶意请求再次请求落度 db 中
        redisService.setObj(key, NOT_EXIST_OBJ, 3,
TimeUnit.MINUTES);
    }
    return userDTO;
}
```

批量用户查询实现

定义一个批量查询的接口：

```
Java
/**
 * 批量查询用户信息
 *
 * @param userIdList
 * @return
 */
Map<Long,UserDTO> batchQueryUserInfo(List<Long> userIdList);
```

进行底层的实现：

```Java
@Override
public Map<Long, UserDTO> batchQueryUserInfo(List<Long>
userIdList) {
    if (CollectionUtils.isEmpty(userIdList)) {
        return Maps.newHashMap();
    }
    userIdList = userIdList.stream().filter(id -> id != null && id
> 100).collect(Collectors.toList());
    if (CollectionUtils.isEmpty(userIdList)) {
        return Maps.newHashMap();
    }
    List<String> userInfoKeyList = userIdList.stream().map(userId
->
userProviderCacheKeyBuilder.buildUserInfoKey(userId)).collect(Coll
ectors.toList());
    List<UserDTO> cacheList = redisService.mget(userInfoKeyList,
UserDTO.class);
    if (cacheList.size() == userIdList.size()) {
        return
cacheList.stream().collect(Collectors.toMap(UserDTO::getUserId, x
-> x));
    }
    List<Long> userIdInCache = cacheList.stream().map(userDTO ->
userDTO.getUserId()).collect(Collectors.toList());
    List<Long> userIdNotInCache =
userIdList.stream().filter(userId -> !
userIdInCache.contains(userId)).collect(Collectors.toList());
    //不在缓存的 key 才从数据库查询
    List<UserDTO> totalQueryList =
this.queryUserInfoFromSplitTable(userIdNotInCache);
    totalQueryList.addAll(cacheList);
    List<Long> existUserIdList =
totalQueryList.stream().map(UserDTO::getUserId).collect(Collectors
.toList());
    Map<String, String> mSetMap = new HashMap<>();
    Map<Long, UserDTO> resultMap =
totalQueryList.stream().collect(Collectors.toMap(UserDTO::getUserI
d, x -> x));
    userIdList.forEach(userId -> {
        String cacheKey =
userProviderCacheKeyBuilder.buildUserInfoKey(userId);
        //代表 id 查询不到
```

```
        if (!existUserIdList.contains(userId)) {
            mSetMap.put(cacheKey,
JSON.toJSONString(NOT_EXIST_OBJ));
            resultMap.put(userId,NOT_EXIST_OBJ);
        } else {
            mSetMap.put(cacheKey,
JSON.toJSONString(resultMap.get(userId)));
        }
    });
    //数据库查询出来的对象要重新塞入缓存中
    redisService.mSet(mSetMap, 30 * 60);
    return resultMap;
}
```

将上述的接口以 rpc 的形式暴露出去。

最后在 api 层实现远程调用：

```Java
@GetMapping("/batchQueryUserInfo")
public Map<Long,UserDTO> batchQueryUserInfo(String userIdStr) {
    String[] idStr = userIdStr.split(",");
    List<Long> userIdList = new ArrayList<>();
    for (String userId : idStr) {
        userIdList.add(Long.valueOf(userId));
    }
    return userRpc.batchQueryUserInfo(userIdList);
}
```