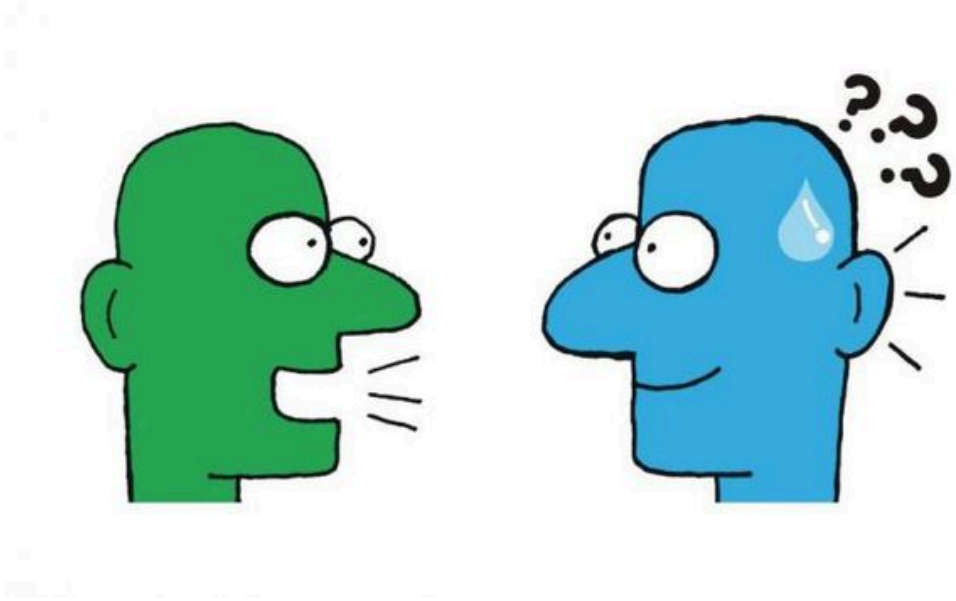
一、规范的目的：

1、高效的沟通-同样的语言、同样的世界：

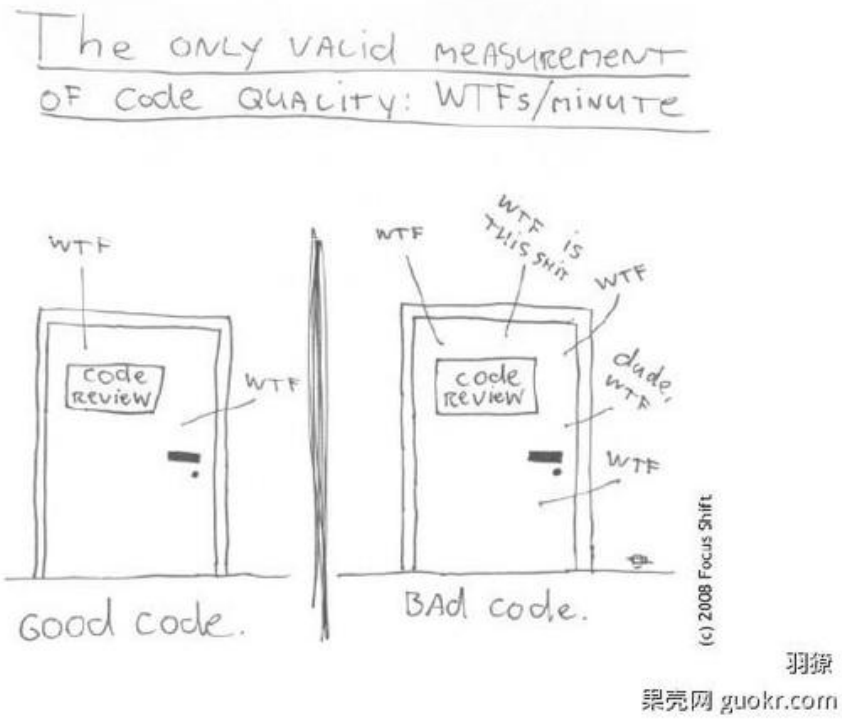


我们尊重个人风格，但是五花八门的个人风格就像不同国家的语言，我们为了交流、了解对方，就必须花费时间去翻译对方的语言。

这个过程中耗费时间、信息没有100%传递。

风格过于个性化的代码，队友很难再短时间内看清楚，CodeReview效果大打折扣。

2、提升工程质量



还是那句话，代码的整洁程度和质量成正比。

- 整洁代码易于测试：模块清晰、单元化强，易于编写测试case。
- 整洁代码易于codeReview：降低队友同步信息的成本，很容易发现问题，减少设计缺陷、bug被忽略的可能。

3、提高“开发迭代”速度

也许有人会质疑：“投入那么多精力把代码写整洁，花费的时间也许会很长，我简单实现立马就能写完。"

我想说：是的，简单实现可以最快把代码写完。

但我所指并非“写代码”，而是“开发迭代”，它包含两层意思：

- 开发与交付： 包括 写代码、CodeReview、测试、bugfix
- 迭代：即下一次“开发与交付”

清晰的代码，可以提高 CodeReview、测试、bugfix 的效率，不仅如此，他还会提高迭代（下一次）开发的效率。

而冗长、不清晰的代码，会导致 CodeReview、测试、bugfix 效率降低，更不用说下一次迭代了。

二、聊一聊

1、老生常谈： 错误码 or 异常

错误码：input&output

（1）描述：一个函数就是一次“输入、输出”，而程序往往需要处理边界问题（如返回数据不合法、网络断掉了），所以这种形式必须将所有边界问题以“输出”的形式返回给调用者。

（2）尝试：

- 提供者代码：

```
/**
```

```
* 接口定义:根据inputA、inputB获得Output

* @param inputA

* @param inputB

* @return OutPut

*/

public BizResult<OutPut> getSimpleCpaTaskById(Long inputA, Long inputB) {

if (inputA == null || inputB == null) {

    return BizResult.fail(InnerBizRetCodeMsg.ParamInvalid);

}

ResultDTO<AdgroupCompositeTaskDTO> resultDTO = null;


//-----------调用接口取得数据

try {

    resultDTO = taskService.getAdgroupTaskById(inputA, SimbaConstants.PRODUCT_ID, inputB);

} catch (Exception e) {

    return BizResult.fail(InnerBizRetCodeMsg.RemoteInvokeException);

}


//-----------检查接口调用成功

if (resultDTO == null || !resultDTO.isSuccess()) {

    return BizResult.fail(InnerBizRetCodeMsg.RemoteInvokeFail);

}


//-----------检查数据是否存在

AdgroupCompositeTaskDTO resultModel = resultDTO.getResult();

if (resultModel == null) {

    return BizResult.fail(InnerBizRetCodeMsg.SimbaCompositeDTONotExist);

}


//-----------检查数据partA是否合法

AdgroupDTO partA = resultModel.getAdgroupDTO();

if (partA == null) {

    return BizResult.fail(InnerBizRetCodeMsg.AdGroupNotExist);
```

```
    }


    //-----------检查数据partB是否合法
    TaskDTO partB = resultModel.getTaskDTO();
    if (partB == null) {
        return BizResult.fail(InnerBizRetCodeMsg.TaskNotExist);
    }


    //-----------数据转换
    return BizResult.success(convert(partA, partB));
```

}

- 调用者代码:

```
    BizResult<OutPut> result = getSimpleCpaTaskById(shopKeeperId, cpaTaskId);
    if(!result.isSuccess())
    {
        syncLogger.error("shopKeeperId:" + shopKeeperId + ",cpaTaskId:" +
cpaTaskId+","+result.getInnerBizRetCodeMsg());
        return false;
    }
```

（3）问题:

- 业务和异常混淆:

业务和异常处理耦合到一起，最后难以区分什么是异常、什么是业务返回。

- 侵入性:

我们通过“输出”覆盖了各式各样的case，所以所有客户端代码也要对所有case进行处理，工作量可能会double，甚至更多（因为调用链是一环扣一环的）。

（4）优点（谈不上优点的优点）:

我不得不说，我曾经在这条路上坚守了很久。

如果一定要说优点，那就是它强迫你必须考虑所有case，做到面面俱到。

有一句话说的好，如果一个case有可能发生，即便几率不到1%，你也要处理。

异常：input&output&exception

（1）描述:

- 提供方：按照业务逻辑，把事情从头做到尾。如果发生边界问题，就通过exception将问题抛出。
- 调用方：根据自身需要，选择捕获or放过异常。

（2）尝试：

- 提供者代码：

```
   /**
  * 接口定义:根据inputA、inputB获得Output
  * @param inputA
  * @param inputB
  * @return OutPut
  * @exception BizException
  */
  public OutPut getSimpleCpaTaskById(Long inputA, Long inputB) {
      if (inputA == null || inputB == null) {
//1、基本参数校验
          throw new BizException("param ilegal");
      }
      ResultDTO<AdgroupCompositeTaskDTO> resultDTO = invokeHSFGetAdgroupTaskById(inputA,
inputB); //2、hsf调用
      AdgroupCompositeTaskDTO resultModel = resultDTO.getResult();
      return convert(resultModel);
//3、结果包装、转换
  }
```

- 调用者v1（不需要对异常进行容错）：

```
  OutPut outPut = getSimpleCpaTaskById(shopKeeperId, cpaTaskId);
```

- 调用者v2（需要对异常进行容错）：

```
  OutPut outPut = null;
  try{
      outPut = getSimpleCpaTaskById(shopKeeperId, cpaTaskId);
  }catch(Exception e){
      doSomeThing();
  }
```

- invokeHSFGetAdgroupTaskById：

```
  private ResultDTO<AdgroupCompositeTaskDTO> invokeHSFGetAdgroupTaskById(Long inputA, Long
inputB) {

      String req = "inputA:"+inputA+",inputB:"+inputB;



      ResultDTO<AdgroupCompositeTaskDTO> resultDTO = null;

      //-----------调用接口取得数据

      try {

          logger.info("[getAdgroupTaskById][req]"+req);
//2.1.a发出去的请求记录日志

          resultDTO = taskService.getAdgroupTaskById(inputA, SimbaConstants.PRODUCT_ID,
inputB);

          logger.info("[getAdgroupTaskById][res]"+JSONOBJECT.toJsonString(resultDTO));
//2.1.b得到的结果记录日志

      } catch (Exception e) {
//2.2异常处理

          logger.info("[getAdgroupTaskById][res]",e);
//2.1.c得到的异常记录日志

          throw new BizException("invoke exception:", e);

      }



      if (!resultDTO.isSuccess()) {
//2.3错误处理

          throw new BizException("invoke fail:" + resultDTO);

      }

      return resultDTO;

  }
```

- convert:

```
OutPut notExist = new OutPut();
private OutPut convert(AdgroupCompositeTaskDTO compositeTaskDTO){
    if (compositeTaskDTO == null) {
        return notExist;
    }
    AdgroupDTO partA = compositeTaskDTO.getAdgroupDTO();
    if (partA == null) {
        throw new BizException("dirty data partA");
    }
    //-----------第4步检查
    TaskDTO partB = compositeTaskDTO.getTaskDTO();
    if (partB == null) {
        throw new BizException("dirty data partB");
    }
    OutPut outPut = new OutPut();
    outPut.setCpaTaskId(partB.getId());
    //填充广告主信息
    outPut.setShopKeeperId(partB.getMemberId());
    outPut.setSellerId(partB.getSellerId());
    outPut.setShopId(partB.getShopId());
    //填充广告状态信息
    outPut.setStartTime(partB.getStartTime());
    outPut.setEndTime(partB.getEndTime());
    outPut.setStatus(partA.getOnlineStatus());
    //填充广告价格信息
    outPut.setPrice(MoneyUtil.toYuan(partB.getBidPrice()));
    outPut.setMaxDailyMoney(MoneyUtil.toYuan(partB.getDailyBudget()));
    //填充实时库存信息
    outPut.setLackInventoryEndTime(partB.getLackInventoryEndTime());
    return outPut;
}
```

（3）优点：

- 清晰、简单：

最大优点在于，工程师可以更加专注于业务流程。我说“更加”，是因为边界问题依然要处理，只是通过了一个相对优雅的方式。

- 对调用者友好：

调用者可以明确区分出什么是边界、什么是业务，无需把边界和业务糅杂在一起处理，把自由放还给调用者。

（4）缺点：

- 为“图省事，随随便便抛异常”提供温床

我不得不说一旦发生这种事情我们还不如回归“错误码”的模式,这个问题我们通过团队规范严格控制。

三、规范

1、专注于讲故事，让边界问题飞一会：

提供方：

- handle every case：只要想到的case就要处理，是返回值、还是包装异常抛出？记录日志？
- 万恶的null：返回值尽量不要用null，可以提前实例化一个对象，用来表示“not exist”
- 讲故事：代码逻辑专注于它的功能，对于边界问题采用异常的形式抛出

调用方：

- 保持自由：根据自身需要，选择捕获或者不捕获异常
- I don't case null：无需对返回值做null判断

2、写小方法

（1）水平：注意方法内部缩进

- 最好保证缩进程度只有1~2级
- 建议采用“短路”方法减少缩进

糟糕的写法：

```
     @Override

    public Result<MSignedResponse<MPointOrderDTO>> getOrder(PointOrderGetOption option) {

    String appSecret = signKeyService.getSignKey(option.getPid());

        if(!StringUtils.isBlank(appSecret)){

            PointConsumeOrderDO orderDO =
pointConsumeOrderDevMapper.getOrderByTradeNo(option.getTradeNo());

            if(orderDO!=null){

                logger.info("[getOrder] option is {}, result is {}", JSON.toJSONString(option),
JSON.toJSONString(orderDO));

                if(orderDO.getExchangeType() != AbuConstants.PRODUCT_COMMON_TYPE){

                    AppInfoDO appInfoDO = DOUtil.getAppInfoByPid(appInfoMapper, option.getPid());

                    if (!isOrderBelongsToApp(orderDO, appInfoDO)) {

                        logger.warn("[getOrder] type is inner exchange product, fail validate app
id. option is {}", JSON.toJSONString(option));

                        return failAppNotFound();

                    }

                    return successResult(convertSignedPointOrder(orderDO,
option.getReqId(),appSecret));

                }else {

                    return successResult(convertSignedPointOrder(orderDO,option.getReqId(),
appSecret));

                }

            }else {

                logger.warn("[getOrder] result is  null. option is {}", JSON.toJSONString(option));

                return failOrderNotFound();

            }

        }else{

            return failedResult();

        }

    }
```

推荐的写法:

```
    @Override

    public Result<MSignedResponse<MPointOrderDTO>> getOrder(PointOrderGetOption option) {

    String appSecret = signKeyService.getSignKey(option.getPid());

    if(StringUtils.isBlank(appSecret)){

        return failedResult();

    }

    PointConsumeOrderDO orderDO =
pointConsumeOrderDevMapper.getOrderByTradeNo(option.getTradeNo());

    if (orderDO == null) {

        logger.warn("[getOrder] result is  null. option is {}", JSON.toJSONString(option));

        return failOrderNotFound();

    }

    logger.info("[getOrder] option is {}, result is {}", JSON.toJSONString(option),
JSON.toJSONString(orderDO));

    if (orderDO.getExchangeType() == AbuConstants.PRODUCT_COMMON_TYPE) {

        return successResult(convertSignedPointOrder(orderDO,option.getReqId(), appSecret));

    }

    AppInfoDO appInfoDO = DOUtil.getAppInfoByPid(appInfoMapper, option.getPid());

    if (!isOrderBelongsToApp(orderDO, appInfoDO)) {

        logger.warn("[getOrder] type is inner exchange product, fail validate app id. option is
{}", JSON.toJSONString(option));

        return failAppNotFound();

    }

    return successResult(convertSignedPointOrder(orderDO, option.getReqId(),appSecret));

    }
```

（2）垂直：方法内部只描述一个层级的事情，注意隐藏低层次细节

不好的做法：

```
public static MCpaTaskDetailDTO convertCpaTaskDetail(CpaTaskBizDO cpaTaskDO) {
    MCpaTaskDetailDTO mCpaTaskDetailDTO = new MCpaTaskDetailDTO();
    mCpaTaskDetailDTO.setCpaTaskId(cpaTaskDO.getCpaTaskId());
    mCpaTaskDetailDTO.setShopId(cpaTaskDO.getShopId());
    mCpaTaskDetailDTO.setSellerId(cpaTaskDO.getSellerId());

    //低层级的细节暴露上来
    List<MCpaActionDTO> cpaActionDTOList = Lists.newArrayList();
    for(CpaActionBizDO cpaActionDO : cpaTaskDO.getCpaActionList()){
        MCpaActionDTO mCpaActionDTO = new MCpaActionDTO();
        mCpaActionDTO.setCpaTaskId(cpaActionDO.getCpaTaskId());
        mCpaActionDTO.setCpaActionId(cpaActionDO.getCpaActionId());
        mCpaActionDTO.setCpaActionType(cpaActionDO.getCpaActionType());
        mCpaActionDTO.setRawContent(cpaActionDO.getContent());
        mCpaActionDTO.setOrdered(cpaActionDO.getOrdered());
        cpaActionDTOList.add(mCpaActionDTO);
    }
    mCpaTaskDetailDTO.setCpaActionList(cpaActionDTOList);

    return mCpaTaskDetailDTO;
}
```

推荐的做法:

```
public static MCpaTaskDetailDTO convertCpaTaskDetail(CpaTaskBizDO cpaTaskDO) {
    MCpaTaskDetailDTO mCpaTaskDetailDTO = new MCpaTaskDetailDTO();
    mCpaTaskDetailDTO.setCpaTaskId(cpaTaskDO.getCpaTaskId());
    mCpaTaskDetailDTO.setShopId(cpaTaskDO.getShopId());
    mCpaTaskDetailDTO.setSellerId(cpaTaskDO.getSellerId());
    mCpaTaskDetailDTO.setCpaActionList(convertCpaActions(cpaTaskDO.getCpaActionList()));
    return mCpaTaskDetailDTO;
}

private static List<MCpaActionDTO> convertCpaActions(List<CpaActionBizDO> cpaActionBizDOs){
    List<MCpaActionDTO> cpaActionDTOList = Lists.newArrayList();
    for(CpaActionBizDO cpaActionDO : cpaActionBizDOs){
        MCpaActionDTO mCpaActionDTO = convertCpaAction(cpaActionDO);
        cpaActionDTOList.add(mCpaActionDTO);
    }
    return cpaActionDTOList;
}

private static MCpaActionDTO convertCpaAction(CpaActionBizDO cpaActionDO){
    MCpaActionDTO mCpaActionDTO = new MCpaActionDTO();
    mCpaActionDTO.setCpaTaskId(cpaActionDO.getCpaTaskId());
    mCpaActionDTO.setCpaActionId(cpaActionDO.getCpaActionId());
    mCpaActionDTO.setCpaActionType(cpaActionDO.getCpaActionType());
    mCpaActionDTO.setRawContent(cpaActionDO.getContent());
    mCpaActionDTO.setOrdered(cpaActionDO.getOrdered());
    return mCpaActionDTO;
}
```

3、描述力强的方法名、参数

（1）方法名使用驼峰形式、动宾短语、动词要准确

- 为方法起个好的名字，英语不够好？

那先用中文描述吧。

（2）假如很难用一个短语描述怎么办

- 是不是模块程度不够，挑战一下自己？
- 是不是思考不够透彻？

4、trycatch什么时候用

（1）最顶层（mtop、controller）写一个最大的try catch

这是最后一道防线，但不是懒惰的温床。

controller、mtop层负责包装用户提示。

（2）低层try catch采用就近处理方式

- handle every case one by one：只在发生异常的地方try、catch，不要写大包围trycatch
- 就近处理：在发生的地方记日志、加报警。

5、日志记录的方法

where？在哪里

when？什么时候

how？上下文是什么

what？结果是什么

6、外部接口，包装细节

1、参数校验

2、调用hsf

```
2.1日志记录


    2.1.a发出去的请求记录日志

    2.1.b得到的结果记录日志

    2.1.c得到的异常记录日志


2.2异常处理


2.3错误处理
```

3、数据类型封装

7、其他：

- 幽灵代码：已经不会有调用的代码，请自行删除；如果这块代码很精美舍不得删，可以放在个人文件夹，放在git上也不会丢。
- 代码格式化、import优化：每次提交代码前，请做代码格式化。

四、愿景

1、高效需要大家共同努力

2、共同营造优秀的代码环境

3、《代码整洁之道》