本文将着重就《当错误、异常发生的时候，日志记录怎么记？抛什么异常》的规范进行落地，旨在统一日志、异常规范，从而保证日志记录准确、监控精准、风格一致。

下面，进入正题：

1、什么时候记录日志?

- 异常发生时：
  异常发生的位置是第一现场，必须立刻对现场进行记录（具体记录方法下面说），不要指望着把日志交给别人处理。
- 对核心节点记录输入、输出：
  关键节点（如麻吉宝对接simba服务）可以记录输入、输出信息，一方面可以用于联调期间调试，另一方面及时保留证据。

2、怎么记录日志 when、where、what、how

when：发生的时间，logback的配置就能支持。

where：

- 类名：　logback的配置就能支持
- location：方法名（or 自定义名称）,logback的配置就能支持方法名

what：

- 大致方向：　这里需要团队共建一份ErrorCat表，**PE对ErrorCat表的内容进行监控。**
- 到底发生了什么事情：简单的一句话，都可以大大提高问题的精准度。比如大致方向已经说明“调用HSF的时候超时了”，那么这里就可以描述“具体是哪个HSF超时了，哪个业务功能使用的HSF超时了”。直接提高日志的跟踪效率

how：

- 怎么发生的：说的就是所谓的问题现场，“你干了什么，搞成了什么样子”。

举个栗子：when：红色 where：绿色 what：蓝色 how：紫色

2015-07-16 22:46:53,556 ERROR [CommonHandler] param.error on call 检验用户提交答案,cause error data ： {"callTime":"2015-07-16 22:46:53","callLocation":"检验用户提交答案","params":{"context":: {"userId"}}

3、异常怎么抛

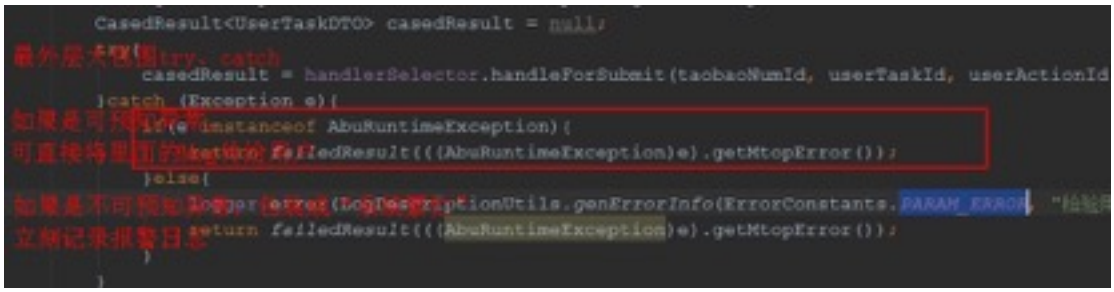
日志已经记录完毕、现场能够完整的恢复，然后我们做什么？

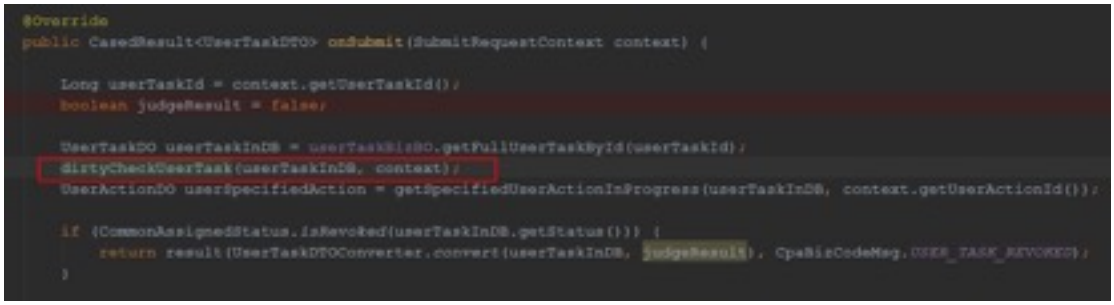
- 放心的抛异常：
  “异常”是业务逻辑的边界，严格的说它并不是业务逻辑，如果为了处理这个边界问

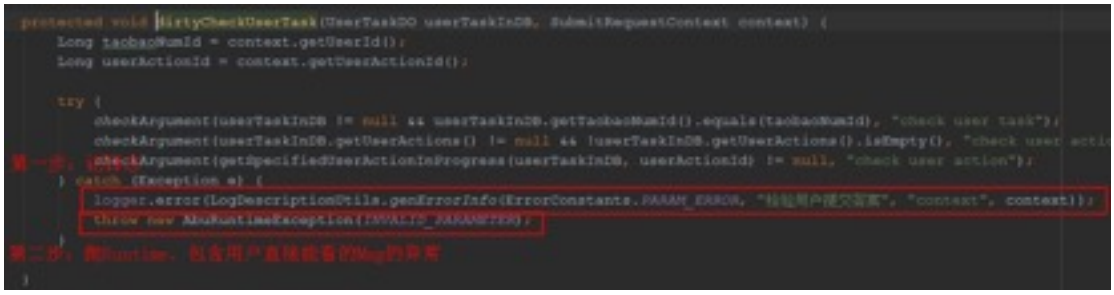
题，我们要写很多边界处理代码（不整洁、阅读性差、不易迭代）。“异常”可以把直接从业务代码那个跳出，保证代码整洁性。

- 抛Runtime异常
  Runtime异常，就是在暗示“不指望调用方必须处理，因为这并不是业务中的一部分。”
- 抛用户可以直接看的异常
  由于Runtime具有unchecked的特性，所以一旦抛出就很可能被抛到最上层才被处理。正好利用这个特性，我们要将直接可供用户查看的ErrorMsg放在里面。
- 最外层trycatch
  写服务端的代码必须这么做，我不多说了。

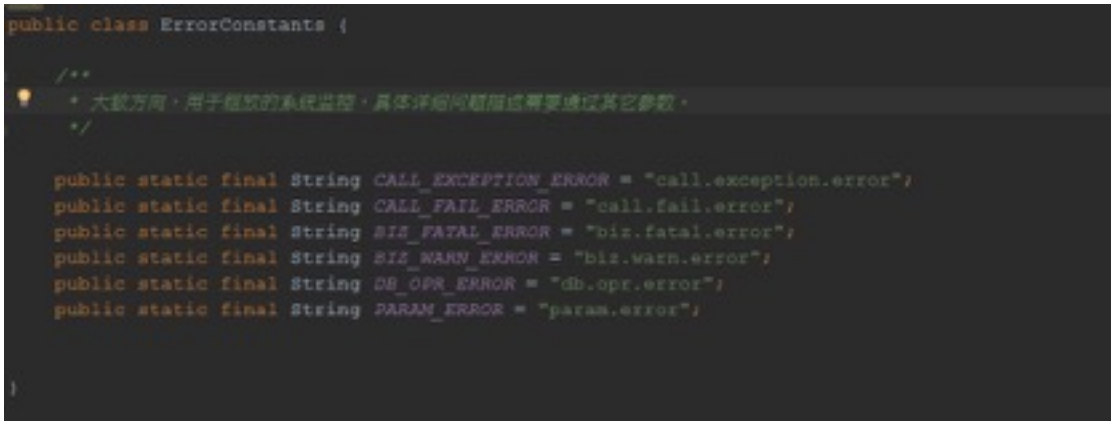
4、代码怎么写







5、ErrorCat(PE监控的内容)



```
public class ErrorConstants {

    /**
     * 大致方向，用于程序的系统监控，具体详细问题描述需要通过其它参数。
     */

    public static final String CALL_EXCEPTION_ERROR = "call.exception.error";
    public static final String CALL_FAIL_ERROR = "call.fail.error";
    public static final String BIZ_FATAL_ERROR = "biz.fatal.error";
    public static final String BIZ_WARN_ERROR = "biz.warn.error";
    public static final String DB_OPR_ERROR = "db.opr.error";
    public static final String PARAM_ERROR = "param.error";


}
```

6、结论

- 常量再细化一下，基于常量做关键字、频率的分级监控。
- 异常发生的位置立即处理（记日志）
- CasedResult专注于业务、AbuRuntimeException用于直达用户传递用户级的消息
- 最上层trycatch捕获所有异常
- alert.log、action.log(关键操作log)、3rd-service.log、system.log(第三方)**********