链滴

【总结】CodeReview 自查要注意的点

作者：xiaodaojava

原文链接：https://ld246.com/article/1609069642463

来源网站：链滴

总述

细数过来。进入阿里大家庭一个多月了，一个月以来，参加了不少CodeReview,虽有开发规约的指引但在Review的过程中，还是会有不少问题暴露出来，本文会总结在CodeReview之前，有哪些可以先查的点，更好的保证代码的健壮性。

代码结构

在CodeReview之前，我们先对代码结构做一次剖析，开头，我们先从最本质的面向对象说起，面向象，有属性，有方法，然后我们对属性做了很多的修饰，比如加static 声明为静态，加 final 声明为量，加 private 声明为私有。同样，我们也会对方法做很多修饰，然后会在方法中调用别的方法，因，在真正讨论CodeReview之前，我们得先讨论下代码结构，如下所示，一个文件中，是分为以下几模块

```
import xxxx
public class XiaoDao{

    public 类型 属性名

    public 回参 方法名(入参){
        一段处理逻辑

        调用其他的方法

            一段逻辑处理

        返回
    }
}
```

要有了上面大致的结构后，我们就可以逐段的去自检。

代码CR

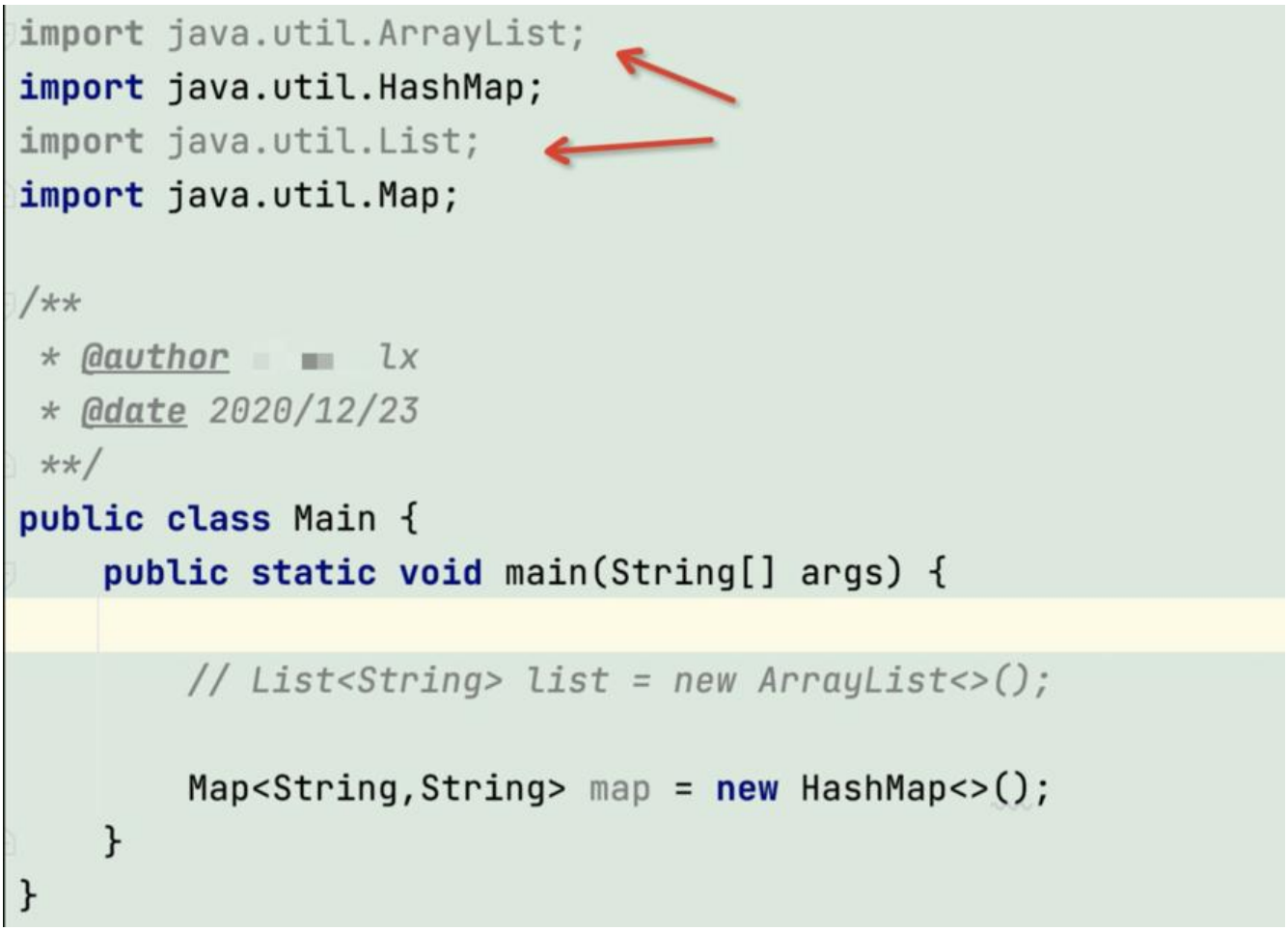
当提到代码CR的时候，那一定是分两大部分， 一部分是代码层面，即import是否合理，属性/方法修是否合理，工具类使用是否合理。第二部分则是看业务逻辑，即写的代码有没有完成即定的产研需求本文也会从这两个方面和大家一起探讨如何自检

代码层面的自检

代码层面的自检，主要是上面所列的代码结构是否是符合现有的业务，代码场景，本文将以两份代码一个对比

import引包

import这个区域经常会被我们忽略掉，因为在新引入一个类的时候 ，IDEA会帮我们自动引入，但是我们删代码的时候，又不会帮我们自动删 除,所以这时候就有了多余的引用，如下所求：



```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 * @author  lx
 * @date 2020/12/23
 **/
public class Main {
    public static void main(String[] args) {

        // List<String> list = new ArrayList<>();

        Map<String,String> map = new HashMap<>();
    }
}
```

如上图所求，对一个数据结构，我们开始想的是用List存，后来还是用Map存，然后把List的声明代给注释掉了，但是上面的import并没有删掉，同样的情况还可能出现在 StringUtils上面，有团队自封装的，有引的Spring里面的，有引的Apache中的，就会在import区域，留下很多无用的引用。

代码格式化

格式化这个事，说大不大，说小不小，就是顺手的一个事，但是有个点要特别注意，只格式化自己修的那一块代码，切记不要全选，不要格式化别人的代码！！

格式化之前：

```
public class Main {

    public static void main(String[] args) {
        TestModel model = new TestModel();
        if("  ".equals(model.getName())){
            System.out.println(JSON.toJSONString(model));
        }else{
            model=new TestModel( name: "  ", age: 27);
        }
        // do something else
    }
```

格式化之后：

```
    public static void main(String[] args) {
        TestModel model = new TestModel();
        if ("  ".equals(model.getName())) {
            System.out.println(JSON.toJSONString(model));
        } else {
            model = new TestModel( name: "  ",  age: 27);
        }
        // do something else
    }
```

public/private

这是一个小细节，可能我们写public会写的比较顺手，或者在开发的时候，把一个公用方法，做了定化改造，或者是把一个私有方法做了通用的抽象等等，在跑了代码逻辑没问题之后,可能就忽略这些小节了，但一个定制化的方法，如果声明成了public，其他开发的小伙伴在不知情的情况下调用了错误逻辑，就有可能引发一些缺陷。如下图所求

```java
public static void main(String[] args) {

    // prepare some data
    onlyForMain();

}

public static void onlyForMain(){
    // do something only for Main Class
}
```

日志

日志这个东西是很难打的一个东西，打少了，无法定位问题，打多了呢，又有太多的无用信息。就本而言，要检查几个必打的地方

入参，调外域接口前的参数，调外域接口返回的参数，进行一大段复杂处理逻辑之后的结果，有异常后的信息，准备返回给上层的返回值，如下所示：

```
public class Main {

    private static final Logger logger = LoggerFactory.getLogger(Main.class);

    public static void main(String[] args) {
        logger.info("Main,main param={}", JSON.toJSONString(args));

        // 经过一系列的复杂的运算得到了param值
        String param = "XiaoDao";
        logger.info("Main,main,key param={}", param);

        try {
            String result = fakeInterface(param);
            logger.info("Main,main,fakeInterface param={},result={}", param, result);
        } catch (Exception e) {
            String msg = String.format("Main,main,fakeInterface,error param=%s", param);
            logger.error(msg, e);
        }

        // 如果有返回值的话，还要记录返回值
        logger.info("Main,main result={}", "result");

    }
```

```
    /**
     * 假装这是外域的接口
     * @param param
     * @return
     */
    private static String fakeInterface(String param) {
        return "fake" + param;
    }

}
```

循环

可以说，我们在写代码的时候，十行有六行都是在对集合(List,Map)处理，如果一个方法中出现对一集合多次遍历的话，就要注意了，是不是可以合并在一起，如下所示：

```
public static void main(String[] args) {
        List<TestModel> modelList = Collections.singletonList(new TestModel());
        // 获取NameList
        List<String> nameList = modelList.stream().map(TestModel::getName).collect(Collectors.
oList());

        // 获取AgeList
        List<Integer> ageList = modelList.stream().map(TestModel::getAge).collect(Collectors.toL
st());

        ////////// 考虑是否可以用下面一个循环代替 //////////

        List<String> names = new ArrayList<>(modelList.size());
        List<Integer> ages = new ArrayList<>(modelList.size());
        for (TestModel model : modelList) {
            names.add(model.getName());
            ages.add(model.getAge());
        }

    }
```

业务CR

发现有些小伙伴在进行CR的时候，不知道要讲什么，直接讲代码吧，没有一个业务前景，听的小伙伴比较懵，讲业务吧，又不知道讲哪些合适。这时候，可以参考本文的表述：

在这次的迭代中，我负责的模块是XXX,其中主要的逻辑是XXXXXXX，涉及到XXX表变更，XXX配置变更，XXX定时任务变更。如涉及到多个配置项和定时任务，可提前列一个表格出来。

然后开始从代码入口处，开始展示代码，就是把上述的代码结构中的每一部分讲解清楚。可参考以下术：

现在我有XXXX信息，要经过XXXX的处理，得到XXXX结果。

然后要调用XXXX方法，得到XXXX结果，然后对XXXX结果进行XXXX的处理，得到XXXXX

最后完成了XXXX业务。

我相信上述这么一段话，在大家开发的时候，一直回响在大家脑海中，CR只不过是把这些话稍加整理再表述出来给大家听。参会的小伙伴也可以顺着讲的业务线，了解这一块的业务，在串数据流的时候也能发现一些处理欠缺可能引起风险的地方。

总结

上述梳理，可以大家平时用以自我Review代码，我一直相信，写代码是一件创造艺术品的过程，是件需要不断打磨的过程，不是说完成了业务功能就可以了，要不断的抽象，整理，沉淀才能写出健壮代码！ 大家一起加油！！