> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

前面一小节集中讲述了 Git 和数据库相关的典型坑点并给出了一些建议。

本节将结合实际的开发经验给出其他各种坑点合集，以帮助大家尽可能地体会开发中可能会遇到的典型坑点，从而尽早避免 。

Java 工程师的进阶之路就是不断的踩坑之路。

聪明人和普通人的区别之一是，聪明人喜欢吸取别人的踩坑经验，而普通人不重视别人的经验，很多道理只有自己亲身经历，遭受挫折才开始重视。

图：好大的一个坑

2.1 二方服务的坑

案例 1：空指针异常

在大一些的公司工作，不免要调用公司其他团队的服务接口，然而二方接口的很多返回值都可能为空。

往往在测试时对接的功能都很正常，结果上线以后，各种空指针异常，囧...

【建议】 写代码代码时一定要注意避免空指针。 具体的避免空指针方法参见第一章的对应小节。

案例 2：幂等性问题

有时候消息可能重复消费，你提供的服务对方也可能会因超时重试，如果没做好幂等，很可能出现重复下单、重复付款等严重的问题。

【建议】对于重复调用会有副作用的接口要注意保持接口的幂等。通常上游传入业务识别号和数据编号，作为幂等的依据，采用数据库唯一键等方式做幂等。

案例 3：二方服务不可用

二方服务不都是可靠的，偶尔可能会出现短暂的不可用。因此大家在和其他团队对接时，要考虑如果对方接口挂掉，我们该如何处理?

比如内容被风控是小概率事件，不应该阻碍核心功能。如果你负责对接风控，如果场景允许，当对方服务异常时可以做降级处理，直接返回风控通过，避免阻塞核心流程。

案例 4：二方接口废弃

我们在对接二方服务时，可能会发现自己找的接口或者对方提供的接口可能被标记废弃。

如果对方服务 jar 包提供了源码包，我们可以看其源码注释，查看是否有废弃的原因并给出替代的新接口，如果没有一定要和对方核实。

案例 5：接口返回值为基本类型

如果对接二方服务时，如果发现对方的接口中如果包含了基本类型的返回值。

因为基本类型的返回值都会有默认值，一定要确认这些默认值对自己的数据有没有副作用。

在某些场景下，很有可能因为这些默认值直造成成一些意想不到坑。

案例 6：SNAPSHOT 包提供的新接口 "丢了"

IDEA 默认不会更新 SNAPSHOT 包，第一次拉取后缓存该 Jar 包。

如果你在 IDEA 中设置 总是更新 SNAPSHOT 可能会发现，二方人员新开发的接口突然找不到了。

因此《手册》中有下面的规定：

> 【强制】线上应用不要依赖 SNAPSHOT 版本 (安全包除外)。
> 说明：不依赖 SNAPSHOT 版本是保证应用发布的幂等性。另外，也可以加快编译时的打包构建。

2.2 环境相关的坑

案例 1：配置错误

很多时候由于我们粗心很有可能上线之前漏了一些配置导致出现一些线上问题。

比如实际开发中就出现某些人将 host 的值配置为 host+port 的形式，而该功能不影响核心流程，程序正常启动，但该功能没有正常工作。

案例 2：测试时发现非常费解的问题

开发中可能偶尔遇到非常诡异的问题，怎么看都应该是对的，但是就是不对，可能卡住很久。

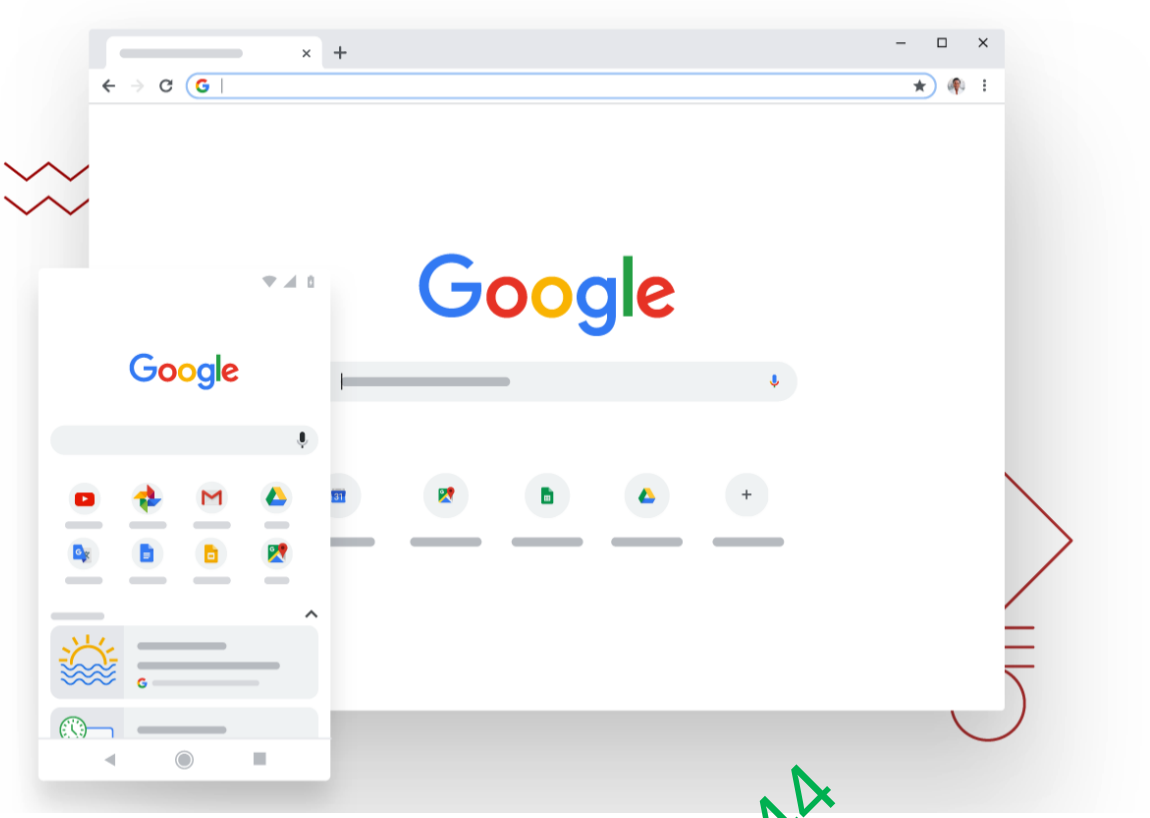
此时要注意是不是环境搞错了。

因为很多大公司都会有好几套环境，如开发环境、QA 环境、QA 性能测试环境、预发环境、预发性能测试环境、和线上环境等。设置同样是 QA 环境还包括通用的 QA 环境和项目特有的 QA 环境。

遇到无法理解的问题，比如发布了项目没报错，但是就是访问不到，大概率是环境搞错了，要注意核对环境。

案例 3：浏览器问题

发现自己的浏览器访问总是有问题，而别人的浏览器可以，orz...



图片来自谷歌浏览器官网。

此时可以尝试开无痕模式，大概率可以解决此类问题。

2.3 测试中遇到的坑

案例 1：场景没回归到

比如你的服务接口提供给 Web 端、 小程序端和安卓 APP 端调用，修改了接口可能只自测了其中两个场景，然后代码上线发现唯一没覆盖的小程序端有 BUG，囧...

【建议】自己的接口修改影响到的地方都要进行回归。

案例 2: 只测试了正常值

作为 Java 开发工程师我们测试自己接口时，总是习惯性测试正常值，而不会甚至想不到去测试异常数据。

接口上线以后，如果出现异常值，很可能就会导致线上问题。

【建议】做好参数防御，编码时要考虑异常情况。

案例 3：服务启动调用了接口没走到断点

在 QA 环境测试时可能会发现服务启动正常，调用的姿势也正确，在接口代码第一行断点都进不来，囧...

很多朋友可能会因为类似问题卡很久，其实此种情况很可能被拦截器拦截掉了...

2.4 经验不足的坑

案例 1：switch 缺少 default 语句

《手册》中规定：在一个 switch 块内，都必须包含一个 default 语句并且放在最后，即使它什么代码也没有。

写 switch 的有些场景是根据类型来做不同的事情，比如商品类型，但是类型可能会增加，但是代码可能会滞后。

因此尽量要编写 default 语句，写 default 语句时要思考未来新增类型时会有什么影响？

【建议】如果走到 default 逻辑，如果可能会有不良影响的情况，即使不抛异常也要打印警告或者错误日志。

案例 2： List 转 Map key 重复问题

如果编写一个 List 转 Map 的函数，可能有些新手会这么写：

```
private static Map<String, Integer> getResult(List<SimpleObject> data) {
    return data.stream()
            .collect(Collectors.toMap(SimpleObject::getName,
SimpleObject::getValue));
}
```

这么写会存在一些问题吗，比如空指针异常，因为集合对象可能为 null, 集合的元素也可能为 null。

还有一个问题是 key 有可能重复，抛出：java.lang.IllegalStateException: Duplicate key 异常信息。

肿么办，orz...

可以参考如下写法：

```
private static Map<String, Integer> getResult(List<SimpleObject> data) {
    if (CollectionUtils.isEmpty(data)) {
        return new HashMap<>(0);
    }
    return data.stream()
            .filter(Objects::nonNull)
            .filter(obj -> Objects.nonNull(obj.getValue()))
            .collect(Collectors.toMap(SimpleObject::getName,
SimpleObject::getValue,
                    Integer::sum));
}
```


案例 3：属性拷贝的坑

开发时，很多人对自己要求不高，为了省事，大量使用 Bean 拷贝工具。

由于某些工具在特殊场景下会触发其自身的 BUG；还有多次属性拷贝通过代码很难看出哪些属性会填入了值，不必要地消耗了精力；修改对象属性时或者类型不对应时无法在编译器报错，增加了出错的风险；

具体建议参考 Java 属性拷贝的正确姿势小节。

案例 4：越权问题

比如你提供一个下载文档的接口，没有校验文档和当前用户的权限关系，而是直接根据文档 ID 就可以下载，那么如果用户发现规律，递增或者递减传入其他用户的文档 ID，就会发生安全问题。

【建议】不要只实现功能，对于需要隔离的资料，要重视权限的校验。

案例 5：消息顺序性

比如有这样一个场景，在短时间内发两次消息通知下游反查你的服务数据，然后更新到他们的库中。

两个消息可能被对方集群中的两台服务器分别消费，反查到的新数据可能先插到了对方的表中，而旧的数据更新到对方库中，导致旧值覆盖新值。

如果可以将两次消息合并为一次，这样就可以避免这个问题；如果是旧版本可丢弃的场景，也可以发送消息时带上时间戳或者版本号，下游更新数据时比对时间戳或版本号，时间更晚或者版本更旧直接丢弃。

案例 6：暴露的服务修改签名

提供给二方用的服务接口，由于需要优化或者升级，而且认为并没有人用，或者已经通知了使用方一起修改，于是大胆地修改了函数签名。

结果新的接口上线以后，发现未评估到的另外一个使用方调用报错，造成故障，orz...

【建议】暴露的接口不到万不得已不要修改函数签名，如果需要修改尽量提供新的接口。

案例 7：没有兜底方案

比如底层新增了一个必传参数，而这个参数是允许传默认的值，开发者认为前端肯定会传入的，就直接透传给底层，结果由于没覆盖所有场景，上线后有一处调用前端没传报错。

【建议】在条件允许的情况下尽可能设计兜底方案。

比如调用二方查询接口时 ip 是必传参数，我们开发时要求前端传给我们，但是我们实际开发时如果不传可以给个默认值，但是走到默认值逻辑时，我们要打印警告日志，引起我们的注意。

这样即使由于某些场景没覆盖到，功能还是 OK 的。

本小节介绍了开发中可能遇到的典型坑点，并给出了对应的建议。

当然还有有各种各样的趟坑姿势，篇幅有限不可能一一介绍，也欢迎大家留言补充。

希望大家能够主动了解开发中可能遇到的坑点，主动积累，从别人的错误中吸取经验，少踩坑。

下一节将根据这几节讲到坑给出 Java 避坑宝典。

}