本章前面两节讲述了 Git 、数据库以及其他方面的各种坑点，并给出了针对某些具体坑点的建议。

本节将综合各种坑点和实际的开发经验，给出更宏观的避坑原则和具体的避坑经验。

2.1 趟坑的原因分析

2.1.1 技术原理理解不到位

由于很多人不重视原理的学习，对很多技术理解不够深入，容易出现误用的情况。

比如 Git 理解不到位，本该从 master 拉取新分支开发新功能，却从另外一个项目分支拉取新的开发分支，导致项目上线带上了本不该发布的功能代码，从而出现故障。

比如对线程池的理解不够深入，使用 Executors 构造无界线程和无界队列的线程池，导致 OOM。

2.1.2 没有养成好的习惯

开发前没有仔细梳理需求的习惯；开发时没有自我代码审查的习惯；测试时没有看错误日志的习惯；上线前没有检查各种配置的习惯；上线过程中和上线后没有关注线上日志的习惯。

2.1.3 经验不够丰富

比如测试时没有充分评估影响面，导致漏测出现故障；

比如线上执行删除操作时，没有开启事务，也没有仔细反复检查；

比如都不认为自己依赖的服务会挂掉，不认为网络抖动会影响到自己的项目；

打印了敏感信息的日志，如果客户的住址、手机号、敏感账户等信息。

比如评估任务之前非常想当然，按照经验评估，而不是核实好每个接口，做好技术方案，留出灵活的时间，导致中间发现很多东西和自己想的不一样，项目经常延期。

2.1.4 缺乏思考

很多人趟坑的另外一个重要原因是思考，不去思考为什么要这么设计，并且设计方案没有在头脑中推演。

由于缺乏思考，产品说啥就做啥，如果产品早期设计有问题，后期大量修改就会跟着被动修改。

2.1.5 缺乏技术追求

很多人开发抱着一种完成任务就好的态度，缺乏技术追求。

2.2 宏观避坑经验

2.2.1 加强学习

重视官方文档。官方文档才是最权威最准确的参考资料，希望可以引起大家足够的重视；

加强专业知识的学习，可以重温大学最重要的四本课：操作系统、计算机网络、数据结构和算法、计算机组成原理，还可以购买黑皮书系列；

平时看书尽量购买经典的技术图书，有些技术图书要反复看；

平时开发时，如果不是特别忙可以进入到源码中，看源码的核心函数的注释和逻辑学习源码；

平时调试问题时，可以使用各种调试技巧来学习源码，比如查看调用栈、执行表达式等；

购买某个技术的专栏。专栏的质量一般都会偏高，大家可以通过专栏的学习，加深对某项技术的理解。

2.2.2 养成好的习惯

要明确每个阶段自己的主要任务，甚至前一个阶段如果提前完成或者有时间可以提前准备后一个阶段的内容。

编码前：认真阅读需求文档，通过思维导图 和各种 UML 图来梳理需求和设计技术方案。

开发中多将自己的代码和 master 进行比对，及时发现自己代码中存在的问题。养成编写单元测试的习惯，单元测试

可以帮助你发现一些粗心导致的错误，帮你发现一些逻辑问题等。写代码时要注意编码规范，日志规范，注释规范等。

养成用静态代码检查插件检查自己代码的习惯，尽可能在编码阶段发现和解决问题。开发过程中一定本地编译通过再提交代码。

测试阶段一定重视错误日志，遇到问题及时修改；测试时不仅要测试正常情况，要以破坏者的角度自测。测试阶段要任何和预期不符的现象都要引起足够的重视。

发布前重新检查上线清单，核查配置是否都已经设置好，通知合作方就绪。

发布阶段及时观察错误日志、观察请求错误情况。

2.2.3 积累开发经验

很多人的思维很奇怪，总是轻视别人的经验，这样容易走弯路，希望大家多从别人身上吸收经验。

每一个自己开发的大项目做完以后都要复盘，反思在从需求分析到上线整个过程中，本次项目自己存在哪些问题，编码中哪个设计不够完善，并总结出可供未来参考的经验。

了解微服务的核心构成，了解每种常用技术存在的主要问题和主要的解决方案。

多看《手册》、《Effective Java》、《重构》、《编写可维护代码的艺术》这饱含者前辈智慧的计算机开发经验的经典图书。

学习和实践高级的调试技巧，帮助自己更快定位问题，更好地学习源码。

要养成 “面向失败编程” 的思维习惯，要考虑各种依赖方出现问题对自己造成的影响，如果有可能可以设计出兜底方案。

多了解和实践软件设计的基本原则：高内聚、低耦合；

开发中多体会设计模式的六大原则：单一职责、里氏替换、依赖倒置、接口隔离、迪米特法则、开闭原则；学习经典设计模式的使用场景，优点、缺点和核心实现逻辑等。

> 注意：这里列举的规则并不是让大家背诵，而是希望大家能实实在在地去理解它们。

当发现公司其他团队或者团队的其他成员的代码出 BUG 甚至发生故障时，要了解出现问题的原因，自己后面开发时要注意规避。

如果有时间可以看看其他同事的技术方案，思考他做方案是为什么会这么设计，这样设计有什么好处等。

多和一些志同道合的朋友交流不涉及具体公司具体代码和业务内容的通用的开发经验。

> 积累一些具体的开发经验，如：

> **加开关**
>
> 如果对某个功能真得没把握，公司提供灰度发布，可以按比例切流量到包含新功能服的务器。如果没提供，可以用 apollo 做功能开关，验证有问题及时关闭；
>
> 如果某个功能有时需要打开，有时需要关闭，设计时可以通过 apollo 来控制，避免打开或关闭时需要修改源码发布。
>
> **加默认值**
>
> 比如调用某个接口，其中 IP 是必传字段，如果不传会报错，如果场景允许，可以在不传时给一个默认值，并加上警告日志，这样就可以避免线上问题。
>
> **加白名单**
>
> 有些数据要走特殊逻辑，可以通过 apollo 加入白名单。
>
> **预留拓展字段**
>
> 有些功能注定会拓展，可以预留一个拓展字段。
>
> 等等。

2.2.4 做技术的思考者

通过本专栏的一些解读，很多朋友会发现：知道是什么，为什么可能比怎么做更重要。

因为很多人之所以不知道该怎么做，恰恰是因为并没有真正理解清楚概念，并没有认真思考过为什么要这样。

2.3 排错技巧

2.3.1 先猜想后验证

大家在遇到坑需要排查问题时，一定要根据已有知识根据可能性列举出几种最大概率的原因，然后再分别验证。

2.3.2 控制变量法

有时候我们分析问题时会发现可能受到多种因素影响，此时我们要尽量让一个因素成为变量，其他的因素成为常量。

这样通过控制其他因素可以有效验证当前的因素是否是错误的主要原因。

2.3.3 缩小范围

通过 F12 看请求和响应信息来定位问题是前端还是后端。

比如 4xx 响应码大都是前端错误； 5xx 响应码大都是后端错误。

2.3.4 换环境大法

比如发现了诡异的问题，我们可以换环境，比如浏览器开无痕模式、换浏览器、换个项目、让同事用他的电脑试试等等。

2.3.5 代码审查大法

代码审查小节给出了非常详细地讲解。

出现问题时，可以重新检查自己的代码逻辑，检查代码是否有性能问题，分析代码可能出错的原因。

2.3.6 善用工具

日志分析大法

遇到问题优先查看日志，平时要熟练掌握 tail、grep、less 等操作日志的指令。

调试大法

前面章节中有讲到 Java 代码调试的基本技巧和高级技巧。

大家通过调试来分析问题时，要灵活运用这些调试技巧。

抓包工具

可以使用 tcpdump 命令抓取网络包来分析问题。

比如分析某个消息是否正常发送出去，可以通过该命令抓取自己的 topic，还可以通过消息的控制台观察消费情况来核实。

反编译和反汇编

对于涉及到源码或者语法糖的问题，大家还可以通过反编译和反汇编来分析问题。

配套监控网站

比如排查消息队列的问题，可以看公司的消息队列对应 topic 和 channel 的相关信息，辅助我们排查问题。

arthas 线上问题诊断工具

可以借助 arthas 的强大功能来分析线上问题。这需要在没有遇到问题时，本地多次训练学习。

2.3.7 搜索引擎

建议大家灵活运用搜索引擎，善于从网上寻找类似错误，找到启发。

多去 StackOverFlow 中去搜索问题。

2.3.8 寻求帮助

紧急的问题自己无法处理，及时寻求帮助。

另外平时遇到一些很难排查的诡异的问题，自己卡住很久可以放一段时间再研究，会发现突然有思路，也可以让同事帮一起研究，有时候自己苦思冥想想不明白的事情，别人很快就可以想出原因，给出更合理的建议等。

最后，建议大家平时没遇到问题的时候多了解别人的坑，本地写 DEMO 演练，这样真正遇到问题时才不会那么方。

本小节分析了开发中遇到坑的原因，主要包括基础不扎实、没有养成好的习惯、缺乏开发经验，缺乏思考等。针对这些问题给出了具体的建议。

希望大家能够更多地从别人的坑中学到经验，增强专业素养，提高编码水平，多产出少犯错。

下一节将对本专栏做最后的总结。

}