

本文由 [简悦 SimpRead](#) 转码，原文地址 www.imooc.com

本章开始我们进入避坑篇，重点讲解开发中的相关坑点和一些避坑经验。

Git 和数据库是我们平时开发中常用到的技术，但是使用不当很容易出坑。

本小节将结合实际的开发经验，讲解 git 和 数据库相关比较有代表性的坑点和如何避坑。

2.1 相关教程、软件

本小节的重点不在于教大家如何学习 Git，而是重点讲解实际开发中可能遇到问题。

本部分小介绍一些经典的 Git 资料，方便大家学习。

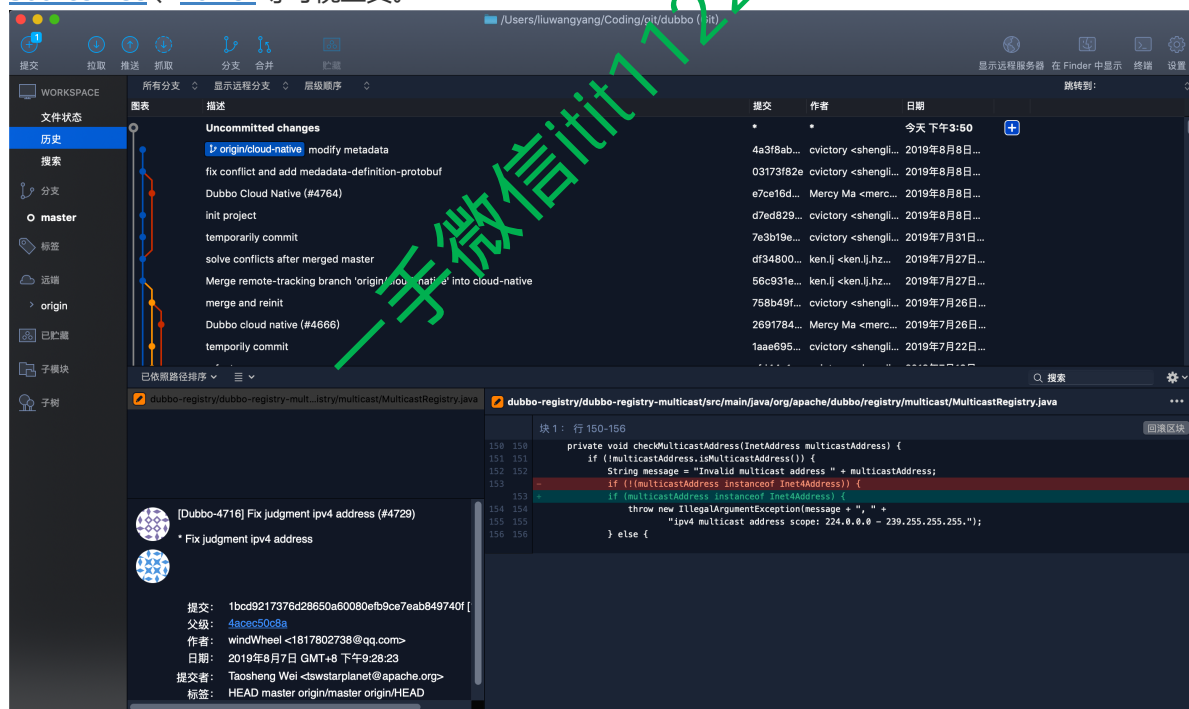
2.1.1 资料

正如本专栏一直强调的：学习技术我们优先去官网看官方文档。

很多人正是因为看官方文档时没时间，才导致遇到 N 多问题浪费更多时间去解决，希望大家一定重视起来。

2.1.2 软件

如果你喜欢原生命令，直接敲命令即可。如果你更喜欢用软件可以用 IDEA 自带的 Git 工具，还可以用 [SourceTree](#)、[Tower](#) 等可视工具。



图：SourceTree 软件界面

2.2 Git 相关的坑点

Git 的相关坑点主要是由于 Git 掌握不熟，还有粗心导致的。

2.2.1 常见错误

案例 1：检出错误

对于有错误提示的错误，一定要认真看错误提示的内容。

如下面的提示：

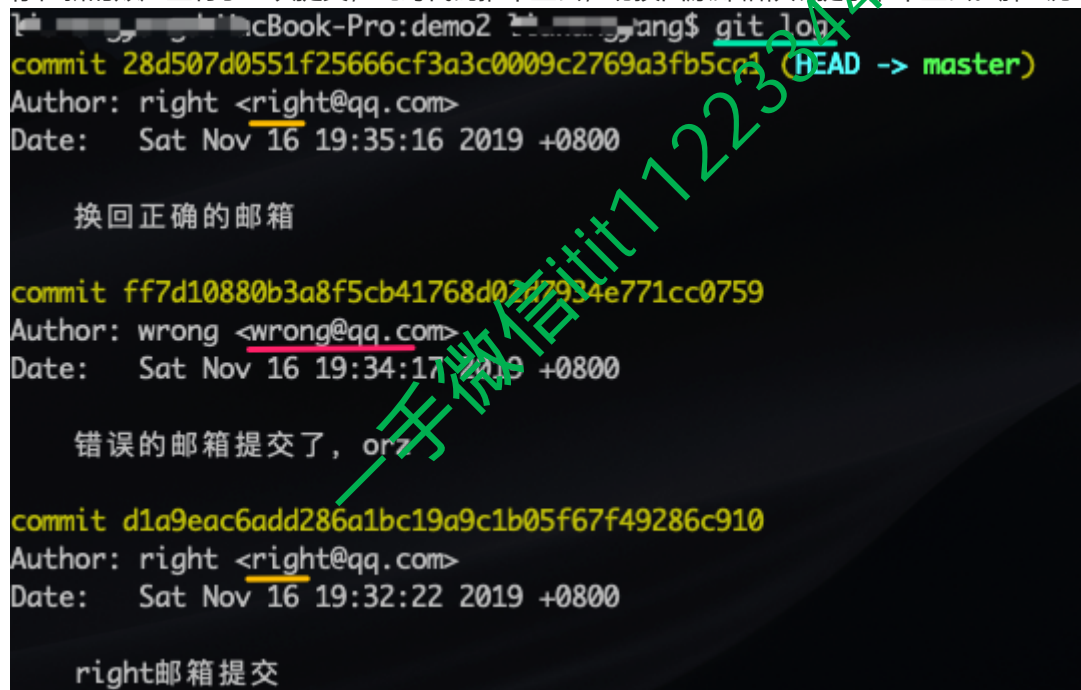
```
ssh: connect to host xxx.com port 22: Connection refused
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

显然要检查是否自己账户有权限或者自己 clone 的地址是否有误。

案例 2：切换邮箱提交错误

如下图所示，开发中可能还会遇到的一种错误是不小心切换了 git 账户，新的账户没有提交权限，然而你在新的账户上有了一次提交，此时代码推不上去，切换回原邮箱依然提示推不上去。肿么办...



The image shows a terminal window with the following content:

```
MacBook-Pro:demo2 wang$ git log
commit 28d507d0551f25666cf3a3c0009c2769a3fb5ca1 (HEAD -> master)
Author: right <right@qq.com>
Date: Sat Nov 16 19:35:16 2019 +0800

    换回正确的邮箱

commit ff7d10880b3a8f5cb41768d02d7934e771cc0759
Author: wrong <wrong@qq.com>
Date: Sat Nov 16 19:34:17 2019 +0800

    错误的邮箱提交了，orz

commit d1a9eac6add286a1bc19a9c1b05f67f49286c910
Author: right <right@qq.com>
Date: Sat Nov 16 19:32:22 2019 +0800

    right邮箱提交
```

A green diagonal watermark with the text '微信号: 172236417' is overlaid on the terminal output.

图：某位同学遇到的类似问题图示

遇到这种问题大家一定要冷静思考，更换账户后为啥还推不上去呢？

通过 `git log` 可以看到提交历史中包含了之前错误账户的提交，因此解决问题的思路就是使用新的账户撤销错误那次提交。

其中一个方法如下：

1. 重新设置回原来的账户；
2. 回退到更换邮箱之前的版本：`git reset --soft d1a9eac6`；
3. 重新提交然后推送即可。

```
ngyangdeMacBook-Pro:demo2 ~$ git reset --soft d1a9eac6
MacBook-Pro:demo2 ~$ git add .
MacBook-Pro:demo2 ~$ git ci -m "重新合并提交"
[master 88b00c4] 重新合并提交
1 file changed, 3 insertions(+)
MacBook-Pro:demo2 ~$ git log
commit 88b00c4ea478ee109d18a907aba15f7bacf28f18 (HEAD -> master)
Author: right <right@qq.com>
Date: Sat Nov 16 19:38:04 2019 +0800

重新合并提交

commit d1a9eac6add286a1bc19a9c1b05f67f49286c910
Author: right <right@qq.com>
Date: Sat Nov 16 19:32:22 2019 +0800

right邮箱提交
```

执行上述步骤后，我们发现提交记录里已经没有 wrong@qq.com 这个错误账户的提交记录，因此重新推送代码就不会再报错。

2.2.2 粗心类错误

案例 1：切分支错误

比如你同时开发多个项目，你本应该按照下面的步骤基于 master 拉出两个功能分支：

1. 基于 master 分支拉出了 feature-a 分支，来开发 A 功能；
2. 然后再从 master 分支拉出 feature-b 分支，来开发 B 功能。

但是由于粗心直接从 feature-a 拉出了 feature-b 分支。

最可怕的是，feature-b 的功能测试都通过且先上线。此时 feature-a 的代码就会被带到线上，极可能导致故障。

【建议】拉取新分支时一定要检查父分支。

案例 2：提前合并到 master

有些功能还未到发布时间，提前一晚合并到 master，由于晚上有其他团队成员紧急发布，导致自己的代码被带到了线上，然而线上相关的表 DDL 还没提交，其他配置还没配置好。

从而发生故障，无故躺枪，杯具...

案例 3: merge 代码冲突解决不当

如果两个人在同一个分支开发或者多个分支合并到主分支容易出现冲突，如果冲突解决不当可能造成故障。

比如冲突时没有了解清楚就删除了别人的代码，导致上线后出现问题。

【建议】合并代码遇到冲突时，如果没有把握，联系和你冲突的人员一起确认。

案例 4：一个分支干了多件事

有些人喜欢偷偷搞点小动作，某个功能专用的分支做了点其他事情，比如顺手优化了某个接口，却不告知测试人员。

导致测试没有覆盖到所有修改的场景，上线以后可能出现问题。

【建议】一个分支只干一件事，如果顺带做了点其它改动一定要告知测试人员。

2.2.3 其他建议

【建议】开发周期较长时，及时合并 master 分支解决冲突，避免最后阶段大量冲突，同时避免自己依赖的接口修改、删除和废弃等情况。

【建议】开发过程中要多将自己的开发分支和 master 代码进行比对，自我 Code Review，有助于及早发现问题。

2.3 数据库相关

2.3.1 案例分析

数据库本身的学习可以参考其他教程，本小节重点讲述实际开发中数据库相关的坑。

案例 1：没有唯一键约束

《手册》中有涉及唯一键的描述：

【强制】业务上具有唯一特性的字段，即使是多个字段的组合，也必须建成唯一索引。

很多人会认为只要先查后插就能保证唯一，实则不然。

在高并发场景下，如果没有同步操作，两个事务同时开启，先后查询到没有数据，然后依次执行插入会出现重复数据。

是不是非高并发场景就没问题呢？结果也是否定的。

一方面别人写代码如果没有先查后插入而是直接插入，无法保证唯一性；另外一方面不是所有的数据都是通过 Java 代码插入到数据库的，如果通过数据平台写入时看到没有唯一键约束就没考虑数据重复的问题，极可能会写入重复的数据。

因此业务上具有唯一性的字段，即使是多个字段组合，也必须创建唯一键。

案例 2：执行 delete 等危险操作没带查询条件

有些朋友手抖，执行 delete 语句时不带条件或者删除条件有误，将可能造成事故。

这里给出一条避坑建议：

【建议】执行删除操作时，要开启事务，同时要先查询核对影响的行数和数据准确性，再执行删除操作；执行删除操作后再次核实，如果情况不对立即回滚。

案例 3：新数据或者功能没考虑对老数据的影响

比如在没下旧接口的情况下要新增接口，而新接口要修改数据库表结构新增一个非 null 字段。

此时如果没考虑老接口，上线以后老接口插入数据时就会报错。

案例 4：时效性要求极高的场景查了备库

有些接口对时效性要求较高，比如支付后还要通过另外一个接口查询支付结果，如果查询支付结果的接口走的是备库，此时恰有一个 DDL 导致主备延时较高，很容易查询出不符合预期的结果。

然而支付的结果影响较大，短暂的主备延迟可能会造成意想不到的后果。

因此给出以下建议：

【建议】大表的 DDL 不要在高峰期执行。

【建议】对于实时性要求极高的接口，请直接查询主库。

案例 5：MyBatis 映射语句问题

一个比较典型的错误案例如下：

```
select * from xxx
where
<if test="a != null">
  a =
</if>
<if test="b != null">
  and b =
</if>
```

如果 a 和 b 同时为 null，则生成的语句将会是: `select * from xxx where` 从而出现语法错误。

此种情况应该使用 标签来代替 where 关键字。

另外一个典型的错误:

```
select * from students
<where>
<if test="a != null">
  a =
</if>
</where>
limit
```

此时，如果 offset 或者 limit 传入负数，就会报错

```
ERROR when execute SQL: SELECT * FROM students limit -1,10
SyntaxError: Parse error on line 1
...FROM students limit -1,10
-----^
Expecting 'NUMBER', got 'MINUS'
```

因此，对于分页的参数一定要做好参数校验。

案例 6：悬停时间较长的事务被 kill

一个朋友在开发中遇到这样一种编码逻辑：查询出所有数据，然后循环根据每个数据的特征查询其他接口作为参数计算出值，然后插入到数据库，循环结束后提交事务。

之前数据量小时都没问题，数据量大以后该函数执行报错。

进行了各种分析，各种调试，各种尝试都没解决。

最终咨询 DBA 才得知根本原因是事务悬停超过 xxx 秒就会被 kill。

可以计算完成将结果每几百个一批批量插入到数据库；可以通过公司的大数据处理平台实现这个功能。

案例 7：表新增了供查询的字段，却没建索引，导致慢查询

建表时很多人会想着创建索引加快查询速度。但是由于新增某个功能加新字段时，很容易忘记加索引。如果数据量较大，并发量较大时，容易导致慢查询。

【建议】新增字段如果作为查询条件时，要思考是否要加索引。

2.3.2 选读

《手册》中有这样一个规定：【强制】页面搜索严禁左模糊或者全模糊，如果需要请走搜索引擎来解决。

给出的说明是：索引文件具有 B-Tree 的最左前缀匹配特征，如果左侧的值未确定，那么无法使用此索引。

因此很多朋友可能会这么写：

```
select * from xxx where name like CONCAT(
```

我们要思考一个问题：**这样写真的能避免左侧模糊查询吗？**

如果参数 name 的值为：“%”或者“% 某字符”呢？

MySQL 中结果就等价于：

```
select * from xxx where name like '%%'
```

或者

```
select * from xxx where name like '%某字符%'
```

依然不会走到索引。

因此对于 like 的参数，大家可以进行参数检查，也可以进行转义处理。

本节结合实际的开发经验讲述了 Git 和数据库中常见的坑点，并给出了一些建议。

希望大家在学习 Git 和数据库相关知识时，要注重思考，多动手实践去验证。

下一节将介绍其它各种坑点，给出对大家实际的学习开发有帮助的建议。

}