

1 复制

1.1 分布式原理

1.2 分布式架构

1.2.1 主从

1.2.2 主备

1.2.3 复合

1.3 docker 搭建读写分离

1.3.1 配置目录文件搭建

1.3.2 启动主从

1.4 sqlalchemy 实现读写分离

1.4.1 自定义 sqlalchemy

1.4.2 集成到项目

2. 分片

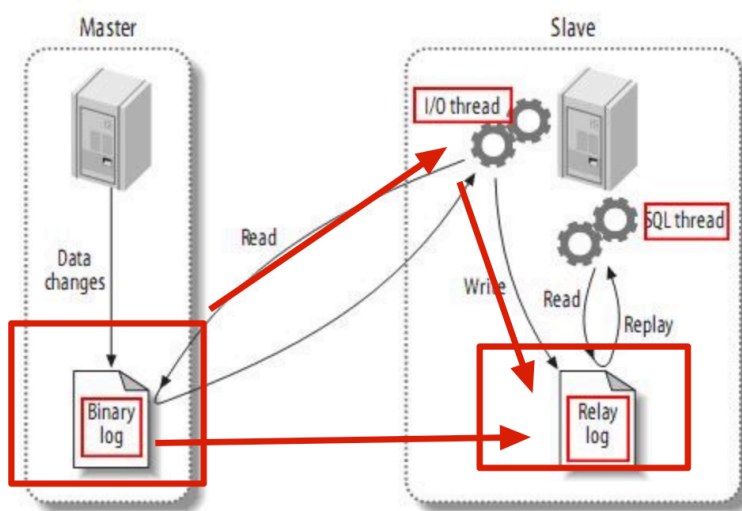
2.1 分片介绍

2.2 垂直拆分

1 复制

1.1 分布式原理

2. 原理



- Mysql的复制 是一个异步的复制过程

二进制日志:

- 存放 SQL 语句

User1: insert

User2: update

User3: delete

users 可以并发执行 sql 语句

slave 节点只能串行执行 sql 语句，回放速度很慢

- 存储数据变化

```
update goods set count = goods.count -1;
```

good 1: count:100 ---> count:99

Good2: count: 1000 ---> 999

Good3: count: 10000---> 9999

Slave 可以并行回放

可能会导致日志非常大

1.2 分布式架构

1.2.1 主从

主从架构

- 性能
 - 一主多从, 读写分离, 提高吞吐量
 - 可用性
 - 主库单点, 一旦挂了, 无法写入
 - 从库高可用
- 写服务性能没有提升, 高可用也没有提升
读都提升

1.2.2 主备

主备架构

- 性能
 - 单库读写, 性能一般
- 可用性
 - 高可用, 一旦主库挂了, 就启用备库
- 这种方案被阿里云、美团等企业广泛使用



1.2.3 复合

高可用复合架构

- 性能
 - 读写分离, 提高吞吐量
- 可用性
 - 高可用, 一旦主库挂了, 就启用备库

写入性能没有提升

读取性能提升

1.3 docker 搭建读写分离

```
10 server-id=1
11 ; 二进制日志
12 log-bin=/var/run/mysqld/mysql-bin
```

master

```
9 , 从库的 id
10 server-id=2
11 ; 重放日志
12 relay_log=/var/run/mysqld/mysql-relay-bin
13 !includedir /etc/mysql/conf.d/
```

从节点

1.3.1 配置目录文件搭建

```
1 $ mkdir mysql-master-slave
2 $ cd mysql-master-slave
3 $ mkdir -p mysql/master/mysql
4 $ mkdir -p mysql/slave/mysql
5 $ touch mysql/master/mysql/my.cnf
6 $ touch mysql/slave/mysql/my.cnf
7 $ touch docker-compose.yml
8 $ mkdir -p volume/mysql/master
9 $ mkdir -p volume/slave/slave
10 $ mkdir -p volume/extra
```

mkdir -p volume/mysql/slave

目录结构如下

1.3.2 启动主从

1. `docker network create cusnet` 创建单独的网路
2. `docker-compose up` 创建容器直接启动
 1. `docker-compose up -d` 后台启动容器

```
File | mysql-bin.000003
Position | 761
```

it: <https://github.com/abdul/mycli>

.l: <https://groups.google.com/forum/#!forum/mycli-users>

ie: <http://mycli.net>

inks to the contributor - Daniel West

```
mysql root@localhost:(none)> CREATE USER 'slave'@'%' IDENTIFIED BY '123456';
```

ry OK, 0 rows affected

ie: 0.028s

```
mysql root@localhost:(none)> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'slave'@'%'
-> ve'@%';
```

ry OK, 0 rows affected

ie: 0.008s

```
mysql root@localhost:(none)> flush privileges;
```

ry OK, 0 rows affected

ie: 0.011s

```
mysql root@localhost:(none)> show master status \G;
```

ow in set

ie: 0.013s

```
mysql root@localhost:(none)> show master status
```

ow in set

ie: 0.012s

```
mysql root@localhost:(none)> show master status \G;
```

ow in set

ie: 0.003s

```
mysql root@localhost:(none)> █
```

me: <http://mycli.net>

hanks to the contributor - Jonathan Bruno

```
mysql root@localhost:(none)> change master to MASTER_HOST='flask-mysql-master',MASTER_U
-> SER='slave',MASTER_PASSWORD='123456',MASTER_LOG_FILE='mysql-bin.000003',MASTER_LOG
-> l-bin.000003",MASTER_LOG_POS=761;
```

uery OK, 0 rows affected

ime: 0.040s

```
mysql root@localhost:(none)> start slave;
```

uery OK, 0 rows affected

ime: 0.006s

```
mysql root@localhost:(none)> show slave status \G;
```

row in set

ime: 0.013s

```
mysql root@localhost:(none)> █
```

```
show slave status \G;
```

```
Relay Master Log File | mysql-bin.000003
Slave_IO_Running      | Yes
Slave_SQL_Running     | Yes
Replicate_Do_DB       |
Replicate_Ignore_DB   |
```

Docker 安装: <https://docs.docker.com/compose/install/>

1.4 sqlalchemy 实现读写分离

目标：读数据库 --连接从库，写数据-->连接主库

python 连接数据库

```
1. 导入: from mysql import Connection
2. conn_master = Connection(host=主库,port=主库,user,password,database)
3. conn_slave = Connection(host=从库,port=从库,user,password,database)

4. 执行写入数据:
5 master_cursor = conn_master.cursor()
6 sql = "update xx set yyy='123123'" insert delete
7 mater_cursor.execute(sql)

8 执行读取数据
9 slave_cursor = conn_slave.curosr()
10 sql = "select * from tb_user"
11 slave_cursor.execute(sql)
```

1.4.1 自定义 sqlalchemy

```
7
8 app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews'
9 # 设置多个数据库的URI (用于数据操作)
10 app.config['SQLALCHEMY_BINDS'] = {
11     "master": 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews',
12     "slave": 'mysql+pymysql://root:123456@127.0.0.1:33307/topnews'
13 }
14
15
```

```

class MySession(SignallingSession):
    def get_bind(self, mapper=None, clause=None):
        """Return the engine or connection for a given model or
        table, using the ``__bind_key__`` if it is set.
        """
        # mapper is None if someone tries to just get a connection
        if mapper is not None:
            try:
                # SA >= 1.3
                persist_selectable = mapper.persist_selectable
            except AttributeError:
                # SA < 1.3
                persist_selectable = mapper.mapped_table

            info = getattr(persist_selectable, 'info', {})
            bind_key = info.get('bind_key')
            if bind_key is not None:
                state = get_state(self.app)
                return state.db.get_engine(self.app, bind=bind_key)

            state = get_state(self.app)
            if self._flushing:
                # 写入数据
                print('-----写入数据-----')
                return state.db.get_engine(self.app, bind='master')
            else:
                # 读取数据
                print('=====读取数据=====')
                return state.db.get_engine(self.app, bind='slave')

```

```

58
59 class MySQLAlchemy(SQLAlchemy):
60
61     def create_session(self, options):
62         """Create the session factory used by :meth:`create_scoped_session`..."""
63
64         return orm.sessionmaker(class_=MySession, db=self, **options)
65
66
67 # 4. db = 自定义 SQLAlchemy 类对象
68 db = MySQLAlchemy(app)
69
70
71

```

```
@app.route("/index")
```

```
def index():
```

```

    print("=====创建用户=====")
    user = User(username="zhangsan", age=100)
    db.session.add(user)
    db.session.commit()

```

访问的是 master

```

    print('=====读取用户信息=====')
    users = db.session.query(User).all()
    for user in users:
        print(user.username)

```

访问的是 slave

```
return 'index'
```

1.4.2 集成到项目

```
10 # mysql配置
11 SQLAlchemy_DATABASE_URI = 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews' # 连接地址
12 SQLAlchemy_TRACK_MODIFICATIONS = True
13 # 主从数据库配置
14 SQLAlchemy_BINDS = {
15     "master": 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews',
16     "slave": 'mysql+pymysql://root:123456@127.0.0.1:33307/topnews'
17 }
18 # redis配置
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

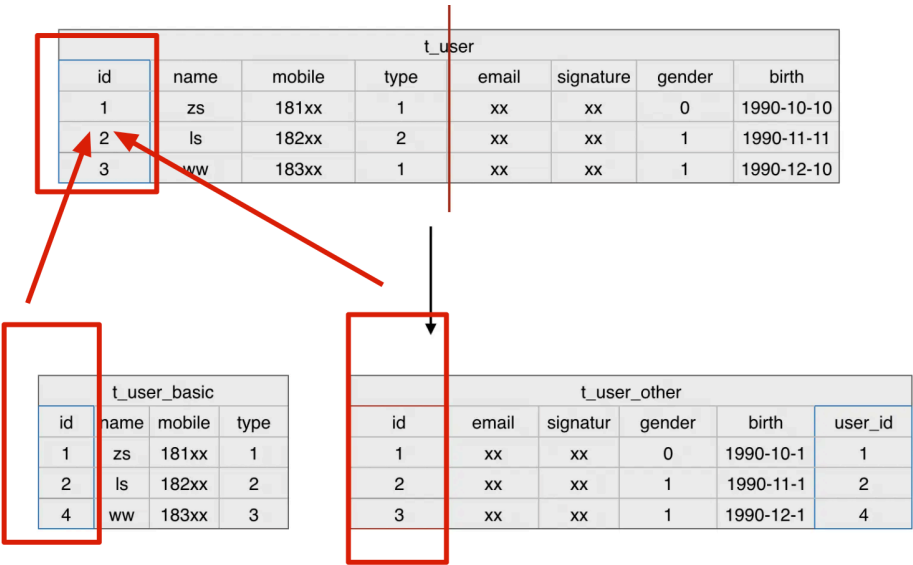
2. 分片

2.1 分片介绍

2.2 垂直拆分

拆分规则

- 相关性
 - 可以将字段根据 业务逻辑 和 使用的相关性 进行分表划分
 - 如: 用户名和密码经常配合使用, 将其分到用户认证表, 生日和邮箱等个人信息经常一起访问, 将其分到用户信息表
- 使用频率
 - 可以将字段根据 常用 和 不常用 进行划分, 并进行分表处理
 - 如: 原始用户表中包含了多个字段, 其中有常用的昵称、手机号等字段, 也包含不常用的邮箱、生日等字段, 可以根据使用频率将其分为两张表: 用户基础信息表 和 用户其他信息表



- 用户数据垂直分表 `user_basic` & `user_profile`
- 文章数据垂直分表 `article_basic` & `article_content` (文章内容较长且只在详情页才需要)

垂直分库

- 将一个数据库中的多张表拆分到多个数据库(服务器节点)中
- 注意点:
 - 由于 本地事务不支持跨库操作, 所以应该将 有相关联性的表放在同一个库中
 - 如: 如果后续头条项目垂直分库, 将用户相关的放在数据库1, 文章相关的放在数据库2

```

1  # 默认
2  数据库  t_user  t_article      多个库之间支持 join 查询
3
4  # 垂直分表
5  数据库  t_user_basic  t_user_detail  t_article_basic  t_article_detail
6
7  # 垂直分库
8  数据库1  t_user_basic      t_user_detail
9  数据库2  t_article_detail  t_article_basic

```

```

10
11  app = Flask(__name__)
12  # 设置多个数据库地址 (用于数据操作)
13  app.config['SQLALCHEMY_BINDS'] = {
14      'db1': 'mysql+pymysql://root:123456@127.0.0.1:33306/db1',
15      'db2': 'mysql+pymysql://root:123456@127.0.0.1:33306/db2'
16  }
17  db = SQLAlchemy(app)
18
19
20  class User(db.Model):
21      __tablename__ = 'tb_user'
22      __bind_key__ = 'db1'
23      id = Column(Integer, primary_key=True)
24      username = Column(String(32))
25      age = Column(Integer)
26
27
28  class Address(db.Model):
29      __tablename__ = 'tb_adr'
30      bind key = 'db2'
31      id = Column(Integer, primary_key=True)
32      detail = Column(String(32))
33      user_id = Column(Integer)
34

```



```

35
36 @app.route('/index')
37 def index():
38     # 创建用户
39     user = User(username='zhangsan', age=100)
40     db.session.add(user)
41     db.session.flush() # 同步到数据库, 能够获取 user.id
42
43     # 创建用户地址
44     adr1 = Address(detail='sh', user_id=user.id)
45     adr2 = Address(detail='bj', user_id=user.id)
46     db.session.add_all([adr1, adr2])
47
48     db.session.commit() # 看似调用一次, 在内部还是不同库分别 commit, 因为不同库, 事务是独立的
49
50     return 'index'
51
52
53 @app.route('/read_data')
54 def read_data():
55     # 先读取 user
56
57     # 因为不同数据库, 所以不能直接 join 查询
58
59     # 再读取用户地址
60
61     addresses = Address.query.filter(Address.user_id == user.id).all()
62     for adr in addresses:
63         print(adr.user_id, '====', adr.detail)
64
65     return 'read data'
66
67
68 if __name__ == "__main__":
69     db.drop_all()

```

写入 db1, 在独立事务中

写入 db2, 在独立的事务中

读取 db1

只能在单库中执行 join 查询

读取 db2