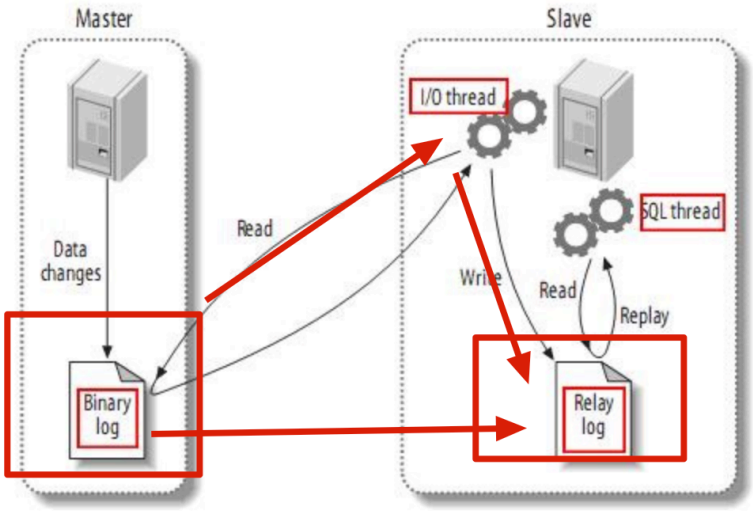
1 复制

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

```
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

1.3.2 启动主从

1. `docker network create cusnet` 创键单独的网路
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://gitter.im/dbcli/mycli
.l: https://groups.google.com/forum/#!forum/mycli-users
ie: http://mycli.net
anks to the contributor - Daniel West
sql root@localhost:(none)> CREATE USER 'slave'@'%' IDENTIFIED BY '123456';
ery OK, 0 rows affected
ie: 0.028s
sql root@localhost:(none)> GRANT REPLICATION SLAVE, REPLICATION CLIENT ON *.* TO 'sla
                        -> ve'@'%';
ery OK, 0 rows affected
ie: 0.008s
sql root@localhost:(none)> flush privileges;
ery OK, 0 rows affected
ie: 0.011s
sql root@localhost:(none)> show master status \G;
ow in set
ie: 0.013s
sql root@localhost:(none)> show master status
ow in set
ie: 0.012s
sql root@localhost:(none)> show master status \G;
ow in set
ie: 0.003s
sql root@localhost:(none)>
```

```
hanks to the contributor - Jonathan Bruno
ysql root@localhost:(none)> change master to MASTER_HOST='flask-mysql-master',MASTER_U
                         -> SER='slave',MASTER_PASSWORD='123456',MASTER_LOG_FILE="mysq
                         -> l-bin.000003",MASTER_LOG_POS=761;
uery OK, 0 rows affected
ime: 0.040s
ysql root@localhost:(none)> start slave;
uery OK, 0 rows affected
ime: 0.006s
ysql root@localhost:(none)> show slave status \G;
 row in set
ime: 0.013s
ysql 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 连接数据库库
1. 导入：from mysql imoprt 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
```

```python
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')
```

```python
class MYSQLAlchemy(SQLAlchemy):
    def create_session(self, options):
        """Create the session factory used by :meth:`create_scoped_session`...."""

        return orm.sessionmaker(class_=MySession, db=self, **options)


# 4. db = 自定义 SQLAlchemy 类对象

db = MYSQLAlchemy(app)
```

```python
@app.route("/index")
def index():
    print("=======创建用户======")
    user = User(username="zhangsan", age=100)
    db.session.add(user)
    db.session.commit()

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

    return 'index'
```

访问的是 master

访问的是 slave

1.4.2 集成到项目

```
# mysql配置
SQLALCHEMY_DATABASE_URI = 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews'  # 连接地址
SQLALCHEMY_TRACK_MODIFICATIONS = True
# 主从数据库配置
SQLALCHEMY_BINDS = {
    "master": 'mysql+pymysql://root:123456@127.0.0.1:33306/topnews',
    "slave": 'mysql+pymysql://root:123456@127.0.0.1:33307/topnews'
}
```

```
from common.models.routing_sqlalchemy import MYSQLAlchemy

# db = SQLAlchemy()
db = MYSQLAlchemy()
```

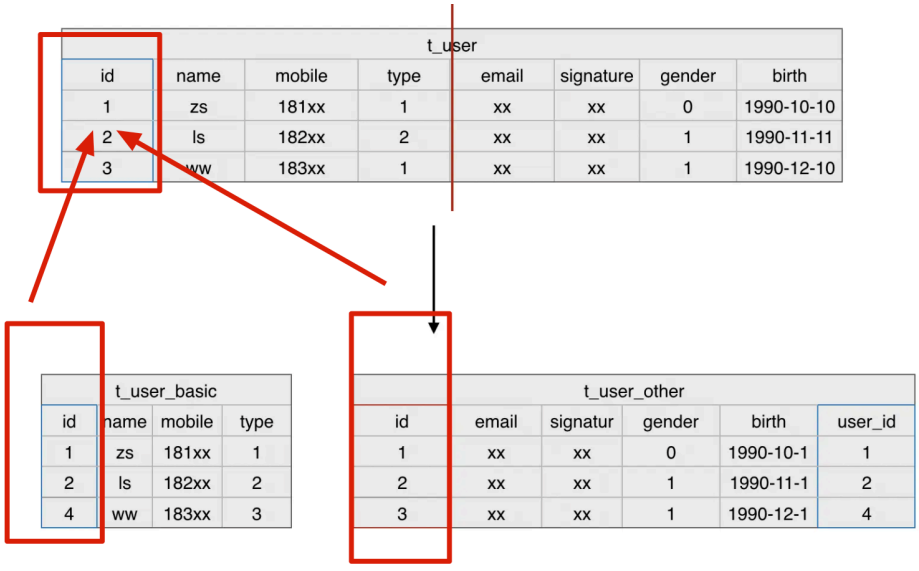
2. 分片

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



t_user

id	name	mobile	type	email	signature	gender	birth
1	zs	181xx	1	xx	xx	0	1990-10-10
2	ls	182xx	2	xx	xx	1	1990-11-11
3	ww	183xx	1	xx	xx	1	1990-12-10

t_user_basic

id	name	mobile	type
1	zs	181xx	1
2	ls	182xx	2
4	ww	183xx	3

t_user_other

id	email	signatur	gender	birth	user_id
1	xx	xx	0	1990-10-1	1
2	xx	xx	1	1990-11-1	2
3	xx	xx	1	1990-12-1	4

- 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

```
# 默认
数据库    t_user  t_article          多个库之间支持 jion 查询

# 垂直分表
数据库    t_user_basic    t_user_detail      t_article_basic    t_article_detail

# 垂直分库
数据库1   t_user_basic          t_user_detail
数据库2   t_article_detail      t_article_basic
```

```python

app = Flask(__name__)
# 设置多个数据库地址 （用于数据操作）
app.config['SQLALCHEMY_BINDS'] = {
    'db1': 'mysql+pymysql://root:123456@127.0.0.1:33306/db1',
    'db2': 'mysql+pymysql://root:123456@127.0.0.1:33306/db2'
}
db = SQLAlchemy(app)


class User(db.Model):
    __tablename__ = 'tb_user'
    __bind_key__ = 'db1'
    id = Column(Integer, primary_key=True)
    username = Column(String(32))
    age = Column(Integer)


class Address(db.Model):
    __tablename__ = 'tb_adr'
    __bind_key__ = 'db2'
    id = Column(Integer, primary_key=True)
    detail = Column(String(32))
    user_id = Column(Integer)
```

```python
@app.route('/index')
def index():
    # 创建用户
    user = User(username='zhangsan', age=100)
    db.session.add(user)
    db.session.flush()  # 同步到数据库，能够获取 user.id

    # 创建用户地址
    adr1 = Address(detail='sh', user_id=user.id)
    adr2 = Address(detail='bj', user_id=user.id)
    db.session.add_all([adr1, adr2])

    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

    return 'index'


@app.route('/read_data')
def read_data():
    # 先读取 user
```

写入 db1，在独立事务中

写入 db2，在独立的事务中

```python
@app.route('/read_data')
def read_data():
    # 先读取 user
    user = User.query.filter(User.username == 'zhangsan').first()

    # 因为不同数据库，所以不能直接 join 查询

    # 再读取用户地址
    addresses = Address.query.filter(Address.user_id == user.id).all()
    for adr in addresses:
        print(adr.user_id, '=====', adr.detail)

    return 'read data'


if __name__ == "__main__":
    db.drop_all()
```

读取 db1

只能在单库中执行 join 查询

读取 db2