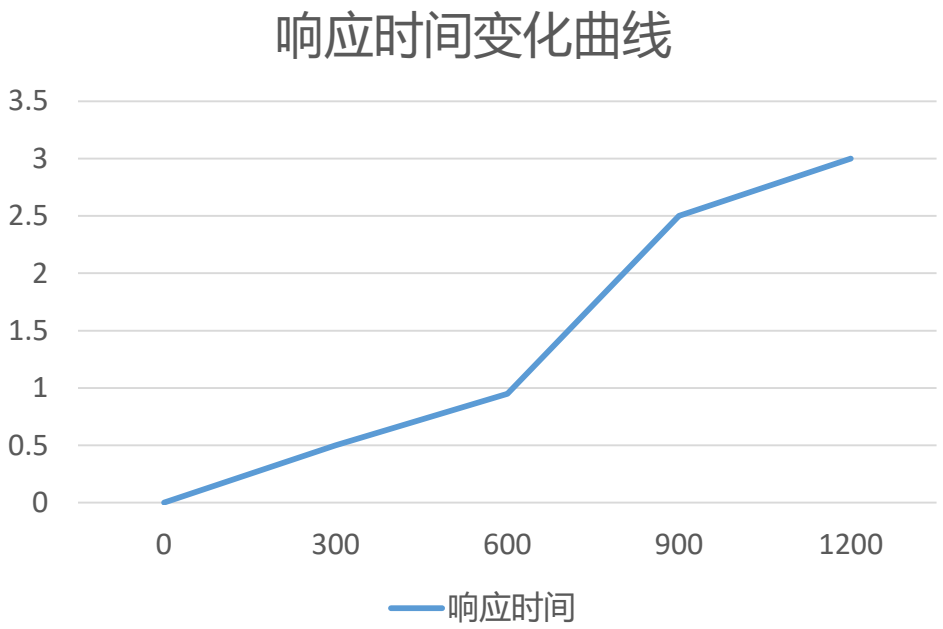
性能（1）

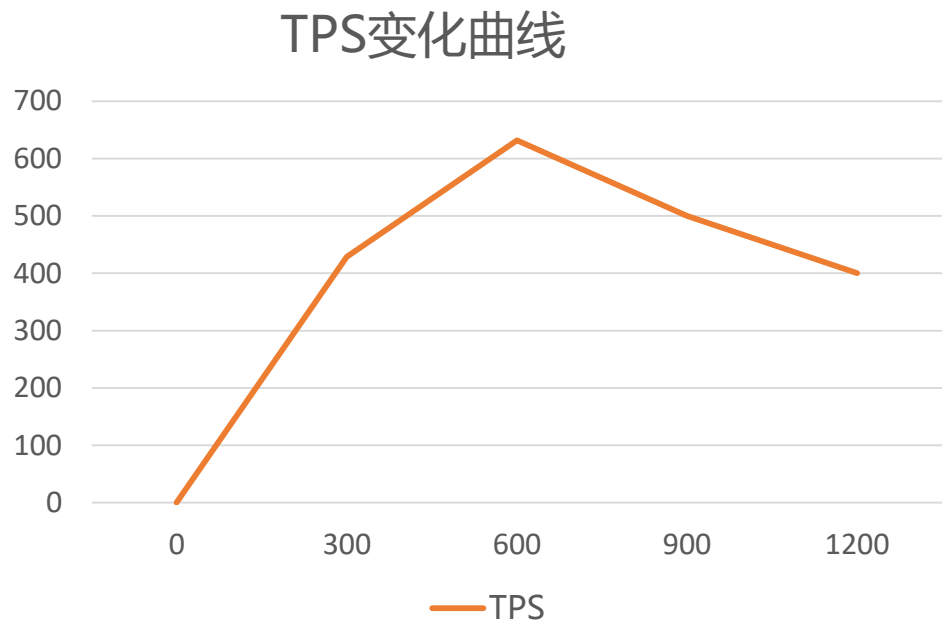
有效提升单个请求性能

taojie@zhucai

并发数： 系统同时处理的请求数
吞吐量（TPS/QPS）： 系统每秒处理完成的事务/请求的数量
响应时间（RT）：系统处理完一个请求的平均耗时

并发量 / 响应时间 = 吞吐量





响应时间

系统

并发请求

怎么更快的吃包子呢?

更多人一起吃?

缩短吃一个包子的时间。

处理单个请求多久合理?

RT < 1s

时间都去哪了？

```
long carwl = System.currentTimeMillis();
Document document = Jsoup.connect("https://www.baidu.com").get();
System.out.println(System.currentTimeMillis() - carwl);
long loop = System.currentTimeMillis();
for(int i=0;i<100000;i++) {
        String x = new String(document.text().getBytes());
}
System.out.println(System.currentTimeMillis() - loop);
```

1568
1536

对比一下

```
long carwl = System.currentTimeMillis();
Document document = Jsoup.connect("https://www.baidu.com").get();
System.out.println(System.currentTimeMillis() - carwl);
long loop = System.currentTimeMillis();
for(int i=0;i<100000;i++) {
        Calendar c = new GregorianCalendar();
        Exception e = new Exception();
}
System.out.println(System.currentTimeMillis() - loop);
```

1555
220

再对比一下

```
long carwl = System.currentTimeMillis();
Document document = Jsoup.connect("https://www.baidu.com").get();
System.out.println(System.currentTimeMillis() - carwl);
long loop = System.currentTimeMillis();
for(int i=0;i<100000;i++) {
        int p = i;
        p = p * i;
        p = p / 2;
}
System.out.println(System.currentTimeMillis() - loop);
```

1481
4

和外部资源访问比起来，
代码运行的性能消耗几乎可以忽略不计。

一个请求会用到最多的外部资源
——DB

DB操作为什么慢

- 数据库操作IO（磁盘、网络）居多
- 代码运算是CPU和内存操作居多

对一个请求的响应时间而言，
DB操作的时间和次数是决定性因素。

提升一个请求响应时间的最有效方法

- 减少DB操作次数
- 提升DB操作效率

减少DB操作

- 避免ORM思路
- 避免N+1查询

ORM操作

```
User user = userRepository.findOne(userId);
Long companyUserId=XaUtil.getCompanyUserIdByUser(user);

List<Role> roleList = roleRepository.findByRoleNameAndCompanyUserIdAndStatusNot(roleName,
companyUserId, XaConstant.Status.delete);

List<Department> deptList =
departmentRepository.findByCompanyUserIdAndLevelAndStatusNot(companyUserId, 1,
XaConstant.Status.delete);

List<User> list_u = userRepository.findByDeparmentIdAndRoleId(deptList.get(0).gettId(),
roleList.get(0).gettId());
```

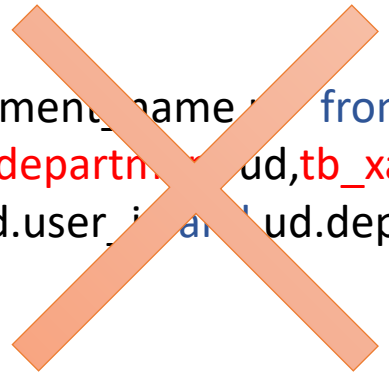
N+1操作

```
List<User> list_u = userRepository.findByDeparmentIdAndRoleId(deptList.get(0).gettId(),
roleList.get(0).gettId());

For {
        List<Department> deptList =
        departmentRepository.findByCompanyUserIdAndLevelAndStatusNot(companyUserId, 1,
        XaConstant.Status.delete);
}
```

消除N+1的方法：
1.连表查询

```
select d.department_name,u.* from tb_xa_user u
join tb_xa_user_department ud on u.id = ud.user_id
join tb_xa_department d on ud.department_id = d.id
```



```
select d.department_name ,u.* from tb_xa_user
u ,tb_xa_user_departm ud,tb_xa_department d
where u.id = ud.user_i and ud.department_id = d.id;
```

消除N+1的方法：
2.批量查询，程序组织数据

```
List<CompHolder> shList = compHolderDao.compPeriodicReport(compCode);
List<StockHolderNumber> csList = stockHolderNumberDao.compPeriodicReport(compCode);

ImmutableListMultimap<Date, CompHolder> shIndex = Multimaps.index(shList, new Function<CompHolder,
Date>() {
        @Override
        public Date apply(CompHolder input) {
                return input.getRptDate();
        }
});

Map<Date, StockHolderNumber> csIndex = Maps.uniqueIndex(csList, new Function<StockHolderNumber, Date>() {
        @Override
        public Date apply(StockHolderNumber input) {
                return input.getRptDate();
        }
});


SetView<Date> intersection = Sets.intersection(csIndex.keySet(), shIndex.keySet());
° ° ° ° ° °
for (Date key : keys) {
        ° ° ° ° ° °
}
```

消除N+1的方法：
3.局部缓存

```
Page<ContractList> page = contractListRepository.findAll(spec, pageable);
List<ContractList> contractLists = page.getContent();
for (ContractList obj : contractLists){

。。。。。。

    User createUsers =
userRepository.findBytIdAndStatusNot(contractListVo.getCreateUser(),XaConstant.Status.delete);

    contractListVo.setCreateUserName(createUsers.getRealName());

。。。。。。
}
```

```
Page<ContractList> page = contractListRepository.findAll(spec, pageable);
List<ContractList> contractLists = page.getContent();
//add by tao
//用于记录需要查询用户名的用户ID
Set<Long> userIdSet = Sets.newHashSet();
for (ContractList obj : contractLists){

。。。。。。

    //edit by tao
    //User createUsers = userRepository.findBytIdAndStatusNot(contractListVo.getCreateUser(), XaConstant.Status.delete);

    //contractListVo.setCreateUserName(createUsers.getRealName());
    userIdSet.add(contractListVo.getCreateUser());

。。。。。。
}
//add by tao
//循环内查用户，这里本地缓存用户数据，避免反复查询
if(!userIdSet.isEmpty()) {
      List<User> userList = userRepository.findUserById(Lists.newArrayList(userIdSet));
      Map<Long,User> userCache = Maps.uniqueIndex(userList, new Function<User,Long>() {
          @Override
          public Long apply(User user) {
             return user.gettId();
          }
      });
      for(ContractListVo contractListVo : contractList) {
          User u = userCache.get(contractListVo.getCreateUser());
          if(u != null) {
              contractListVo.setCreateUserName(u.getRealName());
          }
      }
}
```

提高DB操作效率

- SQL优化
- 索引

SQL优化

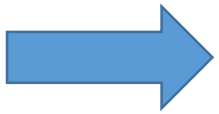
https://mp.weixin.qq.com/s/qjUv9cYPcpDRjmDKP55rJw

索引

- 索引的原理

索引 – 表中部分字段的有序投影

credit_period	credit_status	draw_amount	draw_rate	loan_date	loan_date_ccb	originator
184	REPAYMENTS	88653.60	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93580.56	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93355.74	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93529.48	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93288.00	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	28224.00	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93068.00	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93287.04	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	92491.52	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	93408.00	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
184	REPAYMENTS	84435.92	0.80	2018-11-07 00:00:00	2018-11-08 00:00:00	4
359	REPAYMENT	451244.40	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	818833.97	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	337456.04	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	423361.70	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	1128880.11	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	776436.32	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	WAIT_SUBMIT	(Null)	(Null)	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	884464.45	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	854993.76	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	1123902.90	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	1116416.02	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	689355.04	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	507373.81	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
359	REPAYMENT	243719.27	1.00	2019-05-03 00:00:00	2019-05-03 00:00:00	2
184	REPAYMENTS	1000000.00	1.00	2018-11-09 00:00:00	2018-11-09 00:00:00	2
184	REPAYMENTS	1000000.00	1.00	2018-11-09 00:00:00	2018-11-09 00:00:00	2



credit_period	credit_status
184	REPAYMENTS
184	REPAYMENTS
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	WAIT_SUBMIT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
359	REPAYMENT
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS
184	REPAYMENTS

Select 字段1	->	有索引覆盖则无需回表
from 表1		
where 字段2 = 某些值	->	有索引则无需全表扫描
order by 字段3	->	有索引则无需对结果进一步排序
Limit 100	->	索引已排序可直接取100返回

针对表1的理想索引

字段2，字段3，字段1

索引

- 索引的效果

当前慢日志阈值(long_query_time): 3

全部 | 2018/12/20 00:00:00 — 2018/12/21 13:33:00

执行语句	语句类型	发生时间	执行时间（s）	等待锁时间（s）	结果行数	扫描行数	所属数据库	帐号	IP地址
SELECT * FRO...	SELECT	2018/12/21 11:...	9.09463 s	0.00041 s	10	4054117	zhucai_db	dbuser	192.*.*.8
SELECT * FRO...	SELECT	2018/12/21 11:...	9.16881 s	0.00041 s	10	4054117	zhucai_db	dbuser	192.*.*.8
SELECT * FRO...	SELECT	2018/12/21 11:...	7.52845 s	0.00044 s	10	4054117	zhucai_db	dbuser	192.*.*.19
SELECT * FRO...	SELECT	2018/12/21 11:...	7.56863 s	0.00040 s	10	4054117	zhucai_db	dbuser	192.*.*.8
SELECT * FRO...	SELECT	2018/12/21 11:...	7.40668 s	0.00042 s	10	4054117	zhucai_db	dbuser	192.*.*.8
SELECT * FRO...	SELECT	2018/12/21 11:...	7.57058 s	0.00039 s	10	4054117	zhucai_db	dbuser	192.*.*.4
SELECT * FRO...	SELECT	2018/12/21 11:...	17.46823 s	0.00047 s	10	4054117	zhucai_db	dbuser	192.*.*.4
SELECT * FRO...	SELECT	2018/12/21 11:...	16.85520 s	0.00044 s	10	4054117	zhucai_db	dbuser	192.*.*.19
SELECT * FRO...	SELECT	2018/12/21 11:...	22.31650 s	0.00045 s	10	4054117	zhucai_db	dbuser	192.*.*.19
SELECT * FRO...	SELECT	2018/12/21 11:...	22.55609 s	0.00055 s	10	4054117	zhucai_db	dbuser	192.*.*.19

10 总条数: 35 < 1 2 3 4 >

索引优化方法

- EXPLAIN

```
1  EXPLAIN select d.department_name,u.* from tb_xa_user u
2  join tb_xa_user_department ud on u.id = ud.user_id
3  join tb_xa_department d on ud.department_id = d.id
4  where d.company_user_id = 414
```

信息 | 结果 1 | 剖析 | 状态

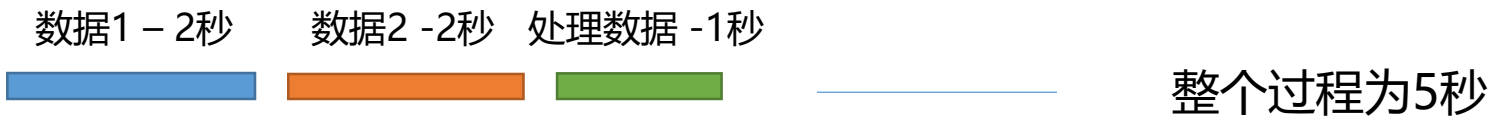
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	d	(Null)	ref	PRIMARY,tb_xa_departme	idx_dept_company_user_id,_leve	9	const	402	100.00	(Null)
1	SIMPLE	ud	(Null)	ref	user_department_departn	user_department_department_ic	9	zhucai_db.d.id	1	100.00	Using index condition
1	SIMPLE	u	(Null)	eq_ref	PRIMARY	PRIMARY	8	zhucai_db.ud.user_id	1	100.00	Using where

https://mp.weixin.qq.com/s/sFAPn9F9jvjhJv82C4H2JQ

DB优化之外最有效手段

- 异步

同步操作



整个过程为5秒

异步操作

整个过程为3秒

异步适用场景

- 主流程之外的附加操作（发短信、发邮件）
- 无需返回的操作（自动任务）
- 有一个上相对独立的长时间任务

异步实现的简单方法

- Runnable / Callable
- Spring @Async
- Guava AsyncEventBus
- JMS / AMQP

Spring @Async

```
public String doSomething() {
        LOGGER.info("doSomething");
        Thread.sleep(2000);
        return "奔跑";
}
```

```
@Async
public Future<String> who() {
        LOGGER.info("who");
        Thread.sleep(3000);
        return new AsyncResult<String>("小筑");
}
```

```java
long now = System.currentTimeMillis();
LOGGER.info("mainThread");
//开始一个异步任务
Future<String> future = asyncServiceImpl.who();
String doSomething = asyncServiceImpl.doSomething();
try {
    String who = future.get();
    System.out.println(who + "在" + doSomething);
} catch (InterruptedException e) {
    LOGGER.error("{}",e);
} catch (ExecutionException e) {
    LOGGER.error("{}",e);
}
System.out.println(System.currentTimeMillis() - now);
```

```
2018-12-18 18:19:37.777  INFO 1776 --- [           main] c.z.s.service.impl.AsyncServiceImpl      : mainThread
2018-12-18 18:19:37.793  INFO 1776 --- [           main] c.z.s.service.impl.AsyncServiceImpl      : doSomething
2018-12-18 18:19:37.793  INFO 1776 --- [         task-1] c.z.s.service.impl.AsyncServiceImpl      : who
小筑在奔跑
3018
```

使用Spring @Async 需要注意

- 使@Async 注解有效

```
//spring boot 项目
@SpringBootApplication
@EnableAsync
public class ServiceApplication {
```

```
<!—非spring boot 的spring项目-->
<task:annotation-driven/> ？ 未确认
```

- 主线程和子线程不能是同一个类

```
@Autowired
private AsyncServiceImpl asyncServiceImpl;

public void test() {
    //开始一个异步任务
    Future<String> future = asyncServiceImpl.who();
    String doSomething = asyncServiceImpl.doSomething();
    。。。。。。
}
```

- 多个自动任务的异步@Scheduled
  - Spring timer的CRON，是一个主线程，每个自动任务的执行会阻塞主线程，所以即使后启动的任务是async，也无法被唤醒。
  - 要让多个Cron任务同时运行需要把先启动的任务async，留出主线程来唤起后续启动的任务

异步操作最重要的事：子线程的异常处理

```
@Async
public Future<String> who() throws Exception {
        LOGGER.info("who");
        Thread.sleep(3000);
        return new AsyncResult<String>("小筑");
}
```

```
Future<String> future;
try {
        future = asyncServiceImpl.who();
} catch (Exception e1) {
        // 不会被执行
        e1.printStackTrace();
}
```

```
try {
    String who = future.get();
    System.out.println(who + "在" + doSomething);
} catch (InterruptedException e) {
    //子线程未结束的情况下中断的异常
    LOGGER.error("{}",e);
} catch (ExecutionException e) {
    //获取实际的异常消息
    Throwable rootCause = Throwables.getRootCause(e);
    System.out.println(rootCause.getMessage());
    //处理特定类型的异常
    if(Throwables.getRootCause(e) instanceof RuntimeException) {
        System.out.println(e.getCause().getMessage());
    }
    //抛出特定类型的异常
    Throwables.propagateIfInstanceOf(Throwables.getRootCause(e), RuntimeException.class);
    //抛出子线程的实际异常
    throw Throwables.propagate(Throwables.getRootCause(e));
}
```

异步方法中未捕捉的异常处理

AsyncUncaughtExceptionHandler

- 减少SQL操作
- 优化SQL性能，SQL编写要为索引留出空间
- 创建合适的索引
- 适当引入异步操作

性能（2）

缓存和请求前置