#### 

#### 一、应用服务修改，TSF

1. 修改Maven setting文件

* <profiles>  
   <profile>  
   <id>nexus</id>  
   <repositories>  
   <repository>  
   <id>central</id>  
   <url>http://repo1.maven.org/maven2</url>  
   <releases>  
   <enabled>true</enabled>  
   </releases>  
   <snapshots>  
   <enabled>true</enabled>  
   </snapshots>  
   </repository>  
   </repositories>  
   <pluginRepositories>  
   <pluginRepository>  
   <id>central</id>  
   <url>http://repo1.maven.org/maven2</url>  
   <releases>  
   <enabled>true</enabled>  
   </releases>  
   <snapshots>  
   <enabled>true</enabled>  
   </snapshots>  
   </pluginRepository>  
   </pluginRepositories>  
   </profile>  
   <profile>  
   <id>qcloud-repo</id>  
   <repositories>  
   <repository>  
   <id>qcloud-central</id>  
   <name>qcloud mirror central</name>  
   <url>http://mirrors.cloud.tencent.com/nexus/repository/maven-public/</url>  
   <snapshots>  
   <enabled>true</enabled>  
   </snapshots>  
   <releases>  
   <enabled>true</enabled>  
   </releases>  
   </repository>  
   </repositories>  
   <pluginRepositories>  
   <pluginRepository>  
   <id>qcloud-plugin-central</id>  
   <url>http://mirrors.cloud.tencent.com/nexus/repository/maven-public/</url>  
   <snapshots>  
   <enabled>true</enabled>  
   </snapshots>  
   <releases>  
   <enabled>true</enabled>  
   </releases>  
   </pluginRepository>  
   </pluginRepositories>  
   </profile>  
   </profiles>  
   <activeProfiles>  
   <activeProfile>nexus</activeProfile>  
   <activeProfile>qcloud-repo</activeProfile>  
   </activeProfiles>

1. 修改parent pom.xml

* <parent>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-dependencies</artifactId>  
   <version>1.10.0-RELEASE</version>  
  </parent>

1. 注释parent pom.xml 中 dependencyManagement

* <dependencyManagement>  
   <dependencies>  
   <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-dependencies</artifactId>  
   <version>Dalston.SR2</version>  
   <type>pom</type>  
   <scope>import</scope>  
   </dependency>  
   </dependencies>  
  </dependencyManagement>

1. 修改对应项目的pom.xml配置，这里以ecooperation为例，注释掉zipkin，如下：

* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-zipkin</artifactId>  
  </dependency>

1. 修改pom.xml配置，删除注册中eureka 的依赖，如下：

* <dependency>  
   <groupId>org.springframework.cloud</groupId>  
   <artifactId>spring-cloud-starter-eureka</artifactId>  
  </dependency>

1. 增加腾讯云的依赖

* <!-- tsf调用链 -->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-sleuth</artifactId>  
  </dependency>  
  <!-- tsf鉴权 -->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-auth</artifactId>  
  </dependency>  
  <!-- tsf路由 -->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-route</artifactId>  
  </dependency>  
  <!-- tsf API注册-->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-swagger</artifactId>  
  </dependency>  
  <!-- tsf限流 -->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-ratelimit</artifactId>  
  </dependency>  
  <!-- tsf加解密 -->  
  <dependency>  
   <groupId>com.tencent.tsf</groupId>  
   <artifactId>spring-cloud-tsf-encrypt</artifactId>  
  </dependency>

1. 修改applicaiton 配置文件，修改内容如下：

* # 删除 eureka 注册中心配置，以下为举例，如果相关更多，请确保删除  
  eureka.client.healthcheck.enabled=true  
  eureka.client.serviceUrl.defaultZone=http://localhost:9011/eureka/  
    
  # 新增 配置项  
  tsf.swagger.enabled=false  
  logging.file=/Users/lishangmin/data/${spring.application.name}/root.log  
  logging.file.level.root=INFO  
    
  # 删除 sleuth 链路追踪配置，以下为举例，如果相关更多，请确保删除  
  spring.sleuth.sampler.percentage=0.1  
  spring.sleuth.sampler.percentage=0.  
    
  # 删除 zipkin 链路追踪配置项，以下为举例，如果相关更多，请确保删除  
  spring.zipkin.base-url=http://localhost:9015  
  spring.zipkin.enabled=true

1. 在主方法增加注解，示例如下，鉴权、路由、限流、链路追踪：

* @SpringBootApplication  
  @ComponentScan(basePackageClasses = {  
   ApiServerApplication.class,  
   LogUtilConfig.class,  
   RedisUtilConfig.class,  
   HbaseUtilConfig.class,  
   PublicToolsConfig.class,  
   NodeCheckConfitg.class})  
  @EnableDiscoveryClient//开启自动注册  
  @EnableConfigurationProperties  
  @EnableFeignClients  
    
  @EnableTsfAuth  
  @EnableTsfRateLimit  
  @EnableTsfRoute  
    
  public class ApiServerApplication extends SpringBootServletInitializer {  
    
   /\*\*  
   \* 方法名称: restTemplate  
   \* 描述: 配置内置负载  
   \* 参数:  
   \* 返回值: org.springframework.web.client.RestTemplate  
   \*/  
   @Bean  
   @LoadBalanced  
   RestTemplate restTemplate() {  
   return new RestTemplate();  
   }  
    
   public static void main(String[] args) {  
   SpringApplication.run(ApiServerApplication.class, args);  
   }  
    
  }

1. 启动脚本增加VM参数

* -Dtsf\_consul\_ip=127.0.0.1 -Dtsf\_consul\_port=8500 -Dtsf\_application\_id=a -Dtsf\_group\_id=b

1. 安装consul，下载地址如下：

* https://www.consul.io/downloads.html

1. 启动注册中心

* # Linux / Unix  
  ./consul agent -dev &  
    
  # Windows  
  consul agent -dev &  
    
  ./consul agent -server -bootstrap-expect=1 -data-dir=/tmp/consul -node=consul -bind=192.168.8.108 -ui -client=0.0.0.0

#### 二、Redis 集群迁移 CKV

1. 增加配置文件属性

* redis.cache.node=62.234.134.35:6379  
  redis.cache.pwd=crs-r51bc20c:ckv@123CKV

1. 修改 RedisClusterProperties 增加属性

* /\*\*  
   \* 单机节点  
  \*/  
  private String node;

1. 修改 JedisClusterConfig

* // 新增  
  @Bean  
  public JedisPool initJedisPool(){  
   JedisPoolConfig jedisPoolConfig = getJedisPool();  
   String[] ipPortPair = redisProperties.getNode().split(":");  
   JedisPool jedisPool = new JedisPool(jedisPoolConfig,ipPortPair[0].trim(),Integer.valueOf(ipPortPair[1].trim()),redisProperties.getConnectionTimeout(),redisProperties.getPwd());  
   return jedisPool;  
  }  
    
  // 注释  
  @Bean  
  public JedisCluster getJedisCluster() {  
   String[] serverArray = redisProperties.getClusterNodes().split(",");//获取服务器数组(这里要相信自己的输入，所以没有考虑空指针问题)  
   Set<HostAndPort> nodes = new HashSet<HostAndPort>();  
    
   for (String ipPort : serverArray) {  
   String[] ipPortPair = ipPort.split(":");  
   nodes.add(new HostAndPort(ipPortPair[0].trim(), Integer.valueOf(ipPortPair[1].trim())));  
   }  
    
   JedisPoolConfig jedisPoolConfig = getJedisPool();  
    
   return new JedisCluster(nodes,redisProperties.getConnectionTimeout(),redisProperties.getSoTimeout(),redisProperties.getMaxAttempts(), redisProperties.getPwd(),jedisPoolConfig);  
  }

1. 修改 RedisJpaTemplate

* // 修改前  
  private JedisCluster jedisCluster;  
    
  // 修改后  
  private JedisPool jedisPool;  
    
  // 新增方法  
  public void returnResource(final Jedis jedis) {  
   if (jedis != null) {  
   jedisPool.returnResourceObject(jedis);  
   }  
  }  
    
  // 使用方式修改为  
  public List<String> hmget(String prefix, String key, String... fileds){  
   Jedis jedis = jedisPool.getResource();  
   try {  
   return jedis.hmget(ROOT\_KET + KEY\_SPLIT + prefix + KEY\_SPLIT + key, fileds);  
   }finally {  
   returnResource(jedis);  
   }  
  }

#### 三、CMQ 迁移

1. 添加CMQ SDK依赖，下载地址如下：

* http://cmqsdk-10016717.cossh.myqcloud.com/qc\_cmq\_java\_sdk\_V1.0.4.zip?\_ga=1.238761863.623572950.1544435534

1. 消费者修改

* 采用轮循拉取消费，示例代码已上传到SVN

1. 生产者新修改

* Queue queue = cmqAccount.getQueue(propertiesInit.getPolicyDumpMq()); //通道name  
  String message = null;  
  try {  
   message = queue.sendMessage(JsonUtil.bean2json(paramMap)); //将mq报文放入通道  
  } catch (Exception e) {  
   e.printStackTrace();  
  }  
  logger.info("policyDump推送mqid：:"+message);

#### 四、Hbase迁移

1. 修改HBase地址

* hbase.zookeeper.quorum=10.0.3.14  
  hbase.zookeeper.port=2181

#### 五、服务联调

1. 从原来Dubbo RPC调用方式修改为 Restful

* 采用Feign的调用方式，将原有Dubbo依赖去除

#### 六、数据库迁移（ecooperation（接口服务）、policyserver（承保服务）、policydumpserver（保单转储服务）三个服务一致）

1. pom.xml添加Tbase驱动

<dependency>  
 <groupId>org.postgresql</groupId>  
 <artifactId>postgresql</artifactId>  
 <version>42.2.5</version>  
 </dependency>

1. 修改application.properties配置文件中数据源相关配置

spring.tbase.datasource.url=jdbc:postgresql://154.8.190.95:11000/postgres? useSSL=false&useUnicode=true&characterEncoding=UTF-8  
 spring.tbase.datasource.username=picc  
 spring.tbase.datasource.password=picc@123  
 spring.tbase.datasource.driverClassName=org.postgresql.Driver  
 spring.tbase.datasource.initialSize=1  
 spring.tbase.datasource.minIdle=1  
 spring.tbase.datasource.maxActive=20  
 spring.tbase.datasource.maxWait=60000  
 spring.tbase.datasource.timeBetweenEvictionRunsMillis=60000  
 spring.tbase.datasource.minEvictableIdleTimeMillis=300000

1. 数据库表结构修改

* 分表分库规则修改，删除relationShip表，policyBasic、policyBusiness、policyInsured、policyComInfo、policyDump、policycheck表id字段，policyBasic、policyBusiness、policyInsured、policyComInfo、policyDump表以proposalno（投保单号做分片），policycheck表分片规则不变，以identifynumber（证件号码）做分片，新增businessno\_mapper表，做保单号和投保单号映射。数据结构文档见svn，POC-Tencent-Tbase数据结构。

1. mybatis相关映射修改（与数据库表结构一致）

* mybatis使用Tbase数据库，json字段无mybatis相关映射类型，改为taxt。

1. 业务代码修改，修改查询规则，根据投保单号查询，封装relationShip表改为封装businessno\_mapper表。

#### 七、请求周边系统相关修改

1. elife系统高保额校验、黑名单校验、出单后高保额推数做挡板开发，调用方法直接返回成功标志位。
2. 微服务项目组单号微服务、算费微服务，dubbo调用转restful，采用FeignClient。

/\*\*  
 \* 算费微服务  
 \*/  
 @FeignClient(value = "${micro.service.calculate}")  
 public interface CalculateService {  
   
 @RequestMapping(value = "/calculate/calculate", consumes = {MediaType.APPLICATION\_JSON\_VALUE})  
 CalcResponseRoot calculate(RequestRootNormal message);

/\*\* \* 单号微服务 \*/ @FeignClient(value = "${micro.service.no}") public interface NumberService {   
 @RequestMapping(value = "/number/normalID", consumes = {MediaType.APPLICATION*JSON*VALUE}) ResponseMsg normal(ReqMessage message);

}

```properties  
 #####算费微服务、单号微服务注册中心serverName  
 micro.service.no=provider-number  
 micro.service.calculate=provider-calculate  
 ```

需要将单号微服务、算费微服务请求报文对象导入policyserver承保服务。

#### 八、 微服务功能点改造

1：熔断（Hystrix）

A） 引入spring-cloud-starter-feign依赖包（腾讯TSF依赖包中包含此依赖包）；

B） 启动类新增注解@EnableHystrix ；

C） 通过Feign使用Hystrix时，接口类配置，直接在@FeignClient注解中添加fallback属性，属性值为触发熔断时调用的自定义类；

D） 启动Hystrix熔断：properties配置文件新增feign.hystrix.enabled=true ；

E）Hystrix参数配置见 Hystrix参数详解.md

接口

@FeignClient(value = "${micro.service.no}", url = "http://192.168.8.100:18082", fallback = NumberServiceHystrix.class)  
public interface NumberService {  
  
 @PostMapping(value = "/number/internetID", consumes = {MediaType.APPLICATION\_JSON\_VALUE})  
 ResponseMsg internet(ReqMessage message);  
  
 @RequestMapping(value = "/number/normalID", consumes = {MediaType.APPLICATION\_JSON\_VALUE})  
 ResponseMsg normal(ReqMessage message);  
  
 @PostMapping(value = "/number/normalIDs", consumes = {MediaType.APPLICATION\_JSON\_VALUE})  
 List<ResponseMsg> allNormal(List<ReqMessage> messages);  
  
}

fallback类

@Component  
public class NumberServiceHystrix implements NumberService {  
  
 @Override  
 public ResponseMsg internet(ReqMessage message) {  
 throw new PolicyServerException("调用互联网单号服务熔断");  
 }  
  
 @Override  
 public ResponseMsg normal(ReqMessage message) {  
 throw new PolicyServerException("调用单号服务熔断");  
 }  
  
 @Override  
 public List<ResponseMsg> allNormal(List<ReqMessage> messages) {  
 throw new PolicyServerException("集合调用单号服务熔断");  
 }  
  
}

2：微服务添加标签（限流、鉴权、uuid调用链）

//将uuid作为标签  
TsfContext.putTag("uuid",uuid);

3：微服务容错，springcloud原生ribbon

properties配置文件中添加ribbon相关的配置

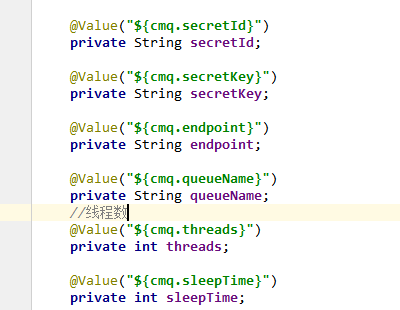
#请求处理超时时间  
ribbon.ReadTimeout: 5000  
#请求连接超时时间  
ribbon.ConnectTimeout: 2000  
#同一实例最大重试次数，不包括首次调用  
ribbon.MaxAutoRetries: 0   
#重试其他实例的最大重试次数，不包括首次所选的server  
ribbon.MaxAutoRetriesNextServer: 1  
#是否对所有操作请求都进行重试  
ribbon.OkToRetryOnAllOperations: false

#### 九、性能调优修改

1：删除api-server微服务（接口服务），将相关业务逻辑迁移至policy-server、api-server服务除报文解析和简单校验（业务校验）之外，并没有做其他操作，而api-server与policy-server服务交互时，request/reponse对于json报文的操作占用了大量系统资源，取消api-server服务减少了四次json报文操作， 节省了大量系统资源的开支。

2：CMQ消费者采用多线程并发消费

多线程较单线程而言，mq的消费效率以成倍方式提升，在系统数据高峰的情况下，一定时间内mq消息队列数据的堆积量很有可能超过mq消息队列可支持的最高数据，多线程消费，线程数以可配置方式设定，能够有效地预防数据高峰时消息队列的数据堆积问题。







3：分布式锁优化

解决相同uuid 资源争抢问题

4：幂等流程优化

原有业务流程：

查询redis，如果redis没有，redis 有或者没有 都会查询 hbase，再去Hbase查询 ， 如果hbase有返回幂等报文承保完成，保存redis。保存hbase，使用redis 并没有意义；

修改后：

Hbase查询 ， 如果hbase有返回幂等报文，保存hbase 。

5：删除补数之后对数据的校验

以policyinfo节点为例，修改前：



修改后：



补数的数据，是由redis缓存中读取，非必要操作影响系统响应时间，代码结构不合理。

6：修改redis存储key和取值时的key不一致问题 部分redis存储是的key值和取值时的key值不一致，导致每次取不到redis中的数据而频繁查询数据库。

7：系统代码结构调整（异常处理、方法调用等）

该保存到redis的没有保存到redis 删除UUID递，增然后保存记录请求次数的业务逻辑。 请求报文保存到日志文件，不需要每次请求都存储到Hbase，增加业务流程复杂

8：系统组件兼容调试

A） Restful调用优化

Ecooperation API 模块中，只有业务逻辑处理，只是做了一次转发，增加序列化和反序列化。 将该部分代码合并到policy-server； 单号模块，增加批量调用接口，保单号 和 投保单号 集中调用，而不是分两次调用，减少连接次数。

b ）SpringHbase组件

Spring Hbase 官方组件，每次连接数据库都会关闭连接，每次连接都会创建新的链接，消耗系统连接资源； 重写 HbaseTemplate组件，采用连接池，减少系统端口的消耗。

C）SpringAync组件优化

增加连接池，控制并发最大数据量。

D） Spring Cacheable组件优化

在300TPS左右，Cacheable能够带来提升，因为使用的是本地内存，减少了系统对redis外部交互的次数。但Cacheable采用的是动态代理

实现，并发上去之后，1.5.10 版本 的SpringBoot Tomcat Loadclass 采用的方法级别 同步锁，会造成线程堵塞，2.0.0版本 SpringBoot 对该问

题作了修复，但是框架升级到2.0.0成本较大，所以弃用cacheable。

E） SpringEnableTomcat系统参数调整

根据也测结果，调整线程数、最大连接数、等待队列大小等参数。

F）数据源连接池改造，druid改为hikaricp

9：腾讯云优化

A） 网络调整

因为POC测试基于公有云，很多都是采 用外网调用方式，后期修改为内网调用。

B） 数据库优化

数据库分区、dn、cn优化、db全局表调整等。

C） CMQ优化

采用ThreadLocal，提高连接的复用次数。

10：试错优化

A） Tomcat

Tomcat NIO 调用 方式修改为 APR ，有一些提升，但是可能会增加运维成本，所以放弃。

B） Feign

Http方式从URLConnection修改为Apache HttpClient没产生实际的提升。

11：监控使用

A） 增加数据库连接池监控

监控慢查询，连接消耗等等，根据反馈，对数据的节点，分表方案，SQL效率作出相应的修改。

B）VisualVM

监控JVM的使用情况，如线程、内存、热点方法，找出性能的瓶颈点，根据反馈，作出相应修改。