论文检测全文报告

基本信息

报告编号：20200226302574911E32E72D0

文档名称：718101530006 初稿
提交方式：上传文档检测
提交时间：2020年02月26日
正文字符数：19507
正文字数：8296
检测范围：大雅全文库

总体结论

文献相似度：17.33%
去除参考文献相似度：17.33%
去除本人已发表论文相似度：17.33%
重复字符数：3381
文献原创度：82.67%
单篇最大相似度：2.22%
单篇最大重复数：434
最相似文献题名：用JAVA API函数实现数据压缩与解压缩

相似片段分布

最密集相似段：1
密集相似段：9
非密集相似段：20
前部相似段：10
中部相似段：9
尾部相似段：11

典型相似文献

相似图书

序号	题名	作者	出处	相似度
1	犯罪预防学	李春雷;靳高风	北京：中国人民大学出版社，2016.08	2.22%
2	Java核心编程技术	张屹;蔡木生	大连：大连理工大学出版社，2015.07	2.22%
3	Java开发实战经典 第2版	李兴华	北京：清华大学出版社，2018.01	2.22%
4	计算机程序设计实践教程 C#语言	李利明;刘卫国	北京：清华大学出版社，2018.01	2.13%
5	Java程序设计案例教程	许敏;史荧中	北京：机械工业出版社，2018.07	1.65%
6	Java面向对象程序设计基础教程	冯洪海	北京：清华大学出版社，2011.06	1.65%
7	Java程序开发实用教程	邱加永	北京：清华大学出版社，2014.04	1.65%
8	JAVA面向对象程序设计 第4版	张桂珠	北京：北京邮电大学出版社，2015.01	1.65%
9	Java程序设计基础	栾颖	北京：清华大学出版社，2014.12	1.65%
10	Java程序设计 第2版	朱庆生	北京：清华大学出版社，2017.01	1.61%
11	Java解惑 英文	布洛克（Bloch J.）;加福特（Gafter N.）	北京：人民邮电出版社，2006.07	1.49%
12	Java编程及应用	杨武;刘贞	北京：高等教育出版社，2004.02	1.28%
13	轻量级J2EE企业应用实战 Struts+Spring+Hibernate整合开发	李刚	北京：电子工业出版社，2007.04	1.28%
14	Java完全自学手册	林树泽	北京：机械工业出版社，2009.01	1.28%

北京大雅图书信息技术有限公司 相似度检测专用章

15	Struts 2 权威指南 基于WebWork核心的MVC开发	李刚	北京：电子工业出版社，2007.09	1.28%
16	JAVA程序设计教程 第2版	迟丽华	北京：清华大学出版社，2008.03	1.28%
17	Java Web应用详解	张丽	北京：北京邮电大学出版社，2015.01	1.28%
18	Java程序设计项目化教程	郑哲	北京：机械工业出版社，2015.02	1.28%
19	Java程序设计教程 第3版	雍俊海	北京：清华大学出版社，2014.03	1.28%
20	Java程序设计培训教程	孙燕	北京：清华大学出版社，2002.06	1.28%
21	JSP网络开发入门与实践	吴建;张旭东	北京：人民邮电出版社，2006.11	1.28%
22	轻量级Java EE企业应用实战 Struts 2+Spring 4+Hibernate整合开发	李刚	北京：电子工业出版社，2014.10	1.28%
23	整合Struts+Hibernate+Spring应用开发详解	李刚	北京：清华大学出版社，2007.11	1.28%
24	Java程序设计教程与项目实训	温秀梅;司亚超	北京：清华大学出版社，2017.08	1.28%
25	21世纪电脑学校 Java程序设计实用教程	张大治	北京：清华大学出版社，2008.04	1.28%
26	Java语言程序设计教程	刘政怡	合肥：安徽大学出版社，2016.11	1.28%
27	Java Web应用开发实用教程	龚永罡;陈秀新	北京：机械工业出版社，2010.05	0.94%
28	Java语言程序设计	周少琦;李铁兵;董东	北京：北京大学出版社，2007.08	0.94%
29	高等学校计算机网络技术课程系列教材 WEB编程技术	丁跃潮;余元辉;叶文来	北京：高等教育出版社，2015.10	0.94%
30	JAVA语言基础与实训	许文宪	济南：山东科学技术出版社，2009.08	0.94%
31	Java程序设计基础与上机指导	赵文靖	北京：清华大学出版社，2006.06	0.94%
32	实战Java高并发程序设计	葛一鸣;郭超	北京：电子工业出版社，2015.10	0.91%
33	Core JavaJava应用程序编程案例	刘甲耀;郑小川;严桂兰	武汉：武汉大学出版社，2009.02	0.86%
34	微服务实战 Dubbox +Spring Boot+Docker	肖睿	北京：人民邮电出版社，2018.07	0.83%
35	Java高并发编程详解 多线程与架构设计	汪文君	北京：机械工业出版社，2018.06	0.74%
36	Java多线程设计模式	结城浩;博硕文化	北京：中国铁道出版社，2005.04	0.67%
37	EFFECTIVE JAVA 第2版 英文版	JOSHUA BLOCH	北京：电子工业出版社，2016.04	0.59%
38	NIO与Socket编程技术指南	高洪岩	北京：机械工业出版社，2018.07	0.58%
39	易学——设计模式	郭志学	北京：人民邮电出版社，2009.04	0.58%
40	基于工作任务的Java程序设计实验教程	冯君;宋锋;谭业武;刘春霞	北京：清华大学出版社，2015	0.58%
41	JAVASE程序设计	李瑞生;何珍祥	北京：中国铁道出版社，2016.12	0.55%
42	C#程序设计案例教程	郭树岩;刘一臻	北京：北京理工大学出版社，2016.12	0.55%
43	21天学编程系列 21天学通Java	郭现杰	北京：电子工业出版社，2016.01	0.55%
44	面向对象与Java程序设计	朱福喜	北京：清华大学出版社，2009.03	0.55%
45	Java语法及网络应用设计	徐迎晓	北京：清华大学出版社，2002.09	0.55%
46	Java应用与开发案例教程	徐谡;徐立;孙计安	北京：清华大学出版社，2005.07	0.55%
47	Java程序设计	贾素玲;王强;陈当阳;曹爽;许珂	北京：清华大学出版社，2007	0.55%
48	21天学通Java 第5版	郭现杰	北京：电子工业出版社，2018.03	0.55%

相似期刊

序号	题名	作者	出处	相似度
1	用JAVA API函数实现数据压缩与解压缩	杨静;孙劲光;何晓军	辽宁工程技术大学学报·自然科学版，2004，第4期	2.22%
2	无线传感器网络应用卫星通信传输大数据的研究与设计	卢涛;缪瑛;高骋飞;崔建刚	电子科学技术，2016，第4期	0.53%

北京大雅图和信息技术有限公司 相似度检测专用章

3	基于Java的多线程快速排序设计与优化	黄志波;赵晴;孙少乙	微型机与应用，2016，第16期	0.41%
4	Java并发模型框架的构建及应用	姜军平;刘伟	西北医学教育，2006，第3期	0.41%
5	Session Facade在分布式系统中的应用研究	石亮	现代电子技术，2008，第10期	0.31%
6	基于MA的分布式入侵检测系统模型的设计	何勇	信阳农业高等专科学校学报，2013，第2期	0.17%
7	贵冶智能工厂建设之应用系统技术架构浅析	徐娟华	铜业工程，2019，第1期	0.16%
8	基于ARM-Linux与JavaEE网络化门禁考勤系统的设计	房好帅;李志鹏;郑哲豪;王春景	电脑知识与技术，2014，第13期	0.11%
9	一种基于微服务的应用框架	张晶;黄小锋	计算机系统应用，2016，第9期	0.1%
10	浅谈视频监控中的数据保存	关伟基	中国安防，2007，第5期	0.09%

相似网络文档

序号	题名	作者	相似度
1	免费毕业论文致谢词		0.79%
2	毕业论文致谢词		0.79%
3	毕业论文致谢——多种版本,毕业论文的致谢词		0.79%
4	毕业论文致谢词精选 一		0.79%
5	教你如何写毕业论文感谢词(精心整理)		0.79%
6	从徐渭的泼墨写意中体验艺术创作的情态自由	胡琏	0.79%
7	挖掘产品信息的变异表达		0.79%
8	毕业论文致谢辞9篇		0.79%
9	2011毕业论文致谢词模版		0.7%
10	机械毕业设计		0.7%
11	2011届毕业论文致谢词模版(免力推荐)		0.7%
12	企业知识型员工的激励问题wps		0.62%
13	基于钯催化C-H活化多氟取代苯与杂环化合物的偶联	周兴健	0.56%
14	Sun Certified Java Programmer SCJP 15		0.55%
15	中国教会女子大学研究	宋爱妮	0.43%

全文对比

上海交通大学本科毕业论文

分布式环境下唯一ID分发系统的设计与实现

学 生：李真

学 号：7181015300006

专 业：计算机科学与技术

导 师：赵玉峰

学校代码：10248

上海交通大学继续教育学院

二〇二〇 年 二月二十五日

大雅 相似度分析 dsa.dayainfo.com

毕业论文声明

本人郑重声明：

1、此毕业论文是本人在指导教师指导下独立进行研究取得的成果。除了特别加以标注和致谢的地方外，本文不包含其他人或其它机构已经发表或撰写过的研究成果。对本文研究做出重要贡献的个人与集体均已在文中作了明确标明。本人完全意识到本声明的法律结果由本人承担。

2、本人完全了解学校、学院有关保留、使用学位论文的规定，同意学校与学院保留并向国家有关部门或机构送交此论文的复印件和电子版，允许此文被查阅和借阅。本人授权上海交通大学继续教育学院可以将此文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本文。

3、若在上海交通大学继续教育学院毕业论文审查小组复审中，发现本文有抄袭，一切后果均由本人承担（包括接受毕业论文成绩不及格、缴纳毕业论文重新写作费、重新写作毕业论文、不能按时获得毕业证书等），与毕业论文指导老师无关。

作者签名： 李真 日期：2020年2月25日

分布式环境下唯一ID分发系统的设计与实现

摘要

随着近些年国内互联网行业的发展，互联网相关服务也越来越多的深入到各行各业，同时也对互联网服务在实用性，便利性等方面提出了越来越高的要求，也变相的推动了互联网技术的发展与进步.“高并发，高可用，高性能”更是越来越成为未来互联网服务提供者能否提供更优质服务无法规避的重要因素。

在这样的互联网行业形势下，“万物互联”已经离我们越来越近，作为“万物”唯一标识的ID 也越来越重要， ID生成服务也在这些年互联网行业大发展的前提下不断升级，不断优化。

系统主要使用当前比较主流的技术如SpringBoot框架， Dubbo分布式中间件， Mysql自定义函数， Redis服务及命令， 雪花算法等对几种比较基础的ID生成服务进行实现.并对几种实现方式进行测试，分析几种实现方式的性能，优缺点，以便于后续可以根据实际情况应用于生产环境中。

关键词：ID生成系统，SpringBoot，Dubbo，Mysql，Redis，雪花算法，分布式

DESIGN AND IMPLEMENTATION OF UNIQUE ID DISTRIBUTION SYSTEM IN DISTRIBUTED ENVIRONMENT

ABSTRACT

With the development of domestic Internet industry in recent years, more and more Internet related services have penetrated into all walks of life. At the same time, it has put forward higher and higher requirements for Internet services in terms of practicability and convenience, and also promoted the development and progress of Internet technology in a disguised way, High performance "is more and more an important factor that Internet service providers can not avoid in the future

In this situation of Internet industry, "Internet of everything" has become more and more close to us, and the ID as the only identification of "everything" is becoming more and more important. The ID generation service has been upgraded and optimized under the premise of the great development of Internet industry in recent years

The system mainly uses the current mainstream technologies such as spring boot framework, Dubbo distributed middleware, MySQL user-defined function, redis service and command, snowflake algorithm to realize several basic ID generation services, and tests several implementation methods, analyzes the performance, advantages and disadvantages of several implementation methods, so as to facilitate the subsequent application in the production environment according to the actual situation

Key words: ID generation system, springboot, Dubbo, mysql, redis, snowflake algorithm, distributed

北京大雅图书信息技术有限公司 相似度检测专用章

目 录

北京大雅因和信息技术有限公司 相似度检测专用章

第一章 绪论

引言

随着近些年国内互联网行业的发展，互联网相关服务也越来越多的深入到各行各业，同时也对互联网服务在实用性，便利性等方面提出了越来越高的要求，也变相的推动了互联网技术的发展与进步."高并发，高可用，高性能"更是越来越成为未来互联网服务提供者能否提供更优质服务无法规避的重要因素。

在这样的互联网行业形势下，"万物互联"已经离我们越来越近，作为"万物"唯一标识的ID 也越来越重要， ID生成服务也在这些年互联网行业大发展的前提下不断升级，不断优化。

ID分发系统概述

系统主要使用当前比较主流的技术如SpringBoot框架， Dubbo分布式中间件， Mysql自定义函数， Redis服务及命令， 雪花算法等对几种比较基础的ID生成服务进行实现.并对几种实现方式进行测试，分析几种实现方式的性能，优缺点，以便于后续可以根据实际情况应用于生产环境中。

系统使用SpringBoot框架和Dubbo分布式中间件作为基础服务，向外部系统提供访问接口，分别使用Mysql自定义函数，Redis incr命令，雪花算法实现三种基本的ID分发系统。

使用Dubbo中间件的服务版本号区分三种服务:

Mysql自定义函数实现方式对应version 1.0。

Redis incr命令实现方式对应version 2.0。

雪花算法实现方式对应version 3.0。

为了保证系统在分布式环境下的可用性，系统将采用Zookeeper作为注册中心，提供服务注册，服务发现，服务管理功能。

系统开发及运行环境

开发使用语言: Java

系统开发环境: IntelliJ IDEA 2018， Maven 3， Dubbo 2.6， SpringBoot 2.0.0

系统运行环境: JDK 1.8， Tomcat 9， Mysql 5.7， Redis 4， Zookeeper 3.5

本章小结

本章主要介绍了唯一ID分发系统在当前互联网行业背景下的必要性，并且大致描述了后文将要介绍的三种ID分发系统基本实现思路，后文将会着重介绍的三种ID分发系统的具体实现。

第二章 系统总体设计

2.1 系统架构图

系统架构图大致描述了唯一ID系统在架构方面的一些基本想法:

Consumer 作为消费者将向Provider传递version 版本号，ID系统中的Provider通过version版本号采用不同的方式向Consumer提供ID分发服

北京大雅因和信息技术有限公司 相似度检测专用章

务。

从图上我们可以看到各个version分别对应的不同的ID分发服务实现方式。

2.2系统功能设计

系统功能时序图大致描述了实际生产环境下ID系统在整个业务流程中所扮演的角色和具体的操作模式。

系统在处理业务逻辑的时候不可避免的要使用到ID分发服务，这个时候只需要在consumer端向provider端传递必须的数据，provider在接收到数据之后将依据制定的规则给出唯一ID。

consumer端需要引入dubbo的consumer服务，并接入zookeeper系统将dubbo的consumer服务注册在zookeeper系统中。

provider端需要对对三种ID分发方式进行具体实现，并将provider服务注册在zookeeper系统中，以便于consumer服务发现provider服务并进行实际应用。

最终会形成基本的dubbo服务体系，遵循dubbo协议，遵循zookeeper服务发现规则。

2.3本章小结

本章通过《系统架构图》和《系统功能时序图》展示了ID系统的基本设计思路和生产环境中的应用方法，从上述图中，可以看到，实际的系统中，将会有两块内容分别为consumer和provider，业务系统通过consumer访问povider，由provider向consumer提供具体的服务。

第三章 功能实现

3.1基于Mysql的唯一ID分发功能实现

3.1.1概述

基于Mysql的唯一ID分发功能作为ID分发功能实现的一种方式，主要依赖于Mysql数据库本身提供的存储过程或自定义函数实现，这里使用自定义函数作为具体的实现方式。

需要注意的点:

使用自定义函数的方式实现，我们就需要保证自定义函数在数据读写过程中的原子性，从而保证每次都能拿到新的ID数据，这样就需要用到数据查询的行级锁，行级锁是依赖于InnoDb驱动而存在，所以在定义存储表的时候需要使用InnoDb作为表的驱动方式。

具体数据库自定义函数需要提供的内容:

入参和出参:

入参作为唯一ID的类型，区分不同业务场景，不同机房，不同主机等ID分发需求，实际Mysql数据类型定义为char。

出参作为唯一ID的返回值，实际Mysql数据类型定义为Bigint。

数据记录:

定义数据表作为ID的存储仓库。

数据递增

使用Mysql数据库语法对数据表ID进行递增操作。

数据响应

使用Mysql数据库语法对数据表ID进行读取。

3.1.2具体实现

北京大雅因和信息技术有限公司 相似度检测专用章

下面对Mysql获取唯一ID的功能进行具体实现

基本思路:

通过Dubbo provider整合Spring Jpa，并通过Jpa的entityManager访问自定义函数，经由函数递增处理后，返回递增后的结果。

数据库获取ID的自定义函数的具体实现:

```
delimiter $$

create function getSeq(seq_name varchar(32)) returns bigint

begin

declare cnt bigint;

declare val bigint;

declare rst bigint;

set cnt = 0;

set val = 0;

set rst = 0;

select count(1), ssn into cnt, val from tbl_increment where ssn_name = seq_name for update;

if cnt = 0 then

insert into tbl_increment(ssn_name, ssn) values(seq_name, 1);

else

if val = 9000000000000000000 then

update tbl_increment set ssn = 1 where ssn_name = seq_name;

else

update tbl_increment set ssn = (val + 1) where ssn_name = seq_name;

end if;

end if;

select ssn into rst from tbl_increment where ssn_name = seq_name for update;

return rst;

end;

delimiter $$
```

查询时使用for update添加行级锁,保证操作原子性

ID存储数据表

```
create table tbl_increment
```

北京大雅因和信息技术有限公司 相似度检测专用章

(

ssn_name char(4) not null,

ssn bigint default 0 not null

);

create index index_increment on tbl_increment (ssn_name);

函数以tbl_increment作为存储数据表，包含ssn_name， ssn两个字段，作为记录ID字段的ssn类型设定为bigint，函数中将递增步长定义为1。

在函数中使用select [column] into [column] from [table]语法获取最终ID

使用update [table] set [column] = [value] 语法设置ID递增，此处递增步数为1。

经过上面的操作得到一个名为"getSeq"的自定义函数，只需要将"getSeq"函数应用于实际的项目中即可。

下面我以Jpa作为ORM工具， hikari作为数据库连接池来简单的展示"getSeq"函数的使用。

新建一个maven项目，引入SpringBoot和Dubbo相关依赖包，添加数据库连接配置如下:

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql:localhost:3306database_schema?useUnicode=true&characterEncoding=utf-8&useSSL=false

username: xxx

password: xxx

hikari: #使用hikari连接池

connection-timeout: 3000

idle-timeout: 600000

max-lifetime: 1800000

maximum-pool-size: 10

pool-name: hikari

register-mbeans: false

jpa:

show-sql: true

hibernate:

ddl-auto: update

database-platform: org.hibernate.dialect.MySQL5InnoDBDialect #使用InnoDb数据库驱动

北京大雅因和信息技术有限公司 相似度检测专用章

open-in-view: true

经过上面的步骤，已经把Jpa相关的基础环境搭建好。

下面需要引入dubbo的配置，此处采用JavaConfig的方式进行配置

```
@Configuration
@DubboComponentScan(basePackages = {"indeed.dubbo.provider.service"})
public class DubboConfig {
**
* 应用名
*
* @return
*
@Bean
public ApplicationConfig applicationConfig() {
ApplicationConfig applicationConfig=new ApplicationConfig();
applicationConfig.setName("provider");
return applicationConfig;
}
**
*  < dubbo:provider timeout="10000"  >
*
* @return
*
@Bean
public ProviderConfig providerConfig() {
ProviderConfig providerConfig=new ProviderConfig();
providerConfig.setTimeout(10000);
return providerConfig;
}
**
* 地址配置  < dubbo:registry address="zookeeper:127.0.0.1:2181" >
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
*
* @return
*
@Bean
public RegistryConfig registryConfig() {
RegistryConfig registryConfig=new RegistryConfig();
registryConfig.setProtocol("zookeeper");
registryConfig.setAddress("192.168.43.172:2281");
return registryConfig;
}
**
* 协议配置，等同于 < dubbo:protocol name="dubbo" port="20880" >
*
* @return ProtocolConfig
*
@Bean
public ProtocolConfig protocolConfig() {
ProtocolConfig protocolConfig=new ProtocolConfig();
protocolConfig.setName("dubbo");
protocolConfig.setPort(20880);
return protocolConfig;
}
}
```

添加ID系统基础服务接口和实现类:

接口:

```
public interface GeneratedService {
**
* 获取唯一ID
* @param salt 盐
* @return
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
*

ResultDto < Long >  getId(String salt);

}
```

实现类:

```
@Service

@com.alibaba.dubbo.config.annotation.Service(version = "1.0", timeout = 3000, interfaceClass=GeneratedService.class, retries=0)

public class MysqlGeneratedService implements GeneratedService {

@Autowired

private EntityManager entityManager;

@Override

@Transactional(propagation=Propagation.REQUIRED, rollbackFor=Exception.class)

public ResultDto < Long >  getId(String salt) {

Query query = entityManager.createNativeQuery("select getSeq('"+salt+"')");

Object singleResult = query.getSingleResult();

if (null == singleResult) {

throw new RuntimeException("id get failure!");

}

Long id = Long.valueOf(singleResult.toString());

return new ResultDto < Long > ().setData(id);

}

}
```

此处Dubbo service version声明为1.0

上面的配置已经把dubbo服务引到项目中并实现了对"getSeq"方法的调用

下面配置启动类

```
@SpringBootApplication

public class AppStarter {

public static void main(String[] args) throws InterruptedException {

new EmbeddedZooKeeper(2281, false).start();

Thread.sleep(1000);

SpringApplication.run(AppStarter.class, args);
```

北京大雅因和信息技术有限公司 相似度检测专用章

}

}

SpringBootApplication注解开启SpringBoot自动装配功能。

EmbeddedZooKeeper以2281端口启动Zookeeper服务。

经过上述的操作已经成功的对基于Mysql的ID分发服务进行了实现。

3.1.3单体测试

基于上述Dubbo实现的provider服务，针对性的需要构建Dubbo consumer服务进行测试

新建一个maven项目，引入 Dubbo相关依赖包，新增Dubbo consumer配置

< dubbo:application name="demo-consumer" >

< dubbo:registry protocol="zookeeper" address="192.168.43.172:2281" >

< dubbo:reference id="generatedService" check="true" interface="indeed.dubbo.api.service.GeneratedService" version="1.0" >

consumer中的 reference配置version="1.0"要与provider中声明的version="1.0"保持一致。

新增consumer服务启动类

public class AppStarter {

public static void main(String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("springdubbo-demo-consumer.xml");

context.start();

GeneratedService demoService =context.getBean(GeneratedService.class);

ResultDto < Long > hello = demoService.getId("test");

System.out.println(hello.getData());

}

}

此时，只需要在consumer运行AppStarter的main方法即可以取到ID的返回值，并且在tbl_increment数据表中看到对应的ssn和ssn_name记录。

再次运行

可以看到启动provider之后，通过consumer可以正常获取到Mysql服务返回的ID，并且在多次运行之后，发现ID是递增的。

3.1.4小结

基于Mysql的唯一ID分发系统主要依赖于“getSsn”自定义函数，保证提供“getSsn”自定义函数的服务的高性能即保证了整个ID分发服务的高性能.这样一来，数据库系统的性能和数据库连接池的管理将会异常重要。

可能出现的问题:

数据库本身的性能瓶颈，数据库宕机导致服务不可用，数据库集群环境下的数据同步问题，频繁查询可能导致的数据库锁表。

北京大雅相似度检测专用章

3.2基于Redis的唯一ID分发功能实现

3.2.1概述

基于Redis的唯一ID分发功能的实现主要依赖于Redis提供的incr命令，

Redis的incr函数需要提供两个变量key和递增步数

需要注意的点:

因为Redis基于内存的特性，为了保证数据在Redis宕机的时候不丢失，需要开启Redis的数据备份功能，开启Redis的RDB或AOF数据持久化策略，保证数据完整性，并根据实际情况适当调整持久化策略规则.

3.2.2具体实现

基于Redis的ID分发系统采用SpringBoot的redisTemplate包装类来进行具体实现，在具体实现中，会通过Dubbo provider调用SpringBoot的redisTemplate，使用redisTemplate访问Redis client，最终通过Redis client调用Redis server的incr命令，以获取到Redis中递增的ID数据.

首先需要引入SpringBoot和Redis相关依赖包，

在redisTemplate的基础上开发相应的工具类，便于后面对Redis的具体操作

具体实现类如下

```
@Component
public class RedisUtil {
@Resource
private RedisTemplate < String, Object >  redisTemplate;
**
* 递增
* @param key 键
* @param delta 要增加几(大于0)
* @return
*
public long incr(String key, long delta) {
if (delta  <  0) {
throw new RuntimeException("递增因子必须大于0");
}
return redisTemplate.opsForValue().increment(key, delta);
}
}
```

此对象引入redisTemplate并新增incr方法调用redisTemplate.opsForValue()的increment最终访问Redis并调用Redis的incr命令，执行对key相

北京大雅因和信息技术有限公司 相似度检测专用章

关的value的递增操作

基于Redis的ID分发系统，Dubbo provider的构建与基于Mysql服务的provider建构方法类似，需要定义ID系统基础服务接口和实现类，

此处接口复用GeneratedService

实现类

```
@Service
@com.alibaba.dubbo.config.annotation.Service(version = "2.0", timeout = 3000, interfaceClass=GeneratedService.class, retries=0)
public class RedisGeneratedService implements GeneratedService {
@Autowired
private RedisUtil redisUtil;
@Override
public ResultDto < Long > getId(String salt) {
return new ResultDto < Long > ().setData(redisUtil.incr("INCR", 1L));
}
}
```

此处将ID的key定义为INCR，递增步长定义为1，Dubbo service version声明为2.0

RedisGeneratedService实现类直接调用自定义的Redis服务类RedisUtil，使用RedisUtil的incr的返回值作为ID服务的返回值，通过Dubbo的Service注解在Zookeeper中注册以便于在provider 中访问， 并使用SpringBoot的Service注解将RedisGeneratedService对象加入到SpringBoot的Bean对象池中交予SpringBoot托管

Dubbo provider的配置与Mysql分发服务相同

上面的配置已经把Dubbo服务引到项目中并实现了redis服务的调用

下面配置启动类

```
@SpringBootApplication
public class AppStarter {
public static void main(String[] args) {
SpringApplication.run(AppStarter.class, args);
}
}
```

SpringBootApplication注解开启SpringBoot自动装配功能

上述操作已经实现了通过Dubbo provider调用Spring redisTemplate访问Redis服务并执行redis incr命令的基本功能

3.2.3单体测试

基于上述Dubbo实现的provider服务，针对性的需要构建Dubbo consumer服务进行测试

北京大雅集团和信息技术有限公司 相似度检测专用章

新建一个maven项目，引入 Dubbo相关依赖包，新增Dubbo consumer配置

< dubbo:application name="demo-consumer" >

< dubbo:registry protocol="zookeeper" address="192.168.43.172:2281" >

< dubbo:reference id="generatedService" check="true" interface="indeed.dubbo.api.service.GeneratedService" version="1.0" >

consumer中的 reference配置version="1.0"要与provider中声明的version="1.0"保持一致。

新增consumer服务启动类

```
public class AppStarter {
public static void main( String[] args ) {
ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("springdubbo-demo-consumer.xml");
context.start();
GeneratedService demoService =context.getBean(GeneratedService.class);
ResultDto < Long >  hello = demoService.getId("test");
System.out.println(hello.getData());
}
}
```

此时，只需要在consumer运行AppStarter的main方法即可以取到ID的返回值，并且在tbl_increment数据表中看到对应的ssn和ssn_name记录。

再次运行

注:生产环境下慎用Redis的keys命令，该命令在key比较多的情况下会导致redis崩溃。

至此基于Redis的ID分发功能测试完成。

3.2.4小结

基于Redis的ID分发系统主要依赖于redis，因此良好的redis服务器环境就显得尤为重要.而Redis本身作为基于内存型数据库，在内存方面也会有比较高的要求。

可能出现的问题:

redis服务器宕机可能导致的数据丢失，redis集群环境数据同步可能不及时。

。

3.3基于雪花算法的唯一ID分发功能实现

3.3.1概述

基于雪花算法的唯一ID分发功能的实现主要依赖于Twitter 开源的分布式 id 生成算法。其核心思想就是：使用一个 64 bit 的 long 型的数字作为全局唯一 id。在分布式系统中的应用十分广泛，且ID 引入了时间戳，基本上保持自增。

需要注意的点:

北京大雅因和信息技术有限公司 相似度检测专用章

雪花算法的实现本身不具有原子性，所以在实际应用的时候，需要在原有实现类的基础上进行包装，这里借助Spring Bean默认为单例的特性将对雪花算法的实现类的调用包装为单例的Spring Bean，以提供给面向业务层的功能调用，保证每次请求返回ID的唯一性。

3.3.2具体实现

基本思路:

Dubbo provider整合SpringBoot框架，声明单例模式的Spring Bean对雪花算法进行包装，并将此Spring Bean交予Dubbo provider访问，从而达到Dubbo provider通过单例访问雪花算法的目的。

本文基于雪花算法的唯一ID分发功能借鉴了网上对雪花算法的实现

具体实现方法如下

```
public class Snowflake implements Serializable {

private static final long serialVersionUID = 1L;

**

* Thu, 04 Nov 2010 01:42:54 GMT

*

private final long twepoch = 1288834974657L;

private final long workerIdBits = 5L;

private final long datacenterIdBits = 5L;

**

* 最大支持机器节点数0~31，一共32个

*

private final long maxWorkerId = -1L ^ (-1L < < workerIdBits);

**

* 最大支持数据中心节点数0~31，一共32个

*

private final long maxDatacenterId = -1L ^ (-1L < < datacenterIdBits);

**

* 序列号12位

*

private final long sequenceBits = 12L;

**

* 机器节点左移12位

*
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
private final long workerIdShift = sequenceBits;

**

* 数据中心节点左移17位

*

private final long datacenterIdShift = sequenceBits + workerIdBits;

**

* 时间毫秒数左移22位

*

private final long timestampLeftShift = sequenceBits + workerIdBits + datacenterIdBits;

**

* 4095

*

private final long sequenceMask = -1L ^ (-1L < < sequenceBits);

private long workerId;

private long datacenterId;

private long sequence = 0L;

private long lastTimestamp = -1L;

private boolean useSystemClock;

**

* 构造

*

* @param workerId 终端ID

* @param datacenterId 数据中心ID

*

public Snowflake(long workerId, long datacenterId) {

this(workerId, datacenterId, false);

}

**

* 构造

*
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
* @param workerId 终端ID
* @param datacenterId 数据中心ID
* @param isUseSystemClock 是否使用 获取当前时间戳
*
public Snowflake(long workerId, long datacenterId, boolean isUseSystemClock) {
if (workerId > maxWorkerId || workerId < 0) {
throw new IllegalArgumentException(String.format("worker Id can't be greater than {} or less than 0", maxWorkerId));
}
if (datacenterId > maxDatacenterId || datacenterId < 0) {
throw new IllegalArgumentException(String.format("datacenter Id can't be greater than {} or less than 0", maxDatacenterId));
}
this.workerId = workerId;
this.datacenterId = datacenterId;
this.useSystemClock = isUseSystemClock;
}
**
* 根据Snowflake的ID，获取机器id
*
* @param id snowflake算法生成的id
* @return 所属机器的id
*
public long getWorkerId(long id) {
return id > > workerIdShift & ~(-1L < < workerIdBits);
}
**
* 根据Snowflake的ID，获取数据中心id
*
* @param id snowflake算法生成的id
* @return 所属数据中心
*
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
public long getDataCenterId(long id) {
return id > > datacenterIdShift & ~(-1L < < datacenterIdBits);
}
**
*根据Snowflake的ID，获取生成时间
*
* @param id snowflake算法生成的id
* @return 生成的时间
*
public long getGenerateDateTime(long id) {
return (id > > timestampLeftShift & ~(-1L < < 41L)) + twepoch;
}
**
* 下一个ID
*
* @return ID
*
public synchronized long nextId() {
long timestamp = genTime();
if (timestamp < lastTimestamp) {
如果服务器时间有问题(时钟后退) 报错。
throw new IllegalStateException(String.format("Clock moved backwards. Refusing to generate id for {}ms", lastTimestamp - timestamp));
}
if (lastTimestamp == timestamp) {
sequence = (sequence + 1) & sequenceMask;
if (sequence == 0) {
timestamp = tilNextMillis(lastTimestamp);
}
} else {
sequence = 0L;
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
}
lastTimestamp = timestamp;
return ((timestamp - twepoch) < < timestampLeftShift) | (datacenterId < < datacenterIdShift) | (workerId < < workerIdShift) | sequence;
}
**
* 下一个ID（字符串形式）
*
* @return ID 字符串形式
*
public String nextIdStr() {
return Long.toString(nextId());
}
**
* 循环等待下一个时间
*
* @param lastTimestamp 上次记录的时间
* @return 下一个时间
*
private long tilNextMillis(long lastTimestamp) {
long timestamp = genTime();
while (timestamp < = lastTimestamp) {
timestamp = genTime();
}
return timestamp;
}
**
* 生成时间戳
*
* @return 时间戳
*
```

北京大雅因和信息技术有限公司 相似度检测专用章

```
private long genTime() {
return System.currentTimeMillis();
}
}
```

实现类中需要传递两个参数workerId， dataCenterId用于标注终端ID和数据中心ID，这里均设置为1L

配置service服务，将雪花算法设置为单例模式

```
@Service
public class SnowflakeService {
private static volatile Snowflake instance;
public Snowflake getInstance() {
if (instance == null) {
synchronized (Snowflake.class) {
if (instance == null) {
instance = new Snowflake(1L, 1L);
}
}
}
return instance;
}
}
```

Dubbo服务的构建与基于Mysql服务建构方法的类似，需要定义ID系统基础服务接口和实现类，

此处接口复用GeneratedService

实现类

```
@Service
@com.alibaba.dubbo.config.annotation.Service(version = "3.0", timeout = 3000, interfaceClass=GeneratedService.class, retries=0)
public class SnowFlakesGeneratedService implements GeneratedService {
@Autowired
private SnowflakeService snowflakeService;
@Override
public ResultDto < Long >  getId(String salt) {
```

北京大雅因和信息技术有限公司 相似度检测专用章

return new ResultDto < Long > ().setData(snowflakeService.getInstance().nextId());

}

}

Dubbo service version声明为3.0

该实现类直接调用雪花算法工具类Snowflake，使用Snowflake的nextId的返回值作为ID服务的返回值，通过Dubbo的Service注解在Zookeeper中注册以便于在provider 中访问， 并使用Spring的Service注解将SnowFlakesGeneratedService对象加入到Spring的Bean对象池中交予Spring托管

Dubbo的配置与Mysql分发服务相同即可

上面的配置已经把dubbo服务引到项目中并实现了redis服务的调用

下面配置启动类

@SpringBootApplication

public class AppStarter {

public static void main(String[] args) {

SpringApplication.run(AppStarter.class, args);

}

}

SpringBootApplication注解开启SpringBoot自动装配功能

上述操作实现了基于雪花算法的唯一ID分发服务

3.3.3单体测试

下面我们对基于雪花算法的ID分发系统进行测试:

基于上述Dubbo实现的provider服务，针对性的需要构建Dubbo consumer服务进行测试.

新建一个maven项目，引入 Dubbo相关依赖包，新增Dubbo consumer配置

< dubbo:application name="demo-consumer" >

< dubbo:registry protocol="zookeeper" address="192.168.43.172:2281" >

< dubbo:reference id="generatedService" check="true" interface="indeed.dubbo.api.service.GeneratedService" version="3.0" >

consumer中的 reference配置version="3.0"要与provider中声明的version="3.0"保持一致。

新增consumer服务启动类。

public class AppStarter {

public static void main(String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("springdubbo-demo-consumer.xml");

context.start();

北京大雅因和信息技术有限公司 相似度检测专用章

GeneratedService demoService =context.getBean(GeneratedService.class);

ResultDto < Long > hello = demoService.getId("test");

System.out.println(hello.getData());

}

}

此时，只需要在consumer运行AppStarter的main方法即可以取到ID的返回值，并且在日志中看到具体的ID记录。

再次运行

至此基于雪花算法的ID分发功能测试完成。

3.3.4小结

雪花算法的实现主要依赖于系统时钟，因此系统时间的稳定至关重要

可能出现的问题

时钟回拨导致的ID重复，服务器集群环境时钟不统一导致的趋势递增

3.4本章小结

本章对三种ID分发系统进行了实现， 三种实现方式在保证ID唯一性，保持操作原子性等方面各不相同，从而导致系对资源的需求也略有差异，通过对三种实现方式的单体测试，我们可以简单了解到系统资源对这三种ID分发系统影响。

第四章 功能测试及性能比较

4.1功能测试方式和方法

功能测试将会通过在consumer服务中开启多线程调用的方式进行，

测试环境: 内存16G，Cpu 8核16线程，同时运行Dubbo provider， Dubbo consumer， Mysql， Redis。

实现代码如下

public class AppStarter {

public static void main(String[] args) {

ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("springdubbo-demo-consumer.xml");

context.start();

GeneratedService demoService =context.getBean(GeneratedService.class);

ExecutorService service = new ThreadPoolExecutor(100, 100,

60L, TimeUnit.SECONDS,

new LinkedBlockingDeque < > (), r - > new Thread(r, "测试")); long start = System.currentTimeMillis();

int amount=1000;

int count=1000;

北京大雅因和信息技术有限公司 相似度检测专用章

```
CountDownLatch countDownLatch = new CountDownLatch(count);
for (int i=0; i < amount; i++) {
service.execute(
() -> {
try {
ResultDto < Long > hello = demoService.getId("test");
System.out.println("线程[" + Thread.currentThread().getId() + "] 获取响应的ID为: " + hello.getData());
} finally {
countDownLatch.countDown();
}
});
}
service.shutdown();
try {
countDownLatch.await();
} catch (InterruptedException e) {
e.printStackTrace();
}
long end = System.currentTimeMillis();
System.out.println("消耗时间: " + (end - start));
}
}
```

首先设置Dubbo consumer服务访问的version为1.0用于测试基于Mysql的ID分发功能

```
< dubbo:reference id="generatedService" check="true" interface="indeed.dubbo.api.service.GeneratedService" version="1.0" >
```

通过更改AppStarter中的amount标识对调用总数进行调整，更改count标识对单次执行线程数进行调整

实现方式调用总数线程数10

执行时间线程数100

从测试结果来看Mysql 在线程数分别为100和1000的时候执行时间大幅度增加，此时查看数据库连接数配置，如下图

可以发现最大连接数为151，调整最大连接数为1024再次测试

实现方式调用总数数据库最大连接数150

北京大雅因和信息技术有限公司 相似度检测专用章

执行时间数据库最大连接数1024执行时间

此时发现当连接数调整之后执行时间有缩短，可以推断基于Mysql的ID分发系统的并发效率和数据库可支持的连接数有关，那么适当的调整数据库的连接数，对系统本身的性能也会有比较大的影响。

4.2性能比较

由上述的测试结果可知道，性能表现上，基于Mysql的ID分发系统依赖于数据库支持的并行连接数，合理的配置数据库并行连接数对以Mysql构建的ID分发系统性能将有比较好的影响。

Redis的单线程的特性保证了数据操作的原子性，基于内存的操作也保证了数据操作的高性能。

雪花算法运用时钟和位运算实现了在获取递增ID方面性能的大幅度提升。

4.3本章小结

从基本的功能里面可以看到，设计本身在多方面的取舍:

基于Mysql的ID分发系统实现，使用Mysql数据库进行数据存储，保证了数据的安全性，但是因为用到了数据库的行级锁，从而影响到ID获取的效率，而增加并行连接数也会相应的影响到数据库的效率，导致虽然并发数上来了，但是单个的请求响应效率下降了，最终导致实际的效果不尽如人意。

基于Redis的ID分发系统，在使用Redis的时候，无可避免的要受到Redis单线程模型的影响，但正是因为Redis的单线程，我们才不需要消耗额外的资源去保证操作的原子性。

基于雪花算法的ID分发系统在直接使用实现类的时候，很容易出现重复的情况，他是无法保证操作的原子性的，所以我们才需要在原有实现的基础上，采用单例的形式进行包装，以保证使用雪花算法获取ID的时候能够保持操作的原子性，能够保证每次获取到的ID都是唯一的。

本章对三种ID分发系统的测试和性能进行比较，可以了解到在实际生产环境中，适当的系统环境和配置对ID分发系统的会有重要影响.那么选择合适的ID分发系统在实际生产环境中就显得尤为重要了。

第五章 应用场景分析

ID分发系统本身的重要型不言而喻，可以应用在业务系统中的各个环节。

下面对上述三种实现方式的应用场景进行分析:

基于Mysql的ID分发功能依赖于Mysql数据库，那么Mysql数据库就是必备的要素.在高并发场景下如何保证Mysql系统的稳定性也是需要考虑的重要因素，同时Mysql数据的最大连接数也将成为瓶颈.故而在使用基于Mysql的ID分发功能时，最好的选择就是系统并发数在Mysql可以承受的最大连接数内，保证服务可用性的同时，保证系统的吞吐量.在传统的"all in one"系统中这种方式也是一种比较适宜的选择。

基于Redis的ID分发功能依赖于Redis数据库，Redis的高可用，高性能就成为基于Redis的ID分发功能需要考虑的重要因素.那么一个稳定的Redis的集群化部署及管理，跨主机的数据同步，网络通信延迟都可能成为基于Redis的ID分发服务的痛点和难点.在目前Redis集群部署规模化，成熟化的情形下，稳定的Redis集群解决方案也越来越多，未来基于Redis的ID分发功能可能会成为更多用户的选择。

基于雪花算法的ID分发功能依赖于系统时钟，一个统一的，稳定的系统时钟同步功能可能会成为使用雪花算法的必要选择.实际上雪花算法因为在系统依赖方面更少，资源占用更少，容易开发，目前也被越来越多的用户所接受。

第六章 总结与展望

通过对三种ID分发实现方式的实现和分析，我们了解到几种方式在实现上的差异性，实用性和优缺点，那么我们是否有更好的方案呢?

我们以上述三种ID分发服务为基础进行整合，使用雪花算法作为对外的最终ID生成策略，Mysql作为存储服务保障数据安全，Redis作为缓存服务提升存储性能。

在系统启动时进行预热，先使用雪花算法取到若干个ID，写入Redis缓存并将最大值，缓存ID总量，获取时间，获取时间间隔计入数据库留存.在实际使用的时候，定时计算Redis缓存中的ID数据余量，按照(缓存ID总量获取时间间隔)的方式计算单位时间内所需要的ID总数量，并以此为依

北京大雅因科信息技术有限公司 相似度检测专用章

据动态自动循环获取ID总数量并进行ID写缓存，最大ID记入数据库的操作，以实现ID资源弹性控制，在Redis中的ID资源消耗完并且没能执行动态自动获取ID的时候，直接并发调用雪花算法产生初始ID总量两倍或者3倍的ID数量，并自动更新ID获取时间，ID获取时间间隔，以待下次动态自动获取ID的时候提升ID的获取频率和获取总量，保证系统可用性.在系统闲时，通过判断(缓存ID总量获取时间间隔)的比值增加相应的规则，适当减少ID的获取频率和获取总量，以达到动态控制ID资源的目的。经过上面的改造就可以增加ID系统的资源控制弹性，提升系统可用性。由此可见,良好的设计，合理的配置，不同应用的相互协同对分布式系统的服务性能提升具有良好作用。

关于ID分发系统的实现和分析已经告一段落，但是ID分发系统的技术演进不会停止，随着"大数据"的普及和ServiceMesh在互联网行业的普遍应用，ID分发系统也必然将在互联网各个业务系统，各种技术场景中扮演重要角色。

参考文献

分布式全局ID的Twitter的Snowflake（雪花算法）

https:my.oschina.netlenglingxblog1610044

论文具体功能实现代码

https:zhuanlan.zhihu.comp46404167

致谢

学位论文是在我的指导老师赵玉峰老师的亲切关怀与细心指导下完成的。从课题的选择到论文的最终完成，赵玉峰老师始终都给予了细心的指导和不懈的支持。

还要感谢的是我的父母，他们培养了我对学习的浓厚的兴趣，让我在漫长的人生旅途中心灵有了虔敬的归依。在未来的日子里，我会更加努力的学习和工作，不辜负父母对我的殷殷期望！

同时也感谢学院为我提供良好的做 毕业设计中曾经帮助过我的良师益友和同学，以及在设计中被我引用或参考的论著的作者。

说明：

1.文献相似度=送检论文中与检测范围所有文献的相似字数/送检论文正文字符数

2.去除参考文献相似度=送检论文中检测范围所有文献（不包括参考文献）的相似字数/送检论文正文字符数

3.去除本人已发表论文相似度=送检论文中与检测范围所有文献（不包括自引）的相似字数/送检论文正文字符数

4.单篇最大相似度：送检论文与某一文献的相似度高于全部其他文献

5.正文字符数:送检论文正文部分的总字符数，包括汉字、非中文字符、标点符号、阿拉伯数字（不计入空格）

6.正文字数：送检论文正文部分的总字数，正文不包括摘要、关键词、目录、图片、表格、附录、参考文献等

北京大雅因和信息技术有限公司 相似度检测专用章