摘 要

本文基于微服务、大数据分析和机器学习，将此三大热门技术在医疗系统服务中结合并实现。

首先，阐述微服务、大数据分析和机器学习在当今“互联网+医疗”行业的发展现状。

其次，从系统技术架构的角度，对本服务系统进行深入分析并实现业务功能。

第三，结合本服务系统中应用到的微服务、大数据分析组件进行详细分析，以及应用到的机器学习算法原理进行深入说明，并对本服务管理平台的数据处理流程，包括数据采集，数据处理，数据预测，数据反馈等核心步骤进行技术剖析， 最终得出本服务管理平台对社会的价值结论。

最后，对“互联网+医疗”行业的发展做出深入分析。

**关键词：**互联网+医疗；微服务；大数据分析；机器学习；

ABSTRACT

Based on the extensive application of three popular technologies in today's mobile Internet industry: micro services, big data analysis and machine learning, the three popular technologies are combined and realized in this service system.

In this paper, the application of these three popular technologies in the service system will be analyzed in depth.

First, the development of micro services, big data analysis and machine learning in today's mobile Internet industry is expounded.

Secondly, from the perspective of microservices application and architecture, the overall architecture design of the service system is expounded.

Thirdly, a brief introduction and principle analysis of the microservices and big data analysis components applied in the service system, as well as an in-depth explanation of the principles of the machine learning algorithm applied. In addition, the data processing process of the service management platform, including data collection, data processing, data prediction, data feedback and other core steps, is analyzed technically, and finally the conclusion of the value of the service management platform to the society is drawn.

Finally, it makes an in-depth analysis of the development of the mobile Internet industry.

**Keywords**: Mobile Internet; Micro service; Big data analysis; Machine learning; The data analysis

目 录

[摘 要 1](#_Toc525506886)

[ABSTRACT 2](#_Toc525506887)

[目 录 3](#_Toc525506888)

[前 言 4](#_Toc525506889)

[1 行业发展现状 5](#_Toc525506890)

[2 系统技术架构设计与分析 7](#_Toc525506891)

[2.1 系统技术架构设计与层次 7](#_Toc525506892)

[2.2 系统技术架构层次阐述 8](#_Toc525506893)

[2.2.1 网络层 8](#_Toc525506894)

[2.2.2 缓存层 8](#_Toc525506895)

[2.2.3 应用层 9](#_Toc525506896)

[2.2.4 数据计算层 10](#_Toc525506897)

[2.2.5 算法层 12](#_Toc525506898)

[2.2.6 数据存储层 13](#_Toc525506899)

[3 核心技术在系统中应用 16](#_Toc525506900)

[3.1 微服务 16](#_Toc525506901)

[3.1.1 技术特点 16](#_Toc525506902)

[3.1.2 实践应用 16](#_Toc525506903)

[3.2 大数据 22](#_Toc525506904)

[3.2.1 技术特点 22](#_Toc525506905)

[3.2.2 实践应用 22](#_Toc525506906)

[3.3 机器学习 24](#_Toc525506907)

[3.3.1 实践应用 24](#_Toc525506908)

[4 发展前景 24](#_Toc525506909)

[结 论 26](#_Toc525506910)

[致 谢 27](#_Toc525506911)

[参考文献 28](#_Toc525506912)

[附 录 29](#_Toc525506913)

前 言

在当今的互联网时代，微服务、大数据分析处理和机器学习的广泛应用已经深入到所有行业，所以，互联网医疗，作为互联网在医疗行业的新应用，代表了医疗行业新的发展方向，有利于解决中国医疗资源不平衡和人们日益增加的健康医疗需求之间的矛盾，是国家积极引导和支持的医疗发展模式。

近年来，随着国家放开二胎政策，生宝宝的妈妈越来越多，全国大大小小的妇幼保健院的办公压力也逐年增加。由于这些妇幼保健院多数是早期建造的，在网络硬件设施上，承受着很大负荷。所以，移动智能办公在当今互联网时代，体现着举足轻重的作用，凭借其智能、方便、易操作、反应快速、承载信息量大等多方面的优势，越来越受人们的青睐。同时，微服务技术的飞速发展日趋完善，很大程序推动了移动智能办工的使用，在互联网时代，以及刚刚产生的物联网都得到广泛应用。

本课题以准妈妈孕期信息服务管理作为平台，搭载智能手机、智能平板等移动终端设备，实现对准妈妈孕期的身体状态、孕检信息等数据的采集[1]，在PC端通过服务平台，实现对采集数据的预测，达到对准妈妈状况的实时监控和各种预测的目的。

本服务系统可基于多种类型的硬件设备及不同的操作系统实现部署的多样化，并基于大数据组件的高兼容，高并发，高性能的处理能力，提高数据采集、数据清洗、数据计算和数据反馈等核心技术能力，以及利用机器学习算法，实现智能预测及信息的准确推送。

# **行业发展现状**

2018年，沉寂两年的互联网医疗重回头条。

四年前，互联网医疗迎来了大热元年，但很快就遭遇资本寒冬。熬过生死线后，幸存的纯线上问诊难有盈利点，转型迫在眉睫。至于那些转换赛道死磕线下服务的项目，也很难说从轻问诊转型重服务何时才能奏效。

但一系列的政府动作正在释放利好信息。4月11日，李克强总理视察上海华山医院，称赞互联网远程医疗。一天之后，国务院常务会议通过了“互联网+医疗健康发展意见”。5天后国新办又公开介绍了相关“发展意见”精神。当月26日，卫健委发布“互联网+医疗健康”实施计划。

在互联网医疗创业者看来，这意味着“互联网+医疗”首度从国家战略的层面被认可。

自从2014年“互联网+医疗”在短短两年内经历了过山车般的大起大落。在2018年“互联网+医疗”行业形式突然转好，在加上AlphaGo的横空出世，使得在互联网+AI的技术实现趋于深度的发展，更可能将此技术应用于医疗行业，而且发展得会更加好。

从内容上，首先让我们先了解一下，互联网医疗的主要内容？在我国，目前有哪些具有代表性的互联网医疗形式？

互联网医疗，作为依托于互联网技术在医疗行业的新应用尝试，包括了以互联网为载体和技术手段的健康教育、医疗信息查询、电子健康档案、疾病风险评估、在线疾病咨询、电子处方、远程会诊、及远程治疗和康复等多种形式的健康管家服务。

近几年在国内有了一定程度的发展，具有代表性的互联网医疗形式有：以患者社区和医生信息为主的爱好医生，以医师评价和挂号为主的好大夫在线，以电子健康档案采集和应用为主的120ehr网和以疾病风险评估为主的宜康网，以及以远程云诊、全程陪诊为主的[就诊通](https://baike.baidu.com/item/%E5%B0%B1%E8%AF%8A%E9%80%9A)网等等。

在互联网企业开始关注面向医院的全流程服务，全景医疗的概念开始出现，同时，以提升就医体验为目的的医疗服务o2o模式备受关注，越来越多的企业开始关注对医疗活动各个阶段中所产生的数据进行采集、存储和处理，医疗大数据被提到重要位置。随着医疗大数据应用的进一步推进，我国当前医疗资源配置碎片化导致的数据碎片化已经成为政府和产业都高度重视并着力解决的问题，由此引出了产业对电子病历、医疗影像数据化、临床数据、生物医药数据等领域的关注。

未来是一个DT（Data Technology）时代，同时随着人工智能、大数据、云计算的崛起，未来数据将扮演关键的作用，数据将成为如同水电煤一样的基础设施。但是，实际上目前数据的价值还远远没有得到充分的挖掘，比如医疗数据、生物基因数据、交通物流数据、零售数据等。

因此，本服务系统志在利用“互联网+医疗”这个新兴的医疗应用，实现并帮助准妈妈在孕期过程中，了解更多的知识，简化各个阶段的检查步骤，利用机器学习方法实现对准妈妈怀孕期间健康情况的指导建议，最终实现碎片化的医疗数据的整合。

# **系统技术架构设计与分析**

## **系统技术架构设计与层次**



图2.1 – 1

如图2.1 - 1所示，本服务系统基于6层架构设计予以实现，从总体来说，采用这样的架构设计可以使得开发人员的专业分工，专注理解其中一层。由于每一层仅仅调用其相邻下一层所提供的程序接口，只需要本层的接口和相邻下一层的接口定义清晰完整，开发人员在开发某一层时就可以像关注集中于这一层所用的功能和技术。亦可以很容易使用新的实现来替换原有层次的实现，只要前后提供的服务接口相同，即可替换。

系统开发过程中，功能需求不断变化，我们可以替换现有的层次以满足新的需求变化，从而降低系统间的依赖。比如业务逻辑层中的业务发生变化，其他两层即表现层以及数据访问层程序也不需要变化，这大大降低了系统各层之间的依赖。而且有利于复用，充分利用现有的功能程序组件，将已经辨识的具有相对独立功能的层应用于新系统的开发，保证新系统开发的过程中，能够将重点集中于辨识和实现应用系统特有的业务功能， 最终缩短系统开发周期，提高系统的质量，更能适应当前互联网行业的架构设计理念。

## **系统技术架构层次阐述**

### 网络层

在传统的Web项目中，并发量小，用户使用的少。所以在低并发的情况下，用户可以直接访问Tomcat服务器，然后Tomcat服务器返回消息给用户。

但为了解决高并发情况，通常情况下，我们会使用负载均衡，也就是我们多增加几个Web服务器。当用户访问的时候，请求可以提交到空闲的Tomcat服务器上。当网站的访问量达到一定程度后，单台服务器不能满足用户的请求时，需要用多台服务器集群可以使用Nginx做反向代理。并且多台服务器可以平均分担负载，不会因为某台服务器负载高宕机而某台服务器闲置的情况。

负载均衡其意思就是分摊到多个操作单元上进行执行，例如Web服务器、FTP服务器、企业关键应用服务器和其它关键任务服务器等，从而共同完成工作任务。简单而言，就是当有2台或以上服务器时，根据规则随机的将请求分发到指定的服务器上处理，负载均衡配置一般都需要同时配置反向代理，通过反向代理跳转到负载均衡。

### 缓存层

在本服务系统中，使用Redis作为数据缓存层的存储工具，其使用的目的：

1、将热点数据如用户登录信息等，放入缓存数据是为了让客户端很少或者甚至不访问数据库服务器进行的数据查询。在高并发情况下，最大程度降低对数据库服务器的访问压力。

2、防数据穿透，对缓存查询加锁。如果KEY不存在，就加锁，然后查DB入缓存，然后解锁；其他进程如果发现有锁就等待，然后等解锁后返回数据或者进入DB查询。

使用Redis不仅可以让代码变得更简短、更易懂、更易维护，而且还可以使代码的运行速度更快（因为用户不需要通过读取数据库来更新数据）。除此之外，在其他许多情况下，Redis的效率和易用性也比关系数据库要好得多。

**内存方面，**Redis将键值存储在主存中，用于快速地读写访问。

Redis支持主从复制，数据读取在Slave节点完成，而数据写入在 Master 节点完成。复制提供可伸缩性和可用性。任何一个Slave节点宕机，其他的Slave节点还可以提供数据访问。

**从数据结构方面来讲，**Redis不仅存储字符串，还支持列表，集合，哈希和有序集合。Redis的聚合数据可以是整数或者浮点数，而Memcached的聚合数据只能是整数。[2]

**并且**Redis支持创建发布和订阅通道，这样Redis客户端可以订阅任意的通道来进行数据消费，并且任何已订阅该通道的客户端可以发布数据。

Redis还可以将内存中的数据定期保存到文件系统中。当Redis节点故障时，数据可以从Redis数据文件恢复。

### 应用层



图2.2.3 – 1

在本服务系统中，应用层使用Spring Boot框架，并设置用户管理、信息管理、数据管理、系统管理和日志管理模块，如图2.2.3 – 1所求。用户管理模块用于维护着准妈妈和医师信息，并将热点数据如准妈妈和医师的登录信息等放置于缓存数据库中。信息管理模块提供准妈妈和医师信息的更新和修改功能，以及准妈妈检查结果信息和孕检过程等流程信息。数据管理模块提供多维度的数据统计展示。系统管理模块提供孕检预约功能、信息管理功能和检查报告进度展示功能。日志管理模块负责收集各个功能模块执行过程的日志，并提供告警和多维度统计功能。

### 数据计算层

#### 实时流程



图2.2.4.1 – 1

在本服务系统的实时流程中（图2.2.4.1 - 1），使用Kafka和Storm作为数据计算层的服务组件。

在使用Kafka时，通过创建用户信息Topic[3]、检查预约Topic、检查告警Topic和用户反馈Topic，Web应用可根据用户业务请求将相应的请求数据放置于相应的TOPIC队列中。

在使用Storm时，定制相应的业务获取相应Topic队列中数据，依次进行数据清洗、格式转换、业务处理、生成日志和数据入库等功能。数据清洗用于处理无效的请求数据和垃圾数据，从而以减少占用不必要的数据处理资源。格式转换用于将用Json格式定义的请求数据转换成MySQL和Elastic Search中定义的数据结构。业务处理即为针对应用层定制相应的业务数据的处理流程。生成日志即针对整个数据清洗层各个模块的处理功能，生成相应的处理日志，并进行实时监控，对发生异常的情况进行实时提醒。数据入库即当正常数据处理完成后，将有结果数据写入到最终的数据存储中。在本服务系统中，将写入到MySQL和Elastic Search。

#### 离线流程



图2.2.4.2 – 1

离线数据是目前整个数据开发的根本和基础，也是目前数据开发的主战场。在本服务系统的离线流程中，如图2.2.4.2 – 1所示，包含了两部分处理流程：

1、数据回溯即实时流程的全部处理过程。当服务系统发生异常，不能正常处理，造成Kafka中若干Topic队列出现积压，致使产出的最终数据不正确。在这种情况下，我们就必须将一段时间内的原始数据通过离线流程进行数据回溯，把最终产出的数据进行数据补全，并重新写入到数据库中。而对于已经存在的数据，将进行数据更新，以供Web应用层进行正确的数据检索。

2、算法模型数据迭代，会针对不同的数据如准妈妈每次检查报告数据和身体、饮食状况的数据分别进行相应业务数据挖掘，挖掘出更多报告中隐含的数据信息，并由营养师结合全部数据信息给出更加专业地指导建议。

### 算法层

之所以在本服务系统架构中设置算法层，其目的在于：将专业的检查报告数据通过算法概括成点数据即综合值，并与标准值进行对比，而后再以趋势图表的方式进行展示，从而使准妈妈在没有医学基础的前提下，能大致了解每次孕检后身体变化情况。

因此，可以看出算法在数据挖掘中起着很大的作用，而且也是整个数据架构的关键点，所有底层数据发挥作用都需要经过数据挖掘和计算框架输出。数据计算层既要通过不同算法满足不同需求的挖掘需要，又要根据数据需求尽可能的实时输出结果。

### 数据存储层

在本服务系统中，采用MySQL作为数据的最终存储、Elastic Search作为Web应用层进行检索的数据存储。

#### MySQL



图2.2.6 – 1

在本服务系统中，从内容上来看，会创建准妈妈的相关数据表，如图2.2.6 - 1所示，如准妈妈基础信息表、预约信息表、日常身体饮食情况记录表和孕检结果信息表，及医师基础信息表等。准妈妈基础信息表存储在医院建档的准妈妈信息，如姓名、身份证号、籍贯、年龄、最后例假等信息。医师基础信息表存储医院在册的医师的详细，准妈妈可根据自己的意向选择合适自己的医师，进行之后的每次孕检。预约信息表存储准妈妈预约下次孕检的时间与医师的信息。当准妈妈预约成功后，到当日检查，本服务系统以短信方式进行提醒，以免过期，错过检查。日常身体饮食情况记录表存储准妈妈每天的身体和饮食相应信息，这部分数据用于数据挖掘和预测业务，从中挖掘出更多的信息提供给准妈妈和准妈妈指定的医师。孕检结果信息表存储准妈妈每次孕检结果报告的数据，以及经过数据挖掘后的数据信息，并结合医师和营养师的建议，得出更为详细的数据报告。

从使用特点来看，之所以采用MySQL作为数据的最终存储，是由于其具有如下特点：

1. 它使用的核心线程是完全多线程，支持多处理器。

2、有多种列类型：1、2、3、4、和8字节长度自有符号／无符号整数、FLOAT、DOUBLE、CHAR、VARCHAR、TEXT、BLOB、DATE、TIME、DATETIME、TIMESTAMP、YEAR、和ENUM类型。

3、它通过一个高度优化的类库实现SQL函数库并像他们能达到的一样快速，通常在查询初始化后不该有任何内存分配。没有内存漏洞。

4、全面支持SQL的GROUP BY和ORDER BY子句，支持聚合函数(COUNT()、COUNT(DISTINCT)、AVG()、STD()、SUM()、MAX()和MIN())。你可以在同一查询中混来自不同数据库的表，支持ANSI SQL的LEFT 0UTER JOIN和ODBC。

5、所有列都有缺省值，你可以用INSERT插入一个表列的子集，那些没用明确给定值的列设置为他们的缺省值。

6、MySQL可以工作在不同的平台上，支持C、C++、Java、Perl、PHP、Python和TCL API。

#### Elastic Search

采用 Elastic Search 作为Web应用层的检索存储，是由于其具有如下特点：

1. 分布式实时文件存储，可将每一个字段存入索引，使其可以被检索到。

2、实时分析的分布式搜索引擎。分布式：索引分拆成多个分片，每个分片可有零个或多个副本。集群中的每个数据节点都可承载一个或多个分片，并且协调和处理各种操作；负载再平衡和路由在大多数情况下自动完成。

1. 可以扩展到上百台服务器，处理PB级别的结构化或非结构化数据。
2. 支持插件机制，分词插件、同步插件、Hadoop插件、可视化插件等。

# **核心技术在系统中应用**

## **微服务**

### 技术特点

在上一章介绍本服务系统架构的应用层中，我们提到了本服务系统的一些主要功能，如用户管理、信息管理、数据管理、系统管理和日志管理模块，都是应用了Spring Cloud微服务架构技术。

那么什么是“微服务架构”呢？简单的说，微服务架构[4]就是将一个完整的应用从数据存储开始垂直拆分成多个不同的服务，**每个服务都能独立部署、独立维护、独立扩展，服务与服务之间通过RESTful API的方式互相调用**。[5]

通常情况下，类似系统服务的设计与开发都是把所有功能模块整合在一个项目中进行开发与部署，这样就会造成开发人员的工作交集过多，势必是有冲动产生，甚至会影响到开发过程的进度，最终导致项目延期等等各种情况。

所以，这正是区别与其他技术的特点，也正是如此，微服务技术是为了解耦以及服务复用而存在的。

### 实践应用

在开发过程中，通常会把微服务技术的实现可以分成三个模块，如图3.1.2 - 1所示：

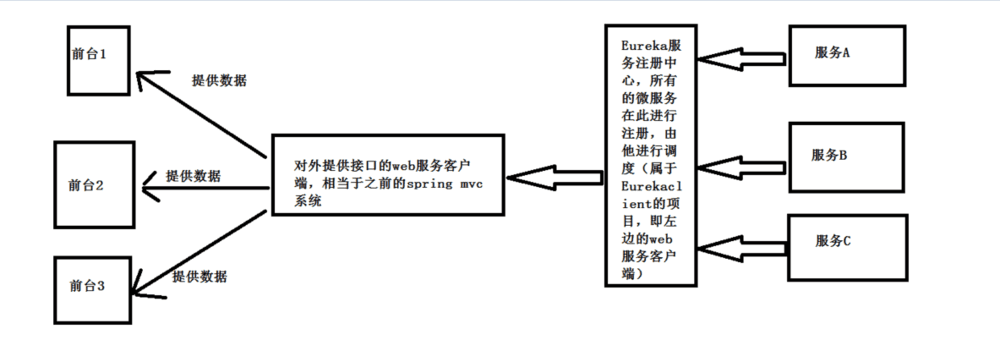


图3.1.2 – 1

从图中我们可以看出，一个微服务架构可以分成三个模块，一个服务的注册中心，一个是服务的客户端（即属于Eureka Client项目），另一个则是提供所有微服务（具体的业务，如用户注册逻辑）的模块。当我们把所有的微服务注册到注册中心之后，我们需要建立一个属于Eureka Clinet的项目，可以通过Discovery Client中的API方法来调用所注册的服务。[6]

接下来，将通过本服务系统的用户注册模块的具体实现代码来说明，微服务技术是怎么实现服务拆分和独立部署的特点。

在开发过程中，使用Eureka服务，将微服务模块连接起来，它是从属于Spring Cloud Netflix （包含服务发现、断路器和监控、智能路由、客户端负载均衡）的子项目。我们需要建立至少三个Spring Boot工程，一个是服务注册中心（Eureka Server），一个服务客户端（Eureka Client），还有一个是用户注册的微服务API（User Registion）。所以，对于其他的功能模块，我们只需要创建与用户注册微服务API类似的子项目，编写相应需求代码，便可实现服务的拆分。在部署时，将各个微服务API注册到服务注册中心，亦可实现独立部署，既而通过服务客户端提供功能模块API的服务。

下面将依次创建这三个Spring Boot工程，实现服务注册中心、服务客户端和用户注册微服务API。

1. 服务注册中心Eureka Server工程

首先，介绍一下配置文件，可看出该工程需要进行哪些配置。内容如下：

# 服务中心的使用端口

pring.application.name=eureka-server

server.port=8761

# 在默认设置下，该服务注册中心也会将自己作为客户端来尝试注册它自己，所以我们需要禁用它的客户端注册行为

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

# eureka.instance.hostname=localhost

# 关闭保护机制，以确保注册中心将不可用的实例正确剔除

eureka.server.enable-self-preservation=false

这样的好处是，我们可以使用相同的代码，但设置不同的服务端口，部署在不同的服务器上，即可微服务的分布式。所以，由此可演化适合不同场景的技术架构。

其次，开发服务中心的启动类，我们只需要编写相应的 main 函数，添加@EnableEurekaServer注解，Spring Boot便可通过此注解，识别到该工程为Eureka服务注册中心。代码量少，简单，易懂，开发快捷，一个微服务的服务中心便创建完成。如下面代码所示。

@SpringBootApplication

@EnableEurekaServer

public class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

二、服务中心客户端

首先，看一下配置文件，与服务端配置的区别。内容如下：

# 服务中心客户端使用的端口

server.port=9001

eureka.instance.hostname=localhost

eureka.client.serviceUrl.defaultZone=http://${eureka.instance.hostname}:8761/eureka/

# 注册到服务中心客户端的功能模块访问信息

spring.application.name=userreg

eureka.instance.preferIpAddress=true

eureka.instance.instance-id=${spring.cloud.client.ipAddress}:${server.port}

其次，与服务中心服务端一样，同样需要一个启动类。与服务端不同的是，需要添加@EnableDiscoveryClient注解，Spring Boot会将该启动类，识别为一个表示能被EurekaClient发现的微服务。代码如下：

@SpringBootApplication

@EnableDiscoveryClient

public class UserregApplication {

public static void main(String[] args) {

SpringApplication.run(UserregApplication.class, args);

}

}

第三，创建相应的业务类，在这样，以用户注册的业务需求为例。在本服务系统中，实现了UserController业务类，并将此业务服务注册到服务中心客户端。代码如下：

@RestController

public class UserController {

@RequestMapping("/userreg")

public Map<String,String> reg()

{

Map<String,String> map=new HashMap<String, String>();

map.put("status","用户注册成功");

return map;

}

}

1. 对外提供接口数据的客户端 Eureka-Client

首先，与之前服务中心服务端和客户端一样，依然需要对比配置文件。在配置文件中，该工程是对外提供接口的，所以他不需要将自己注册到服务端，这和userreg工程是不同的。内容如下：

# 功能模块API使用的端口

server.port=9050

eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka/

# 不需要把自己注册到服务端

eureka.client.register-with-eureka=false

其次，同样需要一个启动类，并在启动类的main函数添加@EnableEurekaClient注解，这样，Spring Boot会将工程识别为EurekaClient的工程，便可以在服务注册中心获取注册的微服务来进行调用。代码如下：

@SpringBootApplication

@EnableEurekaClient

public class MywebApplication {

public static void main(String[] args) {

SpringApplication.run(MywebApplication.class, args);

}

}

第三，添加业务类，并使用DiscoveryClient类来从注册中心获取到userreg微服务的实例。当返回值为List<ServiceInstance>集合时，这就是其他技术服务在不依赖如NGINX这样的负载均衡的服务时，所不能提供和实现的技术能力，因此可以看出微服务架构技术的强大之处。代码如下：

@RestController

public class Myweb {

@Autowired

DiscoveryClient discoveryClient;

@RequestMapping("/userreg")

public Map userreg() {

List<ServiceInstance> list = discoveryClient.getInstances("USERREG");

String serviceUrl = list.get(0).getUri().toString();

System.out.println(serviceUrl);

RestTemplate restTemplate = new RestTemplate();

return restTemplate.getForObject(serviceUrl + "/reg", Map.class);

}

}

至此，我们需要的三个工程后，已经全部完成，然后启动三个工程，并可访问如下地址：

1、服务注册中心服务端：<http://localhost:8761/>

2、服务注册中心客户端，userreg一般不对外提供，可以部署在内网<http://localhost:9001/reg>

3、功能模块的API，对外提供接口的地址，是给前端调用<http://localhost:9050/userreg>



图3.1.2 – 1

所以，从使用微服务架构技术，使本服务系统在开发上实现了简单、快捷、分层、业务扩展等技术特点，在部署上可不依赖如Nginx等负载均衡的技术工具，便可以快速地实现负载均衡和分布式等技术架构。这正是微服务在当今互联网技术公司中发展的原因，充分体现了微服务技术的强大，可以说是今后架构技术的重点发展方向。

## **大数据**

### 技术特点

大数据技术是当今互联网公司应用及处理大数据量信息最常用的实现技术。尽管大数据组件很多，使用任何类似的组件都可能实现自己的业务需求，因此，在设计本服务系统的架构时，就需要选择适合自己的业务场景的组件来实现。所以，针对本服务系统的实现场景，最终选择了Kafka和Storm的两个组件来实现数据处理流程。

使用Kafka的原因是因为在本服务系统的架构设计中，考虑到本服务系统中各种消息结构的不同复杂程度，使用不同的Topic进行分类，与传统消息队列系统不同，在于Kafka的数据持久性、高扩展、高吞吐、高可用、低延迟的特性。

使用Storm的原因是因为考虑到对本服务系统的消息处理上需要采用实时处理，实现低延迟的处理效果。而且在数据计算层的消息体基本上都是采用短消息体，吞吐量基本上在每秒3000至5000个消息，吞吐数据量较小，而且需要在数据处理流程中进行连接计算和API的调用。所以，基于这些业务特点的考虑，在技术造型中，选择Storm，而非Spark Streaming的原因。

### 实践应用

下面以本服务系统在线实时流程[7]对于准妈妈预约检查的消息处理过程作为实例，深度说明一下大数据技术在本服务系统中具体实现。

在这个处理流程中，有两个不同的业务处理。一个是通过微服务API接收消息后将消息写入到Kafka中，另一个是订阅不同Topic进行消费数据进行数据处理和调用其他计算API的过程。

首先，说明第一个业务处理过程，通过微服务的功能模块API接收到消息后，本服务系统会将这些消息根据分类写入到Kafka的Topic中。如下面代码显示：

val props = new Properties()

props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, BROKER\_LIST)

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, classOf[StringSerializer].getName)

props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, classOf[StringSerializer].getName)

val producer = new KafkaProducer[String, String](props)

while ( true ) {

  val ret: Future[RecordMetadata] = producer.send(new ProducerRecord(TOPIC, "uid-" + uid + "|time-" + timestamp, orderMsg))

  val metadata = ret.get  // 打印出 metadata

  println("i=" + i + ",  offset=" + metadata.offset() + ",  partition=" + metadata.partition())

}

producer.close

这样的设计目的是使用消息队列实现了技术解耦，不将写数据库的操作直接暴露给用户，以防止用户通过SQL注入等黑客技术破解应用系统。而且，可以通过简短的代码实现，便可以实现数据接收的业务功能。

其次，第二个业务处理流程才是真正的数据处理过程，是进行数据清洗、数据写入、接口调用的流处理全过程。在实现这个业务过程中，需要通过使用Storm的相关API连接到Kafka的Topic上，并设定并行处理参数，便可以实现多线程处理数据的能力。如下面所示：

String zkRoot = "/storm";

String topic = "t-order";

String clientId = "Topology-Pyramid";

String kafkaZookeeper = "localhost:2181";

BrokerHosts brokerHosts = new ZkHosts(kafkaZookeeper);

SpoutConfig kafkaConfig = new SpoutConfig(brokerHosts, topic, zkRoot, clientId);

kafkaConfig.scheme = new SchemeAsMultiScheme(new StringScheme

kafkaConfig.startOffsetTime = -1;

TopologyBuilder builder = new TopologyBuilder();

builder.setSpout("kafkaSpout", new KafkaSpout(kafkaConfig));

builder.setBolt("OrderIRichBolt", new OrderBolt(), 2).localOrShuffleGrouping("kafkaSpout");

builder.setBolt("DBIRichBolt", new DBBolt(), 3).localOrShuffleGrouping("OrderIRichBolt");

Config config = new Config();

config.setDebug(true);

在这个类中需要指定订阅Kafka的预约检查Topic及Kafka集群的访问地址；OrderBolt类是处理准妈妈检查预约消息的清洗、业务逻辑的实现类，DBBolt类是将最终预约消息写入数据库的实现类。在实现代码中，通过指定三个实现类的处理顺序，并在三个业务实现类中编写相应业务处理逻辑代码，便可以将写入到Kafka Topic中的消息进行消费处理，完成整个针对准妈妈预约检查业务的清洗和数据写入等过程，得到我们最终需要的数据。[8]

## **机器学习**

### 实践应用

在本服务系统中，主要使用了主成分分析法[9]，将二维数据集降维成点投射成一条线，数据集的每个样本都可以用一个值表示。

图3.3.1–1

在图3.3.1 - 1中，参数说明如下：

为第i个检测样本报告

为第i个测试项的权重，取值范围为0 - 1

为第i个测试项的标准值，即为正常范围

为第i个测试项的结果值，即为实际检查结果

计算过程：

将检查报告中，各个检查项定义为特征，检查项权重设为，标准值设为，对应的检查结果为。若检查值在正常范围，即为，若不在正常值范围，根据其检查结果值判断是在上限之外，还是下限之外，计算权重与检查差值乘积的总和值。这样，将二维数据转换成线性数据，可以知道真实值和预测值的匹配程度，更为清晰直观地展示趋势走向[10]。

图3.3.1–2 显示为模拟某位准妈妈前6个月上报的每天身体状态综合值的走势。红色为实际数值的走势，绿色为标准数值的走势。



# **发展前景**

透过本服务系统的实践与应用，我们大致可以了解到微服务、大数据技术和机器学习在“互联网+医疗”行业的重要性及深远的应用性。所以，我们可以“通过现象看本质”，将从研发支撑平台、成果转化和工程应用、人才和能力建设等方面重点突破，创新建设体系与应用模式，全面引领和支撑我国医疗大数据领域的发展，形成完善的产业生态系统。

首先，我们可以建立以医疗大数据应用技术为基础的创新研发平台，组织开展医疗大数据互联互通、医疗大数据整合管理、医疗大数据分析检索、医疗大数据隐私保护、医疗大数据行业应用等研究，进行关键技术的研发、产品化和工程化，快速提高我国医疗大数据应用技术水平，着力探索医疗数据资源的统一标准，形成从多级别的数据资源全面整合共享和规范化，推动医疗数据的互联互通和融合共享。

其次，我们可以培育医疗大数据行业自主创新能力，通过行业领先企业和科研单位的强强联合，发挥技术研发的导向作用，建设开放式、包容式技术研发创新平台和产业联盟，为国内外优势力量提供参与协同创新的沃土，开展医疗大数据关键核心技术和新产品的战略性、前瞻性研发，发展一批具有创新技术路线的医疗大数据的技术模式。

第三，我们可以构建产学研用协同创新联盟促进创新成果转化，通过联合医疗卫生管理部门、医疗卫生服务机构、医疗科研机构、互联网医疗企业和医保医药企业等产业链关键用户单位，构建产学研用协同创新联盟，积极探索医疗大数据应用技术的成果转化，支撑医疗信息化产业链企业开展数据互联互通、共享协同、分析利用等方面的产业链协作。建设技术成果应用示范基地，对接用户单位产业化应用需求，推进技术成果转化与实际应用，推动医疗大数据产业快速发展。

第四，建立医疗大数据应用技术的人才培养平台，为医疗大数据创新工作提供源动力，形成一支结构合理、紧密协作的专病研究、人群队列、公共卫生、生物组学、临床医学、计算机科学等多学科团队。

第五，我们可以通过开展模式创新提升我国医疗全产业链服务能力，聚集全国医疗医药科研创新资源，促进医疗大数据的价值发现，面向行业共性问题和需求，研发一系列医疗大数据服务产品，形成数据驱动的医疗服务决策能力，服务于健康、医药、医保、医疗、大数据多个上下游产业链，提升我国医疗相关产业整体水平。

综上所述，在未来的三到五年，“互联网+医疗”行业将有着更大的发展空间，能够将医疗行业的技术得以质的提高，并且“大数据云化”将是大数据发展到一定程度，更适合该行业的技术产物，甚至将更大化推动“互联网+医疗”行业的发展。

结 论

本论文所取得的成果

在这次毕业设计里，深入地接触微服务架构的搭建、大数据平台环境的搭建和机器学习的深入学习。最终，设计并制件出本服务系统，体验到移动互联网技术应用于医疗行业，以至于应用到其他行业所带给我们的便利。

“互联网+医疗”的发展浅析

就医难是国内医疗面临的最大问题，以互联网化为手段，优化就诊流程，提升患者的就医体验，将会成为未来“互联网+医疗”一直关注的问题。因此，一切围绕这个目的的服务会一直存在，可能贯穿医疗服务的全过程，具体可涵盖：医疗资源查找与匹配、网上挂号、在线问诊、远程诊疗、医药电商、移动医疗等领域。

互联网环境下，医疗服务逐步数字化，这将极大地提高医生与患者直接的相互了解，例如透过先进的影像获取和存储技术，利用大数据分析，获得特定个人的病灶变化情况及同类病例治疗的比较结果，这将为精准治疗提供决策依据。将患者的医疗服务需求精准推送给医生，将医疗服务项目精准提供给患者，实现医疗沟通过程中的双向精准化，才能真正达成精准治疗。

在“互联网+医疗”的驱动下，医疗机构将建立起以患者为中心的全新医疗服务模式，以改善就医体验为目的，逐步实现医疗诊治精准化、医疗组织协同化、医疗服务个性化，将医疗服务扩展到更大范围。未来医疗新模式将在信息技术的推动下向共享、协作、个性化方向发展，“互联网+医疗”将会涌现出更多的应用和模式，而这需要进一步的研究和探索。

致 谢

在论文完成之际，谨向为此文倾注了大量心血的提供大量帮助的老师和家人表示深深地谢意。

向我的指导老师张丽娜致以衷心地感谢，课题的设计以及论文的撰写自始至终都是在张丽娜老师的悉心指导下进行。张丽娜老师在学习科研中给予我无微不至的关怀和指导，使我能够迅速地掌握本专业知识，并解决一个个实际的工程问题，能够迅速地给出处理流程和分析思路，对各方向知识的涉猎。

最后，感谢我的家人，尤其是我的爱人宋晓鹏能够在百忙之中，抽出时间给予我在论文上的指点和帮助。

参考文献

[1] 张思莱.张思莱育儿手记.中国妇女出版社.2017年5月第6次出版

[2] 黄健宏.Redis设计与实现.机械工业出版社.2014年6月第1次出版

[3] Neha Narkhede, Gwen Shapira, Todd Palino. Kafka: The Definitive Guide.

[4] 郑天民. 微服务设计原理与架构. 人民邮电出版社. 2018年5月第1次出版

[5] [美] Sanjay Patni 著 郭理勇 译. RESTful API开发实战. 清华大学出版社. 2018年2月第1次出版

[6] John Carnell. Spring Microservices IN Action. 2018.

[7] 朱松岭. 离线和实时大数据开发实战. 机械工业出版社. 2018年5月第一次出版

[8] 罗聪翼, 龚成志译. Storm应用实践：实时事务处理之策略. 机械工业出版社. 2018年1月第1次出版

[9] 周志华. 机器学习. 清华大学出版社. 2016年1月第1次出版

[10] 李航. 统计学习方法. 清华大学出版社. 2012年3月第1次出版

附 录

**附录A：XX源代码**

XX源程序：

object ProduceOrderMsg {

    def BROKER\_LIST = "localhost:9092"

    def TOPIC = "t-order"

    def main(args: Array[String]): Unit = {

        val props = new Properties()

        props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, BROKER\_LIST)

        props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, classOf[StringSerializer].getName)

        props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, classOf[StringSerializer].getName)

        val producer = new KafkaProducer[String, String](props)

        while ( true ) {

            val ret: Future[RecordMetadata] = producer.send(new ProducerRecord(TOPIC, "uid-" + uid + "|time-" + timestamp, orderMsg))

            val metadata = ret.get  // 打印出 metadata

            println("i=" + i + ",  offset=" + metadata.offset() + ",  partition=" + metadata.partition())

        }

        producer.close

    }

}