社会心理热线服务平台  
软件架构设计说明书

|  |  |  |
| --- | --- | --- |
| 文件状态：  [√] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件标识： | hotline-architecture-draft |
| 当前版本： | v0.1 |
| 作 者： | 王维一 |
| 完成日期： | 2023-05-12 |

版 本 历 史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本/状态 | 作者 | 参与者 | 起止日期 | 备注 |
| v0.1/草稿 | 王维一 | 刘赛威 | 2023/05/12-2023/05/15 |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 概述 3](#_Toc32848848)

[2 设计目标和约束 3](#_Toc32848849)

[3 架构设计 3](#_Toc32848850)

[3.1 总体方案 3](#_Toc32848851)

[3.2 架构说明 4](#_Toc32848852)

[3.2.1 架构图及说明 4](#_Toc32848853)

[3.2.2 架构设计关键点 4](#_Toc32848854)

[3.2.3 高可用性设计 5](#_Toc32848855)

[3.2.4 高性能设计 5](#_Toc32848856)

[3.2.5 可扩展性设计 6](#_Toc32848857)

[3.2.6 安全性设计 6](#_Toc32848858)

[3.2.7 其他设计 6](#_Toc32848859)

[4 部署方案 6](#_Toc32848860)

# 概述

# 社会心理热线服务平台是为提供心理健康服务而设计的在线平台，本架构设计说明书旨在描述该平台的整体架构设计、技术选型、数据流程、安全设计、性能指标等方面的内容。

# 本平台采用微服务架构，其中包括用户服务、咨询服务、聊天服务、评价服务等多个微服务模块。每个微服务模块都可以独立部署、运行和维护，从而保证了系统的高可用性和可扩展性。该平台采用SpringBoot作为后端开发框架，使用MySQL数据库存储数据。前端采用Vue.js、React、UniApp框架进行开发，通过RESTful API进行前后端的交互。

# 从安全性的角度出发，本平台采用了多层安全措施。用户的密码通过加密算法存储，同时还采用了防止SQL注入攻击和XSS攻击等安全措施。在交互过程中，本平台采用HTTPS协议进行数据传输，保证了数据的机密性和完整性。

# 从高性能的角度出发，本平台采用了分布式缓存技术和负载均衡技术。通过Redis缓存用户信息和聊天记录等数据，可以加速数据的访问和查询。同时，通过负载均衡技术，可以将请求分散到我们部署的多台服务器上，从而提高系统的并发处理能力和稳定性。

# 本架构设计说明书旨在为开发团队提供系统架构设计的细节和要求，确保该平台具备良好的可扩展性、可维护性、可靠性和安全性，以满足用户的心理健康服务需求。设计目标和约束

*提示：本节描述会显著影响整个系统架构的系统需求和目标,例如, 包括性能 Performance、成本 Cost、时间 Time、可靠性 Reliability、安全性 Security、合规性 Compliance、技术性 Technology、兼容性 Compatibility。同时，也定义了可能适用于设计和实现策略、开发工具、团队结构、时间表、遗留代码等的约束条件。*

# 架构设计

## 总体方案

微服务架构设计：采用微服务架构设计，各个服务之间采用thrift的方式进行内部调用和通信，并且采用了Dubbo作为服务治理框架，包括负载均衡，降级容灾，注册中心等功能。微服务包括用户服务、咨询服务、聊天服务、评价服务等。每个微服务都可以独立部署、运行和维护，从而保证系统的高可用性和可扩展性。

后端技术选型：采用Spring Boot作为后端开发框架，使用MySQL数据库存储数据。在持久化框架的选择上，我们采用了MyBatis-Plus逆向工程进行开发，最大化减少代码开发量。在日志文件的标准输出上，我们采用了Log4j进行开发，把每一种类型的日志标准化输出到文件，方便后续的排错和线上回归验证。且为了最大化地提高系统性能，我们采用了负载均衡技术，把请求按照一定的逻辑分配到不同的服务器中去。同时，为了保证用户数据的安全，采用多层安全措施，如密码加密、JWT进行用户回话鉴权和防止SQL注入攻击及XSS攻击等。

前端技术选型：采用Vue.js框架进行前端开发，通过RESTful API和后端进行交互。在用户界面设计上，重点考虑用户体验，力求简洁、直观、易用。

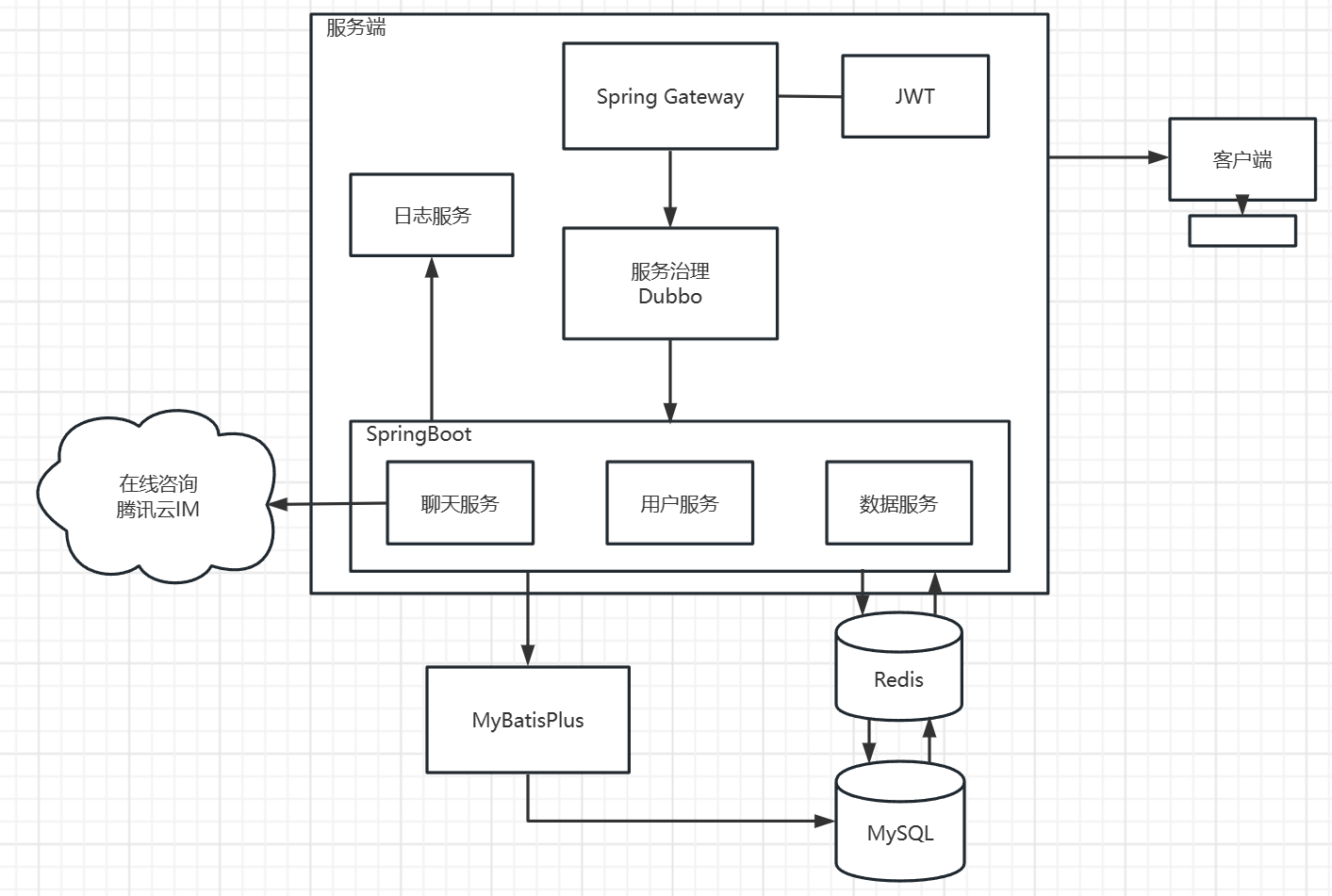
数据流程设计：为了保证用户数据的安全和隐私，用户数据存储在MySQL数据库中，包括用户个人信息和聊天记录等。同时，通过使用分布式缓存技术和负载均衡技术，可以提高系统的并发处理能力和稳定性。

安全设计：本平台采用多层安全措施，包括密码加密、防止SQL注入攻击和XSS攻击等。同时，在交互过程中，采用HTTPS协议进行数据传输，保证了数据的机密性和完整性。

性能指标：为了保证平台的性能，我们将采用负载均衡技术和分布式缓存技术，并进行系统性能测试，以确保系统能够承受大量用户请求，并保证响应速度和稳定性。

## 架构说明

### 架构图及说明



说明可参见整体架构方案

### 架构设计关键点

整体的关键点可以参考架构图给出的示例，其中从流程出发，首先是通过Spring GateWay + JWT进行网关层面的鉴权，这样保证了整个系统的安全性；其次是整个服务的最核心部分，服务治理框架Dubbo，它提供了注册中心、负载均衡、降级容灾、心跳检测等一系列机制来保证系统的高可用性和高扩展性；SpringBoot提供了后端最基础的能力，包括了整个服务的几个最重要的模块：聊天服务、用户服务、数据服务；最后是持久层框架方面，我们选择了MyBatis-Plus来支持持久层的业务，和MySQL数据库进行交互，同时在MySQL的上层还有Redis，基于内存保存用户的相关信息，提升了系统的整体性能。

### 高可用性设计

从整个系统的不同模块出发，分别有以下几部分保证了系统的高可用性：

微服务架构：每一个模块都是解耦且互相独立的，这样可以保证在一个服务因为有问题出现故障时，其他服务仍然能够正常的提供服务，最大程度地保持系统的可用性；此外，我们运用了Dubbo来做整个微服务的服务治理框架，它提供了一系列机制来保证系统的可用性，拿其中的几个特性举例，注册中心提供了一种机制，即所有在运行中的服务都需要以一定的形式注册到注册中心，并且通过心跳检测的机制向注册中心定时发送一个空的TCP报文，这样处理可以保证在一个服务出现故障时，RD能够第一时间感知并排查相关原因进行应对；降级容灾提供了一种机制，即当大量的流量打入，系统无法承受时，微服务框架会使当前提供的服务降级，以支持原本不能支持的流量，从而最大程度地保证系统能够提供服务，而不是直接崩溃或宕机，这些机制都保证了系统的高可用性。

Redis集群：Redis集群指的是我们用多个服务器去部署redis，并且使用redis提供的主从哨兵机制，当redis的其中一个节点下线时，这个机制将会生效，如果是最核心的主节点因故障下线，此时哨兵会通过投票机制选出一个新的主节点，并且把数据全部同步到主节点保证redis集群能够继续提供读写能力，从而保证了系统的可用性。

*提示：支持系统各个功能满足可靠性的设计方案，即系统如何在其应提供服务时间段提供正确的服务*

### 高性能设计

从整个系统的不同模块出发，分别有以下几部分保证了系统的高可用性：

Redis架构：redis是基于内存实现的，相较于磁盘有着更快的读写速度，我们用redis来存储用户的相关信息，这样可以减少系统和磁盘的交互次数及时间，而是直接从内存中读取信息，相比于从磁盘读取，性能大幅提高

微服务架构：由于整个服务被拆解成相互解耦的独立模块，每一个模块的大小都不会太大，这意味着它们的部署会更加轻松，并且每一个模块的体量相对较小，能够发挥更好的性能，从而提高整个系统的速率

数据库架构：对于经常访问的字段，我们建立索引进行查询，这样在查表的时候就不用走全表进行查询，提高速率和性能；此外，对于经常查询到的字段，我们建立联合索引，这样可以减少回表的次数，只需要直接从联合索引中返回结果即可，而不需要回到主索引中再次查询，同样提高了性能和速率。

消息队列架构：我们用消息队列解决了两个问题，其一是10分钟自动断开连接的需求，其二是用户的注册功能，考虑传统的情况，对于10分钟自动断开连接的需求，我们需要不断地设置定时器来对当前的回话进行计时，并且需要一个专门的线程来轮询这个会话以确定是否还有消息的发送，这样的效率是很低的，而采用消息队列的方式，我们只需要用一个延迟队列，当消息超过10分钟的TTL时，直接被消费者消费，而消费者只需要去订阅这个延迟队列，只要有消息就拿出并消费即可，从而提高了性能；对于用户注册的需求，传统的模式是用户发起注册请求->系统响应请求->系统写数据到数据库->系统返回请求，这样串行化的执行大大影响了效率，而现在我们使用消息队列的架构方式，用户发起的请求就是一条消息被放入消息队列中，系统作为生产者，感知到消息后只需要订阅队列并将数据写入数据库返回，而用户作为消费者只需要拿到消费即注册的结果即可，彼此不需要感知的进展过程，而是只关注需要的结果，这样做的好处是大大提高了效率，即系统的性能大大提高。

### 可扩展性设计

系统的可拓展设计主要分为两个方面：纵向拓展性和横向拓展性；

纵向拓展性主要指功能和工具组件上的拓展性。我们根据需求分析得出的领域模型、SDS、用例分析，将系统业务分为一个个模块，模块内部耦合紧密，且力求简洁、贴近实际的API设计，这使得模块内部工作流程简单、调用快速且成本极低，同时模块符合开闭原则，紧密的耦合对修改封闭，抽象的接口使得我们的需求变化可以采用拓展接口的方法实现（对扩展开放）；模块与模块之间，在开发初期为调试和测试的方便性采用单体，开发后期转化为微服务架构，模块之间采用TCP/RPC通信，这样对新模块的加入也会很友好，新业务的创建和消除会变得很简单；同时，在技术层面，我们将不同的工具和服务部署在不同的进程中，而不是部署在同一个jar包中，这样可以让我们的工具可以方便的进行替换和更新；同时我们使用Dubbo或Eureka治理我们的服务，可以做到服务的快速注册或下线，这使我们的系统拥有很好的纵向拓展性。

横向拓展性主要来源于我们部署的网关服务和服务治理组件，gateway可以做到对客户暴露统一的接口（并做一些流量控制和权限管理等），服务限流和负载均衡则让我们更好地管理我们的服务，使得我们可以使用多个虚拟机或容器进行多个服务实例的部署及管理，当流量逐渐增大或突发流量来临时，我们可以采用多实例部署加实施相应后备策略的方式来应对。

### 安全性设计

*提示：保证系统信息安全的设计方案。例如系统提供权限控制功能，权限控制包括两部分：身份识别和人员权限控制。身份识别如何设计，人员权限控制如何设,*

### 其他设计

*提示：上述以外的其他设计考虑点，例如指定开发语言、符合公司的某些标准等*

# 部署方案

*提示：部署视图显示的是系统的实际部署情况，包括硬件要求、服务器部署方式、组网方式，它是为了便于理解系统在一组处理节点上的物理分布。在系统中，只包含有一个部署视图，用来说明各种处理活动在系统各节点的分布。但是，这个部署视图可以在每次迭代过程中都加以改进。部署视图中包括进程、处理器和设备。进程是在自己的内存空间执行的线程；处理器是任何有处理功能的机器，一个进程可以在一个或多个处理器上运行；设备是指没有任何处理功能的机器。*