

- 一、根据业务规划设计表结构
- 二、设计表结构
- 三、使用MyBatis-plus快速实现数据库访问
- 四、用户登录状态查询功能实现要点
  - 前端axios发送请求
  - 合理使用缓存，避免缓存穿透
- 五、用户登录注册功能实现要点
  - 分布式主键生成服务

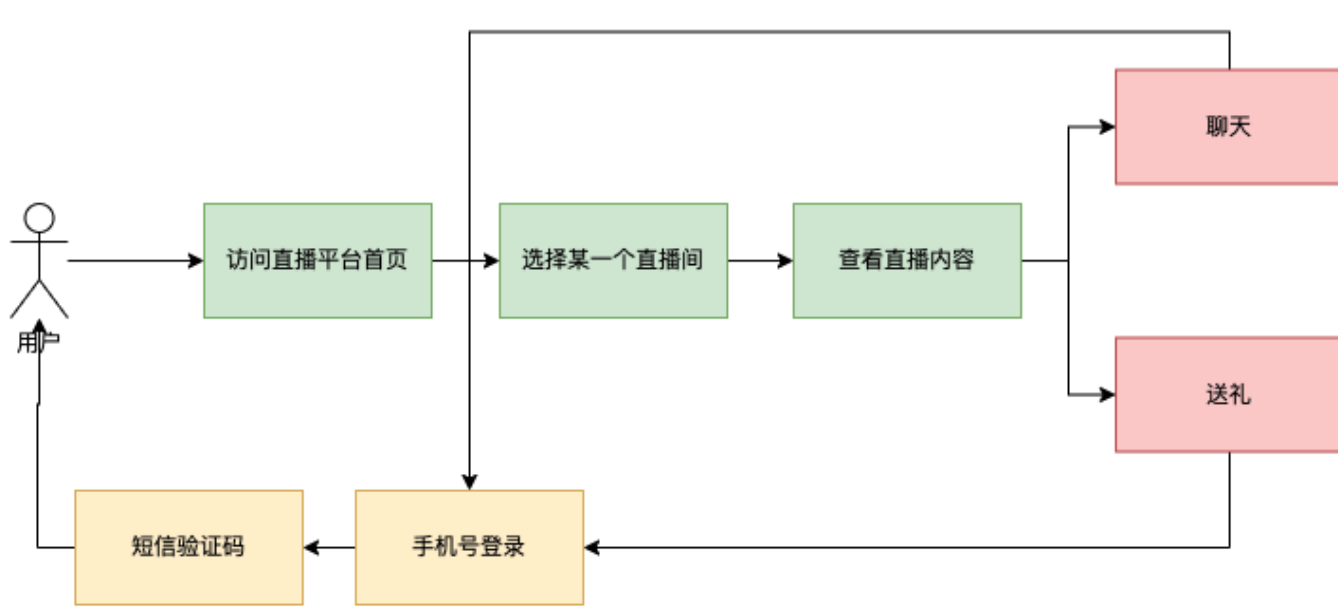
#### 四、用户模块存储方案

阶段目标：在Dubbo服务调用的基础上，实现数据库访问，打通前后端数据通道。

简化业务：在直播首页，可以查询到一个默认的登录用户。

## 一、根据业务规划设计表结构

规划用户权限相关的业务操作：



绿色不需要登录，红色需要登录

## 二、设计表结构

数据库建表语句

```

-- 建库
create database tllive character set utf8mb4 collate utf8mb4_general_ci;
-- 建表
use tllive;
-- 用户表
CREATE TABLE `t_user` (
  `user_id` bigint NOT NULL DEFAULT '-1' COMMENT '用户id',
  `nick_name` varchar(35) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin DEFAULT NULL COMMENT '昵称',
  `avatar` varchar(255) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin DEFAULT NULL COMMENT '头像',
  `true_name` varchar(20) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin DEFAULT NULL COMMENT '真实姓名',
  `sex` tinyint(1) DEFAULT NULL COMMENT '性别 0男, 1女',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP,
  `update_time` datetime DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`user_id`)
) ENGINE=InnoDB COMMENT='用户基础信息表';
-- 插入测试用户
INSERT INTO tllive.t_user (user_id, nick_name, avatar, true_name, sex, create_time, update_time) VALUES (1, 'roy', '/img/avatar.png', 'test', 1, '2024-04-10 17:35:12', '2024-04-10 17:35:15');
-- 用户手机绑定表
CREATE TABLE `t_user_phone` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT COMMENT '主键id',
  `phone` varchar(200) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin NOT NULL DEFAULT '' COMMENT '手机号',
  `user_id` bigint DEFAULT '-1' COMMENT '用户id',
  `status` tinyint DEFAULT '-1' COMMENT '状态(0无效, 1有效)',
  `create_time` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  `update_time` datetime DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `udx_phone` (`phone`),
  KEY `idx_user_id` (`user_id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci COMMENT='用户手机表';
-- 短信推送记录表
CREATE TABLE `t_sms` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT COMMENT '主键id',
  `code` int unsigned DEFAULT '0' COMMENT '验证码',
  `phone` varchar(200) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT '' COMMENT '手机号',
  `send_time` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '发送时间',
  `update_time` datetime DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  KEY `code` (`code`)
) ENGINE=InnoDB AUTO_INCREMENT=1 COMMENT='短信发送记录';

```

## 三、使用MyBatis-plus快速实现数据库访问

三个步骤：

## 1、引入maven依赖

```
主Pom
<mysql.version>8.0.28</mysql.version>
<mybatis-plus.version>3.5.3</mybatis-plus.version>

user-provider
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
</dependency>
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>${mybatis-plus.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

## 2、创建Entity和Mapper

## 3、配置mysql和redis数据源

```
spring:
  application:
    name: tl-live-user-provider

  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.65.210:3306/tllive?useUnicode=true&characterEncoding=utf8
    username: root
    password: root
  data:
    redis:
      host: 192.168.65.210
      port: 6379
      lettuce:
        pool:
          min-idle: 10
          max-active: 100
          max-idle: 10
```

## 3、注册Mapper，使用Mapper快速实现数据库访问。

# 四、用户登录状态查询功能实现要点

## 前端axios发送请求

几个核心问题：

- 前端数据如何发送到后端Controller接口
- 前端如何判断后端响应是否正确

## 合理使用缓存，避免缓存穿透

---

### 核心问题

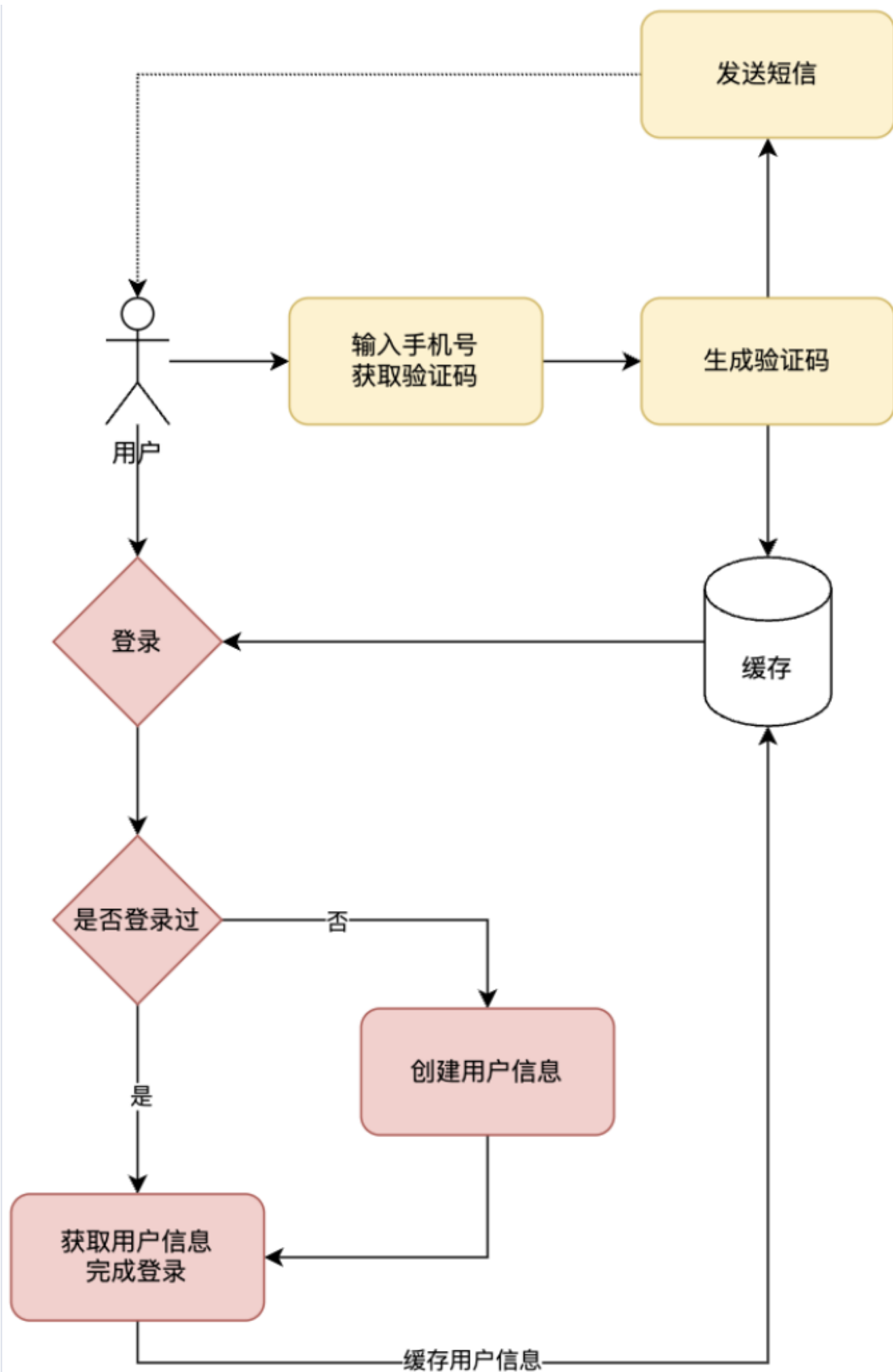
- 缓存穿透：在当前项目中，如果用户登录时，一直用一个不存在的错误用户名进行登录。那么登录请求会穿透缓存，直接进入数据库查询。如果用户一直重复用这个用户名登录，最终就会造成数据库压力过大。
- 解决思路：当前项目当中，采用的解决思路是在Redis缓存当中，给这个用户名添加一个短期的缓存，值是空。这样，下次再有重复的登录请求，就可以在缓存当中直接过滤，而不用到数据库中查询。这样就起到了保护数据库的作用。

当然，解决思路并不止这一种，业界还有另外一种常见的解决思路，是在Redis中构建一个布隆过滤器。过滤器中添加了系统里全部的用户名信息。前端登录请求过来，可以在布隆过滤器中先进行一次判断。如果布隆过滤器判断这个用户名不存在，那么这个用户名就肯定不会在数据库中存在，也就不需要去数据库里查询了。这样也可以对数据库起到一定程度的保护作用。

## 五、用户登录注册功能实现要点

---

用户手机登录业务流程：



# 分布式主键生成服务

分布式主键生成服务是微服务中非常核心的一个服务，也是非常考验程序员对业务问题理解深度的一个场景。

## 1、常见的几种主键生成模式利弊分析

- 数据库自增

优点：简单

缺点：数据库压力大，并且不灵活

- 应用自己生成

常用方案：UUID、NanoID、雪花算法

优点：高性能

缺点：对算法要求高，容易产生ID碰撞。所以可选方案实际上并不多。很容易和业务要求产生冲突。

UUID 和NanoID都不是数字型的，对数据库不太友好。

- 服务统一生成

优点：统一管理ID，不会产生ID碰撞。数据定制比较方便。

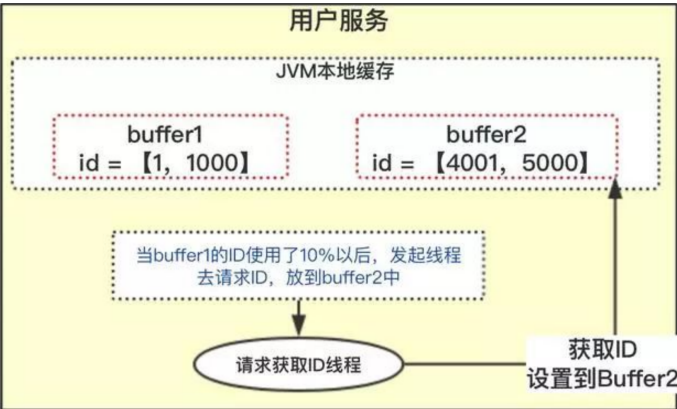
缺点：频繁的网络请求。引入了第三方服务，对服务的高可用要求高。

常见方案：

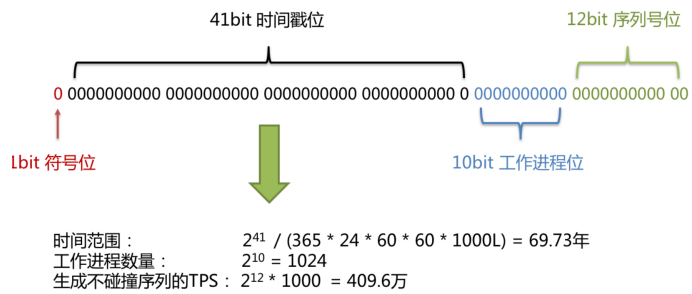
### 1、Segment方案

biz_tag	max_id	step	desc	update_time
user_tag	0	1000	用户ID生成规则	
order_tag	0	2000	订单ID生成规则	

美团Leaf框架的双Buffer缓存方案



Twitter的Snowflake雪花算法方案。



## 2、CosID的主键生成方案分析

- 针对Segment方案

参考美团Leaf方案，需要解决的几个问题：

- 1、强依赖于DB。

CosID可选择多种存储组件。基于多种存储组件实现了相对统一的分配机制。

- 2、双Buffer升级为SegmentChain。

解决了两个核心问题： 1、10%的阈值不太灵活。2、网络不可用时，双Buffer会有点不够用。

- 针对SnowFlake雪花算法方案

雪花算法存在的问题：

- 1、时钟回拨问题
- 2、工作进程位如何设置？
- 3、序列位不递增造成的问题

CosID的雪花算法优化方案：

- 1、机器位自动分配
- 2、序列位递增

## 3、CosID使用实战

CosID使用简单，但是思想不简单

SQL建表语句

```
#单独建一个库
create database tllive_cosid collate utf8mb4_general_ci;

#machine表需要手动创建
create table if not exists cosid_machine
(
    name          varchar(100)      not null comment '{namespace}.{machine_id}',
    namespace      varchar(100)      not null,
    machine_id     integer unsigned  not null default 0,
    last_timestamp bigint unsigned  not null default 0,
    instance_id    varchar(100)      not null default '',
    distribute_time bigint unsigned  not null default 0,
    revert_time    bigint unsigned  not null default 0,
    constraint cosid_machine_pk
        primary key (name)
) engine = InnoDB;

create index idx_namespace on cosid_machine (namespace);
create index idx_instance_id on cosid_machine (instance_id);
#另外还需要一张cosid表，就不用手动创建了。可以自动创建。
```

基础pom依赖



```

<cosid.version>2.6.8</cosid.version>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>me.ahoo.cosid</groupId>
    <artifactId>cosid-spring-boot-starter</artifactId>
    <version>${cosid.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>me.ahoo.cosid</groupId>
    <artifactId>cosid-jdbc</artifactId>
    <version>${cosid.version}</version>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>${mysql.version}</version>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <scope>test</scope>
</dependency>

```

## 前置配置文件

```

spring:
  application:
    name: test
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://192.168.65.212:3306/tllive_cosid?
useUnicode=true&characterEncoding=utf8
    username: root
    password: root

```

- 1、使用SengmentID的share方式生成ID。演示技术+分析后端表数据
- 2、使用SnowflakeID的share方式生成ID， machine手动分配
- 3、使用SnowflakeID的share方式生成ID， machine自动分配；

4、混合使用SegmentID和SnowFlakeID方式，不排除share。--应该会造成bean冲突。

5、混合使用SegmentID和SnowFlakeID方式，排除share，自行创建序列化器。