尚硅谷大数据项目之线上问诊实时数仓（数据接口大屏展示）

（作者：尚硅谷研究院）

版本：V1.0

# 第1章 数据可视化接口

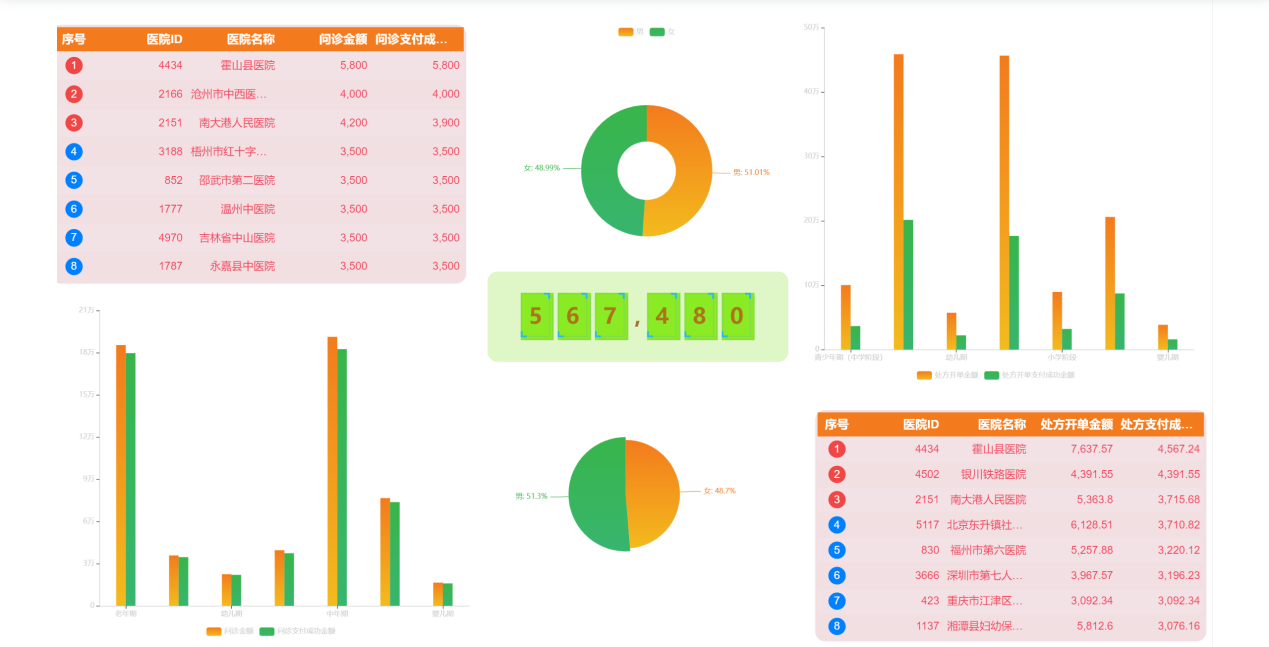
## 1.1 设计思路

DWS 层把轻度聚合的结果保存到Doris中，主要的目的就是提供即时的数据查询、统计、分析服务。这些统计服务一般会以两种形式呈现，一种是面向专业数据分析人员准备的 BI 工具，一种是面向非专业人员的更加直观的数据大屏。

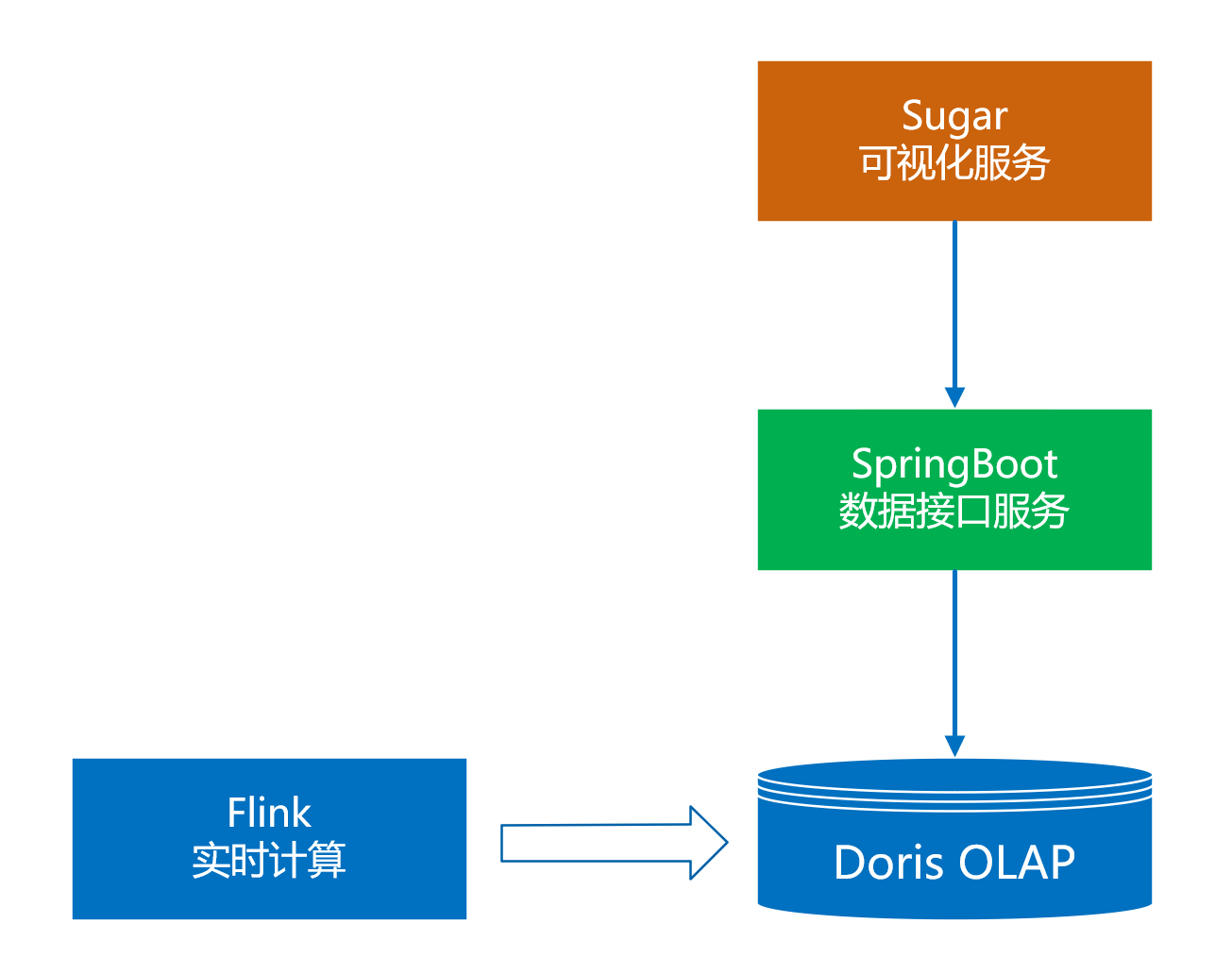
本项目将面向 Sugar 数据大屏开发数据接口服务。

## 1.2 需求梳理

### 1.2.1 最终显示效果图



### 1.2.2 接口执行过程



DWS 层计算结果存储在Doris，本项目将开发数据接口查询Doris中的数据，提供给 Sugar 进行大屏展示。这里主要有两项工作：

* 配置可视化大屏服务。
* 编写数据查询接口以供可视化大屏进行访问。

# 第2章 环境准备

## 2.1 Sugar 简介

### 2.1.1 产品介绍

Sugar是百度云推出的敏捷 BI 和数据可视化平台，目标是解决报表和大屏的数据 BI 分析和可视化问题，解放数据可视化系统的开发人力。

### 2.1.2 使用入口

<https://cloud.baidu.com/product/sugar.html>



### 2.1.3 创建数据大屏

**1）点击【立即使用】后，登录百度账号**

**2）然后首先创建组织**



**3）创建中选择产品【大屏尝鲜版】，首次使用有一个月的试用期**

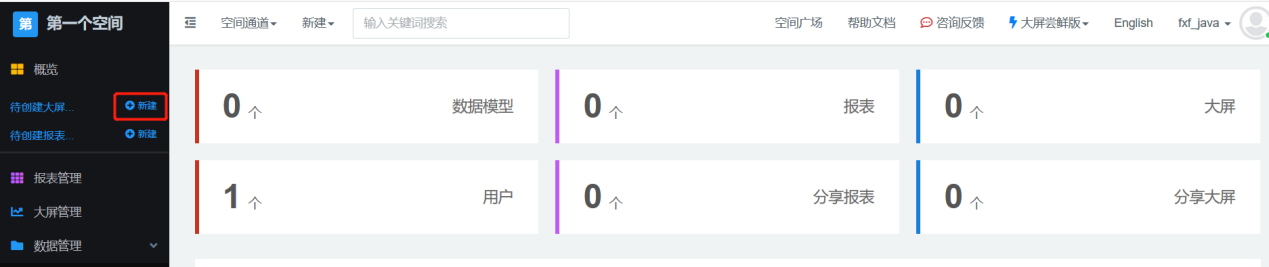


**4）新建好组织后选择【进入组织】**

 **5）然后进入默认的【第一个空间】**

****

**6）在空间中选择【待创建大屏】后的【新建】**

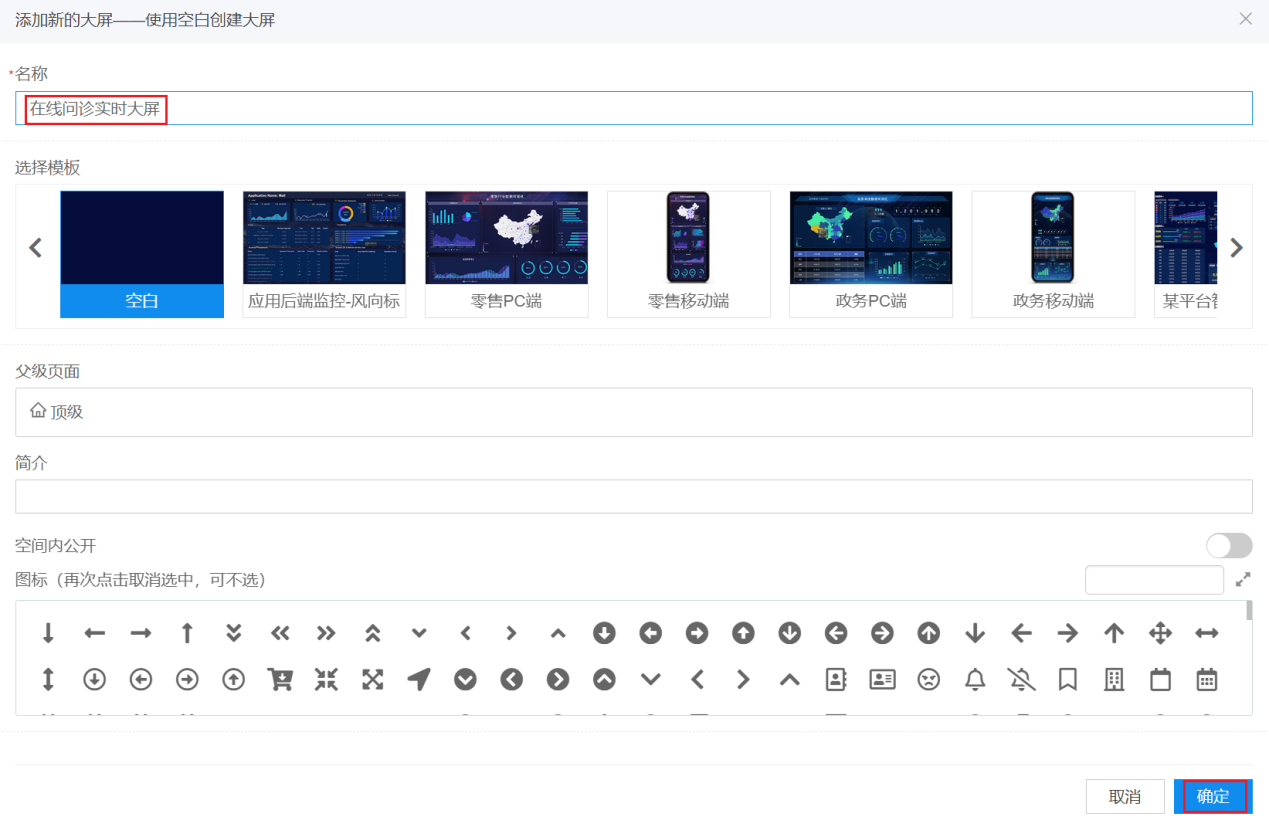


**7）选择大屏的模板**

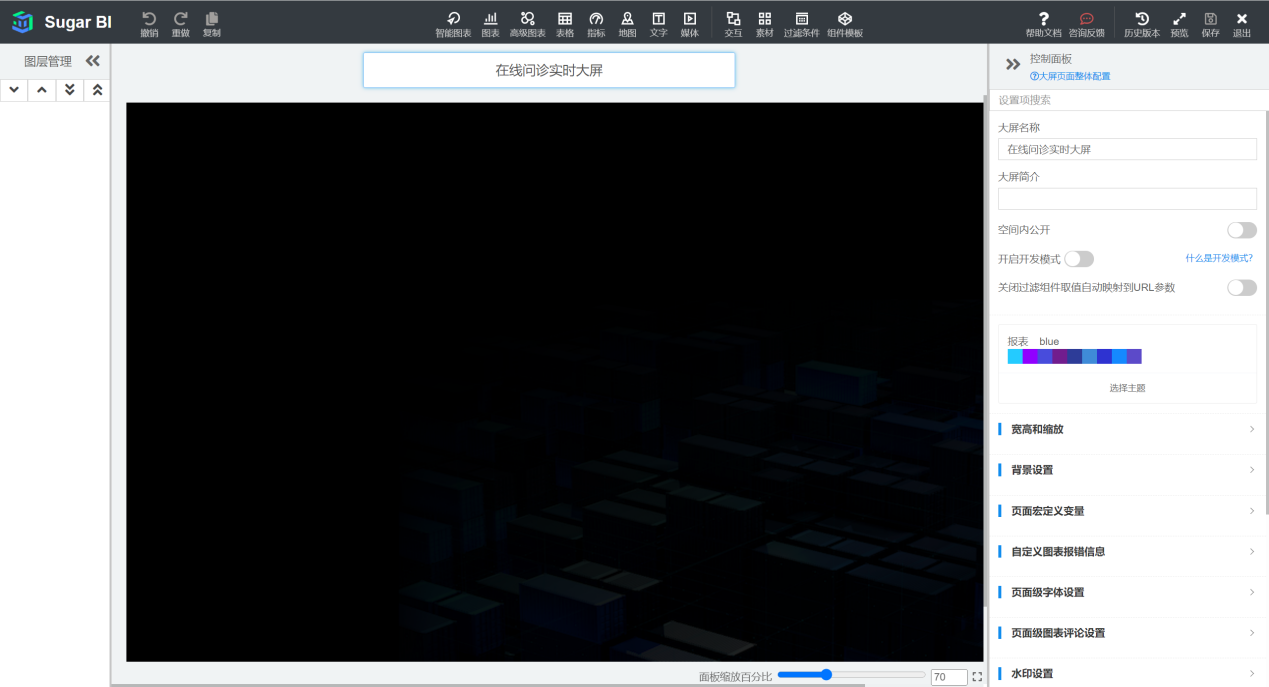


**8）可以选空模板，也可以根据现有的模板进行修改**

我们这里选择空白模板，并指定大屏的名称



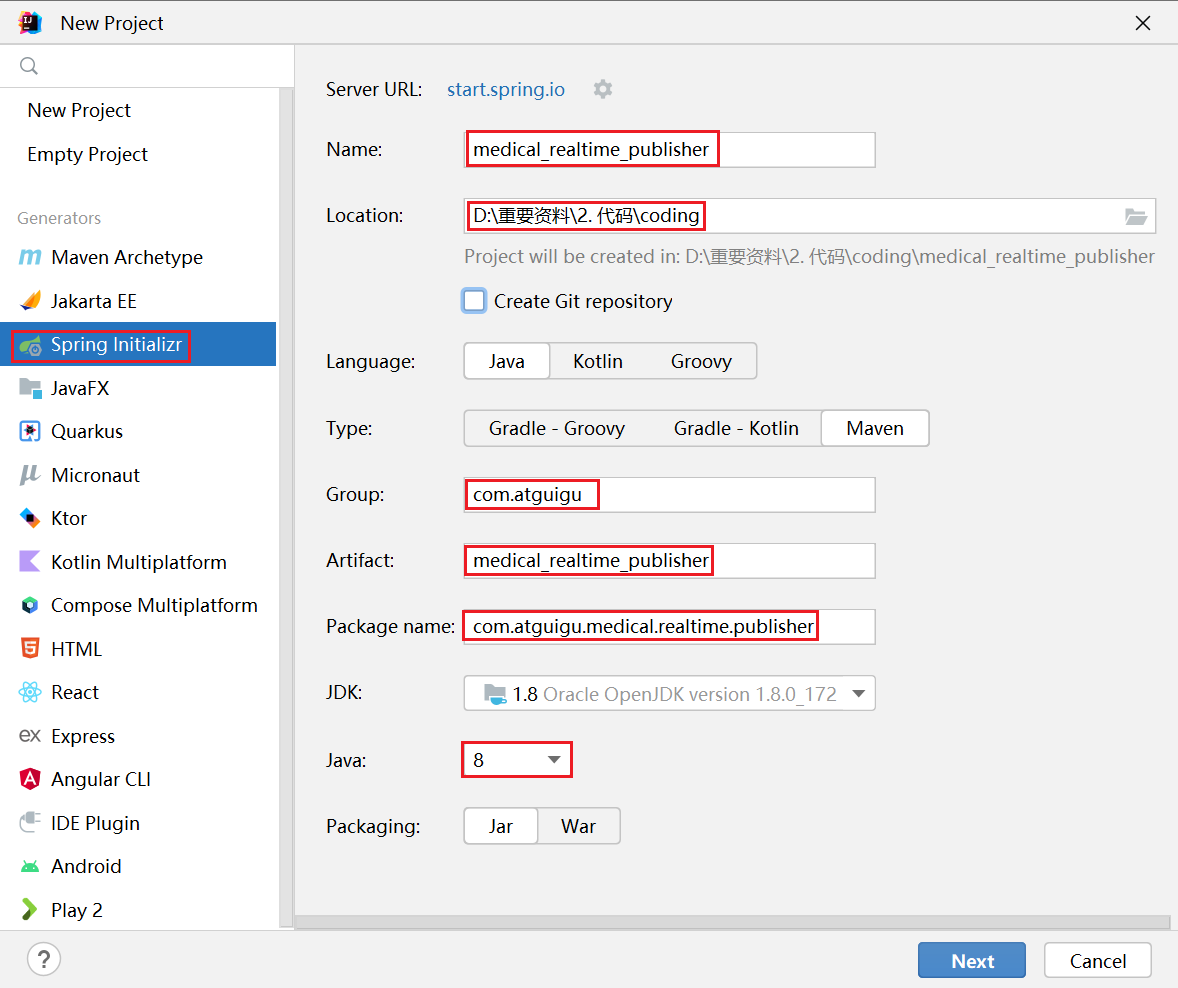
**9）进入大屏的编辑窗口**



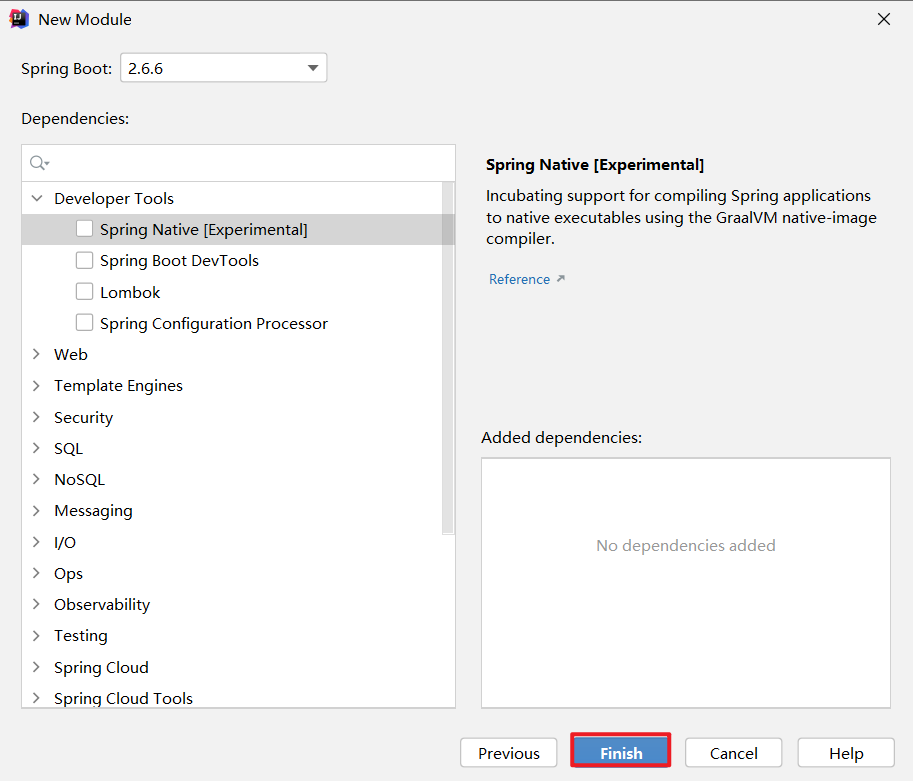
## 2.2 SpringBoot 开发环境构建

### 2.2.1 步骤

**1）新建项目medical\_realtime\_publisher**



此处不勾选，在 pom 文件中添加依赖。



**2）添加依赖**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

<artifactId>mybatis-spring-boot-starter</artifactId>

<version>2.1.3</version>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

<version>3.11</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.16</version>

</dependency>

<dependency>

<groupId>org.apache.doris</groupId>

<artifactId>flink-doris-connector-1.17</artifactId>

<version>1.4.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

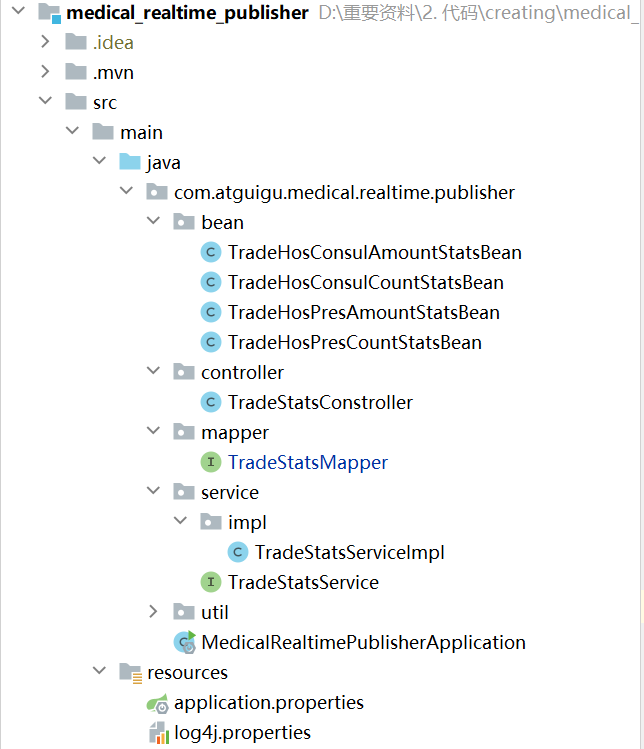
<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

**3）目录结构如下**



**4）在 application.properties 内添加如下内容**

SpringBoot 内嵌了 Tomact，默认端口为 8080。集群 Zookeeper 版本为 3.5.7，该版本提供的 AdminServer 服务端口号也是 8080，为了避免端口冲突，此处将 SpringBoot 内嵌的 Tomcat 容器端口号修改为 8070。

接口对接的数据库为 ClickHouse，需要指定驱动及 url。

server.port=8070

#配置 Doris 驱动以及 URL

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://hadoop102:9030/medical\_realtime

spring.datasource.username=root

spring.datasource.password=aaaaaa

### 2.2.2 SpringBoot 项目分层

**1）表示层（也叫控制层）**

主要任务是拦截并处理请求。表示层代码通常在 controller 包下。

**2）业务层**

业务层代码通常在 service 包下。

service 下会有个名为 impl 的子包，里面放置 service 层接口的实现类。

实现类类名规范：接口名后面加Impl

**3）持久层**

和数据库交互，最常用的框架是 Mybatis，所以也叫 Mapper 层。

持久层代码通常在 mapper 包下。

## 2.3 内网穿透

### 2.3.1 内网穿透简介

内网即局域网。假设局域网中有一台电脑部署了web服务，现在希望所有人都能访问它。很显然，这台电脑只有一个局域网ip，没有公网ip。同一局域网内的设备可以通过局域网ip访问此电脑，而局域网之外的设备无法访问。

内网穿透可以将 ip + 端口唯一标识的本机服务映射为公网域名，局域网之外的设备可以通过该域名访问本机服务。

### 2.3.2 实现步骤

本项目将使用natapp提供的内网穿透工具。

<https://natapp.cn/>

**1）注册并登录**

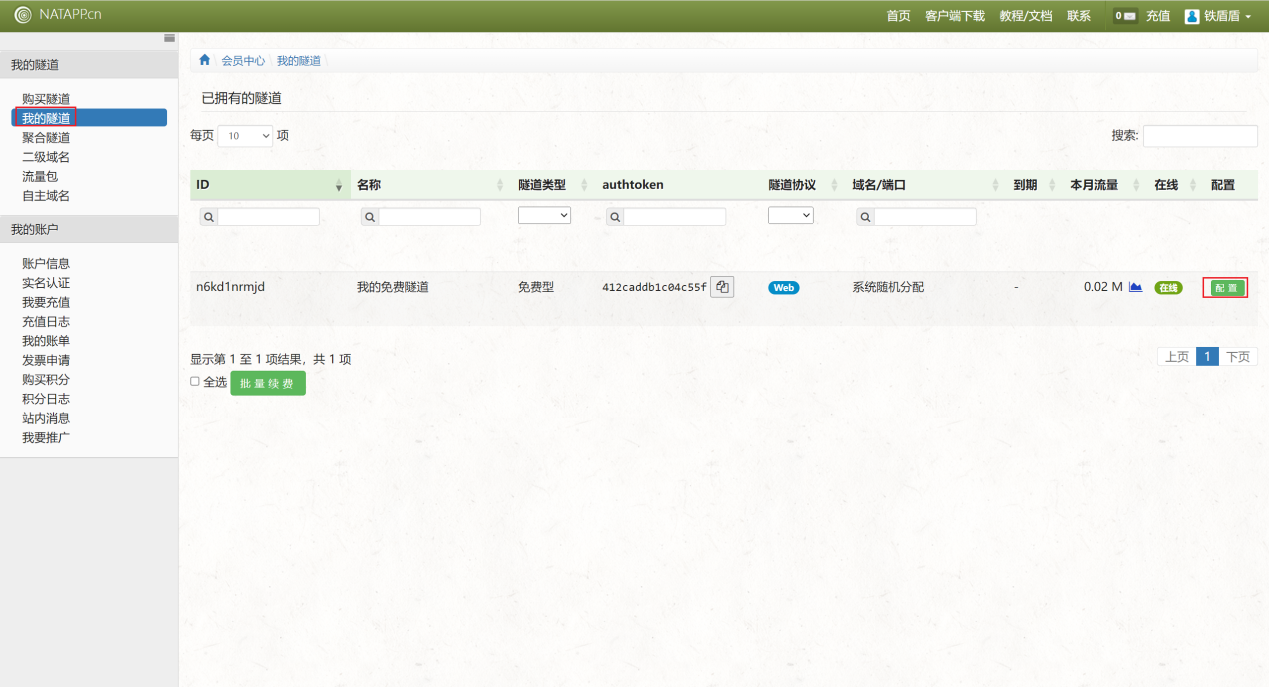


**2）购买免费隧道**



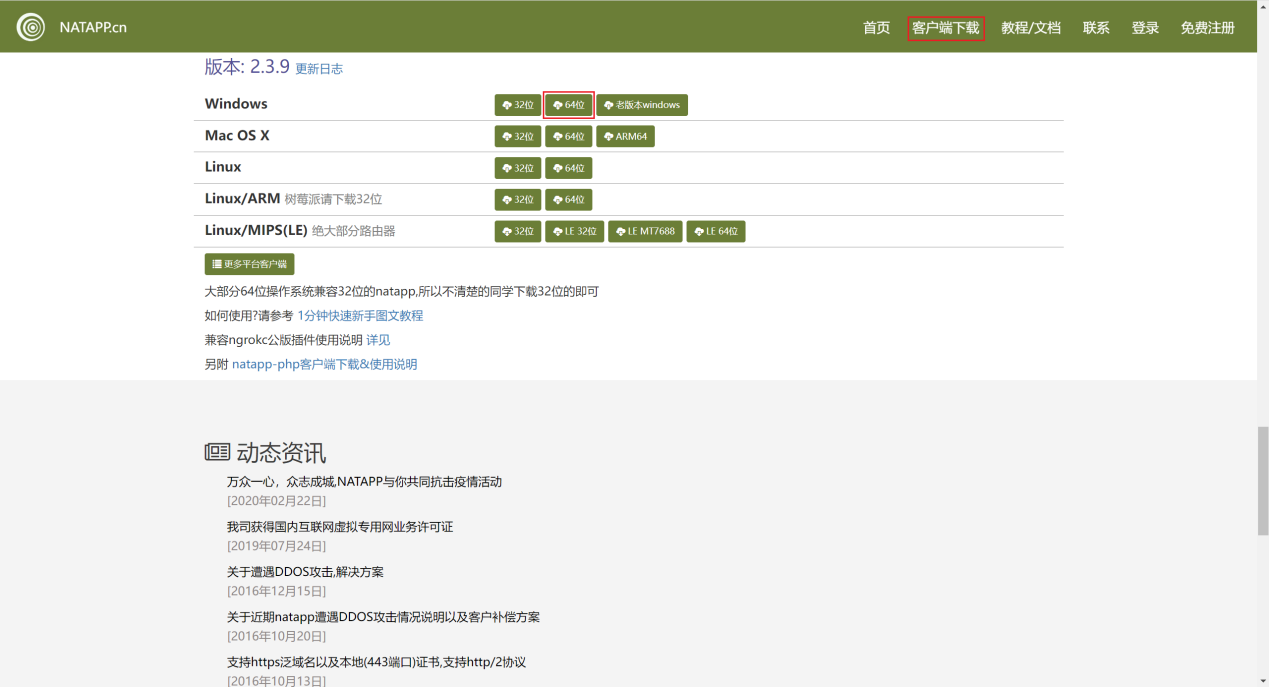


**3）复制authtoken**





**4）下载工具**



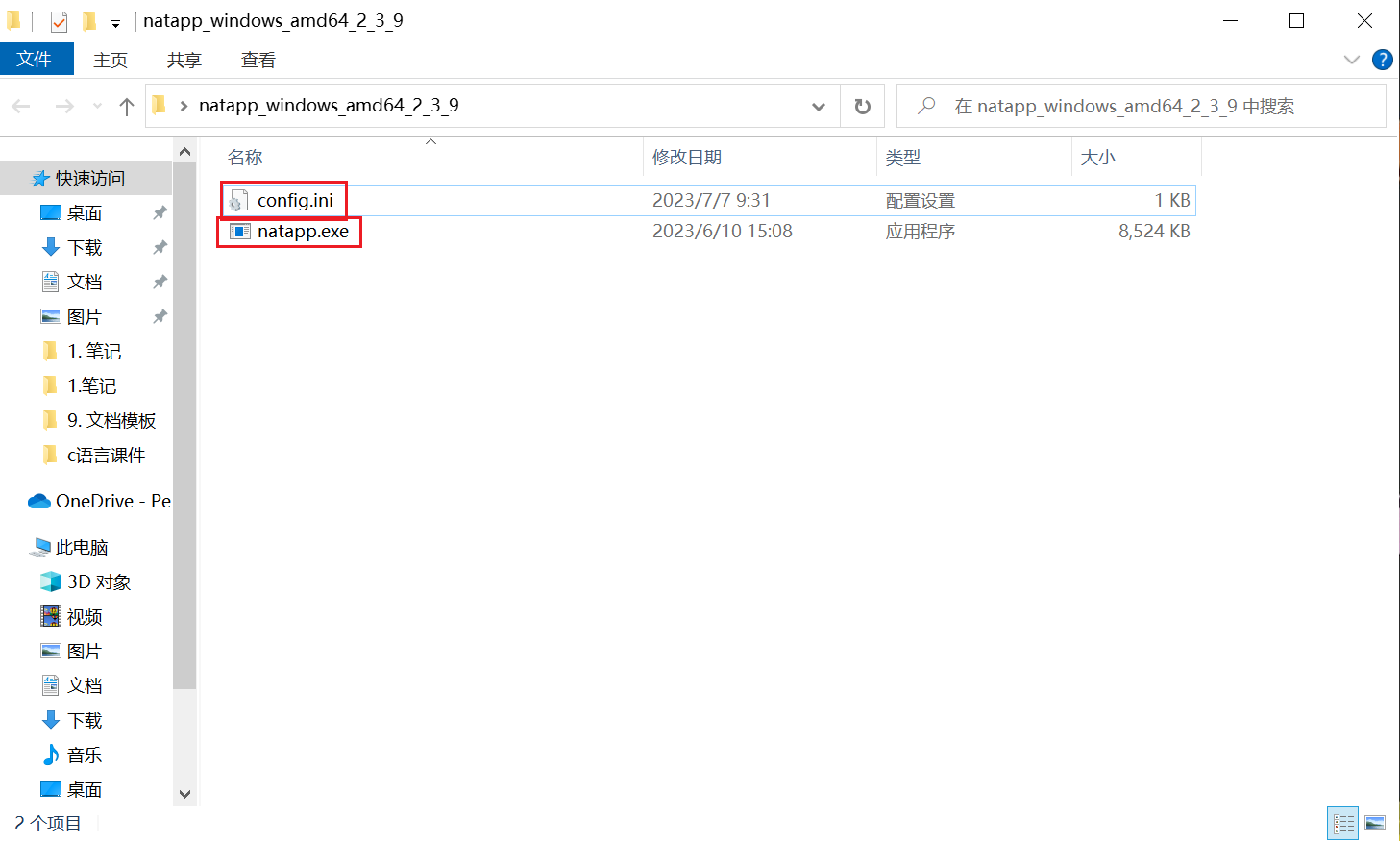
解压得到natapp.exe，将其置于任意目录下。

**5）下载配置文件config.ini**





**6）将配置文件置于natapp.exe同级目录下，如下。**



**7）修改配置文件**



**8）用第三步获得的authtoken替换配置文件中的默认值**

#将本文件放置于natapp同级目录 程序将读取 [default] 段

#在命令行参数模式如 natapp -authtoken=xxx 等相同参数将会覆盖掉此配置

#命令行参数 -config= 可以指定任意config.ini文件

[default]

authtoken=**412caddb1c04c55f** #对应一条隧道的authtoken

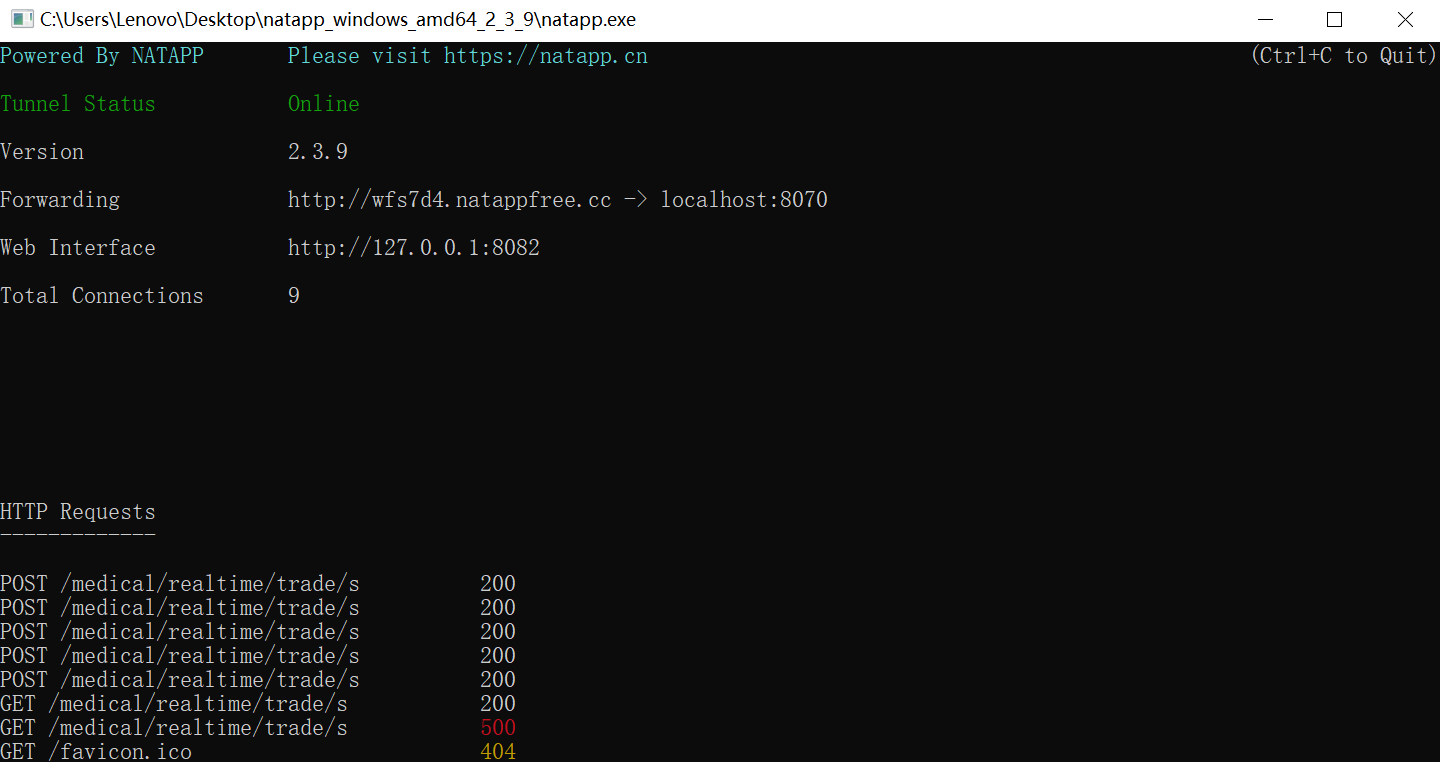
clienttoken= #对应客户端的clienttoken,将会忽略authtoken,若无请留空,

log=none #log 日志文件,可指定本地文件, none=不做记录,stdout=直接屏幕输出 ,默认为none

loglevel=ERROR #日志等级 DEBUG, INFO, WARNING, ERROR 默认为 DEBUG

http\_proxy= #代理设置 如 http://10.123.10.10:3128 非代理上网用户请务必留空

**9）保存退出，双击natapp.exe即可启动隧道，如下。**



# 第3章 数仓开发之ADS层

在Sugar可视化大屏中，一张图表对应一个接口，开发流程类似，本节只选取一张图表作介绍。

## 3.1 交易主题

### 3.1.1 交易综合统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | -- | 问诊金额 | 略。 |
| 当日 | -- | 问诊次数 | 略。 |
| 当日 | -- | 问诊支付成功金额 | 略。 |
| 当日 | -- | 问诊支付成功次数 | 略。 |
| 当日 | -- | 处方开单金额 | 略。 |
| 当日 | -- | 处方开单次数 | 略。 |
| 当日 | -- | 处方开单支付成功金额 | 略。 |
| 当日 | -- | 处方开单支付成功次数 | 略。 |

### 3.1.2 各医院交易统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | 医院 | 问诊金额 | 略。 |
| 当日 | 医院 | 问诊次数 | 略。 |
| 当日 | 医院 | 问诊支付成功金额 | 略。 |
| 当日 | 医院 | 问诊支付成功次数 | 略。 |
| 当日 | 医院 | 处方开单金额 | 略。 |
| 当日 | 医院 | 处方开单次数 | 略。 |
| 当日 | 医院 | 处方开单支付成功金额 | 略。 |
| 当日 | 医院 | 处方开单支付成功次数 | 略。 |

### 3.1.3 各性别患者交易统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | 患者性别 | 问诊金额 | 略。 |
| 当日 | 患者性别 | 问诊次数 | 略。 |
| 当日 | 患者性别 | 问诊支付成功金额 | 略。 |
| 当日 | 患者性别 | 问诊支付成功次数 | 略。 |
| 当日 | 患者性别 | 处方开单金额 | 略。 |
| 当日 | 患者性别 | 处方开单次数 | 略。 |
| 当日 | 患者性别 | 处方开单支付成功金额 | 略。 |
| 当日 | 患者性别 | 处方开单支付成功次数 | 略。 |

### 3.1.4 各年龄段患者交易统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | 患者年龄段 | 问诊金额 | 略。 |
| 当日 | 患者年龄段 | 问诊次数 | 略。 |
| 当日 | 患者年龄段 | 问诊支付成功金额 | 略。 |
| 当日 | 患者年龄段 | 问诊支付成功次数 | 略。 |
| 当日 | 患者年龄段 | 处方开单金额 | 略。 |
| 当日 | 患者年龄段 | 处方开单次数 | 略。 |
| 当日 | 患者年龄段 | 处方开单支付成功金额 | 略。 |
| 当日 | 患者年龄段 | 处方开单支付成功次数 | 略。 |

## 3.2 医生主题

### 3.2.1 医生变动统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | -- | 新增医生数 | 略。 |

## 3.3 用户主题

### 3.3.1 用户变动统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | -- | 新增用户数 | 略。 |
| 当日 | -- | 新增患者数 | 略。 |

## 3.4 评价主题

### 3.4.1 各医院评价统计

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 历史至今 | 医院 | 评价人数 | 略。 |
| 历史至今 | 医院 | 评价次数 | 略。 |
| 历史至今 | 医院 | 好评率 | 好评次数/评级次数 |

## 3.5 实现

**1）需求**

**此处选取交易主题部分指标展示，如下。**

|  |  |  |  |
| --- | --- | --- | --- |
| 统计周期 | 统计粒度 | 指标 | 说明 |
| 当日 | 医院 | 问诊金额 | 略。 |
| 当日 | 医院 | 问诊支付成功金额 | 略。 |

**2）图表选取**

此处选择轮播表格。

**3）数据结构**

图表所需数据结构的获取详见本节 **7）Sugar 配置**。

{

"status": 0,

"msg": "",

"data": {

"columns": [

{

"name": "分组织",

"id": "subsys"

},

{

"name": "部门",

"id": "module"

},

{

"name": "业务组",

"id": "business"

},

{

"name": "姓名",

"id": "name"

},

{

"name": "销售额",

"id": "value",

"accuracy": 2

}

],

"rows": [

{

"subsys": "北京总部",

"module": "市场部",

"business": "第一小分队",

"name": "张三",

"value": 6672389,

"url": "http://www.baidu.com"

},

...

]

}

}

（1）columns

轮播表格的字段。

（2）rows

轮播表格的行。

**4）Mapper 层**

（1）实体类

① TradeHosConsulAmountStatsBean

package com.atguigu.medical.realtime.publisher.bean;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import java.math.BigDecimal;

@Data

@AllArgsConstructor

@Builder

public class TradeHosConsulAmountStatsBean {

// 医院 ID

String hospitalId;

// 医院名称

String hospitalName;

// 问诊金额

BigDecimal consultationAmount;

// 问诊支付成功金额

BigDecimal consultationPaySucAmount;

}

（2）Mapper 接口

package com.atguigu.medical.realtime.publisher.mapper;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulCountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresCountStatsBean;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.Select;

import java.util.List;

public interface TradeStatsMapper {

@Select("select t1.hospital\_id,\n" +

" t1.hospital\_name,\n" +

" consultation\_amount,\n" +

" consultation\_pay\_suc\_amount\n" +

" from\n" +

"(select hospital\_id,\n" +

" hospital\_name,\n" +

" sum(consultation\_amount) consultation\_amount\n" +

"from dws\_trade\_hosp\_pat\_gen\_a\_g\_consul\_win\n" +

"where cur\_date = #{date}\n" +

"group by hospital\_id, hospital\_name) t1\n" +

"left join\n" +

"(select hospital\_id,\n" +

" hospital\_name,\n" +

" sum(consultation\_pay\_suc\_amount) consultation\_pay\_suc\_amount\n" +

"from dws\_trade\_hosp\_pat\_gen\_a\_g\_consul\_pay\_suc\_win\n" +

"where cur\_date = #{date}\n" +

"group by hospital\_id, hospital\_name) t2\n" +

"on t1.hospital\_id = t2.hospital\_id\n" +

"order by consultation\_pay\_suc\_amount desc,\n" +

" consultation\_amount desc\n" +

"limit 10;")

List<TradeHosConsulAmountStatsBean> selectHosConsulAmountStats(@Param("date") String date);

}

（3）注解

① @Select 注解

位于方法定义语句上方，修饰方法。

添加该注解后， Mybatis 会自动实现 JDBC 的环境准备，实现对应的 Mapper 层接口，并在 Tomcat 容器启动时将该实现类加载到容器中，包含 @Service 注解的功能。可以在服务层的 Impl 类中通过 @Autowired 自动装载该实现类。该实现类是单例的。

注意：要将该类加载到容器中，需要在启动类上方添加 @MapperScan 注解，指明 mapper 包路径，如下。

package com.atguigu.medical.realtime.publisher;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**@MapperScan(basePackages = "com.atguigu.medical.realtime.publisher.mapper")**

public class MedicalRealtimePublisherApplication {

public static void main(String[] args) {

SpringApplication.run(MedicalRealtimePublisherApplication.class, args);

}

}

② @Param 注解

位于方法参数数据类型之前，修饰参数。

该注解可以将方法参数接收的值赋给注解参数中的变量，该变量可以在方法上方 @Select 注解的 SQL 语句中使用，调用方式为 #{变量名}。

如 @Param("paramDate")Integer date，将参数 date 接收到的数据赋值给 paramDate，可以在 SQL 中通过 #{paramDate} 获取 paramDate 参数的值。

**5）Service 层**

（1）service 接口

package com.atguigu.medical.realtime.publisher.service;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulCountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresCountStatsBean;

import java.util.List;

public interface TradeStatsService {

List<TradeHosConsulAmountStatsBean> getHosConsulAmountStats(String date);

}

（2）service 实现类

package com.atguigu.medical.realtime.publisher.service.impl;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulCountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresCountStatsBean;

import com.atguigu.medical.realtime.publisher.mapper.TradeStatsMapper;

import com.atguigu.medical.realtime.publisher.service.TradeStatsService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class TradeStatsServiceImpl implements TradeStatsService {

**// 自动装载 Mapper 接口实现类**

@Autowired

private TradeStatsMapper tradeStatsMapper;

@Override

public List<TradeHosConsulAmountStatsBean> getHosConsulAmountStats(String date) {

return tradeStatsMapper.selectHosConsulAmountStats(date);

}

}

（3）注解

① @Service 注解

位于实现类定义语句上方，修饰类。

把该实现类注册成一个组件，这个组件常驻内存，而且默认为单态（即单例，整个服务器进程，该类对象只有一个）。启动时会扫描所有被 @Service 注解修饰的实现类，将它们加载到内存。

② @Autowired 注解

位于成员变量（属性）定义语句上方，修饰类的属性。

① 控制器组件会扫描接口名上方添加了 @Autowired（自动装配）注解的接口，然后去内存的常驻组件中寻找适配组件（实现类），令此处的引用指向适配组件。

② 如果一个接口有多个实现类注册成为了组件，可以通过 @ Qualifier("组件名称") 注解指定不同的适配组件。此处的组件名称在 @Service 注解的参数中指定，如 @Service("组件名称")，只有一个实现类时可以不起名（不传参），直接使用 @Service 注解。

**6）controller 层**

（1）代码

package com.atguigu.medical.realtime.publisher.controller;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosConsulCountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresAmountStatsBean;

import com.atguigu.medical.realtime.publisher.bean.TradeHosPresCountStatsBean;

import com.atguigu.medical.realtime.publisher.service.TradeStatsService;

import com.atguigu.medical.realtime.publisher.util.DateUtil;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController

@RequestMapping("/medical/realtime/trade/stats")

public class TradeStatsConstroller {

**// 自动装载渠道流量统计服务实现类**

@Autowired

private TradeStatsService tradeStatsService;

**// 请求拦截方法**

@RequestMapping("/hospital/consul/amount/stats")

public String getHosConsulAmountStats(@RequestParam(value = "date", defaultValue = "1") String date) {

if (date.equals("1")) {

date = DateUtil.now();

}

List<TradeHosConsulAmountStatsBean> hosConsulAmountStats = tradeStatsService.getHosConsulAmountStats(date);

StringBuilder rowBuilder = new StringBuilder();

String tmp = null;

for (int i = 0; i < hosConsulAmountStats.size(); i++) {

TradeHosConsulAmountStatsBean hosConsulAmountStat = hosConsulAmountStats.get(i);

tmp = " {\n" +

" \"hospital\_id\": \"" + hosConsulAmountStat.getHospitalId() + "\",\n" +

" \"hospital\_name\": \"" + hosConsulAmountStat.getHospitalName() + "\",\n" +

" \"consultation\_amount\": " + hosConsulAmountStat.getConsultationAmount() + ",\n" +

" \"consultation\_pay\_suc\_amount\": " + hosConsulAmountStat.getConsultationPaySucAmount() + "\n" +

" }";

rowBuilder.append(tmp);

if (i < hosConsulAmountStats.size() - 1) {

rowBuilder.append(",");

}

rowBuilder.append("\n");

}

String rows = rowBuilder.toString();

return "{\n" +

" \"status\": 0,\n" +

" \"msg\": \"\",\n" +

" \"data\": {\n" +

" \"columns\": [\n" +

" {\n" +

" \"name\": \"医院ID\",\n" +

" \"id\": \"hospital\_id\"\n" +

" },\n" +

" {\n" +

" \"name\": \"医院名称\",\n" +

" \"id\": \"hospital\_name\"\n" +

" },\n" +

" {\n" +

" \"name\": \"问诊金额\",\n" +

" \"id\": \"consultation\_amount\"\n" +

" },\n" +

" {\n" +

" \"name\": \"问诊支付成功金额\",\n" +

" \"id\": \"consultation\_pay\_suc\_amount\"\n" +

" }\n" +

" ],\n" +

" \"rows\": [\n" +

rows +

" ]\n" +

" }\n" +

"}";

}

}

（2）注解

① @RestController

位于控制类定义语句上方，修饰控制类。

标注方法为控制类，可以将外部请求引入该类的方法中。当方法返回值类型为 String 时，不会发生页面跳转，相当于 @Controller + @ResponseBody。

② @RequestMapping

可以在控制类定义语句上方添加，也可以在方法定义语句上方添加。

示例：@RequestMapping(value = "/traffic", method = RequestMethod.GET)。

该注解用于建立请求URL和请求处理方法之间的对应关系。value 用于指定请求 URL。请求处理方法最终拦截的 URL为类注解指定的路径与方法注解指定的路径拼接所得的 URL。如当前 SpringBoot 服务占用了本机的 8080 端口，类上注解为 @RequestMapping("/gmall ")，方法上方注解为 @RequestMapping("/uvCt")，则最终拦截的 url 为 localhost:8080/gmall/uvCt。控制类上方可以没有该注解。

如果拦截的是 GET 请求，method 参数赋值语句可以省略。此外，SpringBoot 的注解只有一个参数，且参数名称为 value时，可以省略参数名称和等号。

所以，@RequestMapping(value = "/traffic", method = RequestMethod.GET) 可简写为 @RequestMapping("/traffic")。

该注解相当于 @GetMapping("/traffic")

③ @RequestParma("name")

位于方法参数数据类型之前。接收请求路径中键值对(名值对)的参数(通常用于查询条件、辅助参数) 并传递给所修饰的参数。

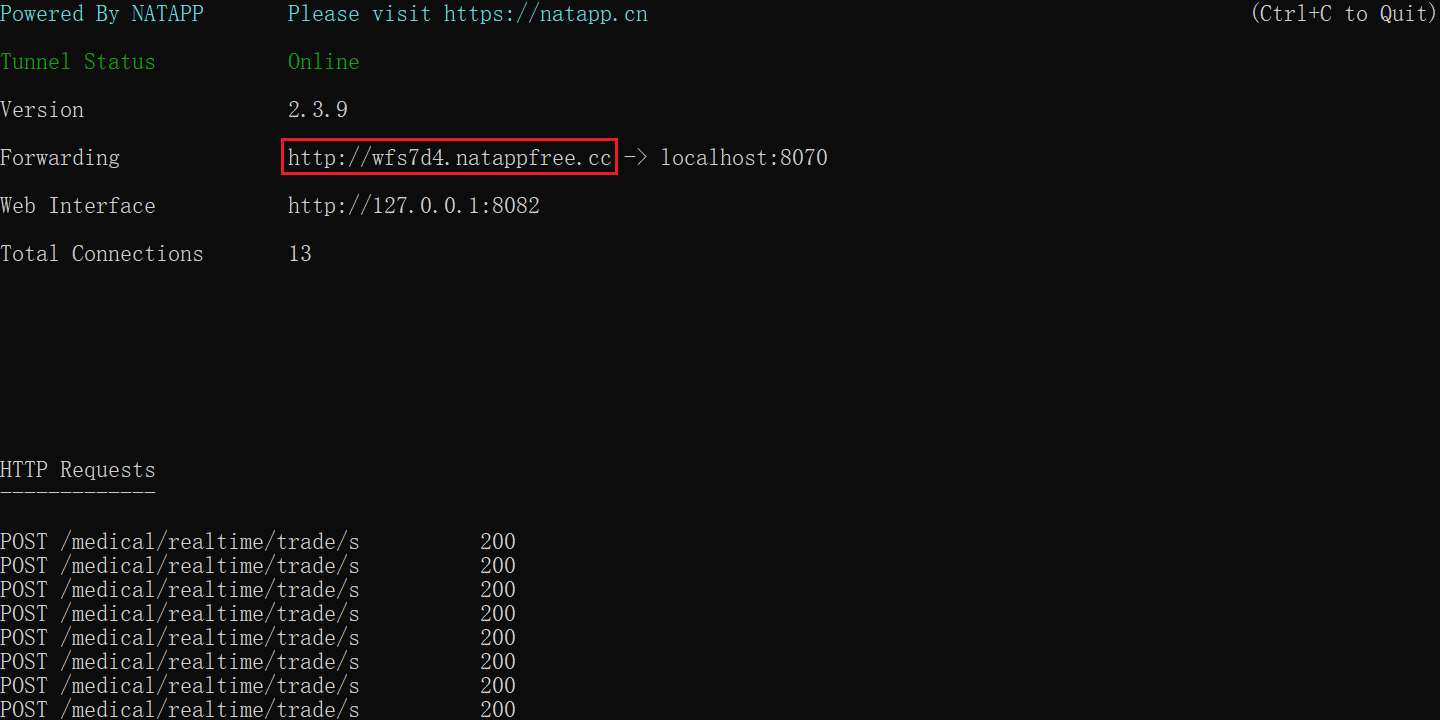
如 URL 为 localhost:8080/test?paramDt=20220221，注解为 @RequestParam("paramDt") Integer dt，则 20220221 会被解析为 Integer 类型数据赋值给 dt 参数。

**7）Sugar 配置**

（1）页面宏定义变量

单击大屏空白处，在右侧“控制面板”中点击“页面宏定义变量”，定义 MEDICAL\_HOST 和 MEDICAL\_DATE。

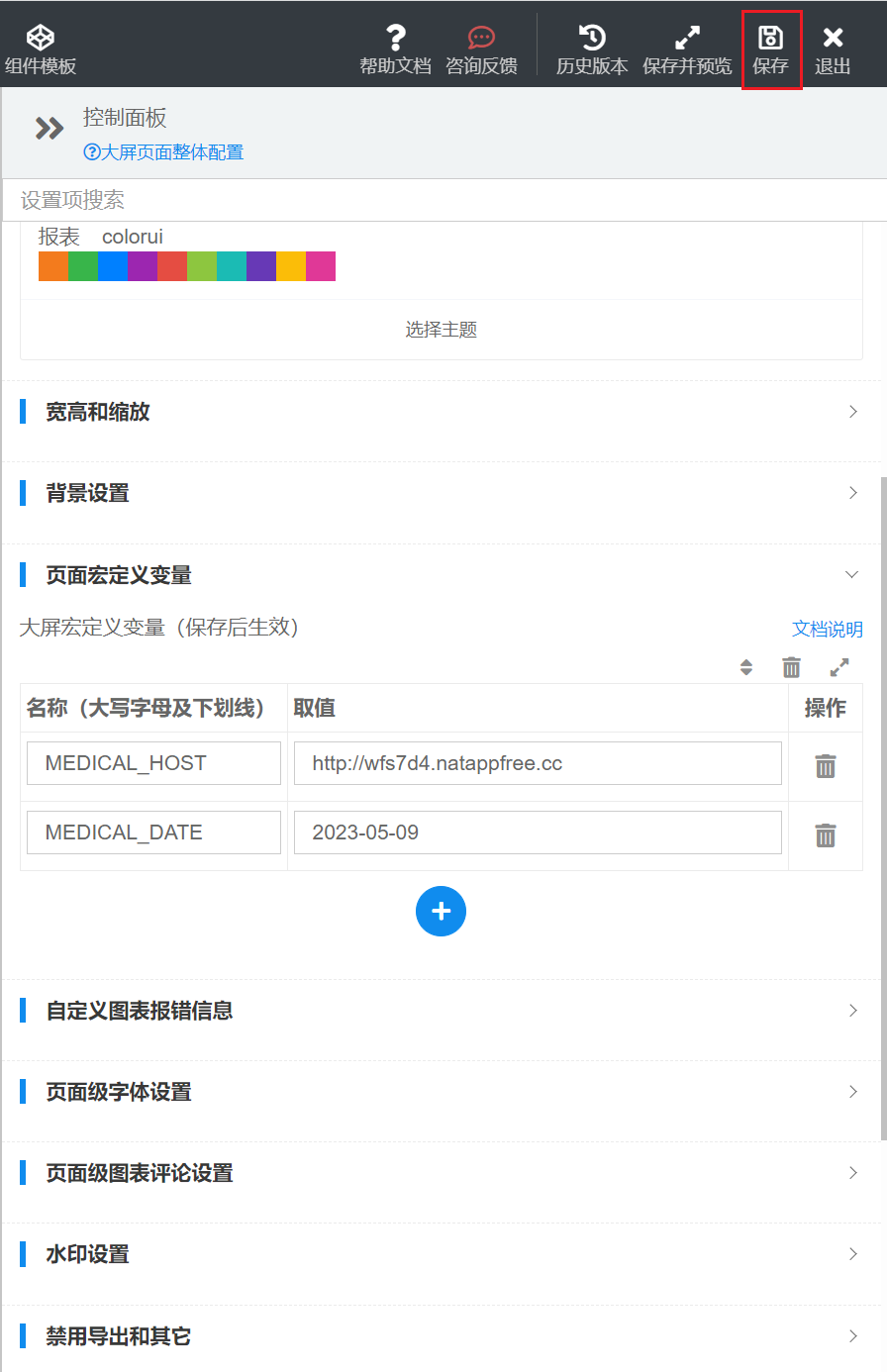
* + MEDICAL\_HOST：本地 8070 端口服务映射的公网域名。在natapp启动后弹出的命令行客户端获取，如下。



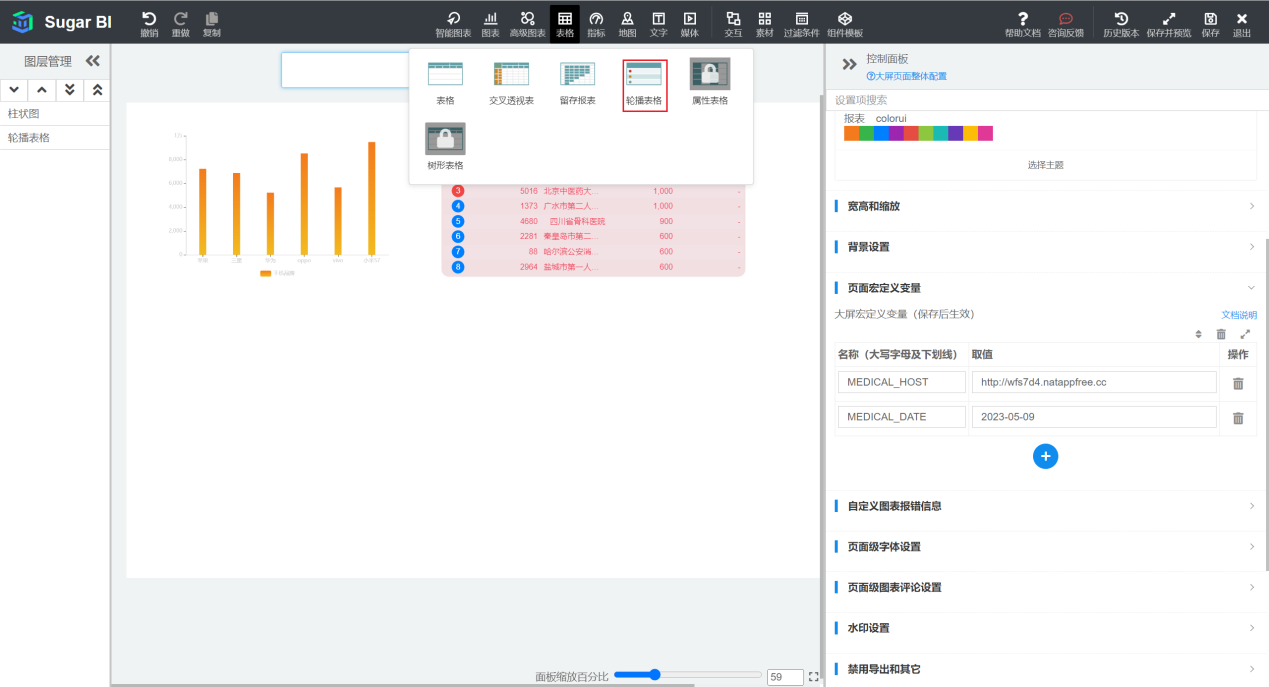
* + MEDICAL\_DATE：当天日期。与数据模拟日期保持一致。



保存生效。



（2）在“图表”中选择“轮播表格”

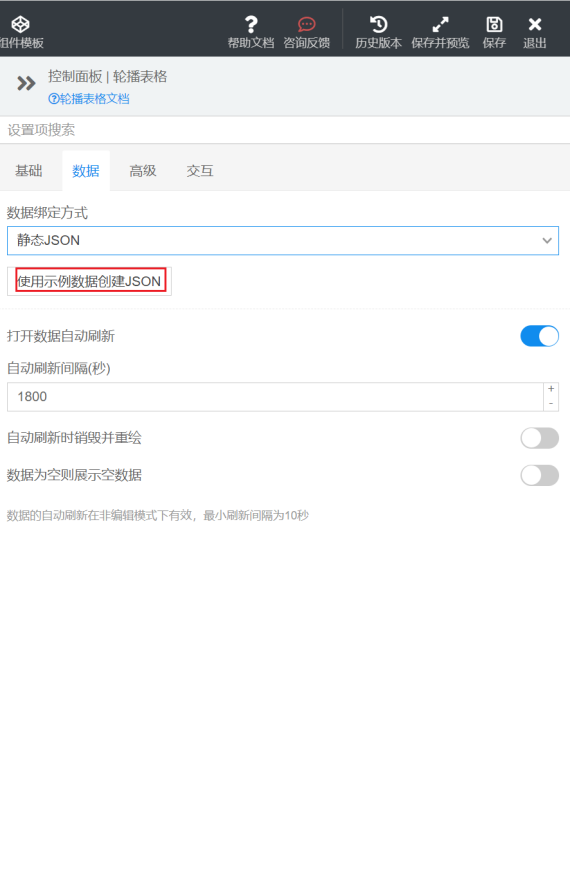


（3）在弹出的控制面板中选择 API 拉取

* + API 拉取：通过给定的数据接口获取数据。我们选择这种方式。
  + 静态 JSON：通过给定的静态 JSON 字符串获取数据。

API 拉取返回的数据格式与静态 JSON 相同，因此可以通过静态 JSON 的示例数据查看数据格式。





（4）输入数据接口的 URL

${MEDICAL\_HOST}/medical/realtime/trade/stats/hospital/consul/amount/stats?date=${MEDICAL\_DATE}

（5）刷新图表，查看效果

