**第12章-数据中心-时序数据库**

**学习目标**

* 了解并能阐述influxDB基本结构和语法操作
* 使用XXL-JOB完成数据统计报表展示

**1、InfluxDB**

**1.1、简介**

**是什么**

**InfluxDB是**一个开源的**时间序列数据库**，特别为处理和分析带有**时间戳的数据**而设计。它由InfluxData公司开发，并使用Go语言编写，旨在提供高性能的数据写入与查询能力。

**应用场景**

* **系统监控**：收集和分析服务器、应用程序的性能指标，如CPU使用率、内存占用、网络流量等。
* **物联网(IoT)**：处理来自传感器的大量实时数据，用于环境监测、智能城市、工业自动化等领域。
* **应用程序指标**：跟踪用户行为、交易量、错误日志等，帮助优化应用性能和用户体验。
* **实时分析**：在金融、电商等行业中，用于实时交易分析、市场趋势预测等。

**特征**

* **时间序列数据**：天然支持时间戳数据的存储与查询，适用于**随时间变化的数据**。
* **高性能**：TSM存储引擎提供高速的数据写入与压缩，优化了时序数据的存储和查询效率。
* **Schemaless**（无结构）：允许灵活的数据模式，每条记录可以有任意数量的列，无需预先定义表结构。
* **Retention Policies**：自动管理数据生命周期，可设定数据**保留策略**，自动删除过期数据。
* **Continuous Queries**：自动执行的聚合查询，定期计算并存储汇总数据，提高查询效率。
* **API支持**：提供HTTP API，支持方便的数据写入和查询，兼容多种数据输入协议，如Graphite、CollectD、OpenTSDB等。

总的来说；非常适合对**实时大量数据进行计算**。

**版本**

InfluxDB 主要有1.x 和 2.x 两个版本；目前企业里面用 InfluxDB 1.X 和 InfluxDB 2.X 都有人在用，数量上 InfluxDB1.X 占多一些。

2020 年 InfluxDB 推出了 2.0 的正式版。2.x 同 1.x 相比，底层引擎原理相差不大；

* 在操作方面：在 2.x 仅需安装 InfluxDB 就能得到一个管理页面，而且附带了定时任务和报警功能。
* 在查询语言方面：1.x 是使用 InfluxQL 进行查询，它的风格近似 SQL。2.x 推出了 FLUX 查询语言，可以使用函数与管道符，是一种更符合时序数据特性的更具表现力的查询语言。

**1.2、相关概念**

InfluxDB的数据模型：

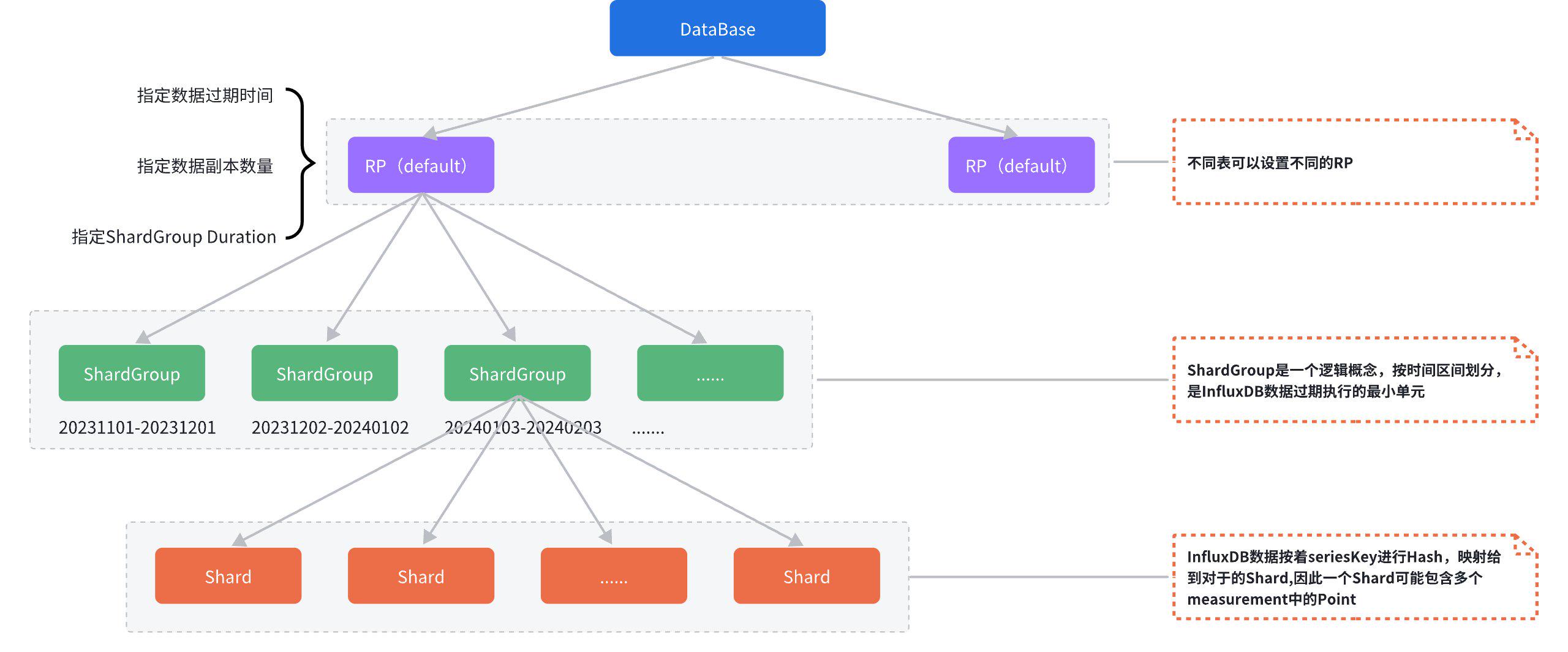


* **Measurement:** 主要用来存储具有相似特征或属于同一类别的数据点集合，类似于关系型数据库中的表（table）；包含了列**Timestamp时间戳，field字段和tag标签**。
* **Field set**: 每组field key和field value的集合，即我们需要的字段，如internale[key] = 76[value], external[key]= 18[value]。不可索引
* **Tag set**: 不同的每组tag key和tag value的集合，如device[key]= dev1[value], buiding[key]= b1[value]。可索引
* **Point：**表里面的一行数据,由时间戳（timestamp）、字段（field）和标签（tag）组成：
* 时间戳time：每条数据记录的时间，也是数据库自动生成的主索引，如果时间戳没有指定。那么InfluxDB就使用当前系统的时间戳（纳秒）
* 字段field：字段包含数据的实际值，可以是各种数据类型（整数、浮点数、字符串、布尔值等）；与标签不同，字段在查询时可以进行数学运算。
* 标签tag：用于索引和过滤数据；通常是字符串类型。
* **series**：series是共同retention policy（数据保留策略），Measurement 和 tag set的集合。

|  |
| --- |
| 在 InfluxDB 表结构中，Field 和 Tag 是用于存储数据的两种不同类型。   1. **Field（字段）**：  * Field 用于存储实际的数值数据，例如温度、湿度等测量值。 * Field 是可变的，可以随时间的推移而改变其值。 * Field 的值可以进行聚合计算，例如求平均值或总和等。 * Field 不适合用于过滤和索引数据，因为它没有元数据信息。  1. **Tag（标签）**：  * Tag 用于存储元数据信息和标识数据的键值对，例如传感器名称、地理位置等。 * Tag 是不可变的，一旦设置就不能更改。 * Tag 的值不能进行聚合计算，只能用于过滤和分组查询。 * Tag 是 InfluxDB 中的主要索引机制之一，可以提高查询性能和数据过滤效率。 |

**1.3、存储架构**

在 InfluxDB 中可以创建多个数据库，不同数据库中的数据文件是隔离存放的，存放在磁盘上的不同目录,每个database 可以有多个RP(retention policy数据保存策略)，但是只有一个默认策略。策略下按照时间段分为多个ShardGroup分片组,每个ShardGroup存储一个时间段的数据。每个shardgroup下分多个shard来存储数据。如下图所示：



* **retention policy：数据保存策略**，用于设置数据保留的时间，每个数据库刚开始会自动创建一个默认的存储策略 autogen，数据保留时间为永久，之后用户可以自己设置，例如保留最近2小时的数据。插入和查询数据时如果不指定存储策略，则使用默认存储策略，且默认存储策略可以修改。InfluxDB 会定期清除过期的数据。
* **ShardGroup：**是一个逻辑概念，按时间区间划分，是InfluxDB数据过期执行的最小单元
* **Shard：**在 InfluxDB 中是一个比较重要的概念，它和 retention policy 相关联。每一个存储策略下会存在许多 shard，每一个 shard 存储一个指定时间段内的数据，并且不重复，例如 7点-8点 的数据落入 shard0 中，8点-9点的数据则落入 shard1 中。每一个 shard 都对应一个底层的 tsm存储引擎。

**1.4、安装**

《四方保险》项目中的InfluxDB的环境是基于docker-compose来进行安装；参考 [第0章-环境搭建](https://j1wtmv7ajj.feishu.cn/wiki/FG5AwAXHCivrwskoxTIcQcSfn4e?from=from_copylink)

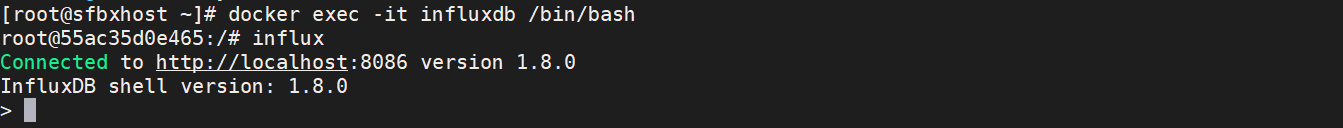
**2、InfluxDB 基础**

**2.1、数据库操作**

**连接InfluxDB**

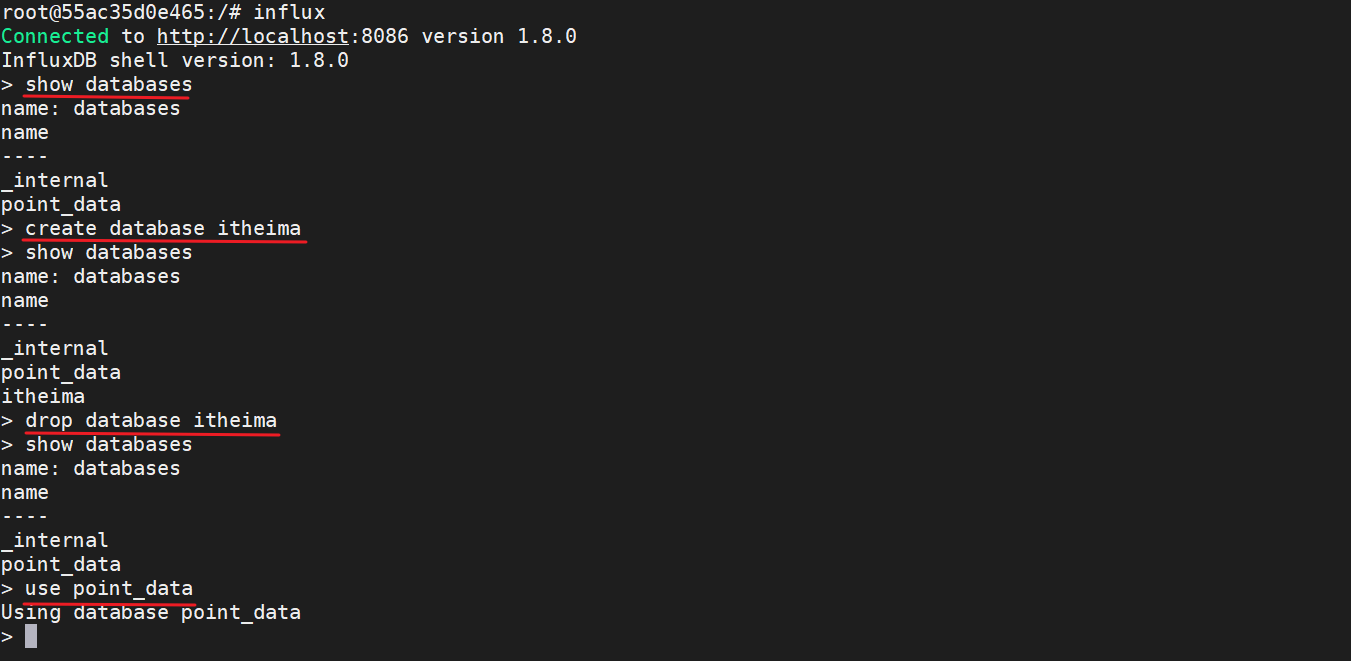
进入InfluxDB的命令行终端，再连接InfluxDB

|  |
| --- |
| Shell # 进入InfluxDB的命令行终端 docker exec -it influxdb /bin/bash  # 连接InfluxDB influx |



**数据库操作**

|  |
| --- |
| Shell # 显示数据库 show databases  # 创建数据库 create database itheima  # 删除数据库 drop database itheima  # 使用数据库 use itheima |

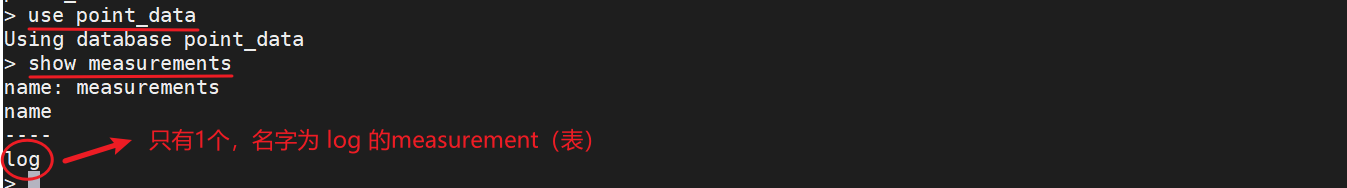


**2.2、数据表操作**

**1）显示所有表**

在InfluxDB当中，并没有表（table）这个概念，取而代之的是MEASUREMENTS，MEASUREMENTS的功能与传统数据库中的表一致，因此我们也可以将MEASUREMENTS称为InfluxDB中的表。

|  |
| --- |
| Shell # 显示所有的 measurement show measurements |

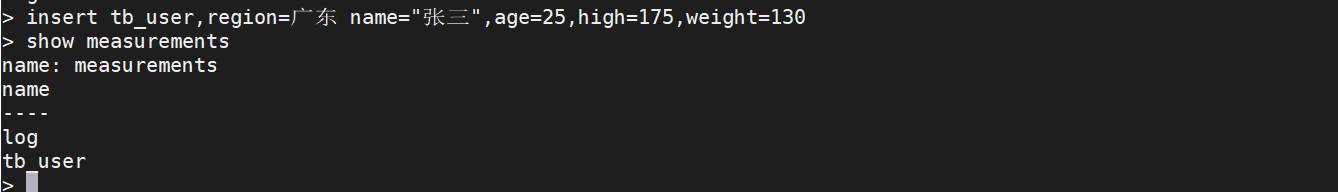


**2）新建表**

InfluxDB中没有显式的新建表的语句，只能通过insert数据的方式来建立新表。语法如下：

|  |
| --- |
| SQL insert measurement+","+tag1=value1,tag2=value2 + 空格 + field1=value1,field2=values2  -- 例如：对measurement为tb\_user的插入数据；有一个tag索引名为region，值为广东；有三个field分别是age,high,weight 对应的值分别为25、175、130 insert tb\_user,region=广东 name="张三",age=25,high=175,weight=130 |

* measurement与 tag 之间使用逗号分隔
* 如果有多个tag，那么tag之间使用逗号分隔
* tag 与 field 之间使用空格分隔
* tag的值都是string类型，value不需要引号包裹
* field如果是string类型的值，需要使用引号包裹



|  |
| --- |
| 也可以预先查询一下数据；语法与关系型类似  select \* from tb\_user |



**3）删除表**

|  |
| --- |
| SQL -- 删除语法 drop measurement 表名  -- 例如：删除名为 tb\_user 的measurement drop measurement tb\_user |

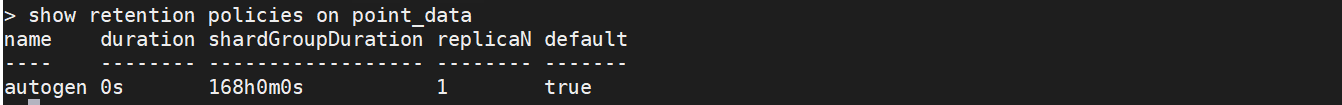


**2.3、数据保存策略**

InfluxDB是一般不建议**直接删除数据记录**的；其提供**数据保存策略**，主要用于指定数据保留时间，超过指定时间，就删除这部分数据。

**1）查看保存策略**

|  |
| --- |
| SQL show retention policies on 数据库名称  -- 例如：查看 point\_data 数据库的保存策略 show retention policies on point\_data |



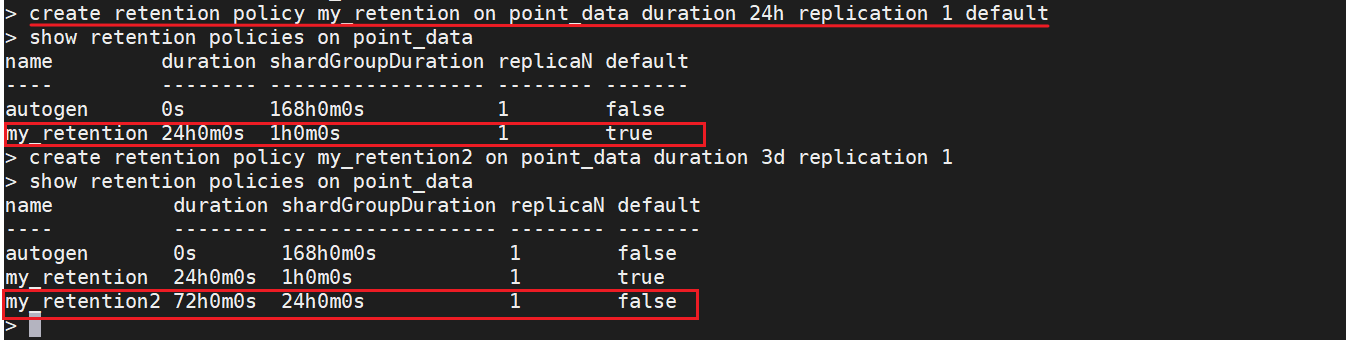
上述的执行结果说明：

1. **name**：这是保留策略的名称。每个数据库可以有多个保留策略，autogen 是默认的保留策略名称，如果你没有为特定的数据点指定保留策略，它将自动使用 autogen。用户也可以自定义其他名称的保留策略。
2. **duration**：表示数据保留的时间长度。这个值决定了数据在数据库中最多能保存多久。时间单位通常是小时(h)、分钟(m)、秒(s)，如 72h 表示数据将被保留72小时。如果设置为 0s，则表示数据保留期无限制。
3. **shardGroupDuration**：分片组的持续时间。InfluxDB使用分片组来组织和存储数据，每个分片组覆盖一个特定的时间范围，这个值定义了每个分片组的时间跨度。查询效率和资源管理在很大程度上取决于这个参数的设定，因为它**影响着数据**如何被索引和**分布(不影响数据的删除与否)**。时间单位同样使用小时、分钟、秒。168h 表示 168小时也就是一周
4. **replicaN**：副本数量。这表示数据在集群中的复制份数。例如，如果设置为 1，则每份数据只有一份副本（数据只存在一份，没有其它）；如果是 2，则每份数据会有两个副本，以此类推。增加副本数量可以提高数据的持久性和可用性，但也会增加存储需求。
5. **default**：标记此保留策略是否为数据库的默认策略。如果此列为 true，则表示当写入数据时未明确指定保留策略，则会使用这个作为默认策略。如果为 false，则表明这是一个非默认策略，需要在写入数据时显式指定才会应用。

|  |
| --- |
| **数据删除**：  1.x 版本中和在关系型中一样使用删除语句；delete from measurement的名称 就删除了整张表数据了，也可以添加条件 delete from measurement的名称 where region='广东' |

**2）创建保存策略**

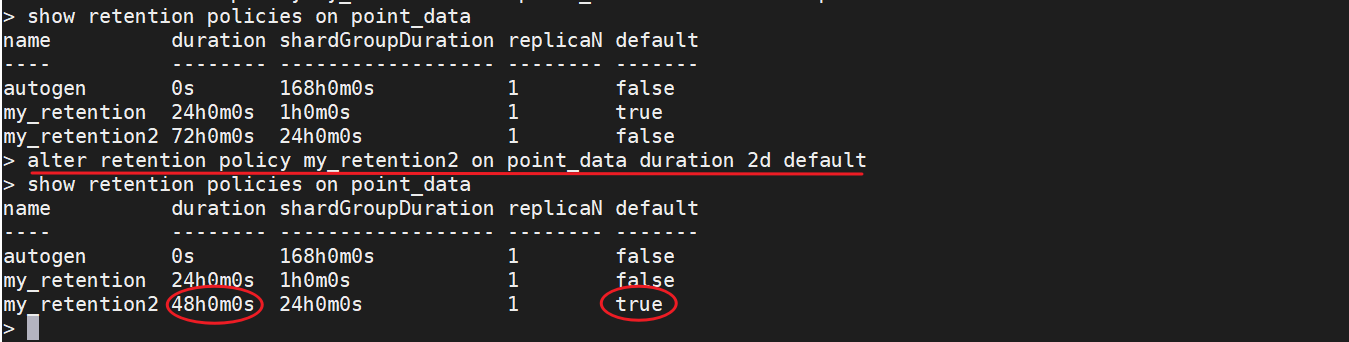
|  |
| --- |
| SQL -- 语法 create retention policy 策略名 on 数据库名 duration 保留时长 replication 副本个数 [default]  -- 示例：创建point\_data数据库的默认保存策略名字为 my\_retention ，保留时长为24小时，副本数1个 create retention policy my\_retention on point\_data duration 24h replication 1 default  -- 示例：同样的，但是保存时长设置为3天，但是不设置为默认的保存策略的话就不加default create retention policy my\_retention2 on point\_data duration 3d replication 1 |



|  |
| --- |
| 保留时长 **24h**：保存24小时，24小时之前的数据将被删除，influxdb具有各种时长单位，比如：h（小时），d（天），w（星期）；  **replication** 1：副本个数，一般为1就可以了；  **default**：设置为默认策略；不默认不加这个关键字  上面查出来的duration和shardGroupDuration不同，数据的保留时长只与duration有关，shardGroupDuration只是说数据分布在哪些时间存储组上（并不是到这个时间就删除） |

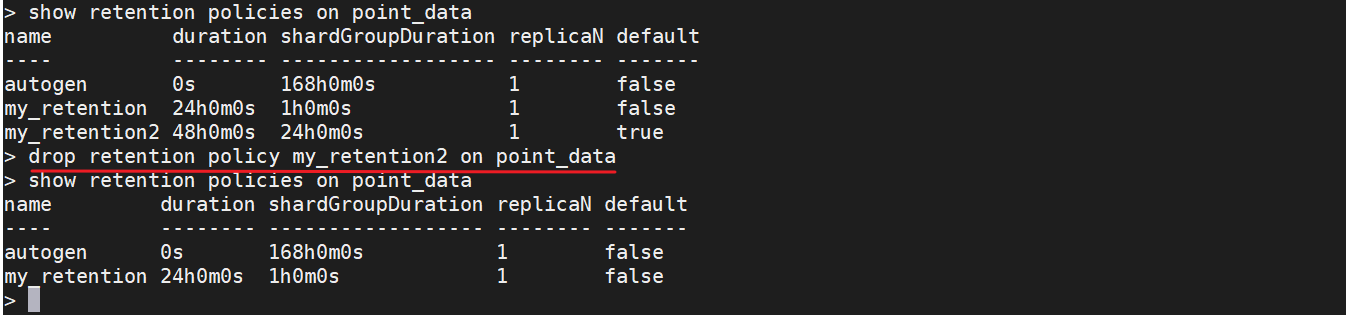
**3）修改保存策略**

|  |
| --- |
| SQL -- 语法 alter retention policy 策略名 on 数据库名 duration 时长 default（可选）  -- 例如：修改point\_data数据库中的my\_retention2策略，保留时长为2天，并设置为默认 alter retention policy my\_retention2 on point\_data duration 2d default |



**4）删除保存策略**

|  |
| --- |
| SQL drop retention policy 策略名 on 数据库名  -- 例如：删除point\_data数据库中策略名为 my\_retention2 的策略 drop retention policy my\_retention2 on point\_data  --- 删除保存策略如果是默认的；则不会自动的指定一个策略为默认；不过可以修改 alter retention policy autogen on point\_data default |



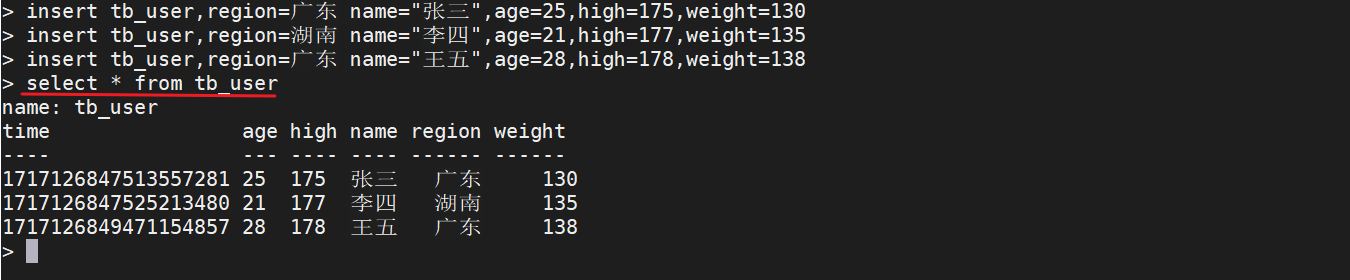
**2.4、数据查询**

InfluxDB基本查询操作和MySQL的基本查询是类似，综合使用如下所示：

|  |
| --- |
| SQL #----综合使用 书写顺序 select distinct \* from '表名' where '限制条件' group by '分组依据' having '过滤条件' order by limit '展示条数' 执行顺序 from -- 查询 where -- 限制条件 **使用单引号，否则无数据返回或报错** group by -- 分组 只能对tags和time进行分组 having -- 过滤条件--》对分组内的数据 order by -- 排序 只能对time进行排序 limit -- 展示条数 distinct -- 去重 select -- 查询的结果 |

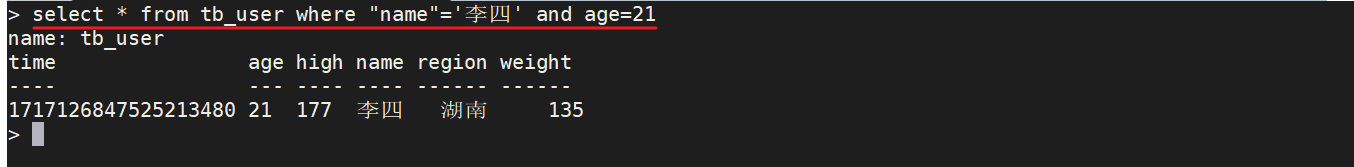
**1）查询全部**

|  |
| --- |
| SQL -- 插入数据 insert tb\_user,region=广东 name="张三",age=25,high=175,weight=130 insert tb\_user,region=湖南 name="李四",age=21,high=177,weight=135 insert tb\_user,region=广东 name="王五",age=28,high=178,weight=138  -- 查询数据 select \* from tb\_user |



**2）条件查询**

|  |
| --- |
| SQL -- 查询名字为 李四，年龄为21 的用户 select \* from tb\_user where "name"='李四' and age=21 |

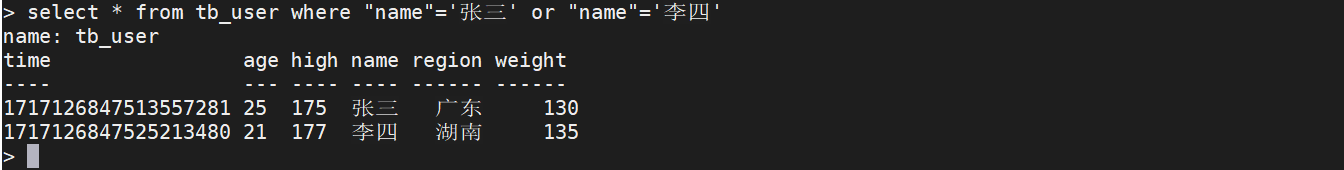


|  |
| --- |
| 上述查询中为何对name添加 双引号；因为name是influxdb的保留关键字，所以添加。一般字段可以不用 |

**3）or查询**

influxDB中没有in的操作，但是有or。

|  |
| --- |
| SQL -- 查询姓名为张三或李四 select \* from tb\_user where "name"='张三' or "name"='李四' |

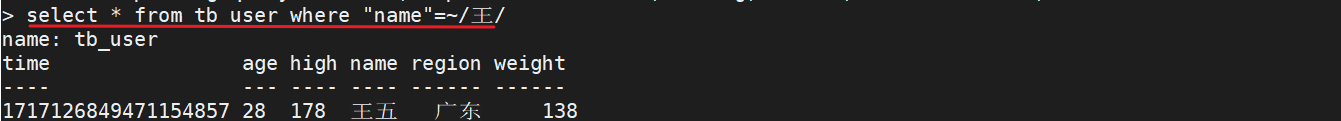


**4）模糊查询**

模糊查询支持正则表达式方式，例如

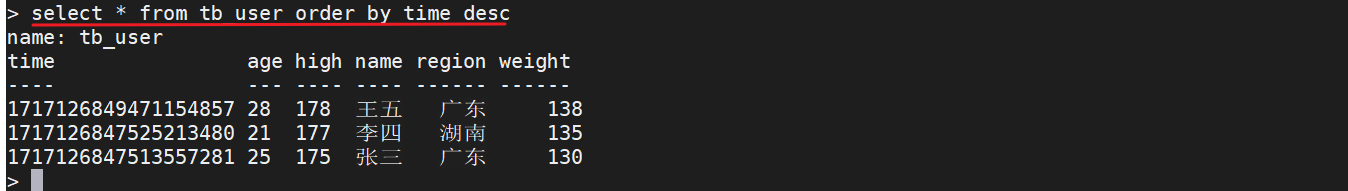
* =~/关键字/ 包含关键字的
* =~/^关键字/ 以关键字开始的
* =~/关键字$/ 以关键字结尾的

|  |
| --- |
| SQL -- 查询名字中包含王的 select \* from tb\_user where "name"=~/王/ |



**5）排序查询**

|  |
| --- |
| SQL -- 只能根据时间排序；根据创建时间降序排序 select \* from tb\_user order by time desc |



**6）去重**

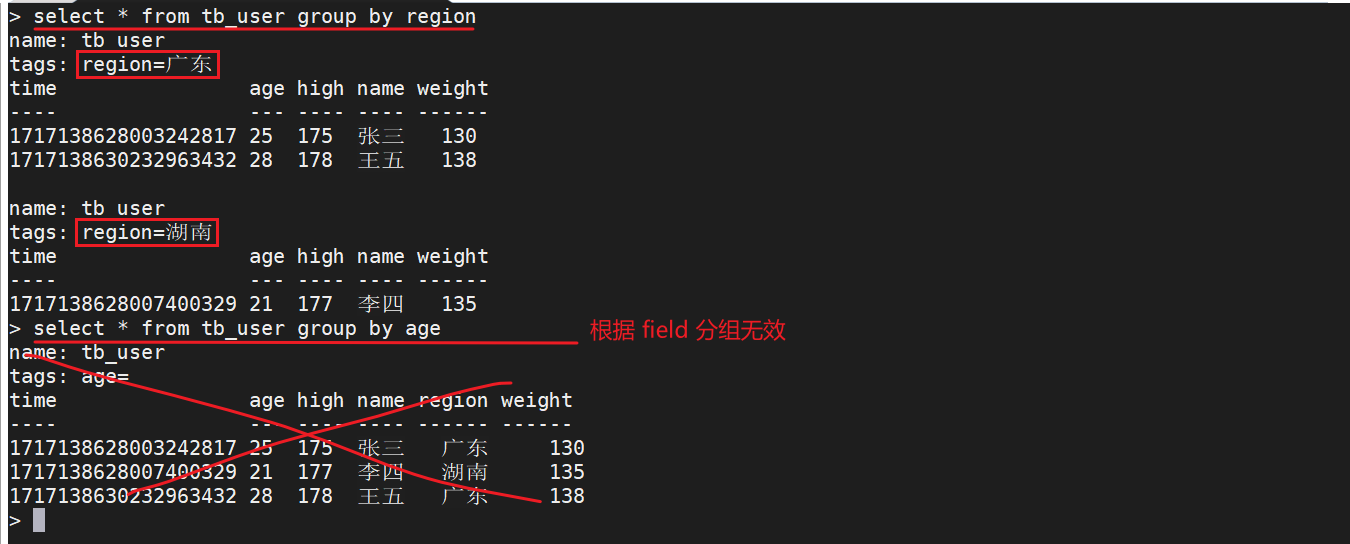
|  |
| --- |
| SQL -- 对age字段去重查询；注意：只能在distinct 之后接一个字段 select distinct age from tb\_user |



**7）分组**

在InfluxDB中，GROUP BY 语句主要用于根据时间序列数据的标签（tags）进行分组。GROUP BY 仅能用于标签（tag）字段，而不能直接用于字段（field）进行分组。

|  |
| --- |
| SQL -- 根据region进行分组 select \* from tb\_user group by region |



**8）聚合函数**

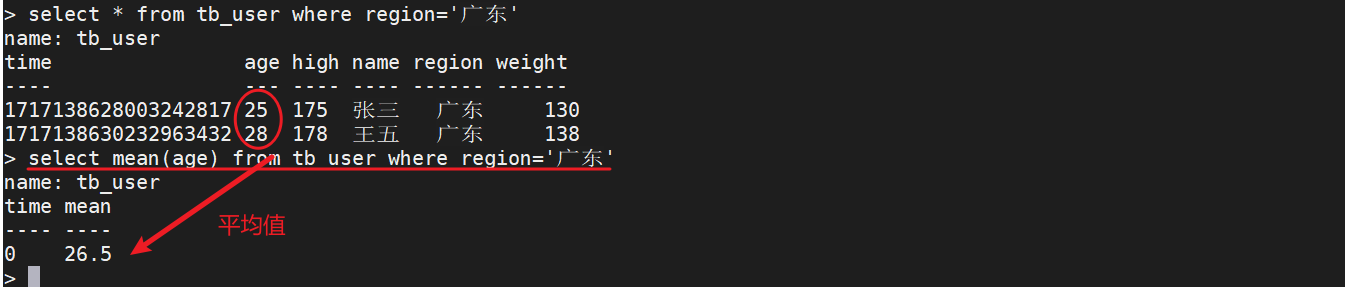
**count()函数**：返回一个（field）字段中的非空值的数量。

|  |
| --- |
| SQL -- 统计一条记录中；每个非空field的总数 select count(\*) from tb\_user |



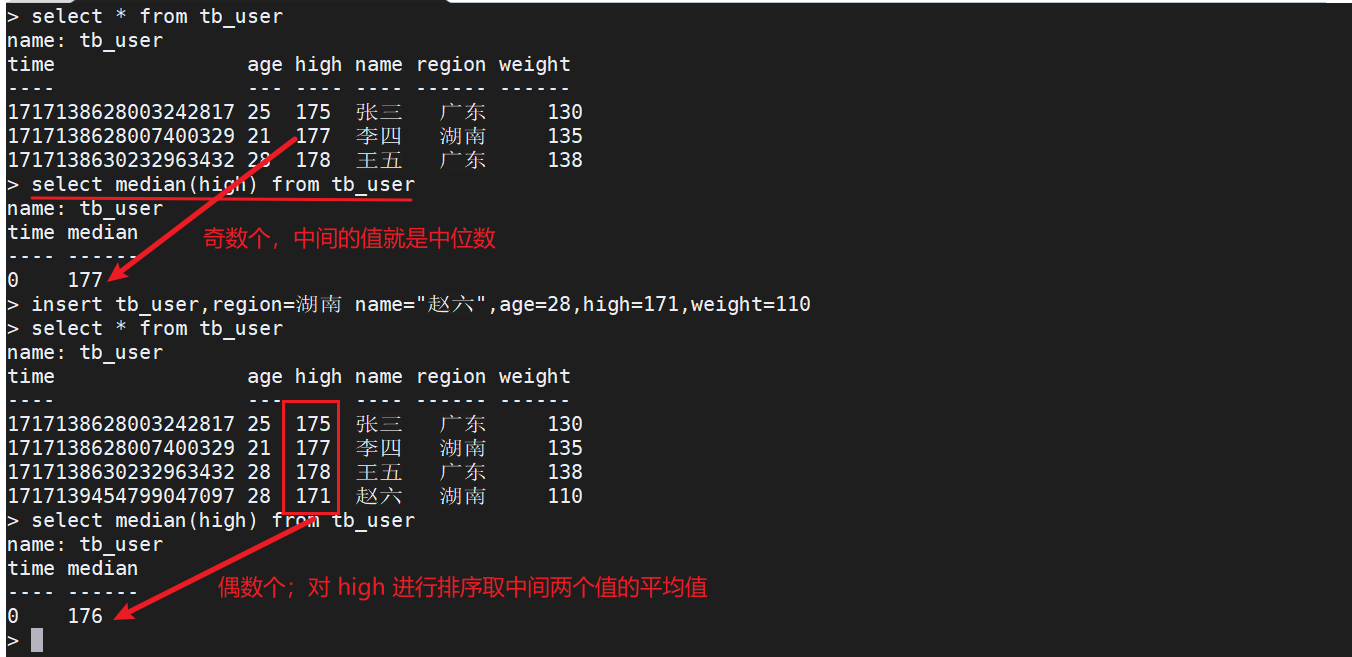
**mean() 函数**：返回一个字段（field）中的值的算术平均值（平均值）。字段类型必须是长整型或float64。

|  |
| --- |
| SQL -- 求用户表的广东地区的用户平均年龄 select mean(age) from tb\_user where region='广东' |



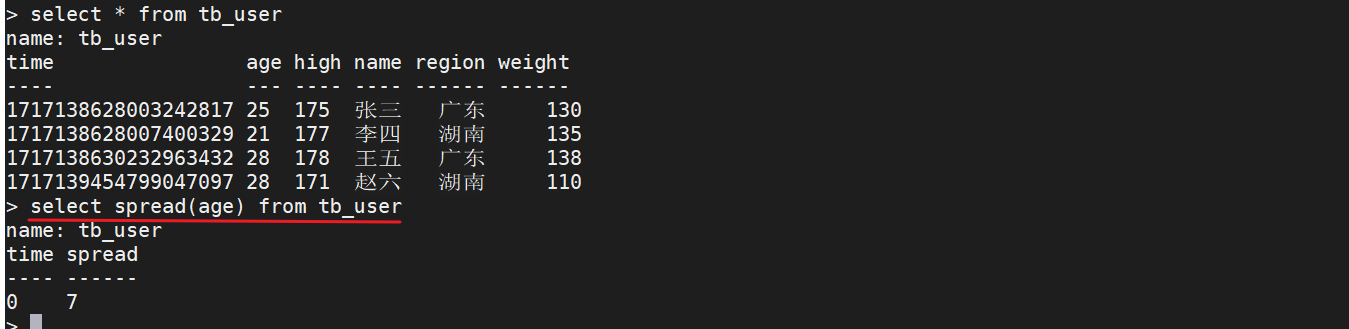
**median()函数：**从单个字段（field）中的排序值返回中间值（中位数）。用于计算一组数值的中位数。中位数是将一组数值**按大小顺序排列后**处于中间位置的数。如果数值集合中有**奇数**个数，中位数就是正**中间的那个数**；如果有**偶数**个数，则中位数是**中间两个数的平均值**。

|  |
| --- |
| SQL -- 查询身高中间值 select median(high) from tb\_user  -- 插入数据后再查看 insert tb\_user,region=湖南 name="赵六",age=28,high=171,weight=110 |



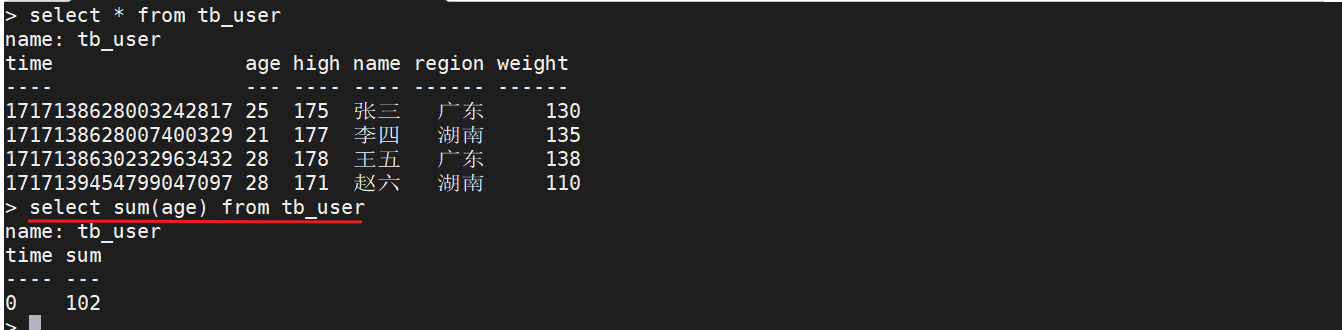
**spread()函数：**返回字段的最小值和最大值之间的差值。数据的类型必须是长整型或float64。

|  |
| --- |
| SQL -- 返回最大与最小年龄之间的差值 select spread(age) from tb\_user |



**sum()函数：**返回一个字段中的所有值的和。字段的类型必须是长整型或float64。

|  |
| --- |
| SQL -- 统计所有年龄之和 select sum(age) from tb\_user |



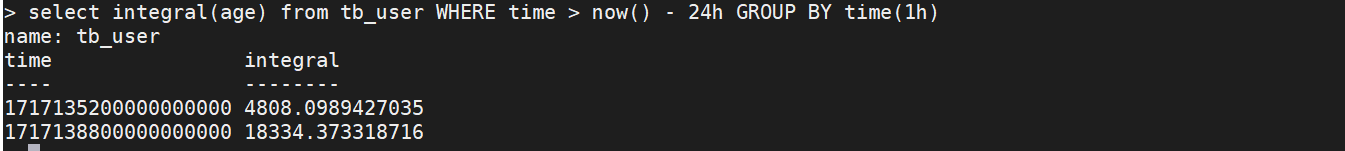
**integral()函数：**用于计算时间序列数据在制定时间范围内的积分值，积分值可以帮助我们更好的理解时间序列数据的趋势和变化。通常应用于计算某个field值随时间的累积总和。它相当于对field值进行连续求和，其中时间序列上的每个点代表了一个小的时间间隔内的“面积”，所有这些“面积”的和即为积分结果。

|  |
| --- |
| SQL -- 语法 SELECT integral(field\_key) FROM measurement\_name WHERE condition GROUP BY time(interval) fill(option) |

* **field\_key**：你想要对其积分的field的名称。
* **measurement\_name**：指定的measurement名称。
* **condition**（可选）：用于筛选数据的条件表达式。
* **time(interval)**（可选）：按照指定的时间间隔进行分组，这对于控制积分的精度很重要。
* **fill(option)**（可选）：指定如何处理数据中的空值或间隙，如none（不填充）、zero（用0填充）等。

|  |
| --- |
| **注意事项**   * **时间序列的连续性**：积分结果的准确性依赖于数据的时间序列是连续的或是以相对固定的时间间隔记录的。数据点之间的间隔过大或缺失可能会导致积分结果失真。 * **时间精度**：使用GROUP BY time(interval)时，积分是基于这个时间间隔来进行的，因此选择合适的时间精度对结果至关重要。 * **应用场景**：integral()常用于计算累积量，比如累计流量、累计电量消耗、温度随时间的累积热量等。 |

|  |
| --- |
| SQL -- 计算过去24小时内；每个小时的年龄累积 ；这样的统计没有实际意义，只是展示语法使用 select integral(age) from tb\_user WHERE time > now() - 24h GROUP BY time(1h) |



**9）分页**

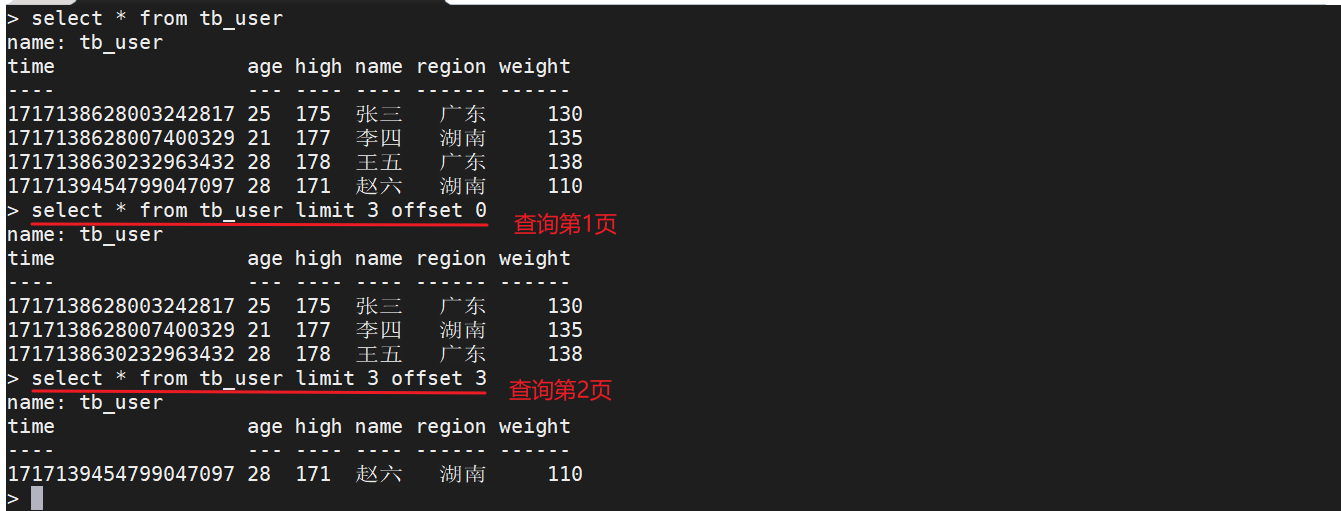
在InfluxDB中，要实现分页效果，通常你会使用 LIMIT 和 OFFSET 这两个关键字结合使用。LIMIT 用于限制返回的结果数量，而 OFFSET 用于指定从结果集中的哪一个位置开始返回结果。这种方式类似于在其他数据库系统中使用 LIMIT m, n 来实现分页，其中 m 是偏移量，n 是每页的数量。

|  |
| --- |
| SQL -- 语法 SELECT \* FROM measurement\_name WHERE condition GROUP BY ... LIMIT n OFFSET m |

* measurement\_name：你想要查询的measurement名称。
* condition（可选）：用于过滤数据的条件表达式。
* GROUP BY ...（可选）：根据需要对结果进行分组。
* LIMIT n：每页显示的记录数。
* OFFSET m：从结果集中的第 m 条记录开始返回，用于实现分页。

示例：

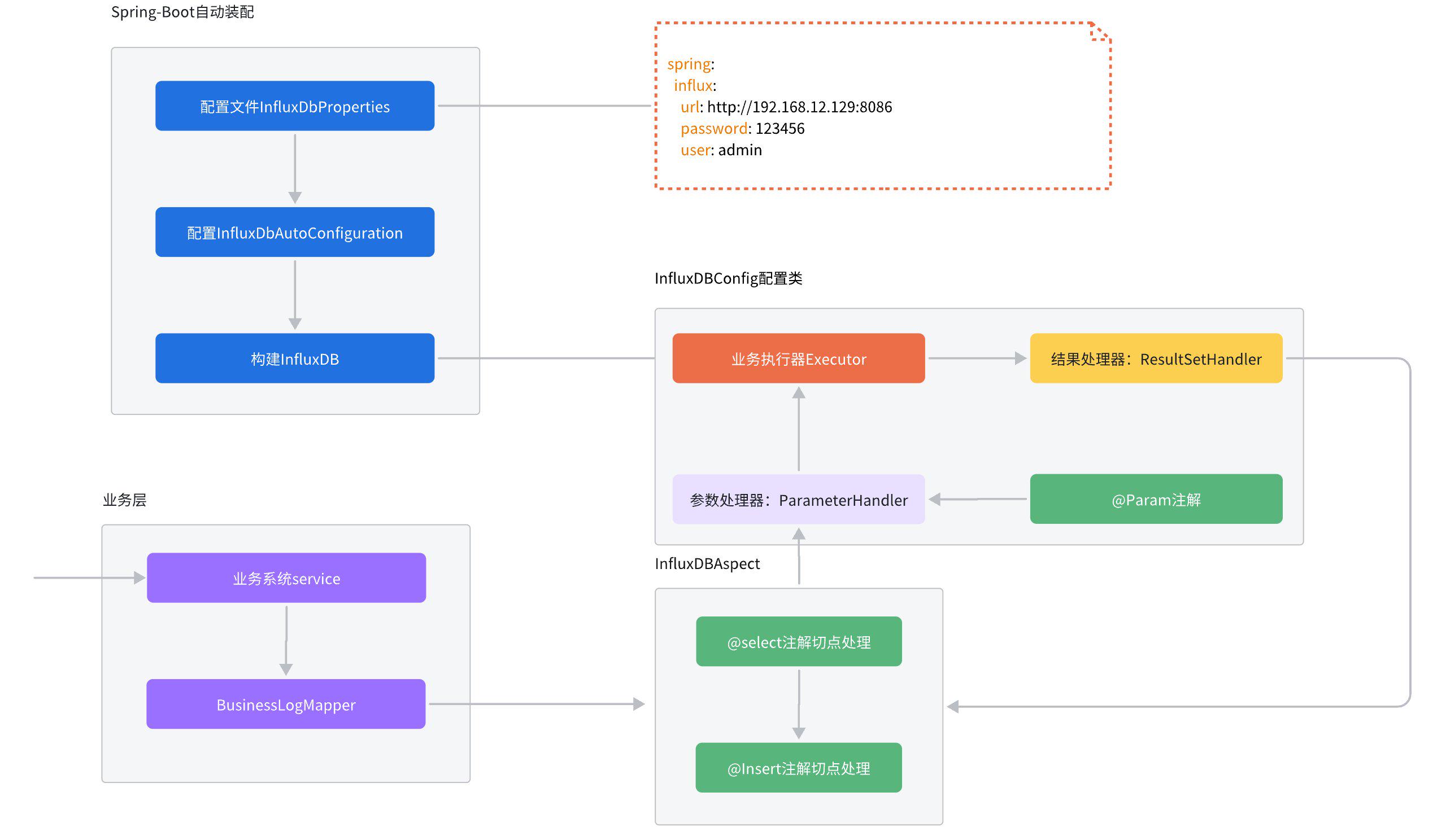
|  |
| --- |
| SQL -- 查询第1页，每页3条数据 select \* from tb\_user limit 3 offset 0  -- 查询第2页，每页3条数据 select \* from tb\_user limit 3 offset 3 |



**3、数据报表开发**

**3.1、InfluxDB持久层封装**

上面我们介绍InfluxDB的基本内容，下面我们来看下如何使用spring-boot来操作InfluxDB数据库，首先我们来看下整个的系统结构图例：



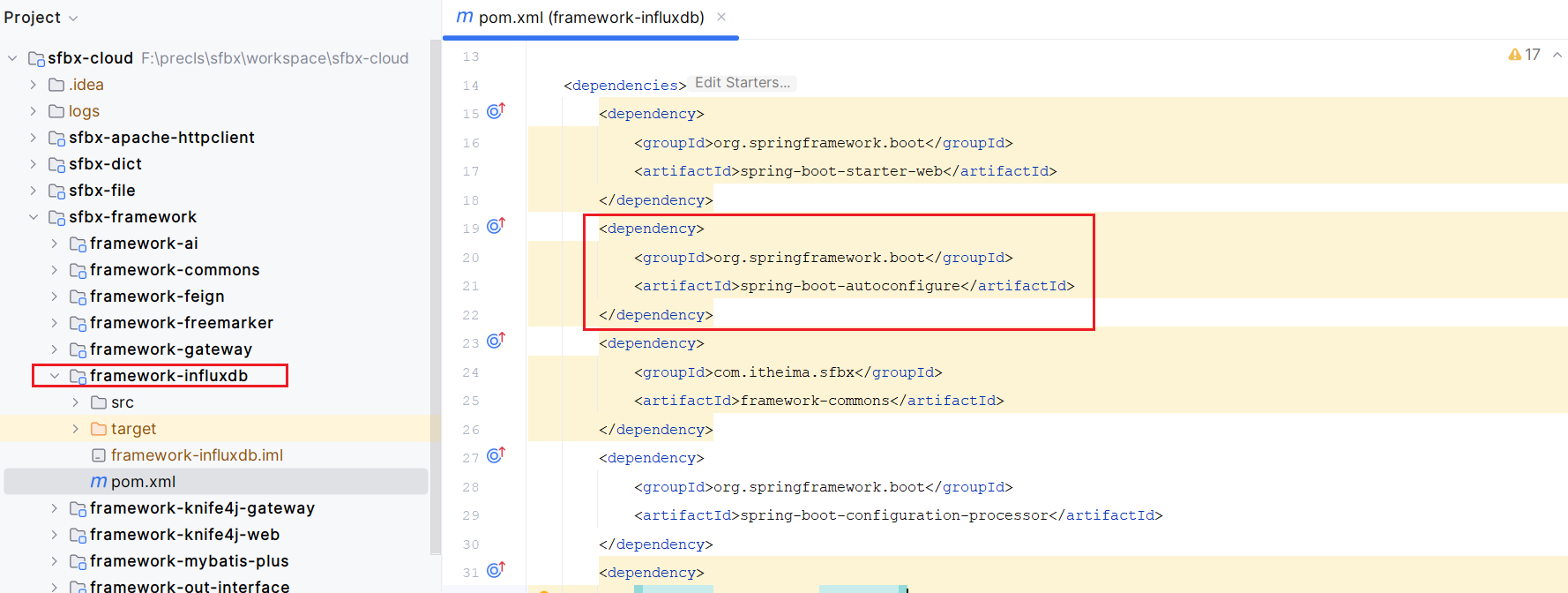
从上图中看我们可以分析出整个InfluxDB持久层封装主要依靠spring-boot自动装配，通过自定注解依赖AOP来实现，整个架构分为3个部分：

* **自动装配**：依赖spring-boot自动装配出InfluxDB对象，并把对象交于IOC容器管理
* **配置管理**：项目启动时，通过InfluxDBConfig构建出业务执行器、参数处理器、结果处理器，并把对象交于IOC容器管理
* **切面处理**：业务系统service调用业务Mapper时，influxDBAspect会对被@select或@Insert注解的方法进行切面处理，封装构建参数处理器，然后通过业务执行器请求influxDB，最后交于结果处理器来封装数据

**3.1.1、自动装配**

因springboot对influxdb已经整合，我们使用只需要引入对应的自动装配的maven坐标再配置influxdb的连接信息即可使用。

1）引入maven坐标

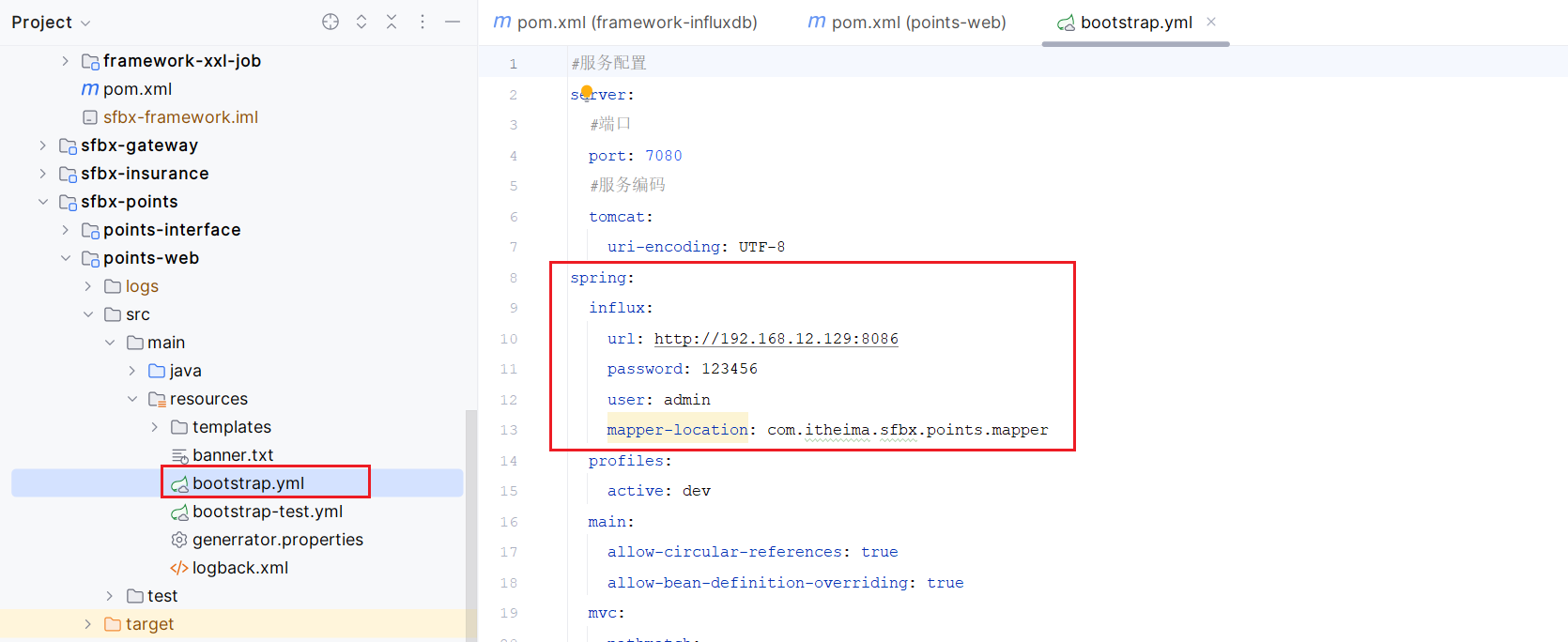


2）配置influxdb连接信息

由于我们的埋点数据是在 point-web 模块工程中处理的；所以在这个工程中引入上述的 framework-influxdb 坐标



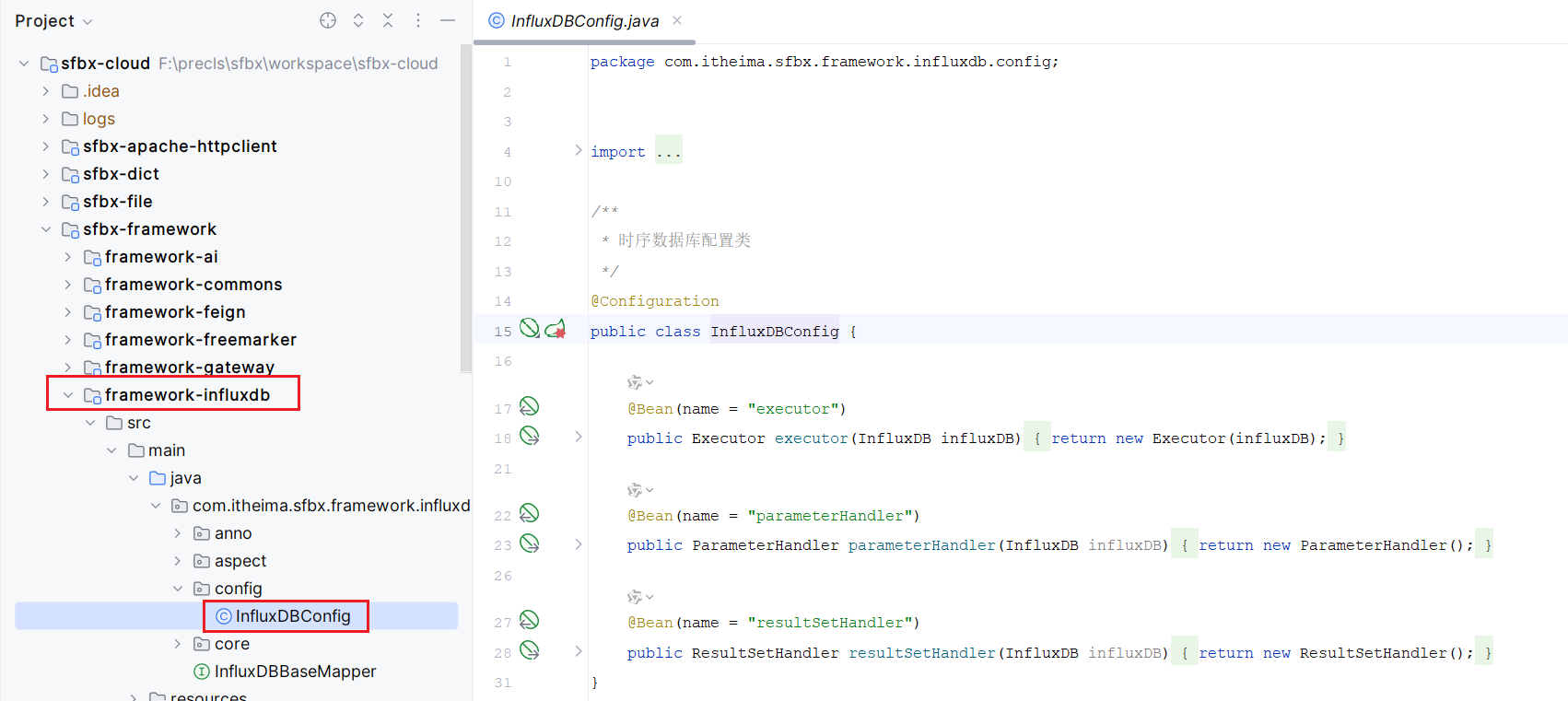
然后在 point-web 配置连接influxdb的连接信息：



至此，自动装配完成。——》InfluxDB Java操作客户端

**3.1.2、配置管理**

因为四方保险中实现了对于influxdb的框架便捷性封装；所以在使用中对于执行influxdb的命令、传入参数的分析、返回结果的映射处理对象都需要配置。在framework-influxdb项目中构建了一个InfluxDBConfig配置类：



业务执行器Executor ：需要从spring-IOC中拿到InfluxDB来完成构建，用于与influxDB进行交互

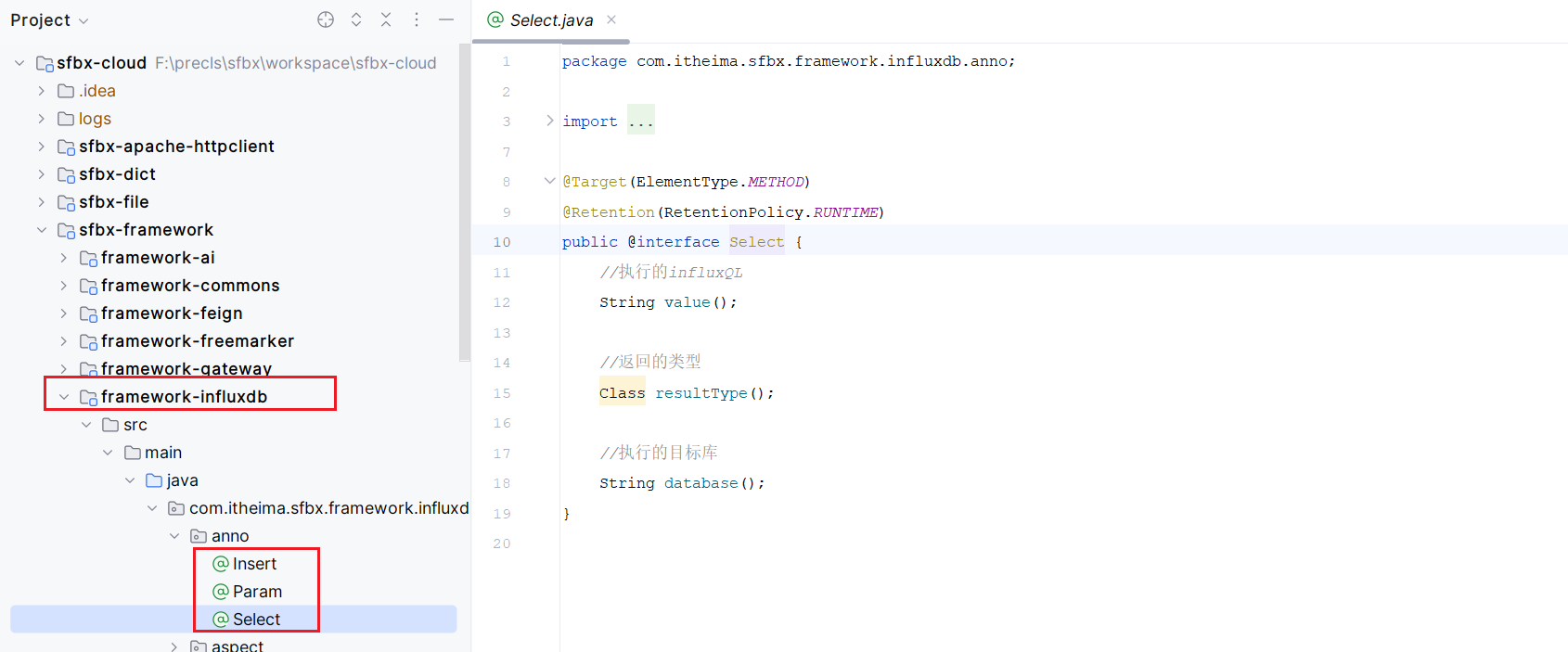
参数处理器ParameterHandler ：用于执行参数的封装处理

结果处理器ResultSetHandler ：用于执行结构的封装处理

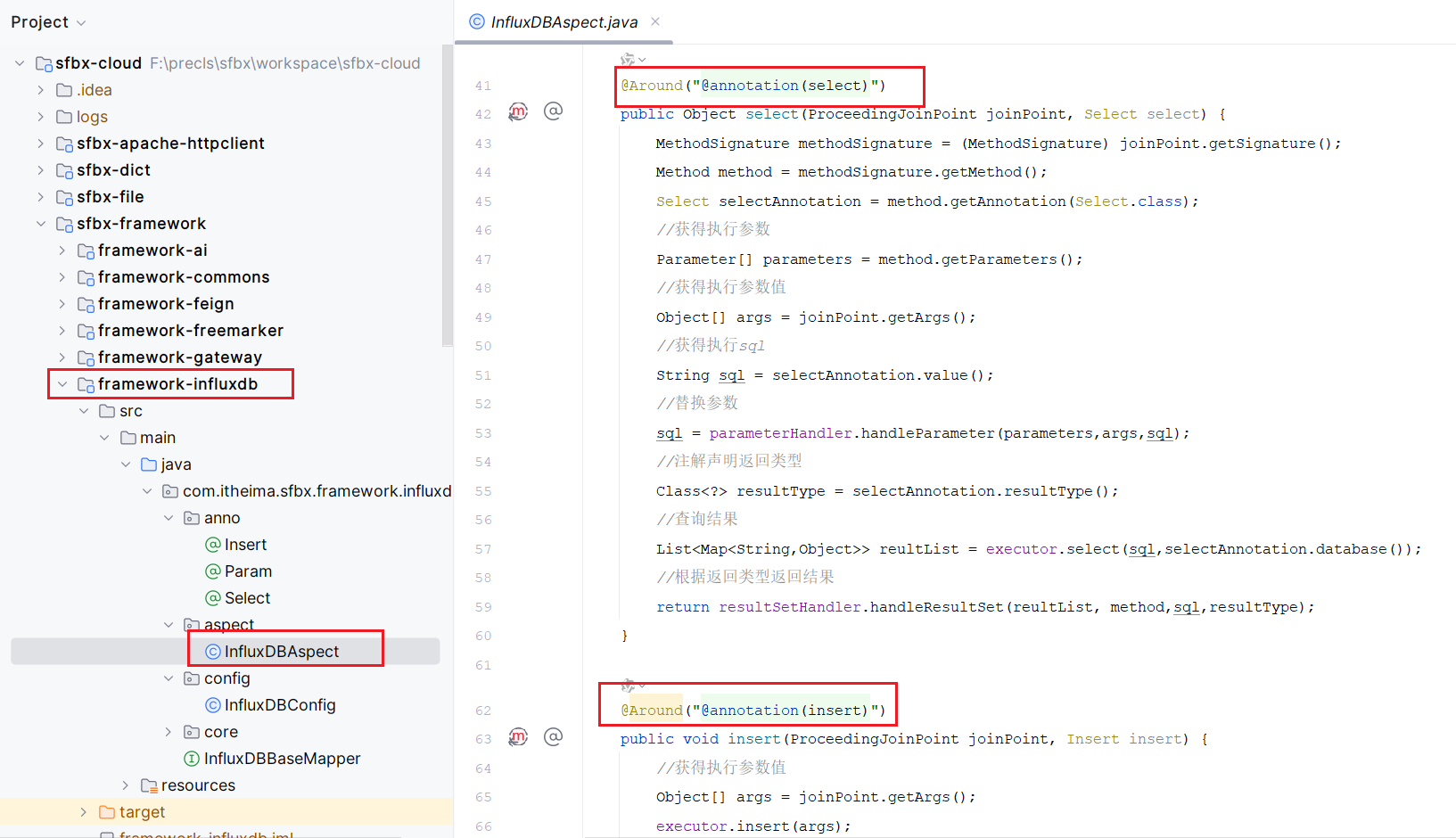
**3.1.3、切面处理**

在使用到了InfluxDB的业务系统中；对应的业务Service调用业务Mapper时；通过自定义注解 @Select和@Insert注解进行查询、插入的处理，这些注解需要有切面解析处理实现参数解析、返回结果映射。

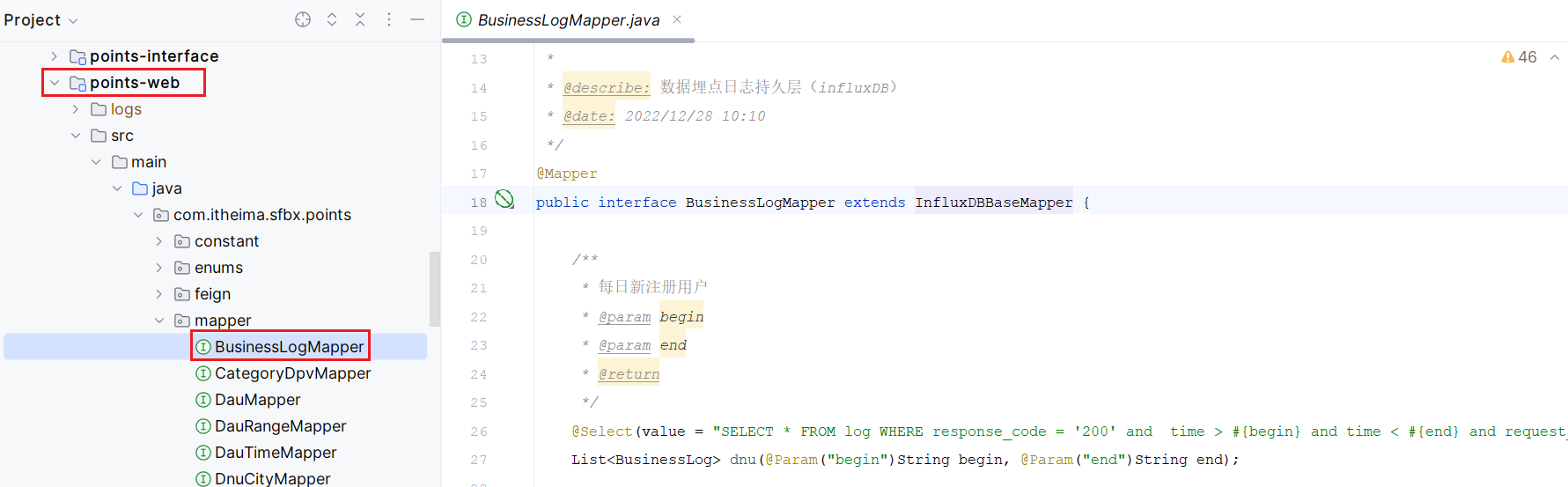
自定义注解：



切面类解析注解并处理参数解析、语句拼接、返回结果映射：



具体使用：



在四方保险中；整合InfluxDB 1.0：

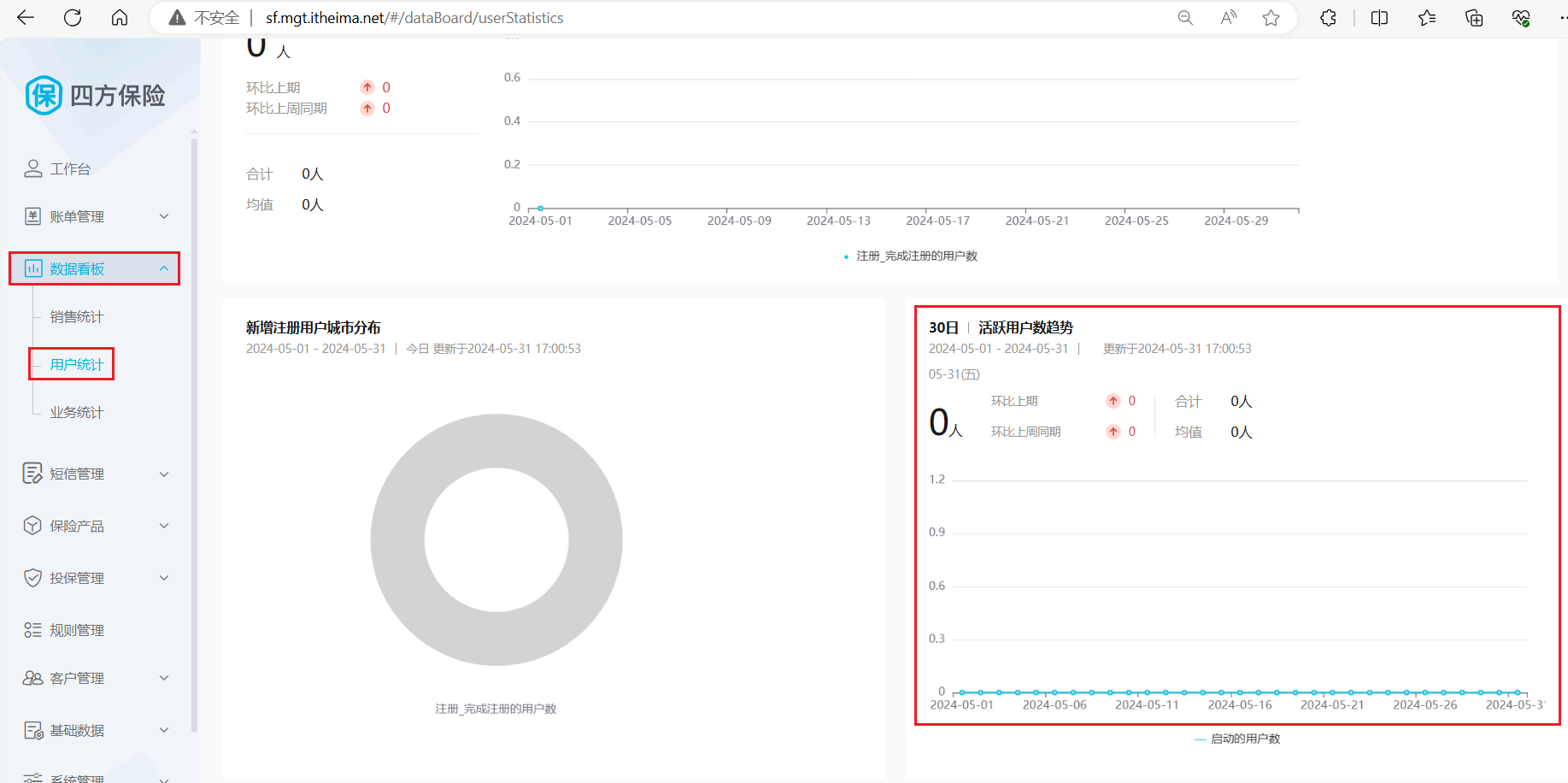
1. 添加influxdb的依赖；——》influxDB核心操作类
2. 配置influxdb的连接信息；
3. 持久层封装InfluxDB

* 编写自定义注解 @Select/@Insert/@Param
* springboot自动装配influxDB持久层处理配置类：Executor/ParameterHandler/ResultSetHandler
* 编写spring aop切面对注解进行拦截解析并执行，将返回结果进行转换

**3.2、30日活跃用户数趋势**

**3.2.1、业务分析**

访问 http://sf.mgt.itheima.net/#/dataBoard/userStatistics

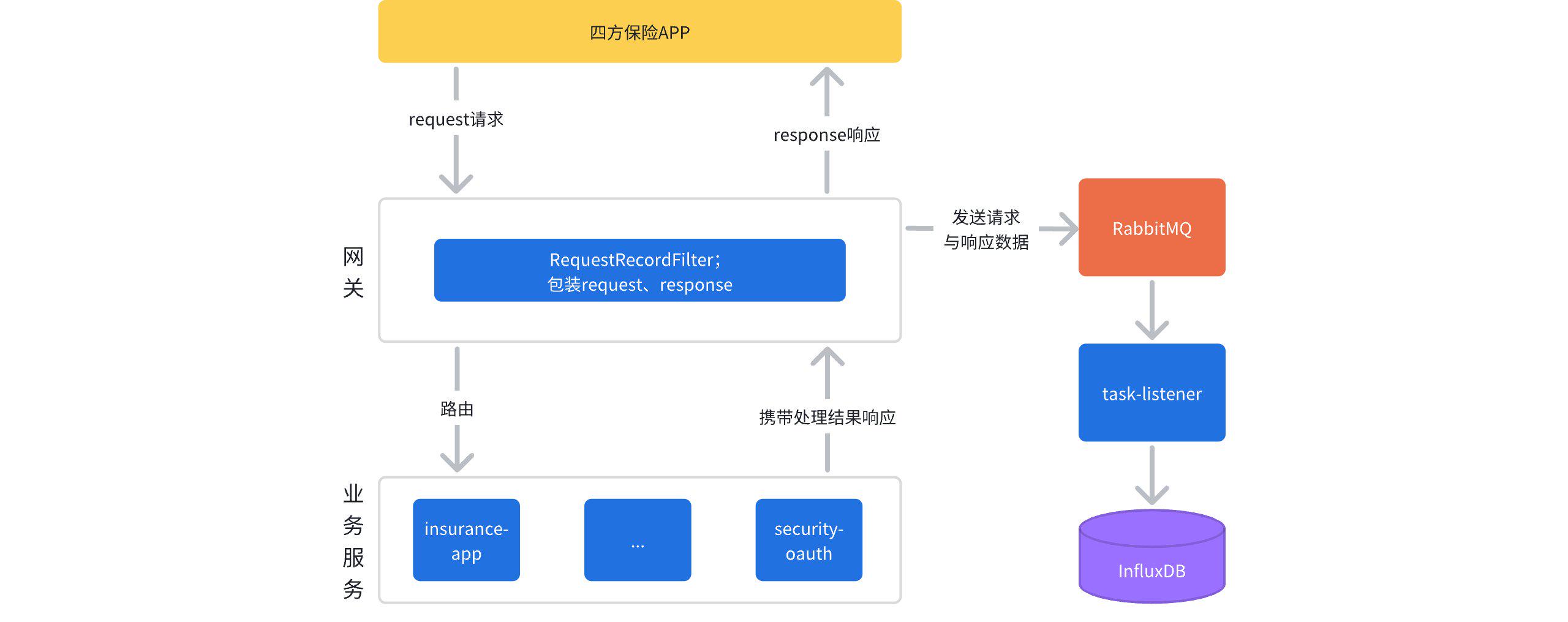


需要实现统计上图中的30日活跃用户数趋势；可以按照如下步骤：

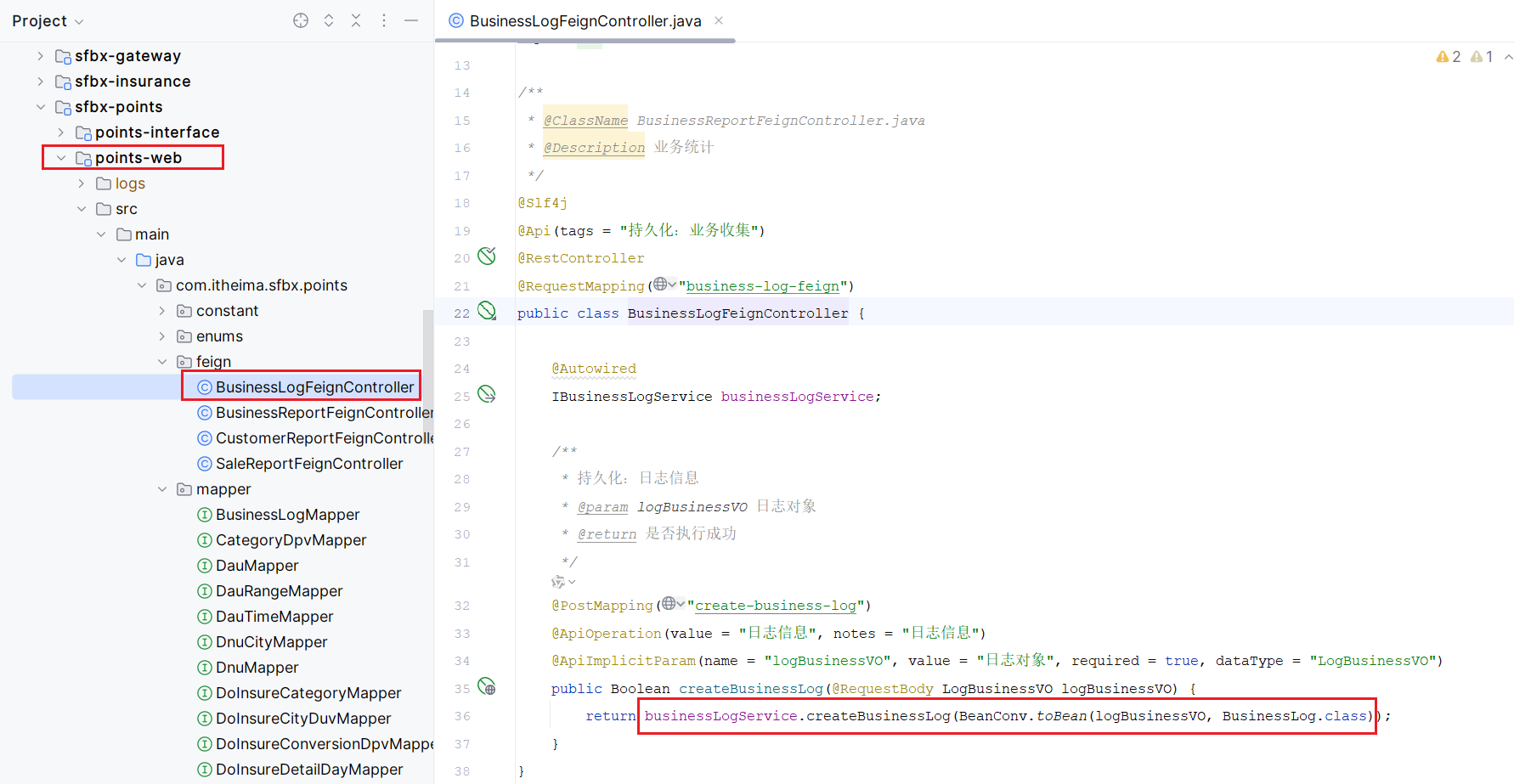
1. **数据来源**：所有的用户行为都通过网关，在网关做数据埋点并通过MQ发送消息存储到InfluxDB
2. **数据清洗**：对于存储到InfluxDB的用户行为数据，可以采用定时任务XXL-JOB分析清洗出对应的结果并存储到MySQL中
3. **数据展示**：在管理后台中利用 point-web 提供的数据接口获取用户行为统计数据并展示

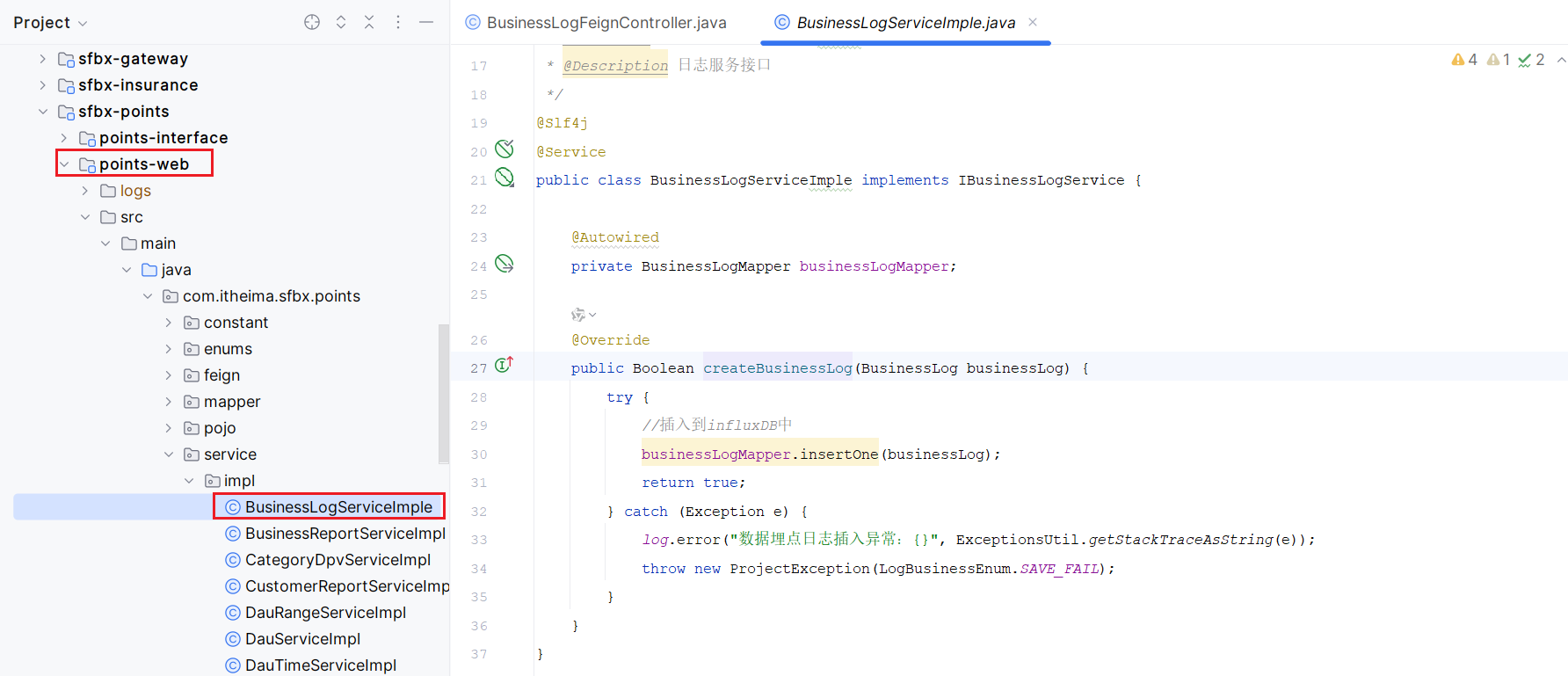
**3.2.2、数据来源**

这一部分就是上一章中的数据采集；我们已经跟踪过代码；流程如：



最终保存日志数据到InfluxDB的处理代码：



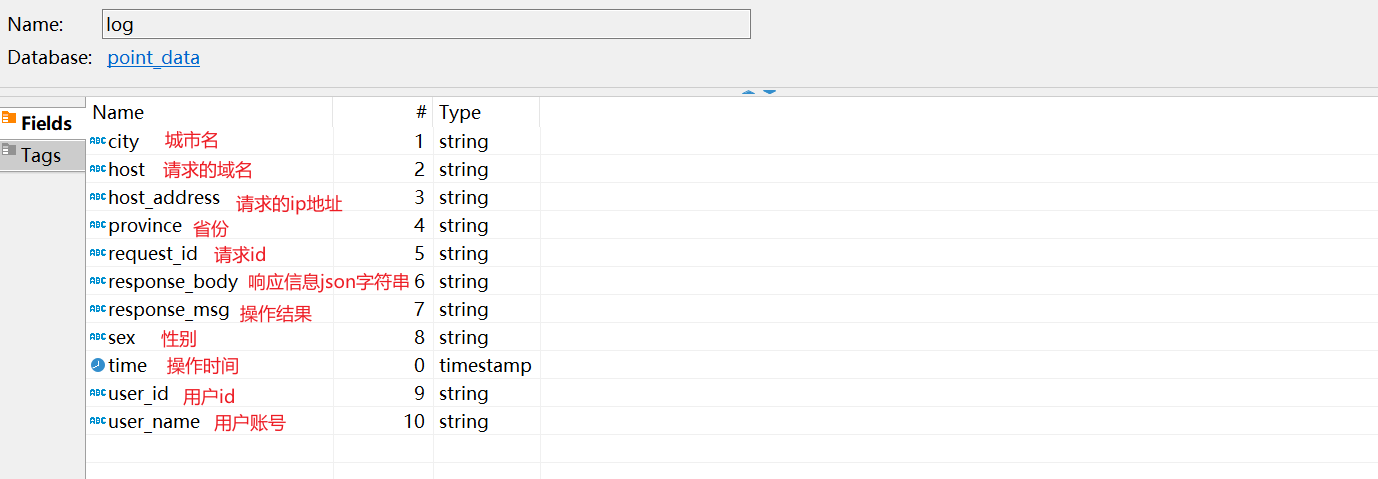


数据采集到InfluxDB中的日志measurement中；该measurement（表）的结构可以如下查看：

|  |
| --- |
| SQL -- 语法 SHOW FIELD KEYS ON "<database\_name>" FROM "<measurement\_name>"  -- 查看log 表结构 show field keys on point\_data from log |



或者直接使用 dbeaver 数据库管理工具查看也可以：



日志在记录到InfluxDB之后；它的数据内容大体如下：



**3.2.3、数据清洗**

**1）分析**

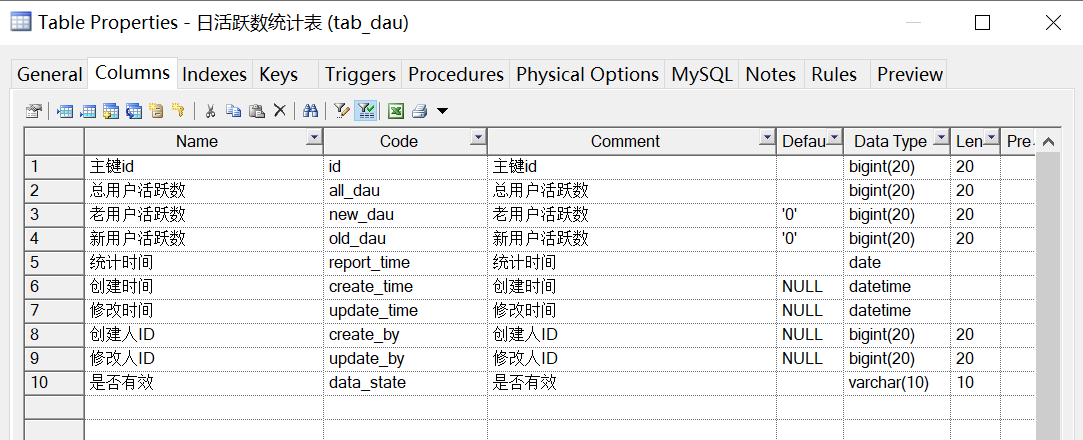
数据的清洗是对记录到InfluxDB的日志数据进行查询统计；而我们的需求是统计 30日活跃用户数趋势 ；如下图：



在做数据分析中有不少概念，我们来了解一下：

|  |
| --- |
| **DAU**：Daily Active User 日活跃用户量。统计一日（统计日）之内，登陆或使用了某个产品的用户数（去重）  **WAU**：Weekly Active Users 周活跃用户量。统计一周（统计日）之内，登陆或使用了某个产品的用户数（去重）  **MAU**：Monthly Active User 月活跃用户量。统计一月（统计日）之内，登陆或使用了某个产品的用户数（去重）  **ACU**：Average Concurrent Users 平均同时在线人数  **PCU**：Peak Concurrent Users 最高同时在线人数  **UV**：Unique Visitor 唯一访问量，即页面被多少人访问过  **PV**：Page View 页面浏览量，即页面被多少人看过 |

根据上述的概念了解到；我们需要统计的是30天的DAU的数据信息；统计日活用户趋势，一般会统计新老用户的日活趋势；所以对应设计如下用于存储，统计的日志中的日活数据表结构：



|  |
| --- |
| **新用户**：界定标准因为业务不同；24小时、一周、一个月等以内注册的用户都可以称为新用户；在四方保险中以当天注册的用户作为新用户 |

分析了数据结构之后也就知道 **数据清洗**，就是将InfluxDB的数据，统计存入到 tab\_dau 表；而且应该每天有好几次的统计：比如01:00,08:00,13:00,18:00,23:00 这些点都统计。

数据的清洗；就是**DAU统计；其实现思路**：

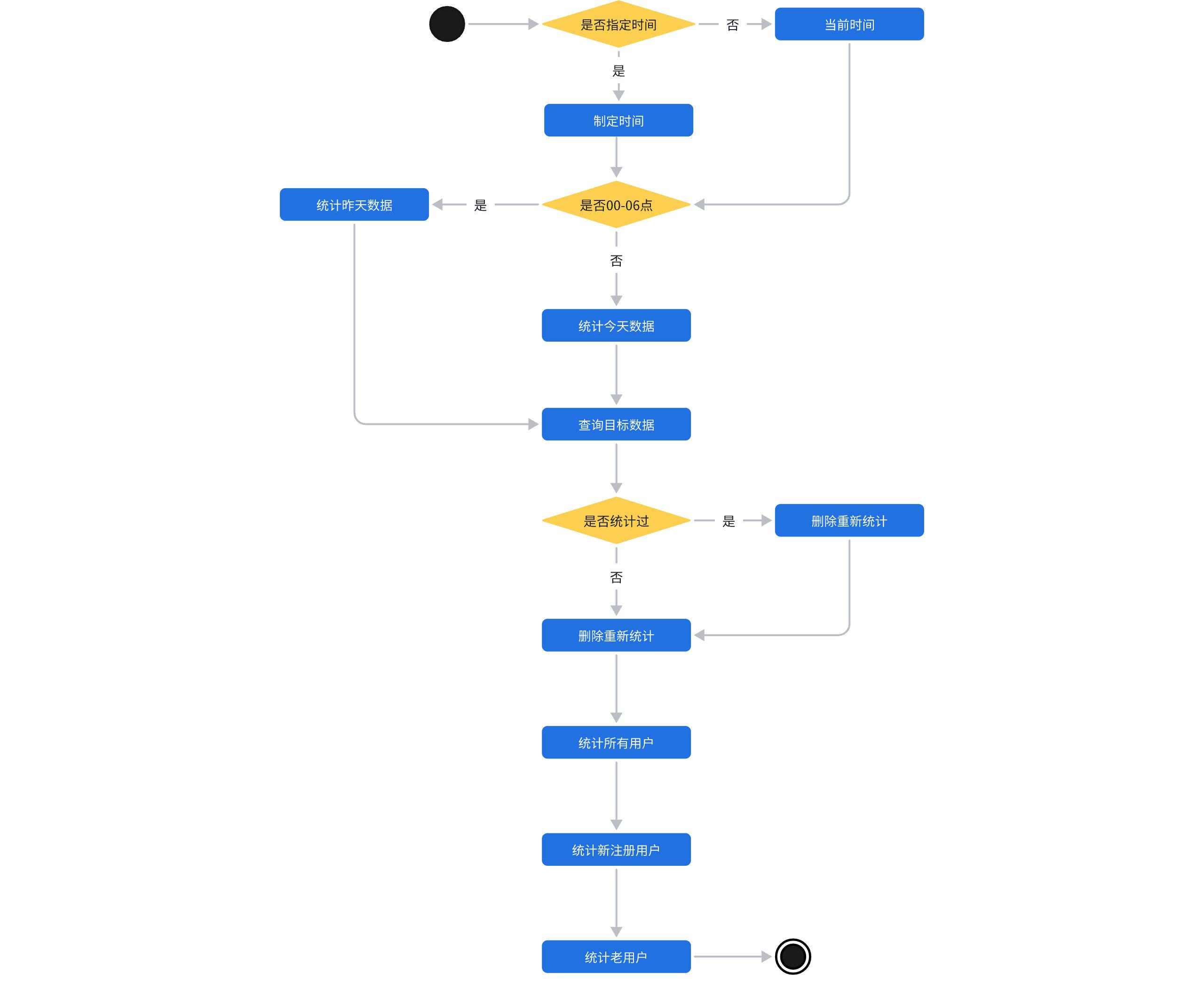
1. 统计这些数据的时候，如果是在 0-6点；这时当天其实还没有太多访问数据，应该统计昨天的日志；
2. 如果已经存在当天的统计数据，则移除重新统计；
3. 到InfluxDB查询 log 表中，当天的 user\_id 去重之后的总个数就是 当天的总活跃数

|  |
| --- |
| SQL -- 查询的SQL语句参考如下：begin、end都是纳秒值 SELECT COUNT(DISTINCT(user\_id)) as user\_id FROM log WHERE time > #{begin} and time < #{end}  --例如；要查询过去30天的活跃用户  SELECT COUNT(DISTINCT(user\_id)) as user\_id FROM log WHERE time > now()-30d   -- 例如 SELECT COUNT(DISTINCT(user\_id)) as user\_id FROM log WHERE time > 1720195200000000000 and time <1720281599000000000 |

|  |
| --- |
| Java */\*\**  *\* 生成某个日期的纳秒值；上述要查询的纳秒值的生产可以参考如下类*  *\*/* public class PointTest {   public static void main(String[] args) {  String beginStr = "2024-07-06 00:00:00";  String endStr = "2024-07-06 23:59:59";   *//将一个日期格式yyyy-MM-dd HH:mm:ss字符串转换为纳秒值* System.*out*.println("beginNanos = " + *genNanoSecond*(beginStr));  System.*out*.println("endNanos = " + *genNanoSecond*(endStr));  }   private static long genNanoSecond(String beginStr) {  LocalDateTime localDateTime = DateUtil.*parseLocalDateTime*(beginStr, "yyyy-MM-dd HH:mm:ss");  Instant instant = localDateTime.atZone(ZoneId.*systemDefault*()).toInstant();  return instant.toEpochMilli() \* 1\_000\_000 + instant.getNano();  } } |

1. 再到InfluxDB查询 log 表中；查询 请求地址包含 register-user 而且响应状态码为 200 的记录总数就是新注册活跃用户总数；
2. 那么 旧的用户活跃总数 = 总活跃数 - 新注册活跃用户总数；
3. 这样 tab\_dau 表中的三大活跃用户数据都有了；保存数据到 tab\_dau表。完成！

使用流程图表示的话；如下图示：



**2）实现**

* 制定定时任务



远程调用客户端：

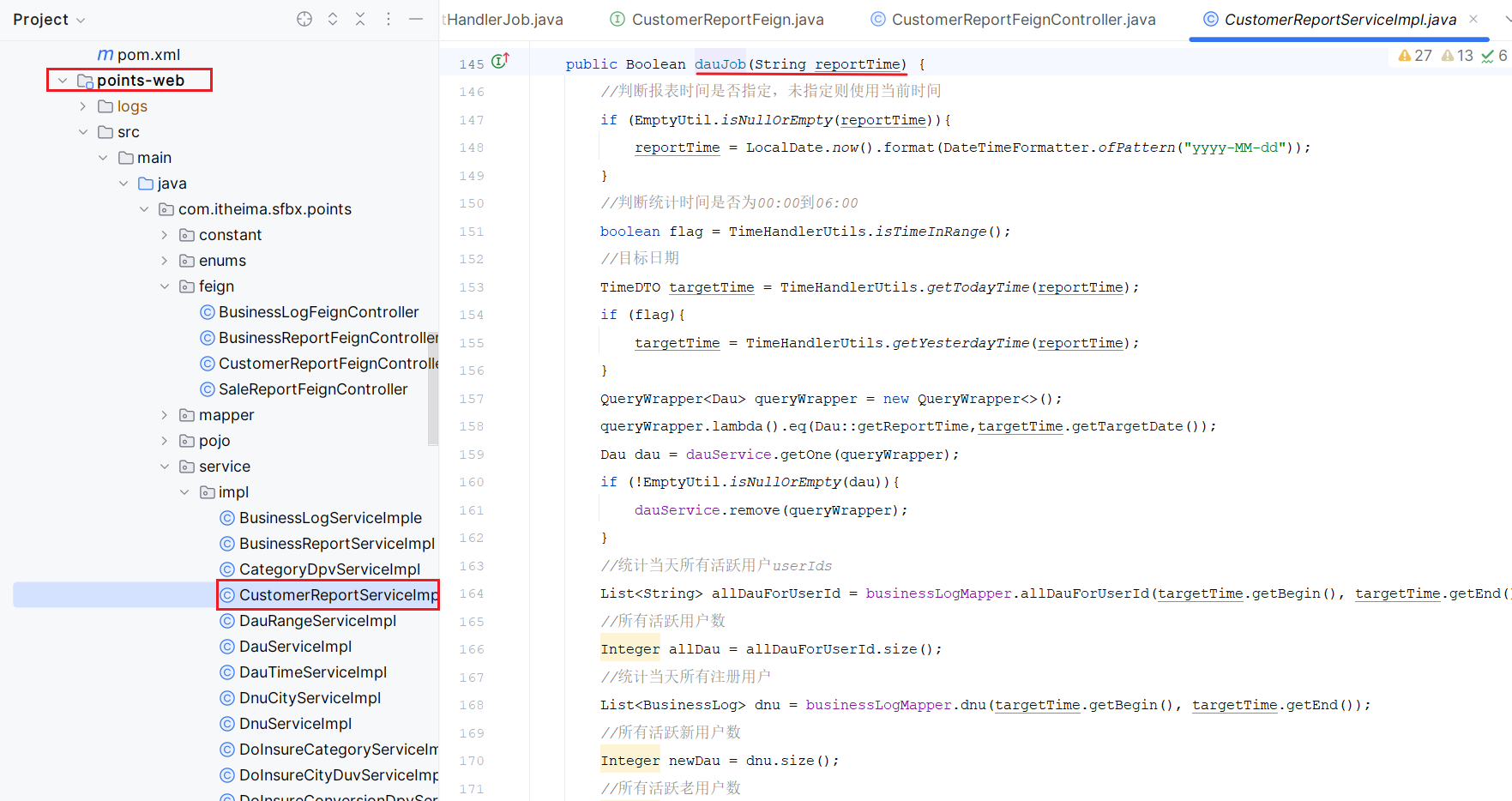


* 实现统计

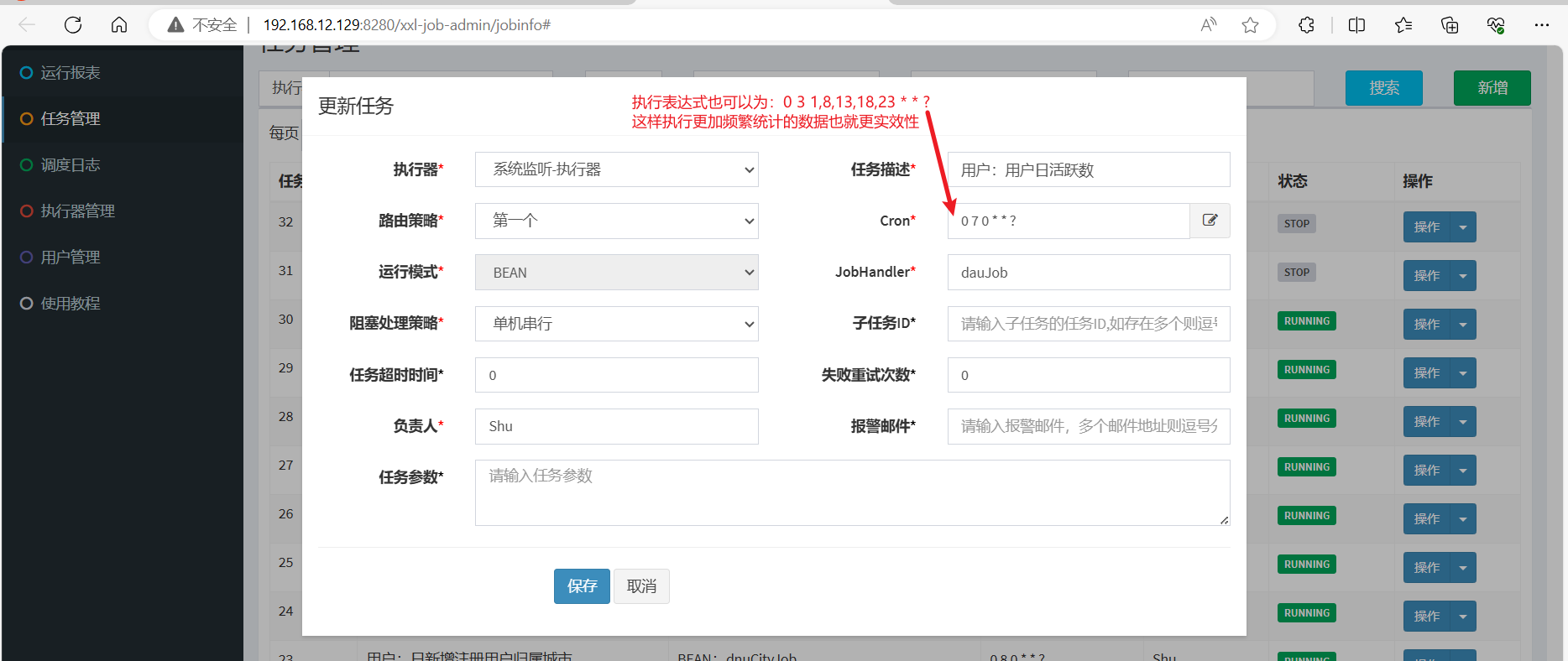
对应远程调用客户端的处理器：



业务实现：



* 配置xxl-job



|  |
| --- |
| 上述的执行时间【0 0 1,8,13,18,23 \* \* ?】任务会在每天01:00,08:00,13:00,18:00,23:00的整点执行，需要注意：   * 01:00执行的这次统计的前一天的数据 * 08:00,13:00,18:00,23:00执行的为统计的当天的数据 |

**3.2.4、数据展示**

**1）分析**

原型界面如下：

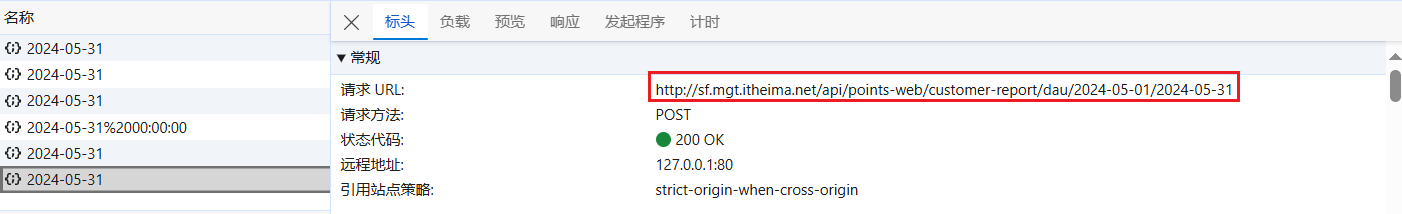


要展示上图中所有的数据，其结果如下：



|  |  |
| --- | --- |
| 功能点 | 功能说明 |
| 30日|活跃用户数趋势 | 采用折线图的方式展示数据变化，并同时展示环比上周，环比上期，30人合计，日均值信息 |

使用F12查看请求地址信息：

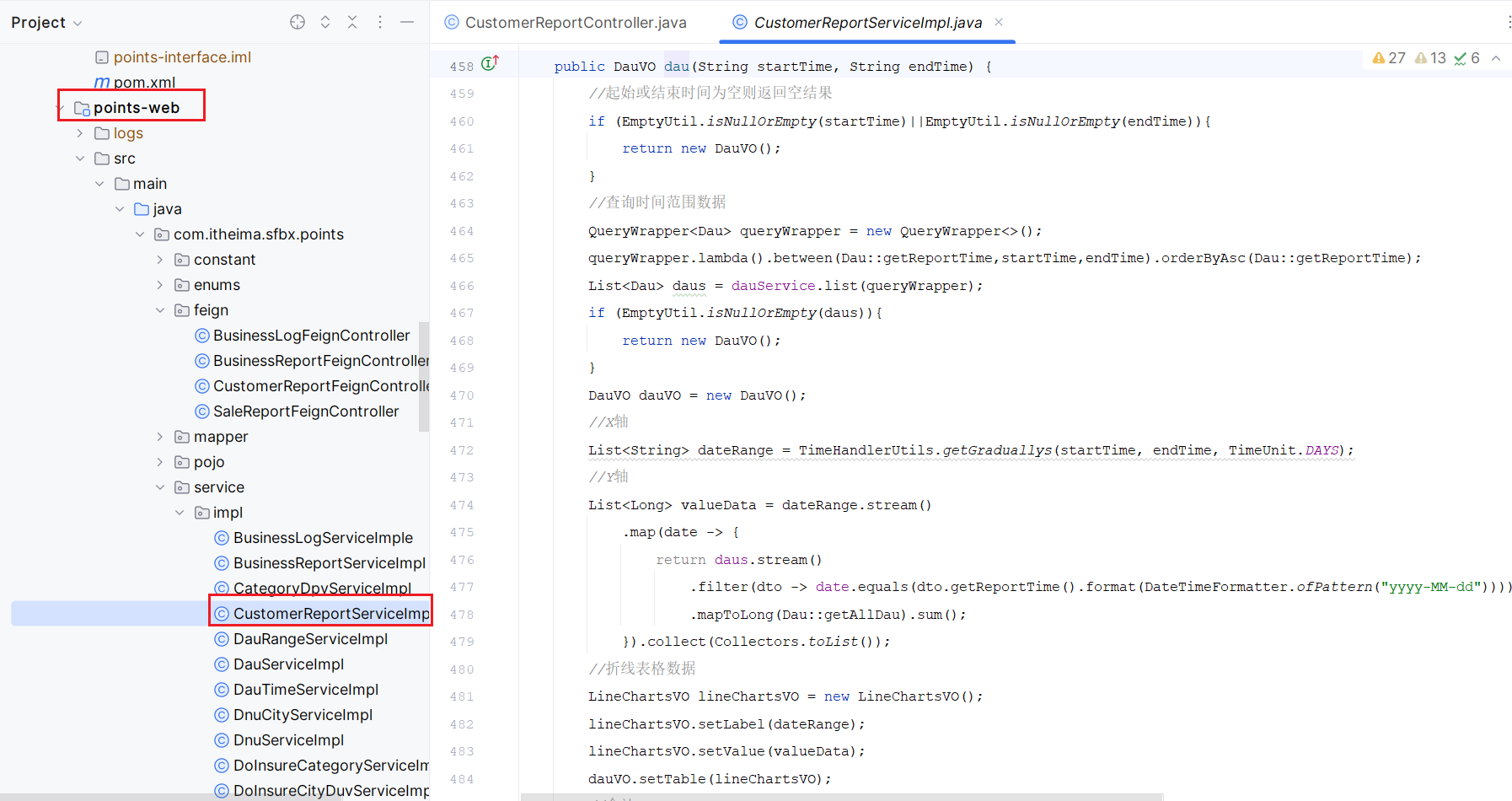


下面我们来参看下对应开发文档，并按照需求文档进行开发：

|  |  |
| --- | --- |
| 功能点 | 开发文档地址 |
| 30日|活跃用户数趋势 | http://sf.mgt.itheima.net:7069/doc.html#/points-web-swagger/%E6%95%B0%E6%8D%AE%E5%B1%95%E7%A4%BA%EF%BC%9A%E7%94%A8%E6%88%B7%E7%BB%9F%E8%AE%A1/dauUsingPOST |

**2）实现**





**3）测试**

在xxl-job管理控制台中；对 用户日活跃数 任务进行执行：



执行之后，可以查看数据库sfbx-log 中的 tab-dau表数据；查看是否有多一条当天的活跃数据记录。

刷新：http://sf.mgt.itheima.net/#/dataBoard/userStatistics

查看页面中 30日活跃用户数趋势 中的数据是否有更新。如果能查看到表中的统计数据；则说明统计成功。

**4、面试题**

**1）InfluxDB你们用来做什么的？**

用于存储用户的系统访问日志；系统中需要利用这些日志信息统计：每日新注册数，注册用户城市分布，活跃用户数趋势（环比昨日、环比上周），新老用户结构，活跃时段分布；

**2）简历中还可以写哪些功能？**

* **基本描述**：独立完成数据看板中用户行为数据统计、销售统计；设计系统统一的网关埋点捕获行为数据，利用MQ异步发送日志信息，独立部署InfluxDB；完成日志数据的存储；再对存储数据进行清洗和统计展示。

**成就**：引入的InfluxDB能够基于时间戳、标签更快有更有效的查询日志，整体日志查询效率提升了83%；

**细节**：比如；我印象比较深的是做的 30日活跃度用户数趋势 的数据看板展示：

* 数据采集：同样基于网关埋点，将非登录等请求、响应都记录到InfluxDB
* 数据清洗：InfluxDB时序数据库，除了它自动设置的时间戳外，我还设置了请求方式、请求路径、响应状态码作为它的tag（标签）这样在统计是否新注册用户的时候可以通过响应码能够大大提高InfluxDB的查询效率。
* 除此以外；还设置了关于日志的 RP保留策略，并不是永久保存，默认我们只保留日志30天，之后InfluxDB会自动删除，保持存储空间的持续性。而这些删掉的数据早就被定时任务已经在第二天就会统计完成形成 dau 信息存入到MYSQL数据库表中永久存在。
* 所以我认为，当时做这块的时候，我考虑到日志数量大查询效率问题，引入了InFluxDB；也兼顾考虑到存储空间过大问题的解决。兼顾了时间与空间复杂度吧；算是印象深刻一些。

**3）追问：怎么能删除日志？如果还要再使用呢？**

* 日志：目前我们是只删除30天以前的，对于我们系统来说已经满足，而且并无问题
* 如果要保留；对于InfluxDB来说也非难事；我就可以实现；改变measurement的 retention policy 保留时长设置为0即可，表示永久存留。
* 或者可以将表（measurement）删除都不影响；因为已经到了其它存储的数据库中tab\_dau

**4）InfluxDB的常用命令说几个听一下**

从数据库——》表——》函数 都与MySQL类似；挑几个说一下。注意一定要说到它独有的几个概念。

Measurement、Tag、Field等。你举一个插入数据的例子就包含到刚刚那几个概念了。

|  |
| --- |
| SQL insert measurement+","+tag1=value1,tag2=value2 + 空格 + field1=value1,field2=values2  -- 例如：对measurement为tb\_user的插入数据；有一个tag索引名为region，值为广东；有三个field分别是age,high,weight 对应的值分别为25、175、130 insert tb\_user,region=广东 name="张三",age=25,high=175,weight=130 |