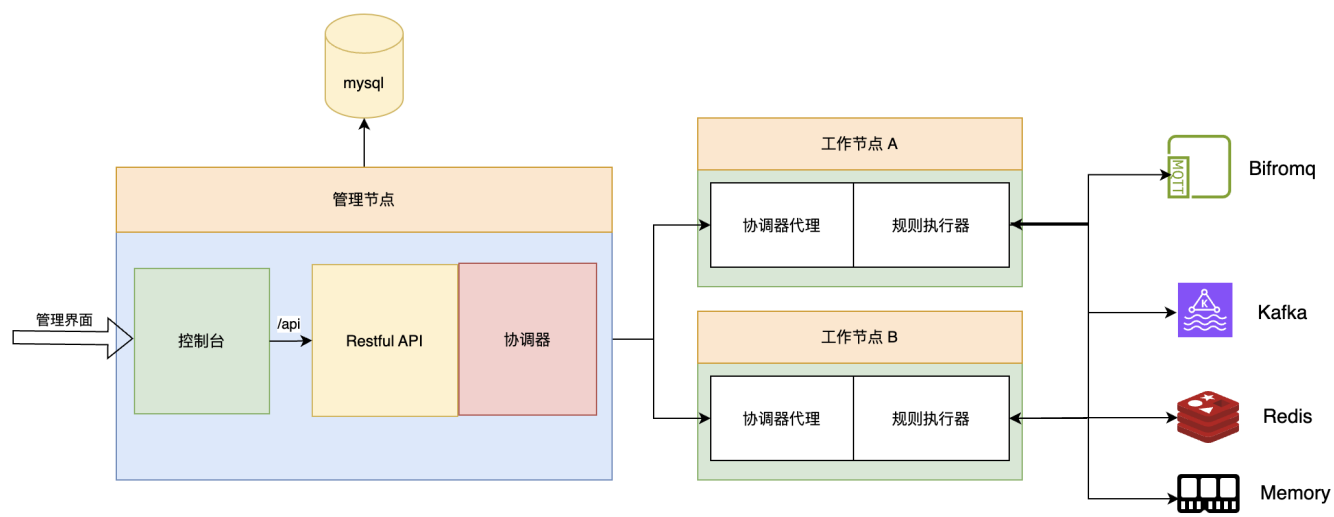


Bifromq 规则引擎设计

一、整体架构

Bifromq 规则引擎设计整体框架如下：



整体来看，整个框架设计实现一个可扩展、分布式的规则引擎，其中包含了数据存储、消息传递、规则处理和系统管理等多个组件。每个组件都有其特定的职责，并通过定义良好的接口和协议相互协作。其中主要组件如下：

1、管理节点：

负责管理和协调其他节点的中心节点，负责规则的分发、任务的调度监控、提供API接口。包括 Restful API 和 协调器俩个子组件。

2、工作节点：

工作节点是从管理节点接收规则、执行规则、上报状态等服务实例，同时根据规则订阅Bifromq MQTT Broker上的消息，处理消息。包含协调器代理 和 规则执行器俩个子组件。

3、管理UI：

提供了一个图形用户界面，允许管理员或用户配置数据源、规则、查看规则执行状态和处理结果等功能。

4、MySQL：

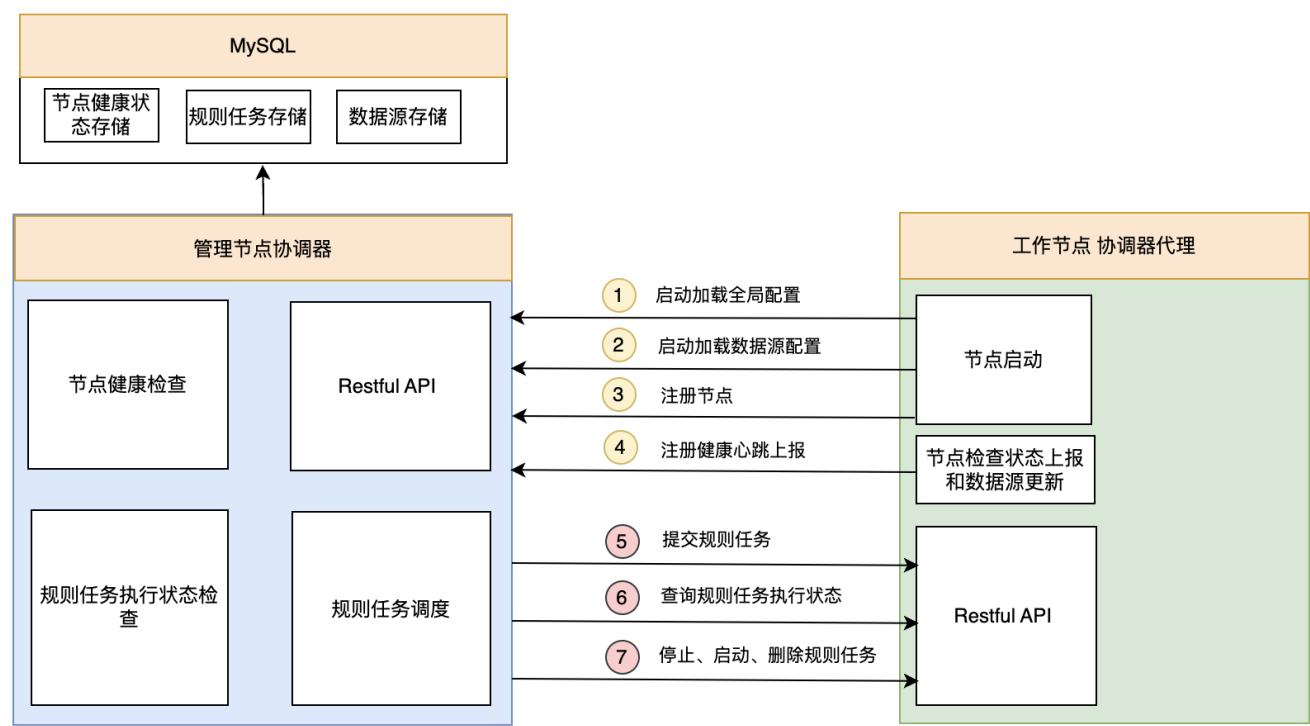
作为持久化存储，用于存储用户和权限，工作节点状态、数据源、规则定义和执行状态等数据。

5、Bifromq, kafka, redis：

Bifromq , kafka, redis 提供规则处理的数据来源和规则处理结果输出存储。

二、协调器工作原理

协调器工作原理如下图：



协调器在这个系统中扮演着核心角色，类似于大脑，负责节点管理、任务调度、状态监控和通信协调。以下是协调器及其代理组件的详细工作流程：

协调器组件：

1. **节点健康检查协程：**定期对工作节点进行健康检查，确保它们正常运行。不健康的节点将从调度队列中移除，以保证任务分配的高效性。
2. **规则任务执行状态检查协程：**通过协调器代理的Restful API，定期检查工作节点上规则任务的执行状态，确保任务按照预期进行。
3. **规则任务调度协程：**根据规则任务的定义和触发条件，智能地在适当的工作节点上调度任务执行。
4. **Restful API协程：**提供Restful API接口，实现与工作节点上的协调器代理的通信。主要负责下发全局配置和数据源配置、节点的注册与注销、节点健康状态的上报等。

协调器代理组件：

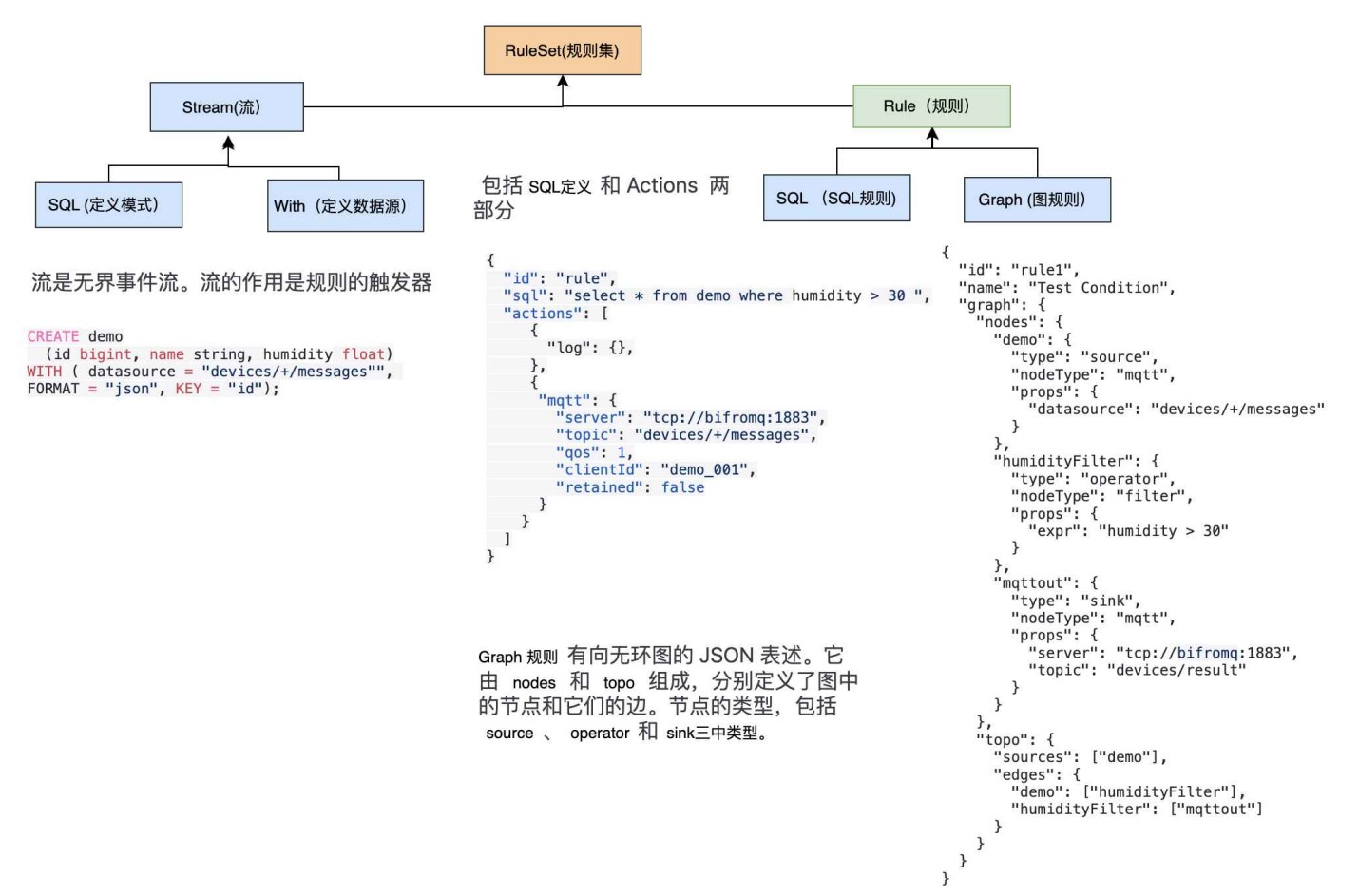
1. **节点启动：**在节点启动时，通过协调器的Restful API获取全局启动配置和数据源配置，同时完成节点的注册，并启动健康状态上报和数据更新协程。
2. **健康状态上报和数据更新协程：**负责将节点的健康状态上报给协调器。同时，如果管理节点上的数据源配置有更新，将同步更新工作节点的状态和数据源配置。

3. **Restful API协程**：提供Restful API接口，与协调器进行通信，负责提交规则任务、查询规则任务的执行状态，以及控制规则任务的启动、停止和删除。

通过这些组件和流程，协调器确保了系统的高效运行和灵活的任务管理，同时也提供了强大的监控和控制能力。

三、规则设计

规则整体结构如下图：



规则集（RuleSet) 规则集能够定义复杂的事件处理逻辑，从数据的接收、处理到最终的响应，形成一个完整的数据处理流程。一个规则集包括 多条Stream（流） 和 多条Rule（规则） 组成。

Stream（流）

流定义需要指定其数据源类型以定义与外部资源的连接方式。数据源作为流使用时，源必须为无界的。在规则中，流的行为类似事件触发器。每个事件都会触发规则的一次计算。

1.1.1 流定义

```
1 CREATE STREAM
2   stream_name
3   ( column_name <data_type> [ ,...n ] )
```

```
4      WITH ( property_name = expression [, ...] );
```

流定义是一个 SQL 语句。它由两部分组成。

- 流的模式定义。其语法与 SQL 表定义相同。这里的模式是可选的。如果它是空的，则流是无模式的。
- 在 WITH 子句中定义连接器类型和行为的属性，如序列化格式。

1.1.2 数据类型

#	数据类型	说明
1	bigint	整数型。
2	float	浮点型。
3	string	文本值，由 Unicode 字符组成。
4	datetime	日期时间类型。
5	boolean	布尔类型，值可以是true 或者 false。
6	bytea	用于存储二进制数据的字节数组。如果在格式为 "JSON" 的流中使用此类型，则传入的数据需要为 base64 编码的字符串。
7	array	数组类型可以是任何简单类型，数组类型或结构类型。
8	struct	复杂类型。

1.1.3 流属性

属性名称	可选	说明
DATASOURCE	否	取决于不同的源类型；如果是 MQTT 源，则为 MQTT 数据源主题名；其它源请参考相关的文档。
FORMAT	是	传入的数据类型，支持 "JSON", "PROTOBUF" 和 "BINARY"，默认为 "JSON"。关于 "BINARY" 类型的更多信息，请参阅 Binary Stream 。该属性是否生效取决于源的类型，某些源自自身解析的时固定私有格式的数据，则该配置不起作用。可支持该属性的源包括 MQTT 和 ZMQ 等。
DELIMITER	是	仅在使用 delimited 格式时生效，用于指定分隔符，默认为逗号。
KEY	是	保留配置，当前未使用该字段。 它将用于 GROUP BY 语句。
TYPE	是	源类型，如未指定，值为 "mqtt"。
CONF_KEY	是	如果需要配置其他配置项，请在此处指定 config 键。 有关更多信息，请参见 MQTT stream 。
TIMESTAMP	是	代表该事件时间戳的字段名。如果有设置，则使用此流的规则将采用事件时间；否则将采用处理时间。详情请看 时间戳管理 。
TIMESTAMP_FORMAT	是	字符串和时间格式转换时使用的默认格式。

1.1.4 样例

```
1 CREATE STREAM demo (  
2     USERID BIGINT,  
3     FIRST_NAME STRING,  
4     LAST_NAME STRING,  
5     NICKNAMES ARRAY(STRING),  
6     Gender BOOLEAN,  
7     ADDRESS STRUCT(STREET_NAME STRING, NUMBER BIGINT),  
8 ) WITH (DATASOURCE="test/", FORMAT="JSON", KEY="USERID", CONF_KEY="demo");
```

Rule(规则)

每条规则都代表了运行的一项计算工作。它定义了连续流数据源作为输入，计算逻辑和结果 sink 作为输出。

1.1.1 参数

参数名	是否可选	说明
id	否	规则 id, 规则 id 在同一 eKuiper 实例中必须唯一。
name	是	规则显示的名字或者描述。
sql	如果 graph 未定义, 则该属性必须定义	为规则运行的 sql 查询
actions	如果 graph 未定义, 则该属性必须定义	Sink 动作数组
graph	如果 sql 未定义, 则该属性必须定义	规则有向无环图的 JSON 表示
options	是	选项列表

1.1.2 SQL 规则

通过指定 `sql` 和 `actions` 属性, `sql` 定义了针对预定义流运行的 SQL 查询, 这将转换数据。然后, 输出的数据可以通过 `action` 路由到多个位置。

```
1 {
2   "id": "rule1",
3   "sql": "SELECT demo.temperature, demo1.temp FROM demo left join demo1 on
4         demo.timestamp = demo1.timestamp where demo.temperature > demo1.temp GROUP BY
5         demo.temperature, HOPPINGWINDOW(ss, 20, 10)",
6   "actions": [
7     {
8       "log": {}
9     },
10    {
11      "mqtt": {
12        "server": "tcp://47.52.67.87:1883",
13        "topic": "demoSink"
14      }
15    ]
16  }
```

1.1.3 图规则

`graph` 属性是有向无环图的 JSON 表述。它由 `nodes` 和 `topo` 组成, 分别定义了图中的节点和它们的边。下面是一个由图形定义的最简单的规则。它定义了3个节点: `demo`, `humidityFilter` 和 `mqttOut`。这个图是线性的, 即 `demo -> humidityFilter -> mqttOut`。该规则将从mqtt(`demo`)读取, 通过湿度过滤(`humidityFilter`)并汇入mqtt(`mqttOut`)。

```

1 {
2   "id": "rule1",
3   "name": "Test Condition",
4   "graph": {
5     "nodes": {
6       "demo": {
7         "type": "source",
8         "nodeType": "mqtt",
9         "props": {
10          "datasource": "devices/+/messages"
11        }
12      },
13      "humidityFilter": {
14        "type": "operator",
15        "nodeType": "filter",
16        "props": {
17          "expr": "humidity > 30"
18        }
19      },
20      "mqttout": {
21        "type": "sink",
22        "nodeType": "mqtt",
23        "props": {
24          "server": "tcp://${mqtt_srv}:1883",
25          "topic": "devices/result"
26        }
27      }
28    },
29    "topo": {
30      "sources": ["demo"],
31      "edges": {
32        "demo": ["humidityFilter"],
33        "humidityFilter": ["mqttout"]
34      }
35    }
36  }
37 }

```

节点

图的 JSON 中的每个节点至少有3个字段：

- type：节点的类型，可以是 `source`、`operator` 和 `sink`。
- nodeType：节点的实现类型，定义了节点的业务逻辑，包括内置类型和由插件定义的扩展类型。
- props：节点的属性。它对每个 nodeType 都是不同的。

节点类型

源节点：

- [Mqtt source](#)：从mqtt 主题读取数据。
- [Redis source](#)：从 Redis 中查询数据，用作查询表。
- [RedisSub source](#)：从 Redis 频道中订阅数据。
- [Memory source](#)：从 内存主题读取数据以形成规则管道。
- [Http pull source](#)：从 http 服务器中拉取数据。
- [Simulator source](#)：生成模拟数据，用于测试。
- [SQL source](#)：定期从关系数据库中拉取数据。
- [Kafka source](#)：从 Kafka 中读取数据

Sink 节点：

用户可以直接使用标准 eKuiper 实例中的内置动作。内建动作的列表如下：

- Mqtt sink：输出到外部 mqtt 服务。
- Rest sink：输出到外部 http 服务器。
- Redis sink：写入 Redis 。
- RedisPub sink：输出到 Redis 消息频道。
- Memory sink：输出到内存主题以形成规则管道。
- Log sink：写入日志，通常只用于调试。
- Nop sink：不输出，用于性能测试。
- SQL Sink

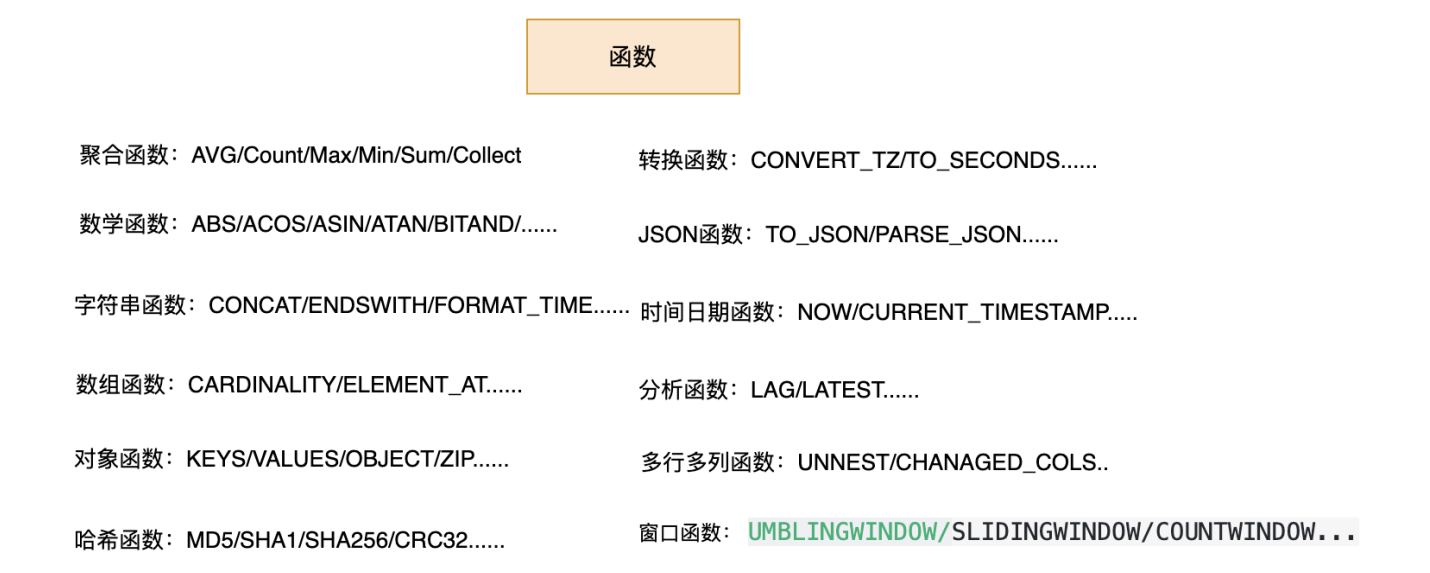
operator 节点类型：

- 函数
- aggfunc
- 过滤
- pick
- 窗口
- join
- groupby
- orderby
- switch

- script

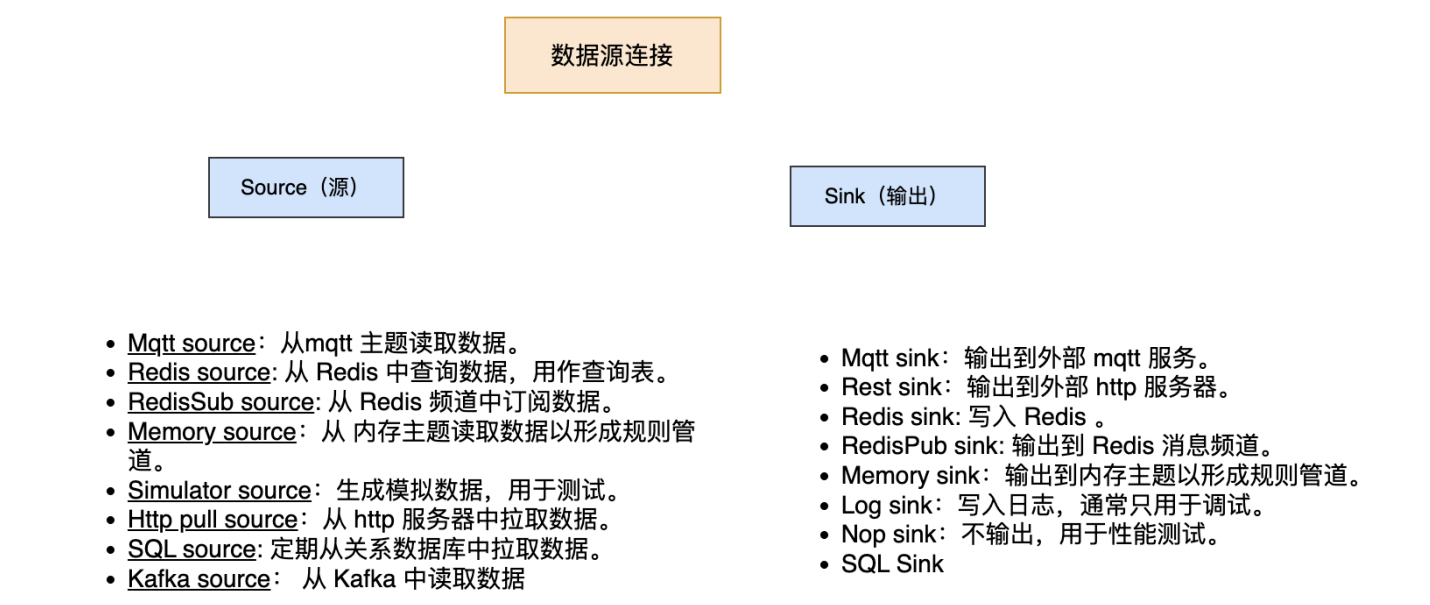
四、函数 Function

支持函数如下图：



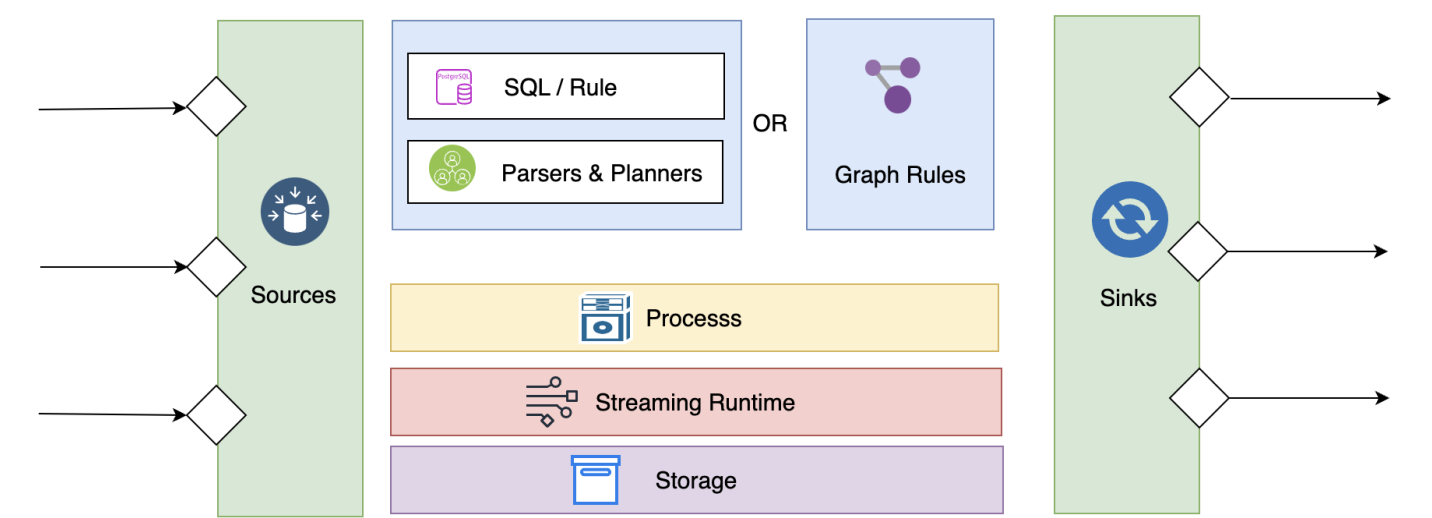
五、数据连接

支持 Source 和 Sink 如下图：



六、执行器

执行器是基于开源 eKuiper 二次开发进行构建，eKuiper 是 Golang 实现的轻量级物联网边缘分析、流式处理开源软件，可以运行在各类资源受限的边缘设备上。eKuiper 的规则引擎是基于 SQL 或基于图形（类似于 Node-RED）的规则。



eKuiper 通过规则/SQL 解析器或图规则解析器将解析、规划和优化规则，使其成为一系列算子的流程，算子可以利用流式运行时和状态存储。算子之间通过 Go 通道进行异步通信。

受益于 Go 的并发模型，规则运行时可以做到：

- 以异步和非阻塞的方式进行通信。
- 充分利用多核计算。
- 算子层可伸缩。
- 规则之间相互隔离。

在 eKuiper 中，计算工作以规则的形式呈现。规则以流的数据源为输入，通过 SQL 定义计算逻辑，将结果输出到动作/sink 中。

规则定义提交后，它将持续运行。它将不断从源获取数据，根据 SQL 逻辑进行计算，并根据结果触发行动。核心计算组件包括 Rule、Source、Sink。

六、部署

在项目 deploy/docker 目录下有个 docker-compose.yaml 文件，内容如下：

```
1 version: '3.9'
2
```

```
3 networks:
4   bifromq-net:
5     external: false
6
7 services:
8   db:
9     image: registry.cn-hangzhou.aliyuncs.com/2456868764/mysql:5.7 # 使用MySQL
    5.7镜像, 你可以选择其他版本
10    networks:
11      - bifromq-net
12    ports:
13      - "3308:3306/tcp"
14    volumes:
15      - ./data/mysql:/var/lib/mysql # 将数据库文件存储在卷中, 以便持久化存储
16      - ./data/init:/docker-entrypoint-initdb.d
17    restart: always # 容器退出时总是重启
18    environment:
19      MYSQL_ROOT_PASSWORD: 123456 # 设置root用户的密码
20      MYSQL_DATABASE: engine # 创建并初始化一个数据库
21      MYSQL_USER: dev # 创建一个新用户
22      MYSQL_PASSWORD: 123456 # 设置新用户的密码
23
24   bifromq_engine:
25     image: registry.cn-hangzhou.aliyuncs.com/2456868764/bifromq_engine:v1.0.0
26     command: ["serve", "--api-port=8080", "--coordinator-port=8081", "--
    dns=root:123456@tcp(db:3306)/engine?charset=utf8mb4&parseTime=True&loc=Local"]
27     environment:
28       - JWT_SIGNING_KEY=bifromq
29     networks:
30       - bifromq-net
31     ports:
32       - "9080:8080/tcp"
33       - "8081:8081/tcp"
34     volumes:
35       - ./data/engine:/data
36     restart: always
37     depends_on:
38       - db
39   bifromq_ui:
40     image: registry.cn-hangzhou.aliyuncs.com/2456868764/bifromq_ui:v1.0.1
41     environment:
42       - ACCESS_CODE=lobe66
43     networks:
44       - bifromq-net
45     ports:
46       - "8090:80/tcp"
47     restart: always
```

```
48     depends_on:
49         - db
50         - bifromq_engine
51     bifromq-server:
52         image: registry.cn-hangzhou.aliyuncs.com/2456868764/bifromq:latest
53         networks:
54             - bifromq-net
55         ports:
56             - "1883:1883/tcp"
57         restart: always
58     redis-server:
59         image: registry.cn-hangzhou.aliyuncs.com/2456868764/redis:latest # 使用最新
版本的Redis镜像
60         environment:
61             - ALLOW_EMPTY_PASSWORD=yes
62         networks:
63             - bifromq-net
64         volumes:
65             - ./data/redis:/bitnami/redis # 持久化Redis数据
66         ports:
67             - "6379:6379" # 将容器的6379端口映射到宿主机的6379端口
68         restart: always # 容器退出时总是重启
69 # Kafka服务定义
70     kafka-server:
71         image: registry.cn-hangzhou.aliyuncs.com/2456868764/kafka:latest
72         networks:
73             - bifromq-net
74         volumes:
75             - ./data/kafka:/bitnami/kafka
76         ports:
77             - '9092:9092'
78         environment:
79             - KAFKA_CFG_NODE_ID=0
80             - KAFKA_CFG_PROCESS_ROLES=controller,broker
81             - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092,CONTROLLER://:9093
82             -
KAFKA_CFG_LISTENER_SECURITY_PROTOCOL_MAP=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTE
XT
83             - KAFKA_CFG_CONTROLLER_QUORUM_VOTERS=0@kafka-server:9093
84             - KAFKA_CFG_CONTROLLER_LISTENER_NAMES=CONTROLLER
85 # bifromq rule engine job
86     bifromq-rule-engine-joba:
87         image: registry.cn-hangzhou.aliyuncs.com/2456868764/ekuiperd:v1.0.0
88         environment:
89             - NODE_IP=bifromq-rule-engine-joba
90             - NODE_PORT=9081
91             - NODE_NAME=bifromq-rule-engine-joba
```

```




92     - NODE_TAG=job,joba
93     - COORDINATOR_HOST=bifromq_engine:8081
94     networks:
95     - bifromq-net
96     restart: always
97     depends_on:
98     - db
99     - bifromq_engine
100  bifromq-rule-engine-jobb:
101     image: registry.cn-hangzhou.aliyuncs.com/2456868764/ekuiperd:v1.0.0
102     environment:
103     - NODE_IP=bifromq-rule-engine-jobb
104     - NODE_POR=9081
105     - NODE_NAME=bifromq-rule-engine-jobb
106     - NODE_TAG=job,jobb
107     - COORDINATOR_HOST=bifromq_engine:8081
108     networks:
109     - bifromq-net
110     restart: always
111     depends_on:
112     - db
113     - bifromq_engine

```

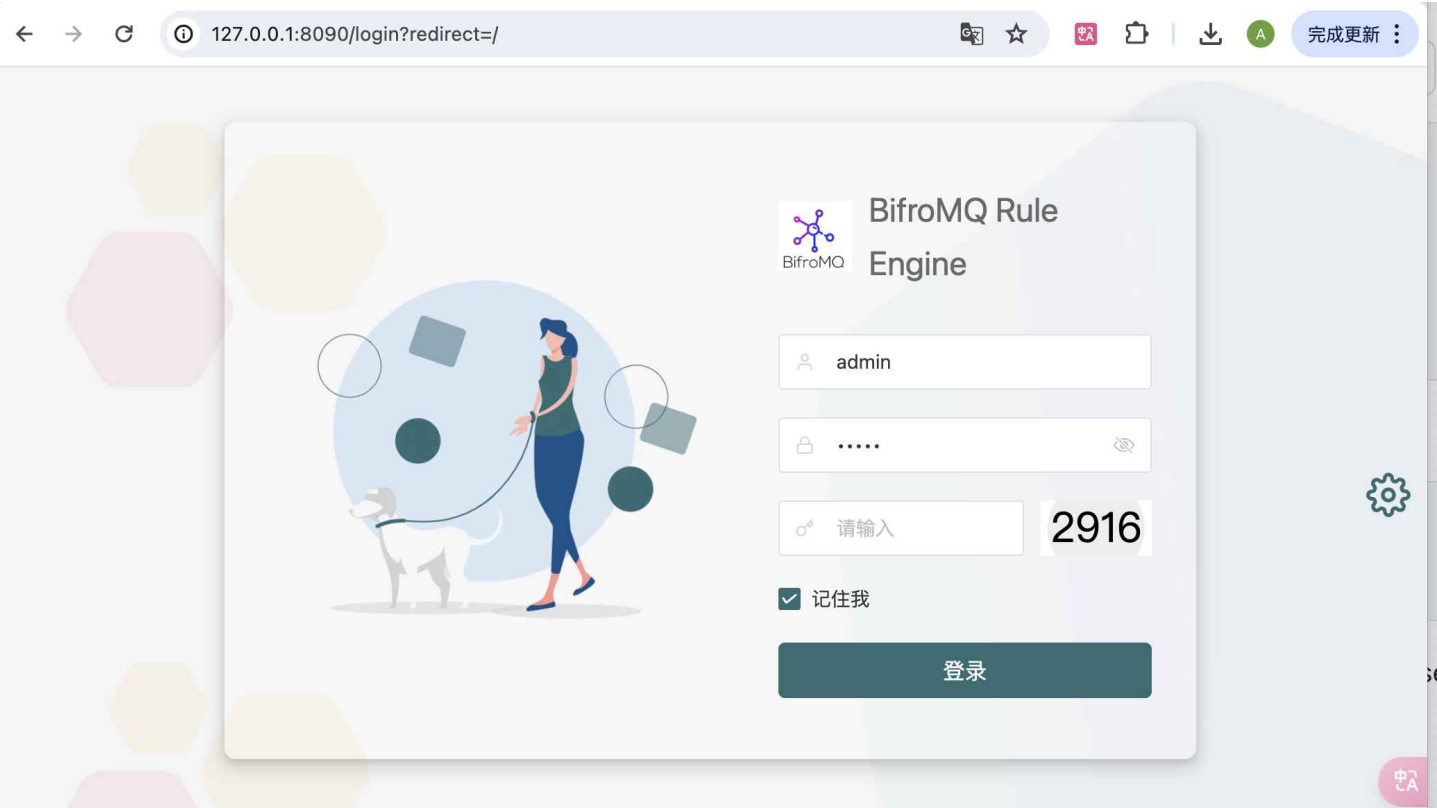
启动命令：

```
1 docker compose up -d
```

启动 包括 数据库mysql db, 管理节点 bifromq engine, 控制台 bifromq ui, bifromq broker server, redis server , kafka server, 工作节点 A： rule engine job a ,工作节点 B： rule engine job b 容器。

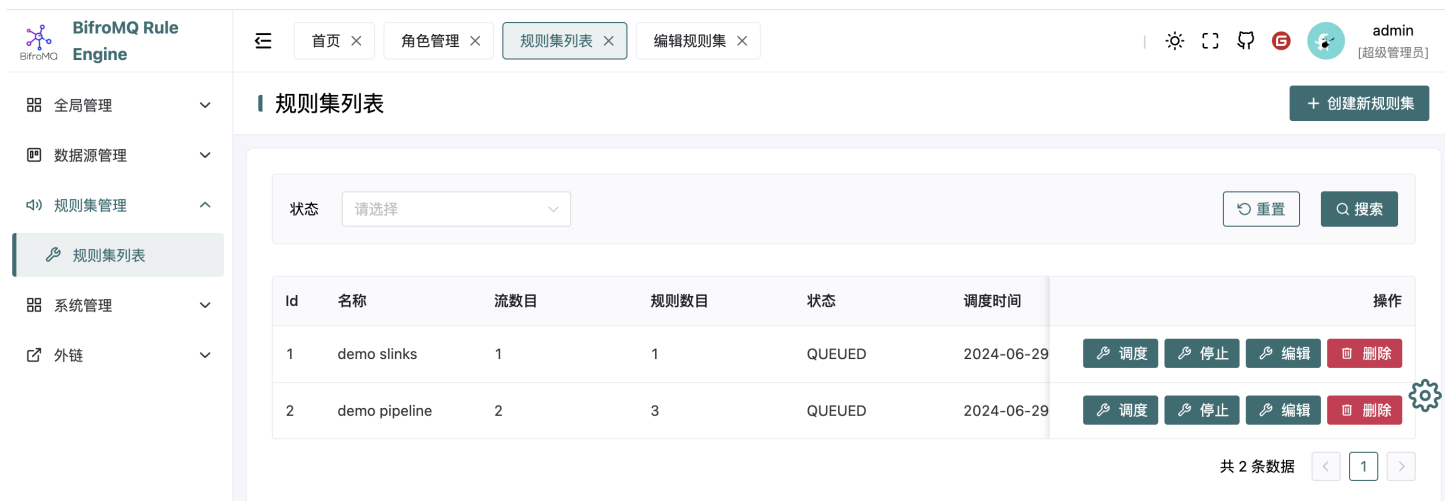
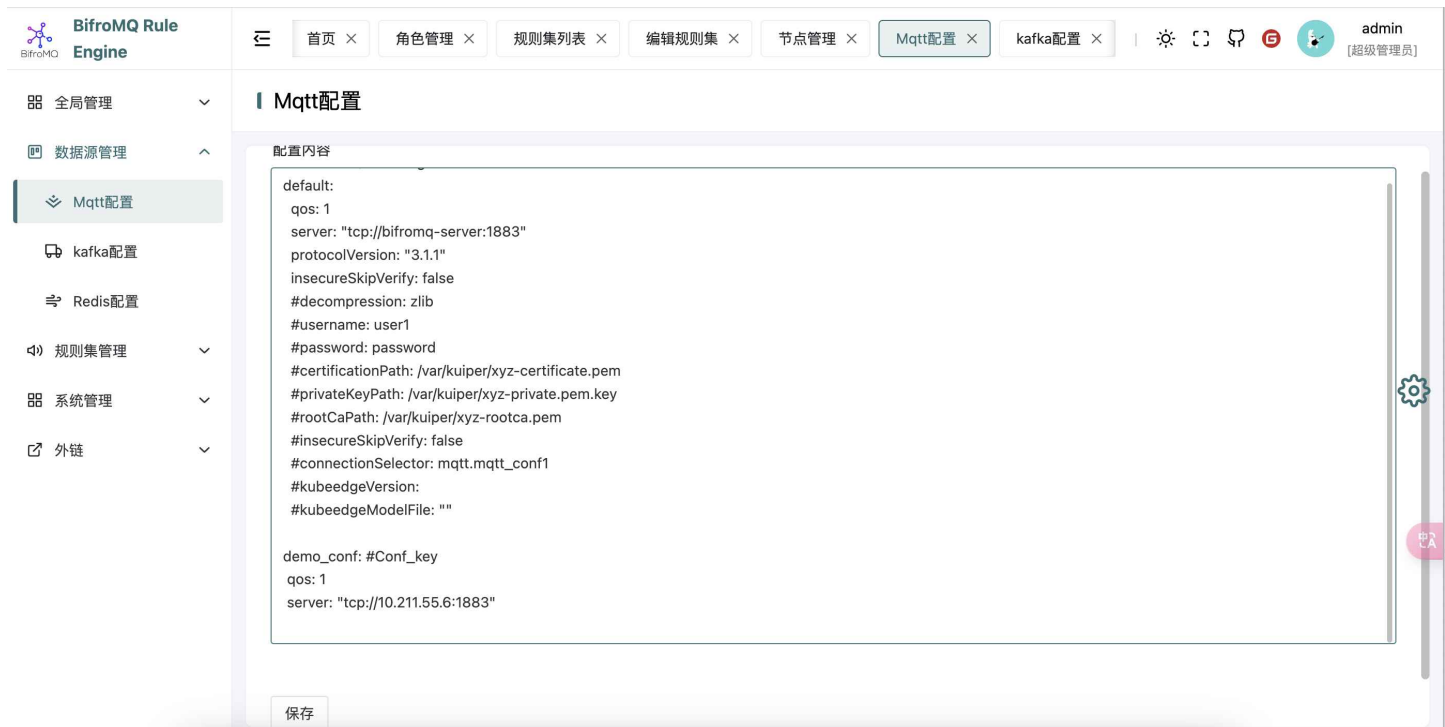
<input type="checkbox"/>	<div><div>▼</div><div> docker</div></div>	Running (7/7)	0 seconds ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>redis-server-1</div><div>a2615a780543 </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>	<div>6379:6379 </div>	1 minute ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>db-1</div><div>26aec781a4f0 </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>	<div>3308:3306 </div>	1 minute ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>kafka-server-1</div><div>c45c2363699e </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>	<div>9092:9092 </div>	1 minute ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>bifromq-server-1</div><div>13c543bc4331 </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>	<div>1883:1883 </div>	1 minute ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>bifromq_engine-1</div><div>dddcfa409e7 </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>	<div><div>8081:8081 </div><div>Show all ports (2)</div></div>	9 seconds ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>bifromq-rule-engine-jol</div><div>7e648778fdb </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>		0 seconds ago	<div><div></div><div>⋮</div><div>🗑</div></div>
<input type="checkbox"/>	<div><div></div><div><div>bifromq-rule-engine-jol</div><div>5b059c2d006e </div></div></div> <div><div>registry.cn-hangzhou.aliyuncs.cc</div><div>Running</div></div>		0 seconds ago	<div><div></div><div>⋮</div><div>🗑</div></div>

访问 <http://127.0.0.1:8090> 进入 bifromq 控制台管理界面，账号和密码 admin/admin。



七、控制台



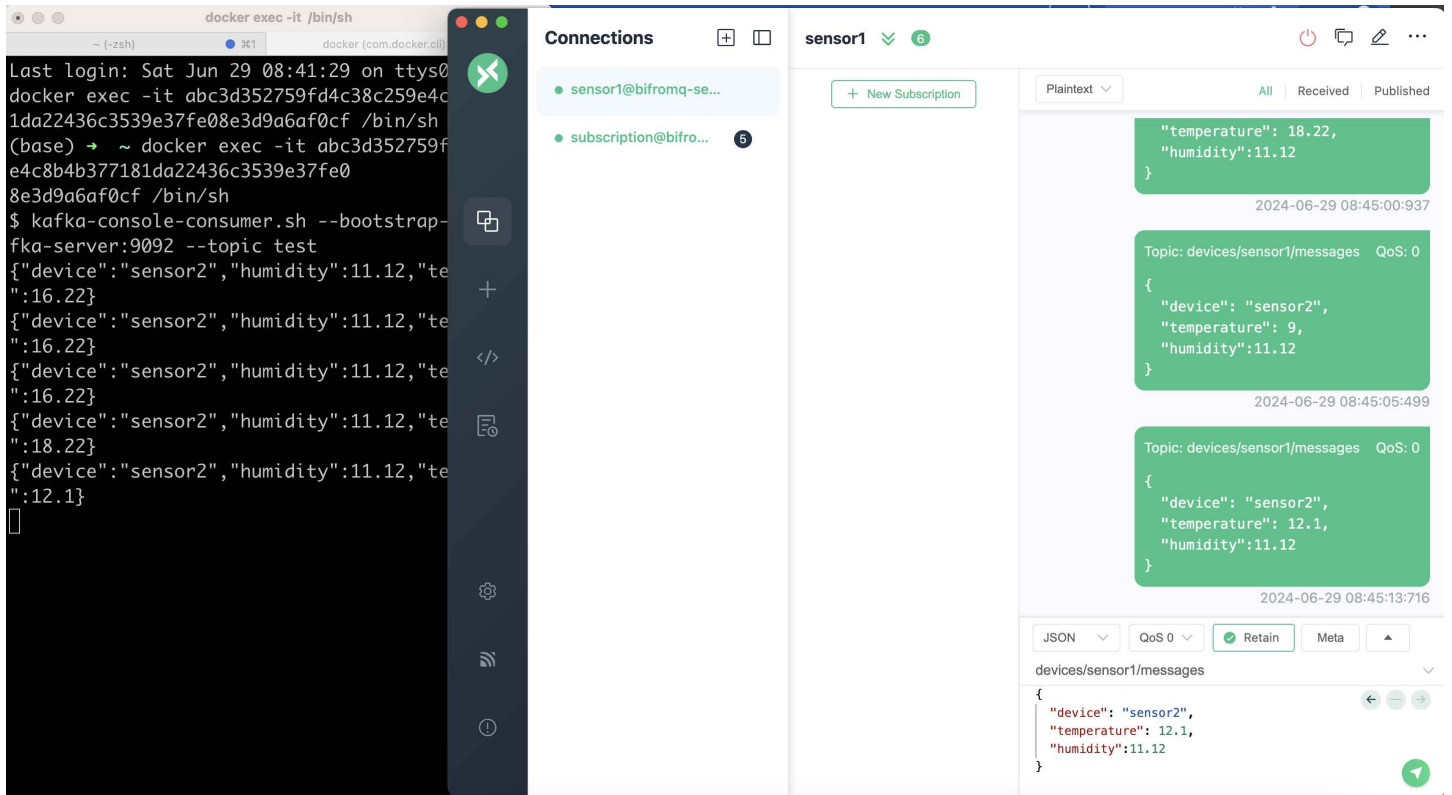


八、案例

1. Demo Slink

```
1 #1 创建源流
2
3 CREAT STREAM demo (device string, temperature float, humidity float) WITH
  (FORMAT="JSON", DATASOURCE="devices/+/messages")
4
5 #2 创建规则和目标
6 {
7   "id": "demo",
8   "sql": "select * from demo where temperature > 10",
9   "actions": [
10    {
11      "log": {}
12    },
13    {
14      "mqtt": {
15        "server": "tcp://bifromq-server:1883",
16        "topic": "devices/allmessages"
17      }
18    },
19    {
20      "kafka": {
21        "brokers": "kafka-server:9092",
22        "topic": "test",
23        "saslAuthType": "none"
24      }
25    }
26  ]
27 }
```

测试结果：



2. 规则管道 - pipeline

以通过将先前规则的结果导入后续规则来形成规则管道。这可以通过使用中间存储或 MQ（例如 mqtt 消息服务器）来实现。通过同时使用 [内存源](#) 和 [目标](#)，我们可以创建没有外部依赖的规则管道。

```
1 #1 创建源流
2
3 CREAT STREAM pipeline (device string, temperature float, humidity float) WITH
  (DATASOURCE="devices/pipeline", FORMAT="JSON")"
4
5
6 #2 创建规则和内存目标
7 {
8   "id": "pipeline-rule1",
9   "sql": "SELECT * FROM pipeline WHERE isNull(temperature)=false",
10  "actions": [
11    {
12      "log": {}
13    },
14    {
15      "memory": {
16        "topic": "devices/ch1/sensor1"
17      }
18    }
19  ]
```

```

20 }
21 #3 从内存主题创建一个流
22
23 CREATE STREAM sensor1 (temperature FLOAT, humidity FLOAT)
24 WITH (DATASOURCE="devices/+/sensor1", FORMAT="JSON", TYPE="memory")
25
26 #4 从内存主题创建另一个要使用的规则
27 {
28   "id": "rule2-1",
29   "sql": "SELECT avg(temperature) FROM sensor1 GROUP BY CountWindow(10)",
30   "actions": [
31     {
32       "log": {}
33     },
34     {
35       "memory": {
36         "topic": "analytic/sensors"
37       }
38     }
39   ]
40 }
41
42 {
43   "id": "rule2-2",
44   "sql": "SELECT temperature + 273.15 as k FROM sensor1",
45   "actions": [
46     {
47       "log": {}
48     }
49   ]
50 }

```

结果:

```

1 time="2024-06-29T09:45:52+08:00" level=info msg="sink result for rule pipeline-
  rule1: [{\"device\":\"sensor2\", \"humidity\":11.12, \"temperature\":12.1}]"
  file="sink/log_sink.go:32" rule=pipeline-rule1
2 time="2024-06-29T09:45:52+08:00" level=info msg="sink result for rule rule2-2:
  [{\"k\":285.25}]" file="sink/log_sink.go:32" rule=rule2-2
3 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule pipeline-
  rule1: [{\"device\":\"sensor2\", \"humidity\":11.12, \"temperature\":12.1}]"
  file="sink/log_sink.go:32" rule=pipeline-rule1
4 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule pipeline-
  rule1: [{\"device\":\"sensor2\", \"humidity\":11.12, \"temperature\":12.1}]"
  file="sink/log_sink.go:32" rule=pipeline-rule1

```

```
5 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule rule2-2:
  [{"k\":\"285.25}]" file="sink/log_sink.go:32" rule=rule2-2
6 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule rule2-2:
  [{"k\":\"285.25}]" file="sink/log_sink.go:32" rule=rule2-2
7 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule rule2-2:
  [{"k\":\"285.25}]" file="sink/log_sink.go:32" rule=rule2-2
8 time="2024-06-29T09:45:53+08:00" level=info msg="sink result for rule pipeline-
  rule1: [{"device\":\"sensor2\", \"humidity\":11.12, \"temperature\":12.1}]"
  file="sink/log_sink.go:32" rule=pipeline-rule1
```

九、项目代码和进度

2.1 项目代码

管理节点和控制台：

1. https://atomgit.com/haoyu/bifromq_engine (atomgit)
2. <https://github.com/2456868764/mqtt-rabbit> (github)

ekuiper 二次开发代码分支

1. <https://github.com/2456868764/ekuiper/tree/feat-engine>

2. Prebuild

build 项目之前，需要下载项目 ekuiper 特定分支到本地 external 目录执行 make prebuild

```
1 ## prebuild
2 make prebuild
```

3. 项目进度

1. 整体架构完成
2. 正在逐步完善控制台

