# 基于消息队列MOM的分布式信息采集分析系统

## 一、技术栈

系统使用Golang和TypeScript作为主力开发语言，借助Makefile作为项目构建工具，使用RabbitMQ作为消息中间件，使用AMQP作为Go中对MQ使用的封装，使用Gin提供前端页面服务和WebSocket API，使用GORM提供数据持久化，以便于进行长序列的数据分析，并且保证系统主要部分都是无状态服务，具有良好的可扩展性和容错性。

## 二、系统架构

系统分为三个主要部分：

- 数据收集的gather部分

- 数据分析的analysis部分

- 数据展示的display部分

最后的display部分使用了前后端架构，并借助WebSocket实现了数据的实时刷新。同时借助Go的编程模型，将WebSocket和AMQP两个组件组合起来，形成了一个通用的数据展示接口。

数据展示的前端部分使用了已经成熟的Vue3+ECharts来完成。借助高阶函数和闭包等语言特性，前端部分的代码量大幅减少。

数据收集部分，使用Go的携程模型实现了多个并发的模拟收集数据的MQ客户端。

数据分析部分，借助GORM操作数据库，实现了数据的持久化存储，以便获取历史数据来完成极值、均值和方差的计算。同时，这部分从一个消息队列获取数据，再将计算结果存储到另一个消息队列，同时使用数据库持久化分析结果。

整个系统使用基于Socket封装的MQ，进行相对松散地组合，保证了系统的灵活性和扩展性。

## 三、系统演示

系统可以使用docker-compose直接部署，配置文件如下：

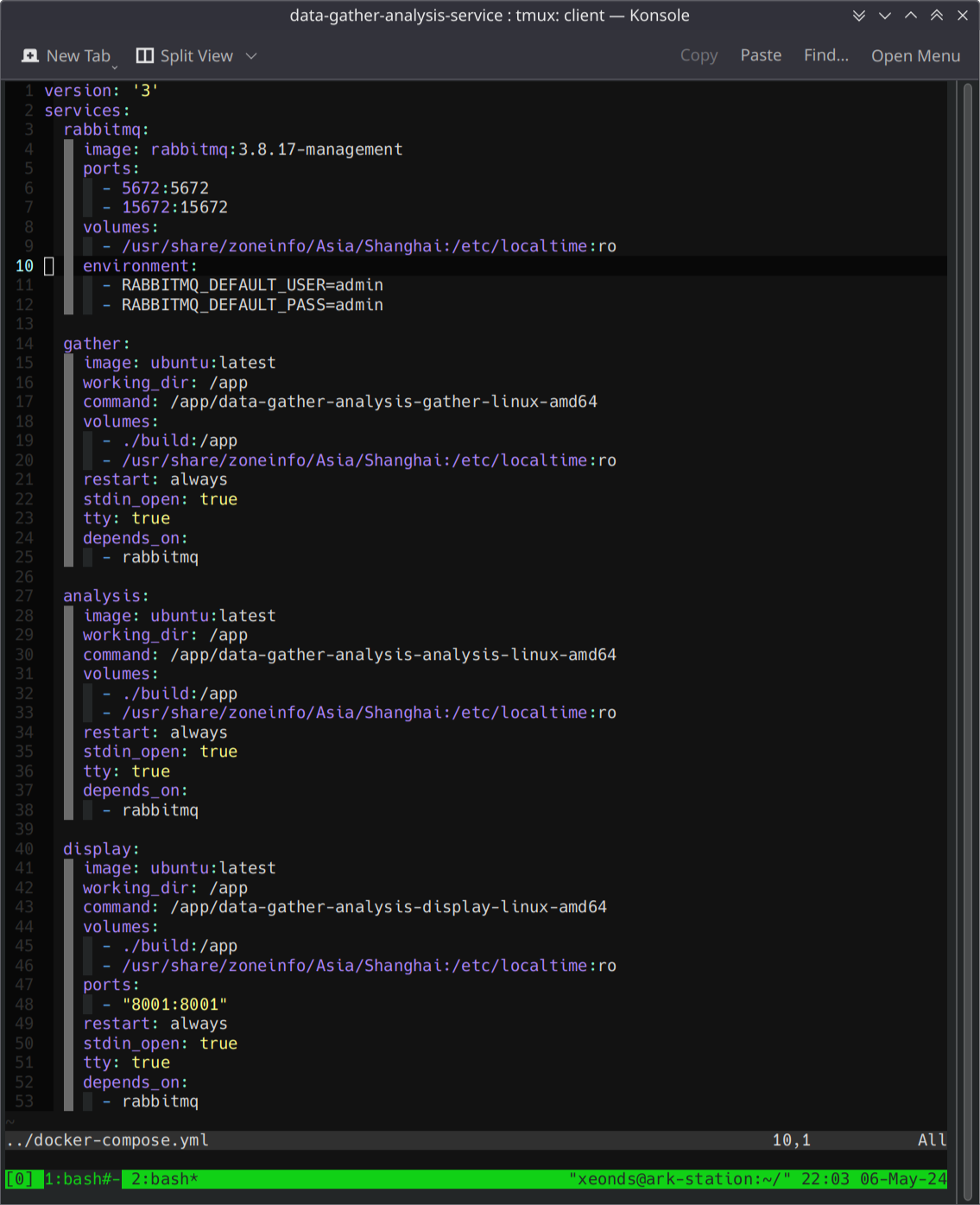


图 1 docker-compose.yml

可以使用如下的配置文件来启动系统：

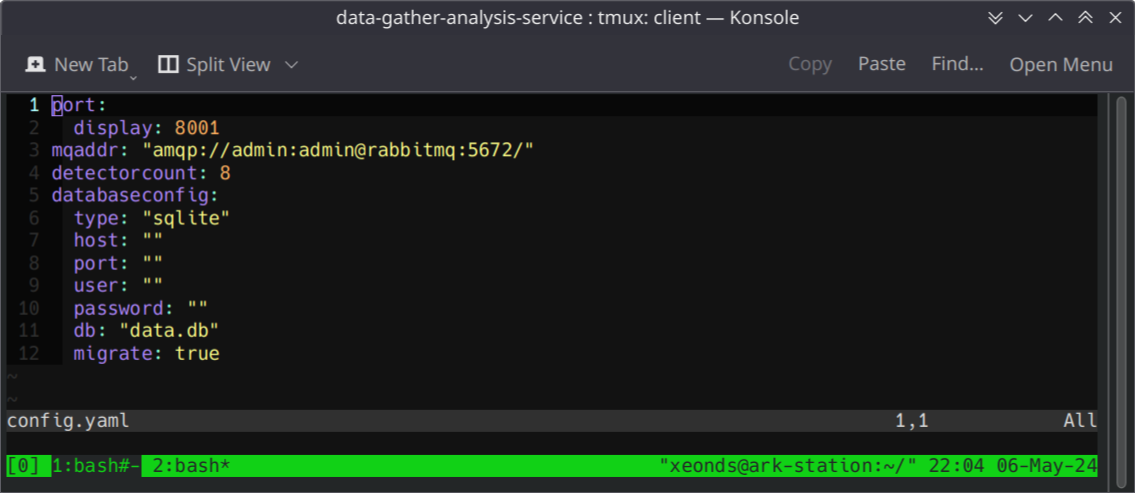


图 2 config.yaml

启动后，后端部分效果如图所示：

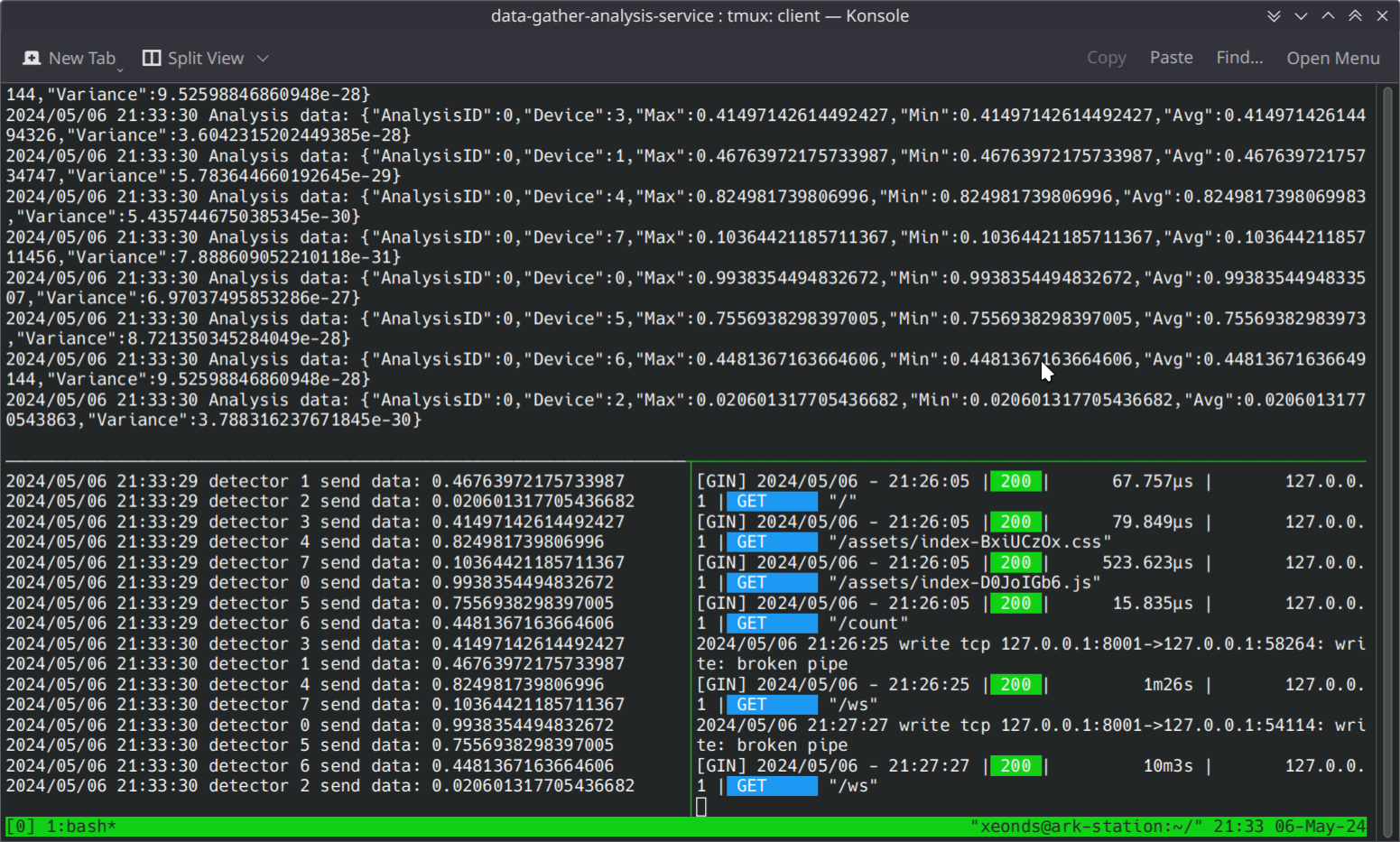


图 3 系统数据采集、数据分析、数据展示（后端）三部分运行中

此时打开浏览器，访问http://localhost:8001/即可打开数据展示面板：

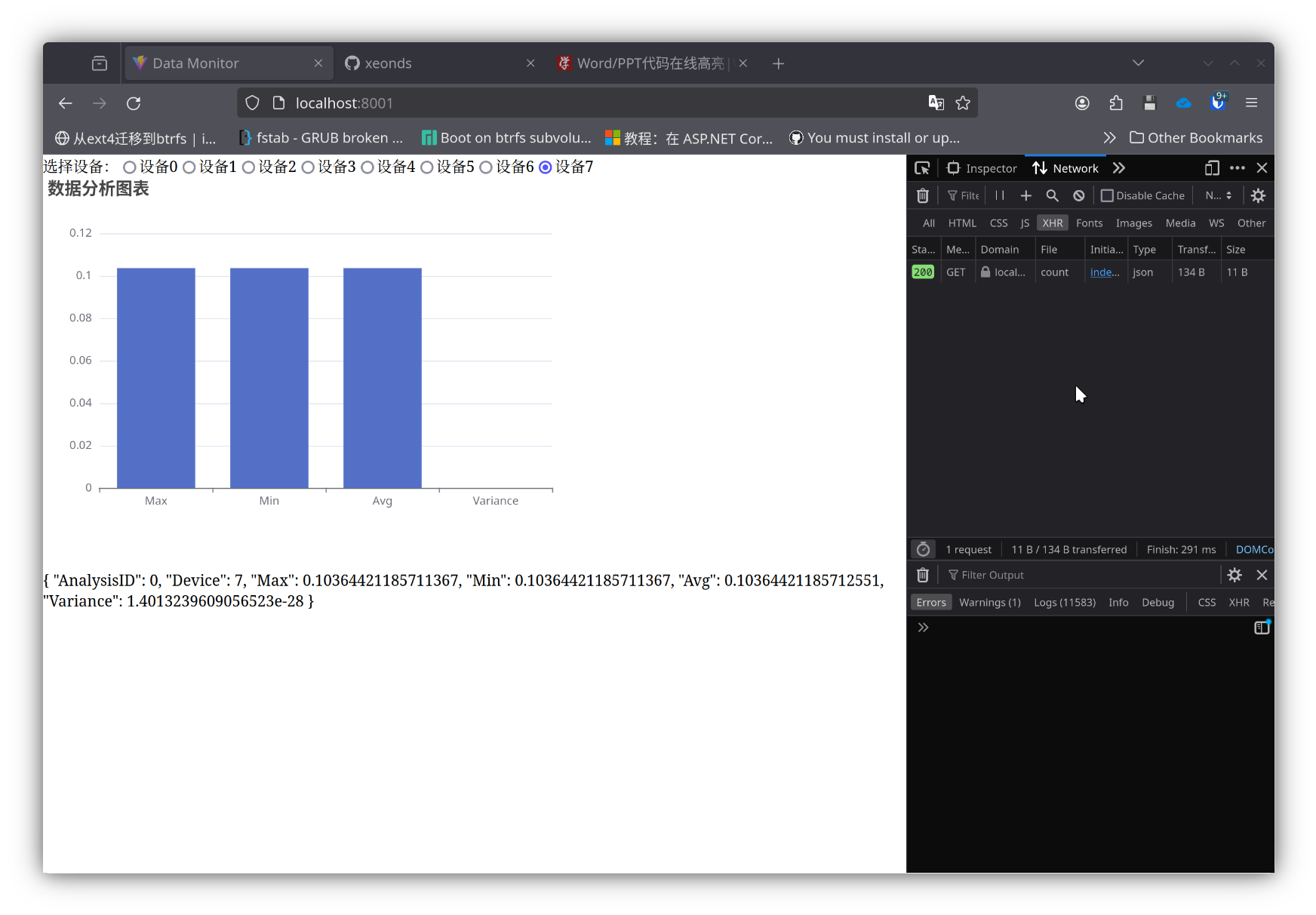


图 4 数据展示面板

最上边能选择后端的数据采集客户端，下面分别是该客户端的分析值统计图，以及详细的数值内容。

## 四、源代码

前端部分节选：

1. <script lang="ts" setup>
2. import { ref, onMounted, watch, Ref } from 'vue';
3. import \* as echarts from 'echarts';
5. const selectedDevice: Ref<any> = ref(0);
6. const deviceList: Ref<any[]> = ref([]);
7. const analysisData: Ref<any> = ref({});
8. const chart: Ref<any> = ref(null);
9. const ws: Ref<WebSocket> = ref({} as WebSocket)
10. const url = ref('ws://localhost:8001/ws')
11. const data: Ref<any> = ref({})
13. const fetchDeviceList = () =>
14. fetch('/count')
15. .then(response => response.json())
16. .then(data => deviceList.value = Array.from({ length: data.count }, (\_, i) => ({ id: i, name: `设备${i}` })))
17. .catch(error => console.error('Error fetching device list:', error));
19. const initWebSocket = (addr: string, callback: Function) => {
20. const socket = new WebSocket(addr)
21. socket.addEventListener('open', () => {
22. console.log('connected')
23. })
24. socket.addEventListener('close', () => {
25. console.log('disconnected')
26. })
27. socket.addEventListener('message', (event) => {
28. console.log(event.data);
29. callback(event.data);
30. })
31. return socket
32. }
34. const conn = () => {
35. ws.value = initWebSocket(
36. url.value,
37. (e: string) => {
38. const parsed = JSON.parse(e)
39. data.value[parsed.Device] = parsed
40. analysisData.value = data.value[selectedDevice.value]
41. })
42. }
44. watch(selectedDevice, () => analysisData.value = data.value[selectedDevice.value]);
46. watch(analysisData, () => {
47. if (analysisData.value) chart.value.setOption({
48. title: { text: '数据分析图表' },
49. xAxis: {
50. type: 'category',
51. data: ['Max', 'Min', 'Avg', 'Variance']
52. },
53. yAxis: { type: 'value' },
54. series: [{
55. data: [analysisData.value.Max, analysisData.value.Min, analysisData.value.Avg, analysisData.value.Variance],
56. type: 'bar'
57. }]
58. });
59. }, { deep: true });
61. onMounted(() => {
62. fetchDeviceList();
63. chart.value = echarts.init(document.getElementById('chart') as HTMLDivElement);
64. conn();
65. });
66. </script>

数据采集部分源码：

1. package main
3. import (
4. "data-gather-analysis-service/config"
5. "data-gather-analysis-service/lib"
6. "data-gather-analysis-service/model"
7. "encoding/json"
8. "log"
9. "math/rand"
10. "time"
12. "github.com/streadway/amqp"
13. )
15. func detector(id int) func(\*amqp.Channel, amqp.Queue) {
16. return func(ch \*amqp.Channel, q amqp.Queue) {
17. msg := model.Data{
18. Device: id,
19. Data:   rand.Float64(),
20. }
21. data, err := json.Marshal(msg)
22. if err != nil {
23. log.Println(err)
24. }
26. for range time.Tick(time.Second) {
27. if err = ch.Publish(
28. "",
29. q.Name,
30. false,
31. false,
32. amqp.Publishing{
33. ContentType: "application/json",
34. Body:        data,
35. },
36. ); err != nil {
37. log.Println(err)
38. }
39. log.Println("detector", id, "send data:", msg.Data)
40. }
41. }
42. }
44. func main() {
45. config := lib.LoadConfig[config.Config]()
46. conn, err := amqp.Dial(config.MQaddr)
47. if err != nil {
48. panic(err)
49. }
50. log.Println("connect to rabbitmq success")
51. defer conn.Close()
53. ch, err := conn.Channel()
54. if err != nil {
55. panic(err)
56. }
57. defer ch.Close()
58. log.Println("open channel success")
60. q, err := ch.QueueDeclare(
61. "data\_queue",
62. false,
63. false,
64. false,
65. false,
66. nil,
67. )
68. if err != nil {
69. log.Println(err)
70. }
71. log.Println("declare queue success")
73. // 根据配置模拟若干个采集终端
74. log.Println(config.DetectorCount, "detector(s) will be started")
75. for i := 0; i < config.DetectorCount; i++ {
76. log.Println("start detector", i)
77. go detector(i)(ch, q)
78. }
80. select {}
81. }

数据分析部分源码：

1. package main
3. import (
4. "data-gather-analysis-service/config"
5. "data-gather-analysis-service/lib"
6. "data-gather-analysis-service/model"
7. "encoding/json"
8. "log"
10. "github.com/streadway/amqp"
11. "gorm.io/gorm"
12. )
14. func main() {
15. config := lib.LoadConfig[config.Config]()
16. db := lib.NewDB(&config.DatabaseConfig, func(db \*gorm.DB) error {
17. return db.AutoMigrate(&model.Data{}, &model.Analysis{})
18. })
19. conn, err := amqp.Dial(config.MQaddr)
20. if err != nil {
21. log.Println(err)
23. }
24. defer conn.Close()
26. ch, err := conn.Channel()
27. if err != nil {
28. log.Println(err)
29. }
30. defer ch.Close()
32. q, err := ch.QueueDeclare(
33. "analysis\_queue",
34. false,
35. false,
36. false,
37. false,
38. nil,
39. )
40. if err != nil {
41. log.Println(err)
42. }
44. msgs, err := ch.Consume(
45. "data\_queue",
46. "",
47. true,
48. false,
49. false,
50. false,
51. nil,
52. )
53. if err != nil {
54. log.Println(err)
55. }
57. for msg := range msgs {
58. recv := new(model.Data)
59. if err := json.Unmarshal(msg.Body, recv); err != nil {
60. log.Println(err)
61. }
62. result := analysis(db)(recv)
63. data, \_ := json.Marshal(result)
65. if err := ch.Publish(
66. "",
67. q.Name,
68. false,
69. false,
70. amqp.Publishing{
71. ContentType: "application/json",
72. Body:        data,
73. },
74. ); err != nil {
75. log.Println(err)
76. }
77. log.Println("Analysis data:", string(data))
78. }
80. select {}
81. }
83. func analysis(db \*gorm.DB) func(data \*model.Data) \*model.Analysis {
84. return func(data \*model.Data) \*model.Analysis {
85. var sum float64
86. db.Create(&data)
87. dataSet := make([]model.Data, 0)
88. db.Where("device = ?", data.Device).Find(&dataSet)
89. for \_, d := range dataSet {
90. sum += d.Data
91. }
93. avg := sum / float64(len(dataSet))
95. var variance float64
96. for \_, d := range dataSet {
97. variance += (d.Data - avg) \* (d.Data - avg)
98. }
99. variance /= float64(len(dataSet))
101. max, min := 0.0, 0.0
102. if len(dataSet) > 0 {
103. max = dataSet[0].Data
104. min = dataSet[0].Data
105. }
106. for \_, d := range dataSet {
107. if d.Data > max {
108. max = d.Data
109. }
110. if d.Data < min {
111. min = d.Data
112. }
113. }
115. return &model.Analysis{
116. Device:   dataSet[0].Device,
117. Max:      max,
118. Min:      min,
119. Avg:      avg,
120. Variance: variance,
121. }
122. }
123. }

数据展示部分源码（后端）：

1. package main
3. import (
4. "data-gather-analysis-service/config"
5. "data-gather-analysis-service/lib"
6. "fmt"
7. "log"
8. "net/http"
10. "github.com/gin-gonic/gin"
11. "github.com/gorilla/websocket"
12. "github.com/streadway/amqp"
13. )
15. func main() {
16. config := lib.LoadConfig[config.Config]()
17. conn, err := amqp.Dial(config.MQaddr)
18. if err != nil {
19. log.Println(err)
20. }
21. defer conn.Close()
23. router := gin.Default()
24. router.GET("/ws", handleWebSocket(conn))
25. router.GET("/count", handleGetCount(config))
26. router.NoRoute(gin.WrapH(http.FileServer(http.Dir("dist/"))))
28. panic(router.Run(fmt.Sprint(":", config.Port.Display)))
29. }
31. func handleWebSocket(conn \*amqp.Connection) gin.HandlerFunc {
32. return func(c \*gin.Context) {
33. srcConn, err := (&websocket.Upgrader{
34. CheckOrigin: func(r \*http.Request) bool {
35. return true
36. },
37. }).Upgrade(c.Writer, c.Request, nil)
38. if err != nil {
39. log.Println(err)
40. return
41. }
42. defer srcConn.Close()
44. ch, err := conn.Channel()
45. if err != nil {
46. log.Println(err)
47. return
48. }
49. defer ch.Close()
51. q, err := ch.QueueDeclare(
52. "analysis\_queue", // queue name
53. false,            // durable
54. false,            // delete when unused
55. false,            // exclusive
56. false,            // no-wait
57. nil,              // arguments
58. )
59. if err != nil {
60. log.Println(err)
61. return
62. }
64. msgs, err := ch.Consume(
65. q.Name, // queue
66. "",     // consumer
67. true,   // auto-ack
68. false,  // exclusive
69. false,  // no-local
70. false,  // no-wait
71. nil,    // args
72. )
73. if err != nil {
74. log.Println(err)
75. return
76. }
78. for msg := range msgs {
79. err := srcConn.WriteMessage(websocket.TextMessage, msg.Body)
80. if err != nil {
81. log.Println(err)
82. return
83. }
84. }
85. select {}
86. }
87. }
89. func handleGetCount(config \*config.Config) gin.HandlerFunc {
90. return func(c \*gin.Context) {
91. c.JSON(200, gin.H{
92. "count": config.DetectorCount})
93. }
94. }