芋道源码 2023-11-23

WebSocket 实时通信

1. 功能简介

项目的 `yudao-spring-boot-starter-websocket` 组件，基于 Spring WebSocket 进行二次封装，实现了更加简单的使用方式。例如说，WebSocket 的认证、Session 的管理、WebSocket 集群的消息广播等等。

> **疑问：为什么不使用 Netty 实现 WebSocket?**
> Netty 的学习和使用门槛较高，对大家可能不够友好，而 Spring WebSocket 足够满足 99.99% 的场景。

1.1 Token 身份认证

① 在 WebSocket 连接建立时，通过 QueryString 的 `token` 参数，进行认证。例如说：`ws://127.0.0.1:48080/ws?token=xxx` 。
由于 WebSocket 是基于 HTTP 建立连接，所以它的认证可以复用项目的 TokenAuthenticationFilter 实现。

> **为什么 token 不使用 Header 传递?**
> WebSocket 不支持 Header 传递，所以只能使用 QueryString 传递。

② 认证完成后，会通过 LoginUserHandshakeInterceptor 拦截器，将用户信息存储到 WebSocket Session 的 `attributes` 中。
这样，后续可以使用 WebSocketFrameworkUtils 获取用户信息，例如说：

```
// WebSocketFrameworkUtils.java

// ① 获取当前用户
public static LoginUser getLoginUser(WebSocketSession session)

// ② 获得当前用户的类型
public static Integer getLoginUserType(WebSocketSession session)

// ③ 获得当前用户的编号
public static Integer getLoginUserType(WebSocketSession session)
```

```
// ④ 获得当前用户的租户编号
public static Long getTenantId(WebSocketSession session)
```

1.2 Session 会话管理

每个前端和后端建立的 WebSocket 连接，对应后端的一个 WebSocketSession 会话对象。由于后续需要对 WebSocketSession 进行消息的发送，所以需要进行管理。

① WebSocketSession 的管理，由 WebSocketSessionManager 定义接口，由 WebSocketSessionManagerImpl 具体实现。

```
// 添加和移除 Session
void addSession(WebSocketSession session);
void removeSession(WebSocketSession session);

// 获得 Session，多种维度
WebSocketSession getSession(String id); // Session 编号
Collection<WebSocketSession> getSessionList(Integer userType); // 用户类型
Collection<WebSocketSession> getSessionList(Integer userType, Long userId); // 用
```

② WebSocket 建立和关闭连接时，通过 WebSocketSessionHandlerDecorator 处理器，分别调用 WebSocketSessionManager 进行 Session 的添加和移除。

1.3 Message 消息格式

WebSocket 默认使用“文本”进行通信，而业务需要按照不同类型的消息，进行不同的处理。因此，项目定义了 JsonWebSocketMessage 消息对象，包含 `type` 消息类型 + `content` 消息内容。

和 Spring MVC 对比，可以理解为：

	标识	方法	参数
Spring MVC	URL + Method 等	Controller 的 Method 方法	QueryString 或 RequestBody 等
项目 WebSocket	`type` 消息类型	WebSocketMessageListener 实现类	解析 `content` 消息内容后的 Message 对象

具体 JsonWebSocketMessage 和 WebSocketMessageListener 详细说明，参见「1.4 Message 消息接收」小节。

1.4 Message 消息接收

① WebSocket 接收到项目后，会先交给 JsonWebSocketMessageHandler 消息处理器，将消息解析成 JsonWebSocketMessage 对象。
之后，根据 `type` 消息类型，获得到 WebSocketMessageListener 实现类，并将 `content` 消息内容进一步解析成 Message 对象，交给它进行处理。
② 具体案例，可见 DemoWebSocketMessageListener 、DemoSendMessage 类。

1.5 Message 消息推送

① 项目的 WebSocketMessageSender 接口，定义了给 Session 发送消息的方法。如下所示：

```
// WebSocketMessageSender.java

// ① 发送消息给指定用户
void send(Integer userType, Long userId, String messageType, String messageConte
default void sendObject(Integer userType, Long userId, String messageType, Objec
    send(userType, userId, messageType, JsonUtils.toJsonString(messageContent));
}

// ② 发送消息给指定用户类型
void send(Integer userType, String messageType, String messageContent);
default void sendObject(Integer userType, String messageType, Object messageCont
    send(userType, messageType, JsonUtils.toJsonString(messageContent));
}

// ③ 发送消息给指定 Session
void send(String sessionId, String messageType, String messageContent);
default void sendObject(String sessionId, String messageType, Object messageCont
    send(sessionId, messageType, JsonUtils.toJsonString(messageContent));
}
```

② WebSocketMessageSender 有多种实现类，如下：

实现类	是否支持 WebSocket 集群	前置要求
LocalWebSocketMessageSender	❌	无
RedisWebSocketMessageSender	✅	开启《消息队列（Redis）》
RocketMQWebSocketMessageSender	✅	开启《消息队列（RocketMQ）》
KafkaWebSocketMessageSender	✅	开启《消息队列（Kafka）》

实现类	是否支持 WebSocket 集群	前置要求
RabbitMQWebSocketMessageSender	✅	开启《消息队列（RabbitMQ）》

> **疑问：什么是 WebSocket 集群?**
>
> 在后端部署多个 Java 进程时，会形成 WebSocket 集群。此时，就会存在跨进程的消息推送问题。例如说，连接 A 进程的 WebSocket 的用户，想要发送消息给连接 B 进程的 WebSocket 用户。
>
> 😁 如何解决呢？消息不直接发送给用户 WebSocketSession，而是先发给 Redis、RocketMQ 等消息队列，再由每个 Java 进程监听该消息，分别判断判断该用户 WebSocket 是否连接的是自己，如果是，则进行消息推送。

默认配置下，使用 LocalWebSocketMessageSender 本地发送消息，不支持 WebSocket 集群。可通过修改 `application.yaml` 配置文件的 `yudao.websocket.sender-type` 来切换，如下：

```
yudao:
  websocket:
    enable: true # websocket的开关
    path: /infra/ws # 路径
    sender-type: redis # 消息发送的类型，可选值为 local、redis、rocketmq、kafka、rab
    sender-rocketmq:
      topic: ${spring.application.name}-websocket # 消息发送的 RocketMQ Topic
      consumer-group: ${spring.application.name}-websocket-consumer # 消息发送的
    sender-rabbitmq:
      exchange: ${spring.application.name}-websocket-exchange # 消息发送的 Rabbit
      queue: ${spring.application.name}-websocket-queue # 消息发送的 RabbitMQ Que
    sender-kafka:
      topic: ${spring.application.name}-websocket # 消息发送的 Kafka Topic
      consumer-group: ${spring.application.name}-websocket-consumer # 消息发送的
```

另外，默认的 WebSocket 连接地址是 `ws://127.0.0.1:48080/infra/ws` ，可通过 `yudao.websocket.path` 配置项进行修改。

2. 使用方案

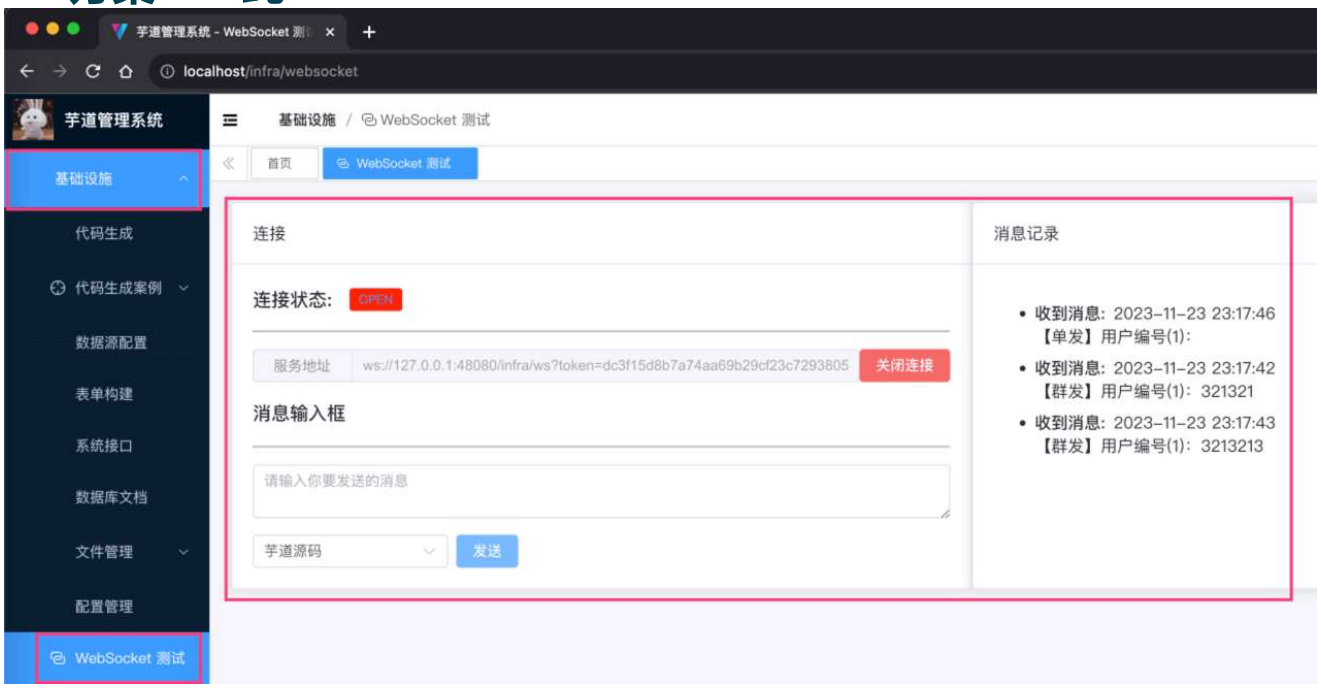
目前有 2 种使用方案，分别是：

方案名	上行	下行
方案一：纯 WebSocket	WebSocket	WebSocket
方案二：WebSocket + HTTP	HTTP	WebSocket

> **疑问：什么是上行？什么是下行？**
>
> - 上行：指的是“前端”发送消息给“后端”，WebSocket 和 HTTP 都可以。
> - 下行：指的是“后端”发送消息给“前端”，只能使用 WebSocket。

> **友情提示：下文中提到的所有配置，项目都已经配置好。你只需要按照下文的步骤，进行调试即可，了解每个配置的作用即可。**

2.1 方案一：纯 WebSocket



- 前端：见 [基础设施 -> WebSocket 测试] 菜单，对应 /views/infra/websocket/index.vue 界面
- 后端：见 `yudao-module-infra-biz` 模块，对应 DemoWebSocketMessageListener 监听器

基于 WebSocket 实现的单聊和群聊，暂时不支持消息的持久化（刷新后，消息会消息）。建议，多多调试，更好的理解 WebSocket 流程。

2.1.1 后端代码

① 在 `yudao-module-infra-biz` 模块的 `pom.xml` 文件中，引入 `yudao-spring-boot-starter-websocket` 依赖。如下所示：

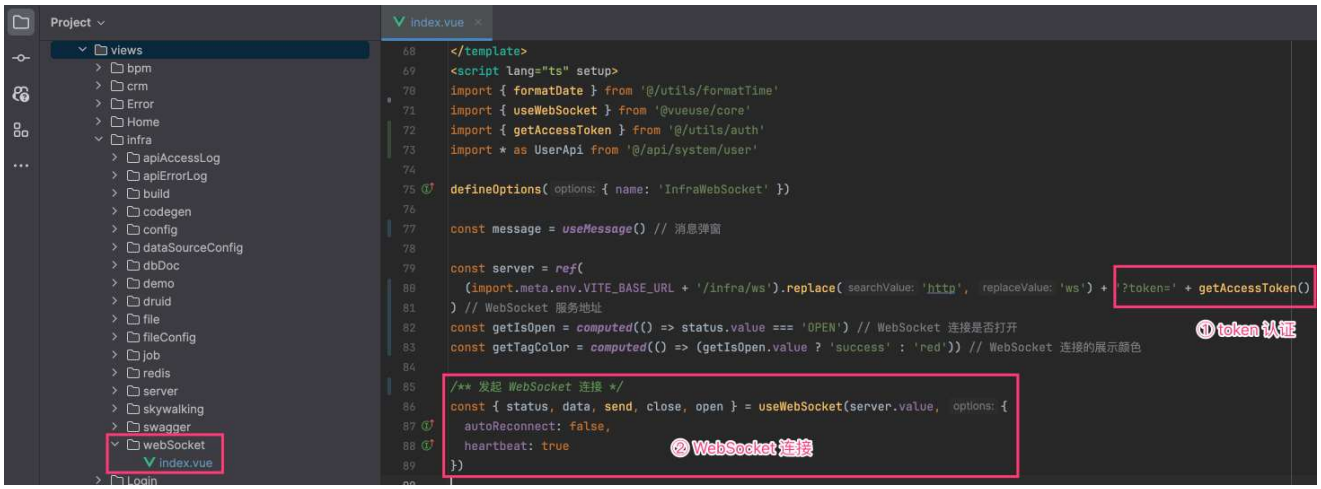
```
<dependency>
    <groupId>cn.iocoder.boot</groupId>
    <artifactId>yudao-spring-boot-starter-websocket</artifactId>
</dependency>
```

② 新建 DemoWebSocketMessageListener 类，实现对应消息的处理。如下图所示：

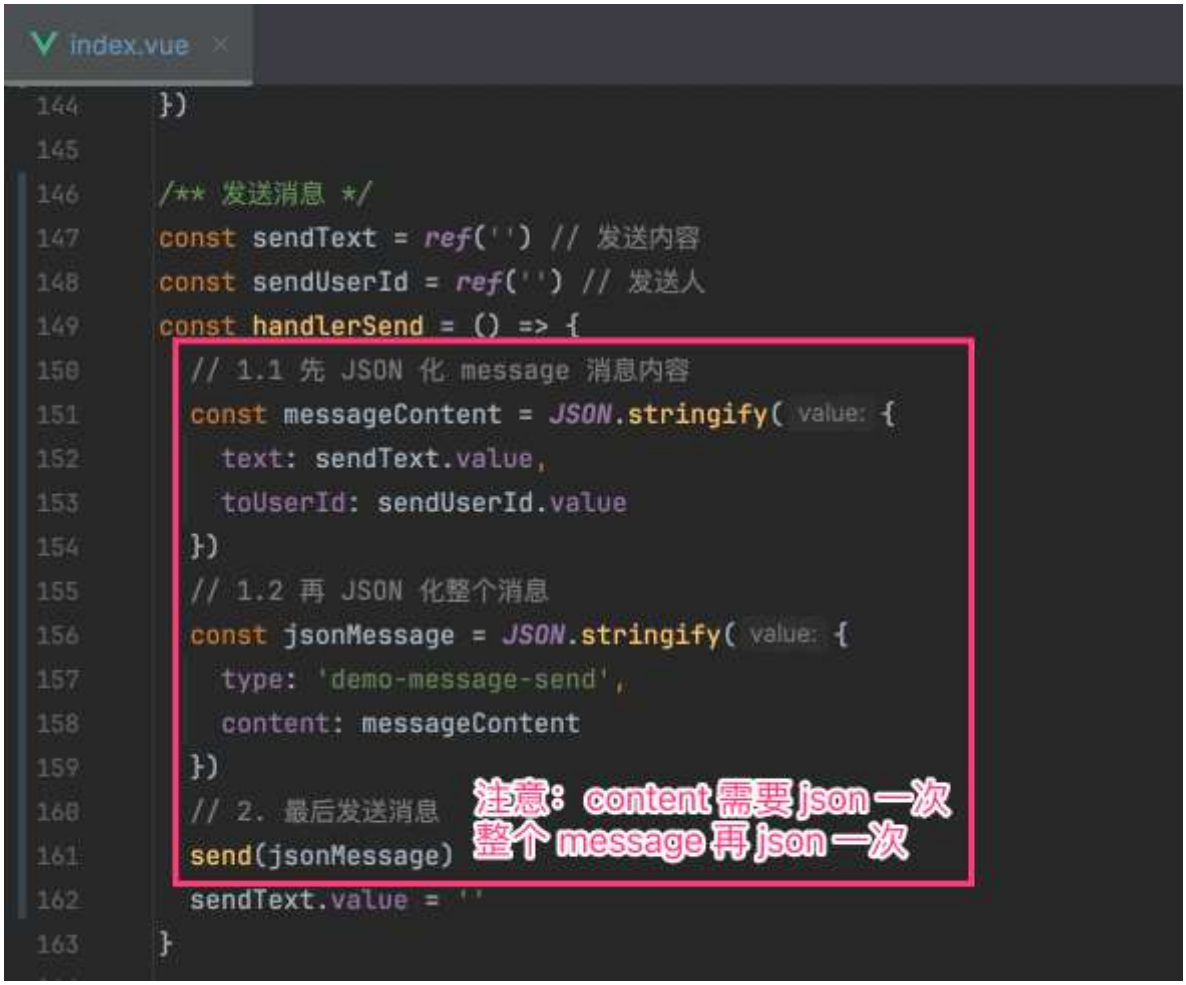

DemoWebSocketMessageListener 类

2.1.2 前端代码

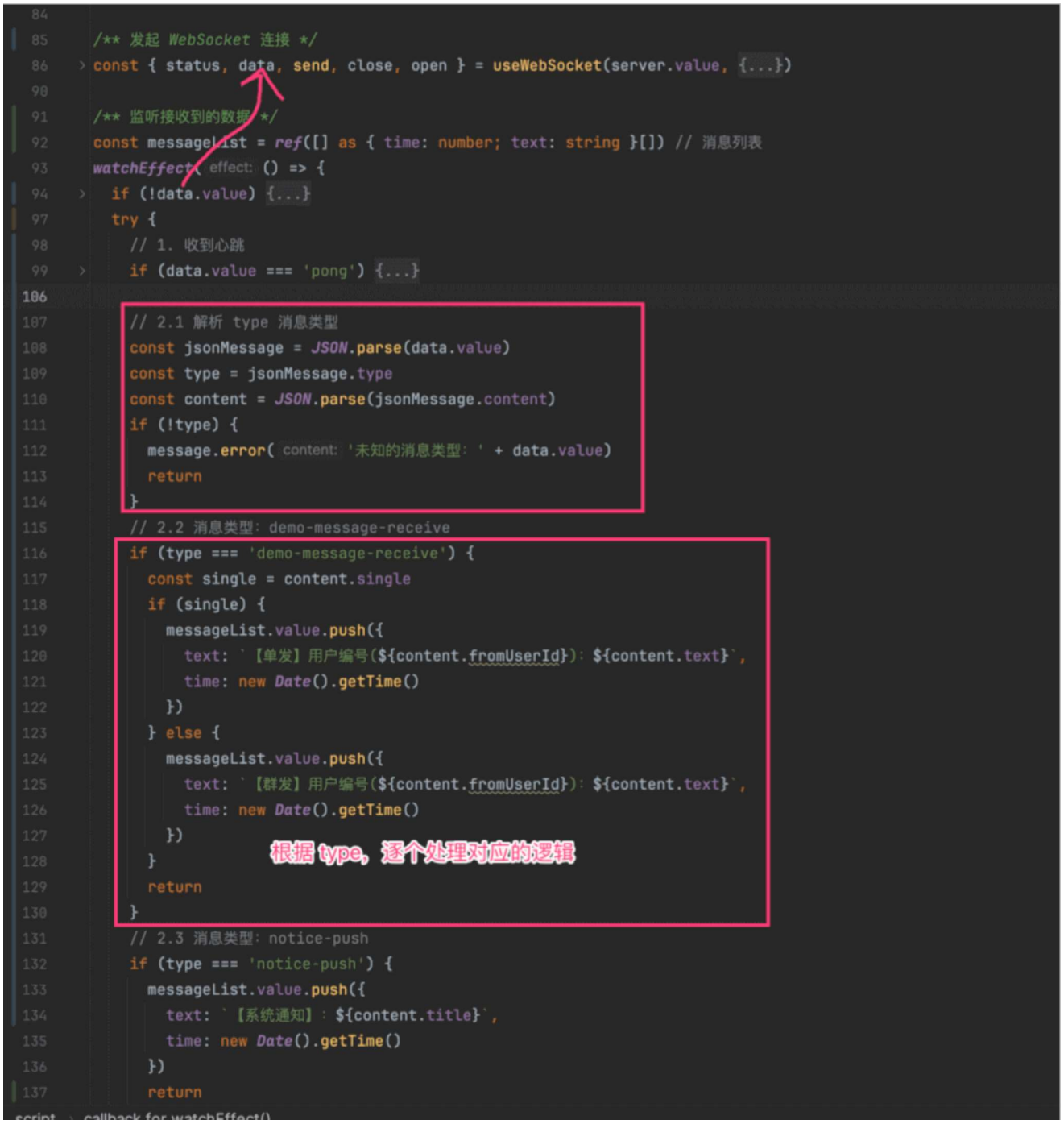
① 建立 WebSocket 连接，如下图所示：



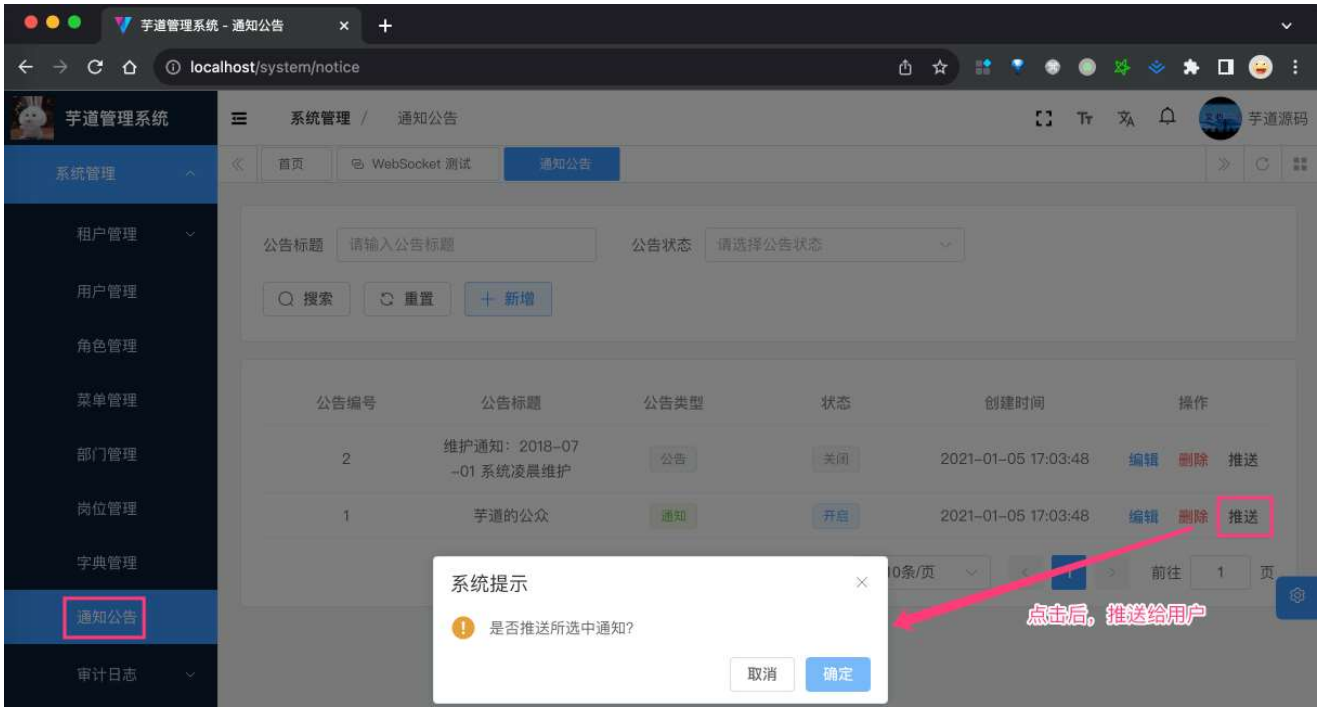
② 发送 WebSocket 消息，如下图所示：



③ 接收 WebSocket 消息。如下图所示：

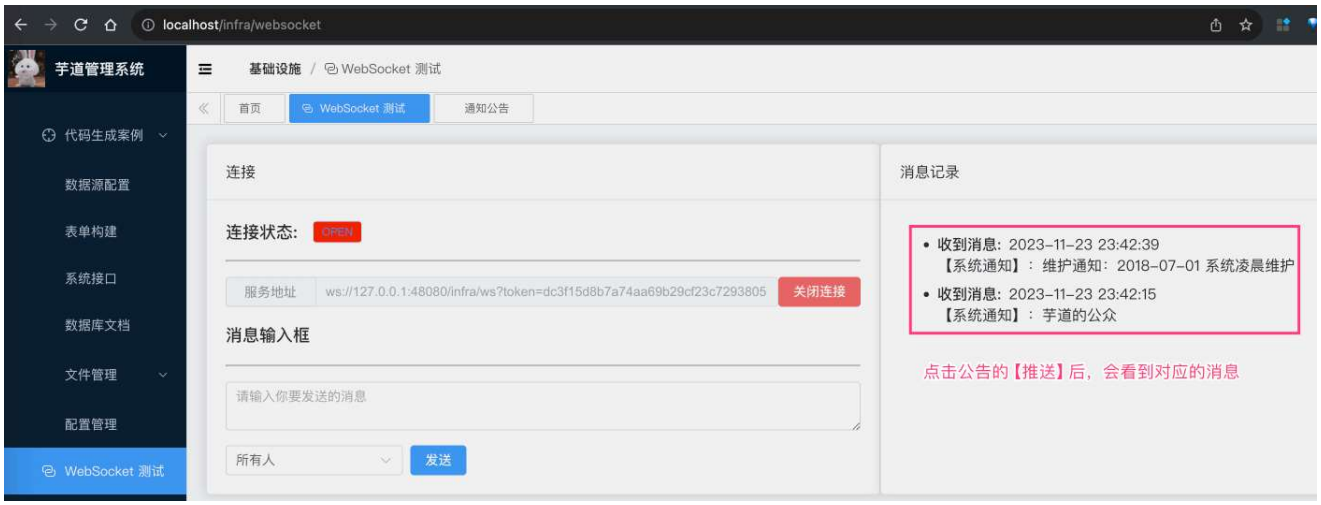


2.2 方案二：WebSocket + HTTP



- 前端：见 [系统管理 -> 通知公告] 菜单，对应 /views/system/notice/index.vue 界面的【推送】按钮
- 后端：见 `yudao-module-system-biz` 模块，对应 DemoWebSocketMessageListener 监听器

点击某条公告的【推送】按钮，仅仅推送给所有在线用户。由于 WebSocket 目前暂时没全局建立，所以还是使用 [基础设施 -> WebSocket 测试] 菜单演示。如下图所示：



2.2.1 后端代码

【相同】① 在 `yudao-module-infra-biz` 模块的 `pom.xml` 文件中，引入 `yudao-spring-boot-starter-websocket` 依赖。

【不同】② 在 `yudao-module-system-biz` 模块的 `pom.xml` 文件中，引入 `yudao-module-infra-api` 依赖。如下所示：

```
<dependency>
    <groupId>cn.iocoder.boot</groupId>
    <artifactId>yudao-module-infra-api</artifactId>
```

```
        <version>${revision}</version>
    </dependency>
```

【不同】③ 在 `yudao-module-system-biz` 模块，在 NoticeController 类中，新建 `#push(...)` 方法，用于推送公告消息。如下图所示：

```
@PostMapping("/push")
@Operation(summary = "推送通知公告", description = "只发送给 websocket 连接在线的用户")
@Parameter(name = "id", description = "编号", required = true, example = "1024")
@PreAuthorize("@ss.hasPermission('system:notice:update')")
public CommonResult<Boolean> push(@RequestParam("id") Long id) {
    NoticeDO notice = noticeService.getNotice(id);
    Assert.notNull(notice, errorMsgTemplate: "公告不能为空");
    // 通过 websocket 推送给在线的用户
    webSocketSenderApi.sendObject(UserTypeEnum.ADMIN.getValue(), messageType: "notice-push", notice);
    return success( data: true);
}
```

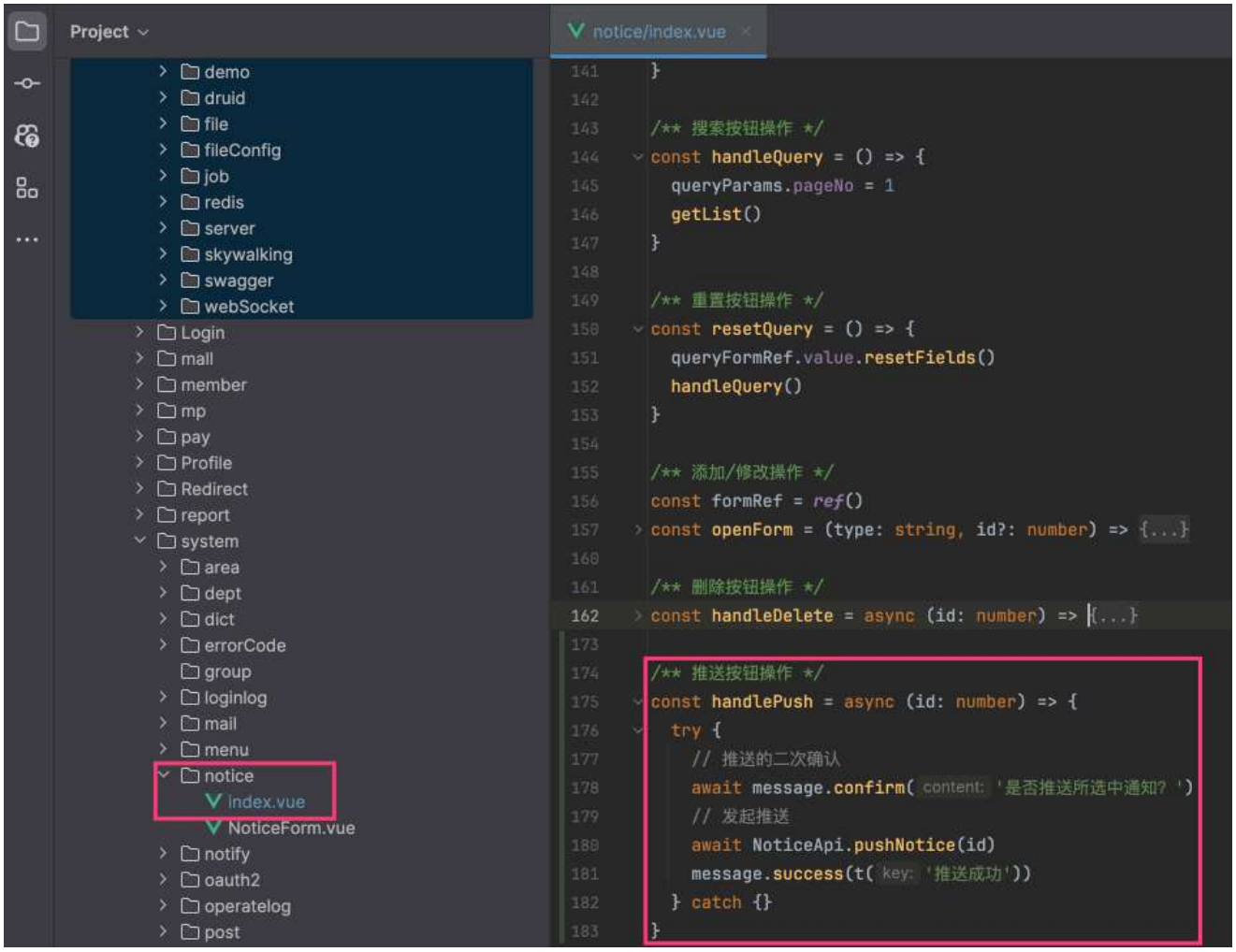
本质上，它替代了方案一的 DemoWebSocketMessageListener 类，走 HTTP 上行消息，替代 WebSocket 上行消息。

> **疑问：WebSocketSenderApi 是什么?**
>
> 它是由 `yudao-module-infra-biz` 对 WebSocketMessageSender 的封装，因为只有它（ `yudao-module-infra-biz` ）可以访问到 WebSocketMessageSender 的实现类，所以需要通过 API 的方式，暴露给其它模块使用。
>
> 这也是为什么 `yudao-module-system-biz` 模块，需要引入 `yudao-module-infra-api` 依赖的原因。

2.2.2 前端代码

【相同】① 建立 WebSocket 连接，和方案一相同，不重复截图。

【不同】② 发送 HTTP 消息，如下图所示：



本质上，它替代了方案一的 WebSocket 上行消息，走 HTTP 上行消息。

【相同】③ 接收 WebSocket 消息，和方案一相同，不重复截图。

2.3 如何选择?

我个人是倾向于方案二的，使用 HTTP 上行消息，使用 WebSocket 下行消息。原因如下：

① `yudao-module-infra-biz` 扮演一个 WebSocket 服务的角色，可以通过它来主动发送（下行）消息给前端。这样，未来如果使用 MQTT 中间件（例如说，EMQX、阿里云 MQTT、腾讯云 MQTT 等）替换现有 WebSocket 也比较方便。

② HTTP 上行消息，相比 WebSocket 上行消息来说，更加方便，也比较符合我们的编码习惯。

③ 在微服务架构下，多个服务是拆分开的，无法提供相同的 WebSocket 连接。例如说，`yudao-module-infra-biz` 和 `yudao-module-system-biz` 两个服务都需要有 WebSocket 推送能力时，需要前端分别连接它们两个服务。

考虑到 `ruoyi-vue-pro` 和 `yudao-cloud` 架构的统一性，还是只让 `yudao-module-infra-biz` 提供 WebSocket 服务：

- 前端连接 `yudao-module-infra-biz` 的 WebSocket 服务，其它服务通过 `yudao-module-infra-biz` 下行消息。
- 前端 HTTP 上行消息时，还是通过 HTTP 调用各个服务。

ps：如果你只用 `ruoyi-vue-pro` 单体架构，不会存在 ③ 的困扰，方案一也没问题。

one more thing~ 后续我们会使用 WebSocket 实现 IM 即时通信功能，敬请期待。

← SaaS 多租户【数据库隔离】 异常处理（错误码）→