1、订阅路由变更事件RouterChangeEvent

2、 根据路由类型(本地Or远程)，如果是本地路由，则发送踢人消息到客户端；否则广播踢人消息到MQ；

3、订阅MQ的广播消息，如果conn在本地机器，发送踢人消息到客户端；

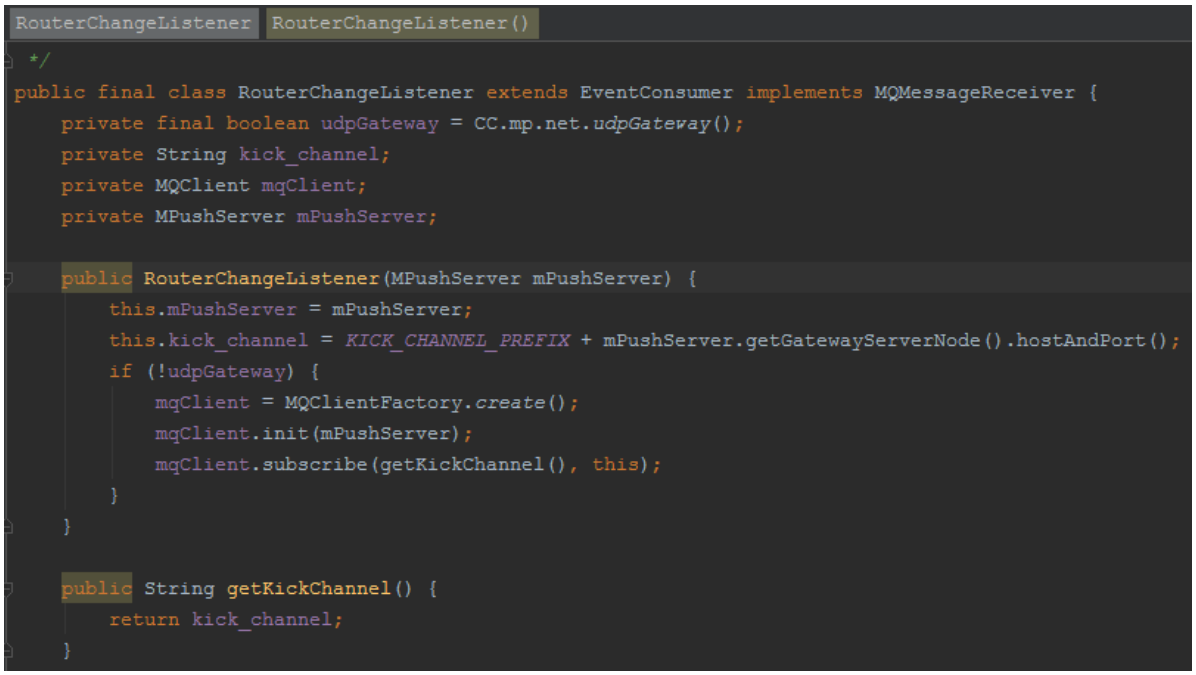
初始化

主要是完成：

1、获取MQClient实例

2、初始化得到MQ线程池，用于异步执行订阅到的消息处理

3、订阅主题消息



```
RouterChangeListener  RouterChangeListener()
 */
public final class RouterChangeListener extends EventConsumer implements MQMessageReceiver {
    private final boolean udpGateway = CC.mp.net.udpGateway();
    private String kick_channel;
    private MQClient mqClient;
    private MPushServer mPushServer;

    public RouterChangeListener(MPushServer mPushServer) {
        this.mPushServer = mPushServer;
        this.kick_channel = KICK_CHANNEL_PREFIX + mPushServer.getGatewayServerNode().hostAndPort();
        if (!udpGateway) {
            mqClient = MQClientFactory.create();
            mqClient.init(mPushServer);
            mqClient.subscribe(getKickChannel(), this);
        }
    }

    public String getKickChannel() {
        return kick_channel;
    }
```

kick_channel 变量为订阅的主题 ，/mpush/kick/127.0.0.1:8080

通过SPI机制，加载mpush-cache中MQClientFactory的实现类RedisMQClientFactory，得到MQClient的实例ListenerDispatcher；

调用ListenerDispatcher#init方法，获取redis线程池Executor，用于异步执行订阅到的MQ消息处理任务；

```
ListenerDispatcher  onMessage()
public final class ListenerDispatcher implements MQClient {

    private final Map<String, List<MQMessageReceiver>> subscribes = Maps.newTreeMap();

    private final Subscriber subscriber;

    private Executor executor;

    @Override
    public void init(MPushContext context) {
        executor = ((MonitorService) context.getMonitor()).getThreadPoolManager().getRedisExecutor();
    }

    public ListenerDispatcher() {
        this.subscriber = new Subscriber(this);
    }
```

调用ListenerDispatcher#subcribe方法，订阅redis中kick_channel 的消息；

同时，保存消息主题与消息处理类的映射关系；

```
public void subscribe(String channel, MQMessageReceiver listener) {
    subscribes.computeIfAbsent(channel, k -> Lists.newArrayList()).add(listener);
    RedisManager.I.subscribe(subscriber, channel);
}
```

发布消息

```
@Override
public void publish(String topic, Object message) {
    RedisManager
}

public Subscribe
    return subsc
}
```

Usages of publish(String, Object) in Project Files (4 usages found)

RouterChangeListener.java	146	mqClient.publish(Constants.getKickChannel(location.getHostAndPort()), message);
UserEventConsumer.java	54	mqClient.publish(ONLINE_CHANNEL, event.getUserId());
UserEventConsumer.java	61	mqClient.publish(OFFLINE_CHANNEL, event.getUserId());
UserManager.java	77	mqClient.publish(Constants.getKickChannel(location.getHostAndPort()), message);

整个应用有4个地方调用发布消息：

1、踢人消息（路由信息变更）

2、在线消息

3、离线消息

4、踢人消息（用户登录）

下面主要是看路由信息变更时的踢人逻辑：

RouterCenter register()

```
    */
    public boolean register(String userId, Connection connection) {
        ClientLocation location = ClientLocation
                .from(connection)
                .setHost(mPushServer.getGatewayServerNode().getHost())
                .setPort(mPushServer.getGatewayServerNode().getPort());

        LocalRouter localRouter = new LocalRouter(connection);
        RemoteRouter remoteRouter = new RemoteRouter(location);

        LocalRouter oldLocalRouter = null;
        RemoteRouter oldRemoteRouter = null;
        try {
            oldLocalRouter = localRouterManager.register(userId, localRouter);
            oldRemoteRouter = remoteRouterManager.register(userId, remoteRouter);
        } catch (Exception e) {
            LOGGER.error("register router ex, userId={}, connection={}", userId, connection, e);
        }

        if (oldLocalRouter != null) {
            EventBus.post(new RouterChangeEvent(userId, oldLocalRouter));
            LOGGER.info("register router success, find old local router={}, userId={}", oldLocalRouter, userId);
        }

        if (oldRemoteRouter != null && oldRemoteRouter.isOnline()) {
            EventBus.post(new RouterChangeEvent(userId, oldRemoteRouter));
            LOGGER.info("register router success, find old remote router={}, userId={}", oldRemoteRouter, userId);
        }
        return true;
    }
```

用户注册和连接，产生路由变更事件，由事件总线EventBus.post发布事件；

RouterChangeListener订阅到发布的事件，调用on();

RouterChangeListener on()

```
    @Subscribe
    @AllowConcurrentEvents
    void on(RouterChangeEvent event) {
        String userId = event.userId;
        Router<?> r = event.router;
        if (r.getRouteType().equals(Router.RouterType.LOCAL)) {
            sendKickUserMessage2Client(userId, (LocalRouter) r);
        } else {
            sendKickUserMessage2MQ(userId, (RemoteRouter) r);
        }
    }
```

如果是本地路由信息(客户端的conn连接在此台服务器)，则发送踢人消息到客户端(手机)；

如果是远程路由信息，则广播踢人消息到MQ，由MQ广播通知到远程的那台机器；

RouterChangeListener sendKickUserMessage2MQ()

```
    /**
     * 发送踢人消息到客户端
     *
     * @param userId 当前用户
     * @param router 本地路由信息
     */
    private void sendKickUserMessage2Client(final String userId, final LocalRouter router) {
        Connection connection = router.getRouteValue();
        SessionContext context = connection.getSessionContext();
        KickUserMessage message = KickUserMessage.build(connection);
        message.deviceId = context.deviceId;
        message.userId = userId;
        message.send(future -> {
            if (future.isSuccess()) {
                Logs.CONN.info("kick local connection success, userId={}, router={}, conn={}"
            } else {
                Logs.CONN.warn("kick local connection failure, userId={}, router={}, conn={}"
            }
        });
    }
```

RouterChangeListener sendKickUserMessage2MQ()

```
    private void sendKickUserMessage2MQ(String userId, RemoteRouter remoteRouter) {
        ClientLocation location = remoteRouter.getRouteValue();
        //1.如果目标机器是当前机器，就不要再发送广播了，直接忽略
        if (mPushServer.isTargetMachine(location.getHost(), location.getPort())) {
            Logs.CONN.debug("kick remote router in local pc, ignore remote broadcast, userId={}", userId);
            return;
        }

        if (udpGateway) {
            Connection connection = mPushServer.getUdpGatewayServer().getConnection();
            GatewayKickUserMessage.build(connection)
                    .setUserId(userId)
                    .setClientType(location.getClientType())
                    .setConnId(location.getConnId())
                    .setDeviceId(location.getDeviceId())
                    .setTargetServer(location.getHost())
                    .setTargetPort(location.getPort())
                    .setRecipient(new InetSocketAddress(location.getHost(), location.getPort()))
                    .sendRaw();
        } else {
            //2.发送广播
            //TODO 远程机器可能不存在，需要确认下redis 那个通道如果机器不存在的话，是否会存在消息积压的问题。
            MQKickRemoteMsg message = new MQKickRemoteMsg()
                    .setUserId(userId)
                    .setClientType(location.getClientType())
                    .setConnId(location.getConnId())
                    .setDeviceId(location.getDeviceId())
                    .setTargetServer(location.getHost())
                    .setTargetPort(location.getPort());
            mqClient.publish(Constants.getKickChannel(location.getHostAndPort()), message);
        }
    }
```

上面TODO，作者给埋了个坑；

接收消息

```java
public final class Subscriber extends JedisPubSub {
    private final ListenerDispatcher listenerDispatcher;

    public Subscriber(ListenerDispatcher listenerDispatcher) {
        this.listenerDispatcher = listenerDispatcher;
    }

    @Override
    public void onMessage(String channel, String message) {
        Logs.CACHE.info("onMessage:{},{}", channel, message);
        listenerDispatcher.onMessage(channel, message);
        super.onMessage(channel, message);
    }
```

订阅之后，会调用Subscriber#onMessage()方法，进而调用到ListenerDispatcher#onMessage()

线程池任务，异步执行消息的处理任务；

ListenerDispatcher onMessage()

```java
    public void onMessage(String channel, String message) {
        List<MQMessageReceiver> listeners = subscribes.get(channel);
        if (listeners == null) {
            Logs.CACHE.info("cannot find listener:{}, {}", channel, message);
            return;
        }

        for (MQMessageReceiver listener : listeners) {
            executor.execute(() -> listener.receive(channel, message));
        }
    }
```

比如下面是订阅踢人消息的处理类RouterChangeListener

RouterChangeListener receive()

```
                //2.1删除本地路由信息
                //localRouterManager.unRegister(userId, clientType);
                //2.2发送踢人消息到客户端
                sendKickUserMessage2Client(userId, localRouter);
            } else {
                Logs.CONN.warn("kick router failure target connId not match, localRouter={}, m
            }
        } else {
            Logs.CONN.warn("kick router failure can't find local router, msg={}", msg);
        }
    }

    @Override
    public void receive(String topic, Object message) {
        if (getKickChannel().equals(topic)) {
            KickRemoteMsg msg = Jsons.fromJson(message.toString(), MQKickRemoteMsg.class);
            if (msg != null) {
                onReceiveKickRemoteMsg(msg);
            } else {
                Logs.CONN.warn("receive an error kick message={}", message);
            }
        } else {
            Logs.CONN.warn("receive an error redis channel={}", topic);
        }
    }
}
```

客户端conn连接所在的服务器，接收到踢人消息，发送踢人消息给客户端；

如果客户端conn不在本服务器，直接忽略，就当是垃圾消息；

RouterChangeListener receive()

```
     * 处理远程机器发送的踢人广播.
     * <p>
     * 一台机器发送广播所有的机器都能收到，
     * 包括发送广播的机器，所有要做一次过滤
     *
     * @param msg
     */
    public void onReceiveKickRemoteMsg(KickRemoteMsg msg) {
        //1.如果当前机器不是目标机器，直接忽略
        if (!mPushServer.isTargetMachine(msg.getTargetServer(), msg.getTargetPort())) {
            Logs.CONN.error("receive kick remote msg, target server error, localIp={}, msg={}", ConfigTools.getLocalIp(),
            return;
        }

        //2.查询本地路由，找到要被踢下线的链接，并删除该本地路由
        String userId = msg.getUserId();
        int clientType = msg.getClientType();
        LocalRouterManager localRouterManager = mPushServer.getRouterCenter().getLocalRouterManager();
        LocalRouter localRouter = localRouterManager.lookup(userId, clientType);
        if (localRouter != null) {
            Logs.CONN.info("receive kick remote msg, msg={}", msg);
            if (localRouter.getRouteValue().getId().equals(msg.getConnId())) {//二次校验，防止误杀
                //fix 0.8.1 踢人的时候不再主动删除路由信息，只发踢人消息到客户端，路由信息有客户端主动解绑的时候再处理。
                //2.1删除本地路由信息
                //localRouterManager.unRegister(userId, clientType);
                //2.2发送踢人消息到客户端
                sendKickUserMessage2Client(userId, localRouter);
            } else {
                Logs.CONN.warn("kick router failure target connId not match, localRouter={}, msg={}", localRouter, msg);
            }
        } else {
            Logs.CONN.warn("kick router failure can't find local router, msg={}", msg);
```

RouterChangeListener sendKickUserMessage2MQ()

```java
    /**
     * 发送踢人消息到客户端
     *
     * @param userId 当前用户
     * @param router 本地路由信息
     */
    private void sendKickUserMessage2Client(final String userId, final LocalRouter router) {
        Connection connection = router.getRouteValue();
        SessionContext context = connection.getSessionContext();
        KickUserMessage message = KickUserMessage.build(connection);
        message.deviceId = context.deviceId;
        message.userId = userId;
        message.send(future -> {
            if (future.isSuccess()) {
                Logs.CONN.info("kick local connection success, userId={}, router={}, conn={}",
            } else {
                Logs.CONN.warn("kick local connection failure, userId={}, router={}, conn={}",
            }
        });
    }
```