Centrifugo简单试用

今天尝试用 centrifugo 来做一个在聊天室，以前用workerman做过，相对来说 workerman的配置就显得复杂多了，需要自己搭建PHP环境，

而 centrifugo 就清爽多了，官网下载二进制单文件直接运行，得益于go语言，centrifugo 的性能应该是大于workerman的，

但也得提醒一下，从github上看centrifugo是 2018年底才开始的，目前还在快速更新中，稳定性有一定风险。

测试系统： Debian 9

下载二进制文件：

https://github.com/centrifugal/centrifugo/releases

添加执行权限

```
chmod a+x centrifugo
```

生成配置文件 config.json文件

```
./centrifugo genconfig
```

内容大致这样的

复制

```
{
  "secret": "16bfd798-4f9f-4362-98e8-d88cb4997db2",
  "admin_password": "18d34296-f9c2-4c65-a04a-118cbe020481",
  "admin_secret": "e6adffc9-e54f-41ef-a487-3b172ea962db",
  "api_key": "3fe2e20a-af48-48d1-9aab-51ae88d92c5e"
}
```

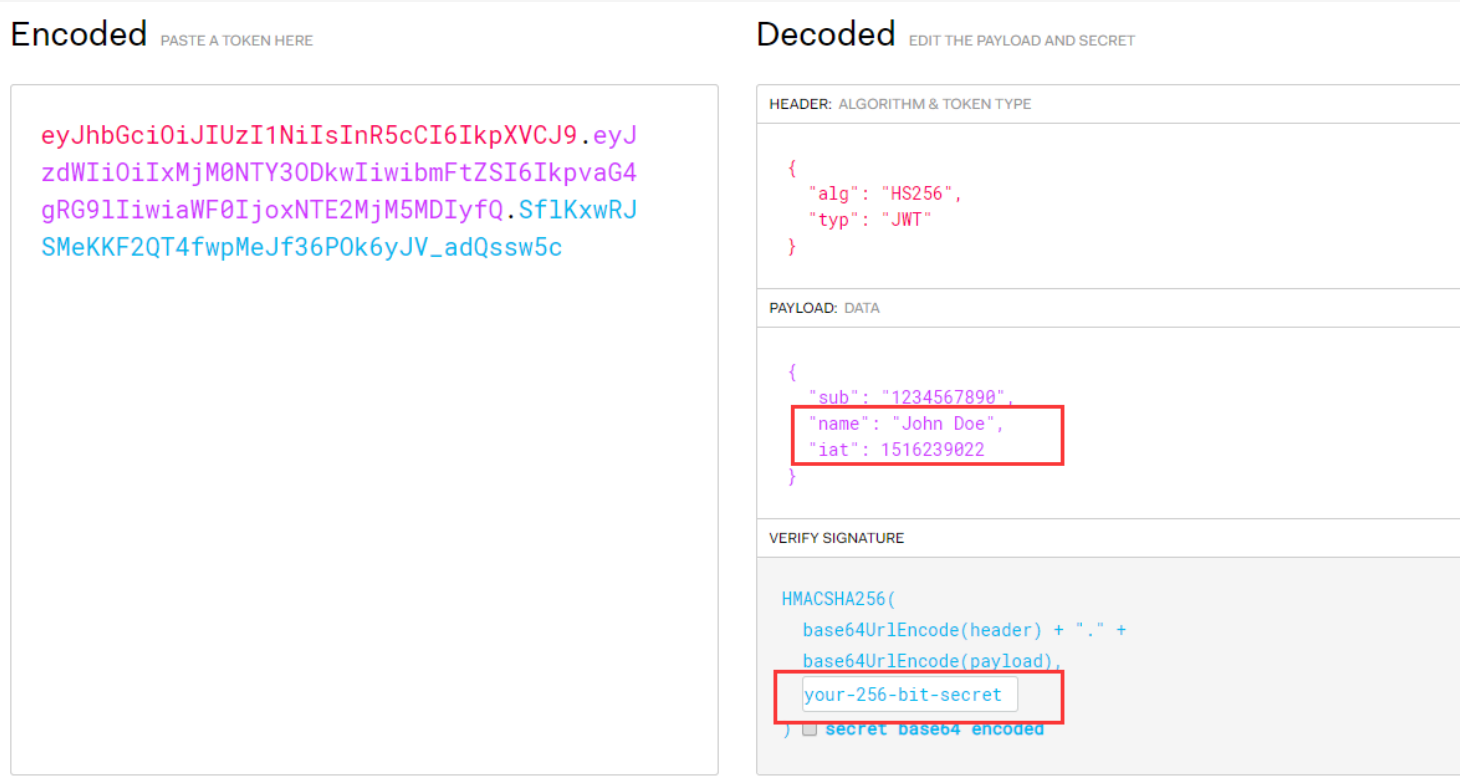
复制

启动 centrifugo

```
./centrifugo  -c config.json
```

前台采用 centrifugo的js客户端，下载地址： https://github.com/centrifugal/centrifuge-js

通过文档得知，要链接到centrifugo服务器，需要一个jwt的token，打开 https://jwt.io/，准本生成一个测试用的jwtToken



PAYLOAD框框里面去掉 name和iat，只保留sub一项，不然连不上，我猜测是centrifugo 不支持

（文档里面确实写了支持exp,info的，实测不行）

VERIFY SIGNATURE 框框里面补充config.json文件里面的secret字段值

这个时候 左边 Encoded框框 里面的一串密文就是我们要用到的

嫌麻烦可以直接使用我生成好的两个（使用的前提是你的config.js secret字段要跟我的一样）：

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxIn0.GvmVuP_7yADlqHk6fB7Tcq2V5EGY98PQw3EkX3DbBmQ
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIn0.ZOIIW5dReDRHEWkBakTJn0-cn0JFL6MIcrkOtElevRk
```

为了接下来的测试方便，我先配置一下config文件的常用选项：

复制

```
{
  "admin": true, # 开启管理员后台
  "posrt": 8000, # 端口
  "debug": true, # 开启debug模式
  "presence": true, # 开启系统状态信息
  "history_size": 100, # 历史消息保存条数，0不保存
  "history_lifetime": 600, # 历史消息保存秒数 0 保存
  "secret": "16bfd798-4f9f-4362-98e8-d88cb4997db2",
  "admin_password": "18d34296-f9c2-4c65-a04a-118cbe020481",
  "admin_secret": "e6adffc9-e54f-41ef-a487-3b172ea962db",
  "api_key": "3fe2e20a-af48-48d1-9aab-51ae88d92c5e",
  "publish": true # 允许客户端直接发布消息到centrifugo通道中，不经过我们自己的服务器，方便没有后端的情况下测试
}
```

复制

测试的html文件是这样的：

复制

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>socktest</title>
    <script src="./centrifuge.min.js"></script>
</head>
<body>
    <button onclick="init()">连接</button>
    <button onclick="closeit()">关闭连接</button>
```

```
    <button onclick="publish()">发送消息</button>
    <button onclick="subscribe()">订阅频道</button>
    <button onclick="unsubscribe()">取消订阅</button>
    <button onclick="history()">拉取历史消息</button>
</body>
</html>
<script>
// js 代码
```

复制

js代码：

复制

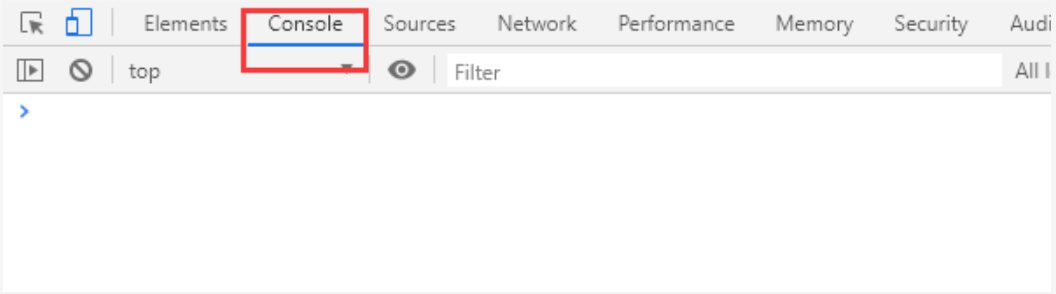
```
function init() {
        // 初始化 centrifugo 客户端
        window.centrifuge = new Centrifuge('ws://localhost:8000/connection/websocket');
        // 设置 token
        centrifuge.setToken("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIn0.ZOIIW5dReDRHEWkBakTJn0-cn0JFI
        // 监听连接事件
        centrifuge.on('connect',function(context) {
            console.log(context)
        })
        // 监听断开连接事件
        centrifuge.on('disconnect', function(context) {
            console.log(context)
        });
        // 启动连接
        centrifuge.connect();
    }
    // 发布消息到通道
    function publish () {
        centrifuge.publish("news", {"input": "hello-publish"}).then(function(res) {
            console.log(res);
        }, function(err) {
            console.log('publish error', err);
        });
    }
```

```
// 订阅频道
function subscribe() {
    // 监听频道内的消息
    var callbacks = {
        "publish": function(message) {
            console.log(message);
        },
        "join": function(message) {
            console.log(message);
        },
        "leave": function(message) {
            console.log(message);
        },
        "subscribe": function(context) {
            console.log(context);
        },
        "error": function(errContext) {
            console.log(err);
        },
        "unsubscribe": function(context) {
            console.log(context);
        }
    }
    window.subscription = centrifuge.subscribe("news", callbacks);
}
// 取消订阅
function unsubscribe() {
    subscription.unsubscribe();
    subscription.removeAllListeners();
}
// 查看历史消息
function history() {
    subscription.history().then(function(message) {
        console.log(message);
    }, function(err) {
        console.log(err);
    });
}
```

```
    // 关闭连接
    function closeit() {
        centrifuge.disconnect();
    }
```

复制

测试流程，开两个窗口，两个窗口的token不一样，打开调试工具的日志界面，



第一个窗口先点击连接，然后订阅频道，第二个窗口同样，

这个时候在第一个窗口点击发送消息，看另外一个窗口有没有接收到消息

centrifugo 提供了一个简易的后台面板：http://www.testserver.com:8000

测试过程中遇到一个问题：如果开启历史消息，开两个客户端测试时发现有个客户端无法发送消息，提示insufficient state（状态信息不足）

关闭历史消息的选项就没有问题

分类: **Centrifugo**

上一篇： react-redux的基本用法
下一篇： 摘要 - Digest

【推荐】博客园携手 AI 驱动开发工具商 Chat2DB 推出联合终身会员
【推荐】抖音旗下AI助手豆包，你的智能百科全书，全免费不限次数
【推荐】轻量又高性能的 SSH 工具 IShell：AI 加持，快人一步

编辑推荐：
· Kafka 的"无锁哲学"：高效消息流动的背后
· 时间轮在 Netty , Kafka 中的设计与实现
· MySQL 优化利器 SHOW PROFILE 的实现原理
· 在.NET Core中使用异步多线程高效率的处理大量数据
· 聊一聊 C#前台线程 如何阻塞程序退出

阅读排行：
· 夜莺 v8 第一个版本来了，开始做有意思的功能了
· 3款.NET开源、功能强大的通讯调试工具，效率提升利器！
· 如何做好技术经理
· 用 Cursor 写出第一个程序
· 【杂谈】Kafka 的"无锁哲学"：高效消息流动的背后

历史上的今天：
2016-04-09 Linux下压缩音频文件