AdminAPI 构架说明

日期: 06-30-2017，版本: v1.0.0

- AdminAPI 构架说明
  - 概述
  - 后端
    - 全局变量:
    - UserModel(用户模型):
    - UserController(用户控制器):
  - 前端
    - 用户主界面:
    - 用户界面长轮询:
    - 用户界面调取系统环境信息:
  - 其他

概述

AdminAPI 是一个使用长轮询机制实现信息推送的交互式平台，结合传统 MVC 模型与后端 JSON 格式 API 进行无刷新实时推送。本项目在 OSX 环境下构设完成，并参考微软在近年对使用 Mac 系统开发 .NET 项目的最新支持与指导。开发环境为微软在2015年04月新发布的 VisualStudioCode (软件主页)（.NET开发具体环境构架)，并使用其内置 NuGet 程序包管理系统引入 .NetCore 实现本地部署。项目数据库使用微软公司 ASP.NET 提供的 EntityFrameworkCore (资料) 以简化数据库接入代码。

平台默认群体推送，基本功能包括接收与发送群体信息和动作，并广播登录登出情况。附加功能包括指定式静音功能以屏蔽指定用户信息。平台带有验证系统，需登录注册后即可使用上述功能。

后端

全局变量:

平台后端在初始时生成一个名为 `Globals.pushStore` 的全局变量，用于内存储全局推送信息，屏蔽情况和其他用户细节。

```
public class Store{
        // 内置变量

        // 用于存储用户发送过的信息。使用哈希表用户名(钥)对应字符串列(值)的格式。
        private Hashtable _store;

        // 用于存储是否要向用户推送信息的布尔值。在存储用户信息时，此哈希表内值会被改写为真，并触发长轮询。
        private Hashtable _shouldUpdateStore;

        // 用于存储全部在线用户的用户名。
        private List<string> _userIds;

        // 用于记忆上次向用户推送的终止位置，防止重复推送。
        private Hashtable _nextStartStore;

        // 用于存储全部注册过用户的用户名。
        private List<string> _allUserIds;

        // 用于存储主动屏蔽方的屏蔽对象。
        private Hashtable _silenced;

        // 用于存储被动屏蔽方的屏蔽源对象。
        private Hashtable _beingSilenced;

        // 公共方法

        // 用于在用户注册时，将其用户名与全局变量进行捆绑，以接收与传递信息。
        // 参量:  userId - 用户名
        public void addSubscription(string userId);

        // 用于在用户登出时，取消对其的推送。
        // 参量:  userId - 用户名
```

```
        public void removeSubscription(string userId);

        // 用于记载新的用户动态信息并推送至其他用户。
        // 参量:  userId - 用户名,  newInfo - 新信息
        public void addInfo(string userId, string newInfo);

        // 用于问询是否需要对用户进行新一轮推送。
        // 参量:  userId - 用户名
        public bool shouldUpdate(string userId);

        // 用于问询用户是否在线。
        // 参量:  userId - 用户名
        public bool isSubscribed(string userId);

        // 用于生成针对源用户的系统环境（包括接收与屏蔽细节）
        // 参量:  userId - 源用户名
        public List<string> userStatus(string userId);

        // 用于屏蔽用户
        // 参量:  userId - 主动屏蔽方用户名, targetUserId - 被动屏蔽方用户名
        public void silence(string userId, string targetUserId);

        // 用于解除屏蔽用户
        // 参量:  userId - 主动解除屏蔽方用户名, targetUserId - 被动解除屏蔽方用户名
        public void unsilence(string userId, string targetUserId);
    }
```

UserModel(用户模型):

用户模型。使用 EntityFramework 提供的 ORM 接入方式对用户数据作出调整和存储。

```
    public class User{
        // 数据项

        public long Id { get; set; }          // 用户主键
        public string Username { get; set; }         // 用户名(为简化，目前作为平台主要识别ID使用)
        public string SessionToken { get; set; }          // 用户令牌
        public string PasswordSalt{ get; set; }      // 用户哈希算法加成字符串
        public string PasswordDigest { get; set; }       // 用户哈希算法转换结果
        public string Activities { get; set; }       // 用户所有活动
        public string CreatedAt{ get; set; }         // 用户初始时间
        public string UpdatedAt{ get; set; }         // 用户修改时间(目前不用)
    }
```

UserController(用户控制器):

后端控制器 `UserController` (用户控制器)协调包括验证和推送的全部功能。交互方式包括MVC与API两种。

MVC路径	HTTP 方法	对应方法	功能
/user/show	GET	*Show()*	显示持有当前令牌的用户界面。若令牌为无，则转向至 `/user/new`。
/user/new	GET	*New()*	显示注册新用户界面。若用户已持有令牌，转向至 `/user/show`。
/user/login	GET	*Login()*	显示用户登录界面。若用户已持有令牌，转向至 `/user/show`。
/user/create	POST	*Create()*	注册新用户。Form 参数必须包括 `username` 与 `password`。使用哈希方式转换秘钥并在成功的条件下给予用户 `admin-api-session-token` 令牌。
/user/create_session	POST	*CreateSession()*	登录用户。Form 参数必须包括 `username` 与 `password`。在成功的条件下给予用户 `admin-api-session-token` 令牌。
/user/delete_session	POST	*DeleteSession()*	登出用户。注销用户当前持有的令牌。

API路径	HTTP方法	对应方法	功能
/fetch/{id}	GET	*Fetch(id)*	长轮询接入口。以用户名 `id` 作为参数调取推送信息。
/post/id={userId}&content={content}	POST	*Post()*	发送新信息端口。以用户名 `userId` 和新内容 `content` 作为参数向服务器提示输入新信息。
/delete/id={userId}	DELETE	*Delete()*	取消所有订阅。将持用户名 `userId` 的用户从推送列表移除。
/user_status/id={userId}	GET	*UserStatus()*	调取针对持用户名 `userId` 用户的系统环境信息，包括屏蔽人等细节。
/silence/id={userId}&targetId={targetUserId}	POST	*Silence()*	用于屏蔽用户。为持用户名 `userId` 的主动屏蔽人屏蔽持用户名 `targetUserId` 被动屏蔽人的信息。
/unsilence/id={userId}&targetId={targetUserId}	POST	*Unsilence()*	用于解除屏蔽用户。为持用户名 `userId` 的主动屏蔽人解除对持用户名 `targetUserId` 被动屏蔽人的屏蔽。

前端

用户主界面：

主界面包括左侧系统环境和屏蔽状态一览，中间功能区块和右侧新推送消息栏。主界面在不刷新的情况下应自动更新左侧针对当前用户的系统环境和右侧的新消息列表。

用户界面长轮询：

用户界面使用 `ajax` 作为长轮询起点，传入后端 `/fetch/{id}` 路径，与后端长轮询循环配合，产生及时推送效应。下为长轮询生命周期。

```
// 前端长轮询方程开启通信。
function longPolling(){
    $.ajax({
        url: "fetch/" + dataStore.userId,        // 接入点
        method: "GET",
        data: JSON.stringify({
            userId: dataStore.userId
        }),
        success: function(newInfo){      // 回传接收到有效信息
            // 存入前端存储并将新信息放置于界面上
            dataStore.information = dataStore.information.concat(newInfo);
            for(var i = 0; i < newInfo.length; i++){
                $("#information-list").append("<li class='list-group-item'>" + newInfo[i] + "</li>");
            }
            getUserStatus();         // 调取针对性用户数据(用户界面左栏刷新)
            longPolling();       // 成功收到有效信息后重新开启另一个长轮询周期
        },
        error: function(err){

        }
    })
}
```

```
// 后端长轮询接收通信后，维持请求开放并定时查看新信息。若有新信息，则回传维持的请求。
[HttpGet("fetch/{id}", Name="Fetch")]
public List<string> Fetch(string id){
    while(!Globals.pushStore.shouldUpdate(id)){
        Thread.Sleep(500);       // 0.5秒定时查看新信息
    }

    return Globals.pushStore.newInfo(id);
}
```

用户界面调取系统环境信息:

跟随用户的屏蔽偏好，针对该用户的系统环境发生变化。在新动作产生后，追加调取目前针对用户的系统信息并更新左侧屏蔽栏。

```
function getUserStatus(){
    $.ajax({
        url: "user_status/id=" + dataStore.userId,
        method: "GET",
        success: function(response){
            dataStore.users = response;
            $("#users-list").html("");
            for(var i = 0; i < response.length; i++){
                ...          // (略) 将所得信息放置于网页上
            }
            // 在新生成的屏蔽按键上捆绑屏蔽和解除屏蔽功能
            bindSilenceButtons();
            bindUnsilenceButtons();
        },
        error: function(err){

        }
    })
}
```

其他

本项目使用 Bootswatch.com (链接)提供的 `bootstrap.min.css` 文件作为美化雏形，加由 `site.css` 进行细节美化。