Web 版聊天室

需求分析

实现 web 版聊天室程序

1. 打开主页, 见到登陆页面
2. 登陆成功, 进入主页面
3. 主页面中可以看到当前的频道(房间)列表
4. 点击某个频道, 可以看到频道(房间)中的消息
5. 点击某个频道, 可以发送消息, 此时其他用户也都能看到该消息.

认识 WebSocket

消息推送

消息推送大家都不陌生，比如扣扣消息、某东某宝购物后的系统消息等等都是消息推送，在H5出来之前，消息推送基本上都是使用HTTP请求的，但HTTP请求只能在客户端发起请求后服务端返回消息，而不能再客户端未发起请求时服务端主动推送消息给客户端，而对于HTTP的方式实现消息推送时，有以下几种方式：

轮询方式：客户端定时向服务端发送ajax请求，服务器接收到请求后马上返回消息并关闭连接。
优点：后端程序编写比较容易。
缺点：TCP的建立和关闭操作浪费时间和带宽，请求中有大半是无用，浪费带宽和服务器资源。
实例：适于小型应用。

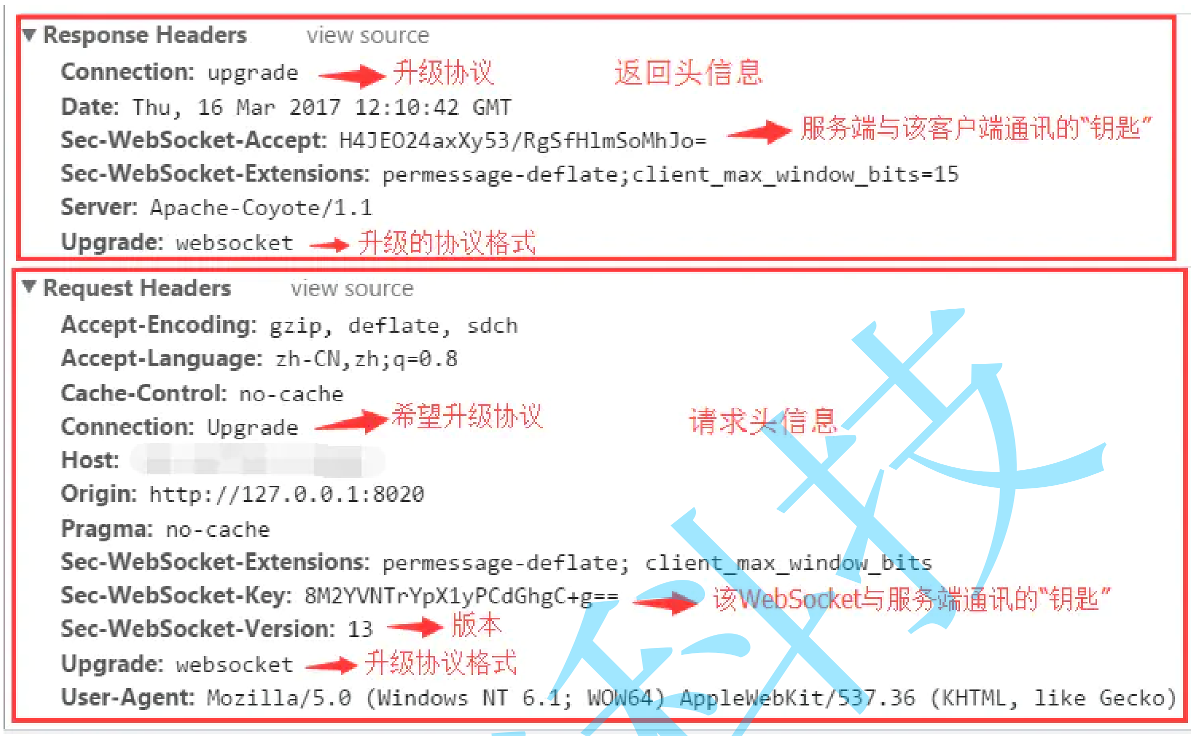
长轮询：客户端向服务器发送Ajax请求，服务器接到请求后hold住连接，直到有新消息才返回响应信息并关闭连接，客户端处理完响应信息后再向服务器发送新的请求。
优点：在无消息的情况下不会频繁的请求，耗费资源小。
缺点：服务器hold连接会消耗资源，返回数据顺序无保证，难于管理维护。
实例：WebQQ、Hi网页版、Facebook IM。

长连接：在页面里嵌入一个隐蔵iframe，将这个隐蔵iframe的src属性设为对一个长连接的请求或是采用xhr请求，服务器端就能源源不断地往客户端输入数据。
优点：消息即时到达，不发无用请求；管理起来也相对方便。
缺点：服务器维护一个长连接会增加开销，当客户端越来越多的时候，server压力大！
实例：Gmail聊天

webSocket：HTML5 WebSocket设计出来的目的就是取代轮询和长连接，使客户端浏览器具备像C/S框架下桌面系统的即时通讯能力，实现了浏览器和服务器全双工通信，建立在TCP之上，虽然WebSocket和HTTP一样通过TCP来传输数据，但WebSocket可以主动的向对方发送或接收数据，就像Socket一样；并且WebSocket需要类似TCP的客户端和服务端通过握手连接，连接成功后才能互相通信。
优点：双向通信、事件驱动、异步、使用ws或wss协议的客户端能够真正实现意义上的推送功能。
缺点：少部分浏览器不支持。
示例：社交聊天（微信、QQ）、弹幕、多玩家玩游戏、协同编辑、股票基金实时报价、体育实况更新、视频会议/聊天、基于位置的应用、在线教育、智能家居等高实时性的场景。

WebSocket 原理简介

WebSocket 协议本质上是一个基于 TCP 的协议。为了建立一个 WebSocket 连接，客户端浏览器首先要向服务器发起一个 HTTP 请求，这个请求和通常的 HTTP 请求不同，包含了一些附加头信息，附加信息如图所示：



```
▼ Response Headers     view source
    Connection: upgrade  → 升级协议          返回头信息
    Date: Thu, 16 Mar 2017 12:10:42 GMT
    Sec-WebSocket-Accept: H4JEO24axXy53/RgSfHlmSoMhJo=  → 服务端与该客户端通讯的"钥匙"
    Sec-WebSocket-Extensions: permessage-deflate;client_max_window_bits=15
    Server: Apache-Coyote/1.1
    Upgrade: websocket  → 升级的协议格式
▼ Request Headers     view source
    Accept-Encoding: gzip, deflate, sdch
    Accept-Language: zh-CN,zh;q=0.8
    Cache-Control: no-cache
    Connection: Upgrade  → 希望升级协议          请求头信息
    Host: [redacted]
    Origin: http://127.0.0.1:8020
    Pragma: no-cache
    Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits
    Sec-WebSocket-Key: 8M2YVNTrYpX1yPCdGhgC+g==  → 该WebSocket与服务端通讯的"钥匙"
    Sec-WebSocket-Version: 13  → 版本
    Upgrade: websocket  → 升级协议格式
    User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
```

使用 Tomcat 中内置的 WebSocket 库

pom.xml 中添加

注意: 必须在 xml 中写上 `<scope>` 字段

```
<dependency>
   <groupId>javax.websocket</groupId>
   <artifactId>javax.websocket-api</artifactId>
   <version>1.1</version>
   <scope>provided</scope>
</dependency>
```

服务器代码

```
@ServerEndpoint(value="/webSocketTest/{userId}")
public class TestWebSocket {
    private String userId = null;

    @OnOpen
    public void onOpen(@PathParam("userId") String userId, Session session)
{
        this.userId = userId;
        System.out.println("打开连接: " + userId);
    }

```

```java
    @OnClose
    public void onClose() {
        System.out.println("关闭连接: " + userId);
    }

    @OnMessage
    public void onMessage(String message, Session session) throws IOException {
        System.out.println("收到消息! " + userId + ": " + message);
        session.getBasicRemote().sendText(message);
    }

    @OnError
    public void onError(Session session, Throwable error) {
        System.out.println("连接出现错误! " + this.userId);
        error.printStackTrace();
    }
}
```

客户端代码

直接复制粘贴

```html
<body>
userId: 1 <br />
<input id="userName" type="text" />
<input id="text" type="text" />
<button onclick="send()"> Send </button>
<button onclick="closeWebSocket()"> Close </button>
<div id="message">   </div>

<script type="text/javascript">
      //判断当前浏览器是否支持WebSocket
      if('WebSocket' in window){
          websocket = new WebSocket("ws://47.98.116.42:8080/java_chatroom/webSocketTest/1");
          console.log("link success")
      }else{
          alert('Not support websocket')
      }

      //连接发生错误的回调方法
      websocket.onerror = function(){
          setMessageInnerHTML("error");
      };

      //连接成功建立的回调方法
      websocket.onopen = function(event){
          setMessageInnerHTML("open");
      }

      //接收到消息的回调方法
      websocket.onmessage = function(event){
          setMessageInnerHTML(event.data);
      }
```

```

        //连接关闭的回调方法
        websocket.onclose = function(){
            setMessageInnerHTML("close");
        }

        //监听窗口关闭事件，当窗口关闭时，主动去关闭websocket连接，防止连接还没断开就关闭窗口，server端会抛异常。
        window.onbeforeunload = function(){
            websocket.close();
        }

        //将消息显示在网页上
        function setMessageInnerHTML(innerHTML){
            document.getElementById('message').innerHTML += innerHTML + '<br/>';
        }

        //关闭连接
        function closeWebSocket(){
            websocket.close();
        }

        //发送消息
        function send(){
            var message = document.getElementById('text').value;
            websocket.send(message);
        }
    </script>

</body>
```

数据库设计

用户表

```
create table user (userId int primary key auto_increment,
                   name varchar(50) unique,
                   password varchar(50),
                   nickName varchar(50),   -- 昵称
                   iconPath varchar(2048), -- 头像路径
                   signature varchar(100),
                   lastLogout DateTime -- 上次登录时间
                  ); -- 个性签名

insert into user values(null, 'test', '123', '蔡徐坤', '', '我擅长唱', now());
insert into user values(null, 'test2', '123', '蔡徐坤2', '', '我擅长跳', now());
insert into user values(null, 'test3', '123', '蔡徐坤3', '', '我擅长rap', now());
insert into user values(null, 'test4', '123', '蔡徐坤4', '', '我擅长篮球', now());
```

频道表

```
create table channel (channelId int primary key auto_increment,
                      channelName varchar(50)
                     );
insert into channel values(null, '体坛赛事');
insert into channel values(null, '娱乐八卦');
insert into channel values(null, '时事新闻');
insert into channel values(null, '午夜情感');
```

消息表

```
create table message (messageId int primary key auto_increment,
                      userId int, -- 谁发的
                      channelId int, -- 发到哪个频道中
                      content text, -- 消息内容是啥
                      sendTime DateTime     -- 发送时间
                     );

insert into message values (null, 1, 1, 'hehe1', now());
insert into message values (null, 1, 1, 'hehe2', now());
insert into message values (null, 1, 1, 'hehe3', now());
```

Model 层实现

创建实体类

User 类表示一个用户

```
public class User {
    int userId;
    String name;
    String password;
    String nickName;
    String iconPath;
    String signature;
    java.sql.TimeStamp lastLogout;
}
```

Channel 类表示一个频道

```
public class Channel {
    private int channelId;
    private String channelName;
}
```

Message 类表示一条消息

```java
public class Message {
    private int messageId;
    private int userId;
    private String nickName; // 这个字段只是为了前后端交互方便, 表中没这个.
    private int channelId;
    private String content;
    private java.sql.TimeStamp sendTime;
}
```

实现 DBUtil

```java
public class DBUtil {
    private static final String URL = "jdbc:mysql://127.0.0.1:3306/java_chatroom?characterEncoding=utf8&useSSL=true";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "";

    private static DataSource dataSource = null;

    private static DataSource getDataSource() {
        if (dataSource == null) {
            synchronized (DBUtil.class) {
                if (dataSource == null) {
                    dataSource = new MysqlDataSource();
                    ((MysqlDataSource)dataSource).setUrl(URL);
                    ((MysqlDataSource)dataSource).setUser(USERNAME);
                    ((MysqlDataSource)dataSource).setPassword(PASSWORD);
                }
            }
        }
        return dataSource;
    }

    public static Connection getConnection() {
        try {
            return getDataSource().getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }

    public static void close(Connection connection,
                             PreparedStatement statement,
                             ResultSet resultSet) {
        try {
            if (resultSet != null) {
                resultSet.close();
            }
            if (statement != null) {
                statement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

实现 UserDao

实现新增用户

实现注册功能需要这个方法.

```
public void add(User user) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "insert into user values(null, ?, ?, ?, ?, ?, now())";
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setString(1, user.getName());
        statement.setString(2, user.getPassword());
        statement.setString(3, user.getNickName());
        statement.setString(4, user.getIconPath());
        statement.setString(5, user.getSignature());
        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("插入用户失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现按名字查找

实现登陆功能需要这个方法

```
public User selectByName(String name) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "select * from user where name = ?";
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setString(1, name);
        resultSet = statement.executeQuery();
        if (resultSet.next()) {
            User user = new User();
            user.setUserId(resultSet.getInt("userId"));
            user.setName(resultSet.getString("name"));
            user.setPassword(resultSet.getString("password"));
            user.setNickName(resultSet.getString("nickName"));
```

```
                user.setIconPath(resultSet.getString("iconPath"));
                user.setSignature(resultSet.getString("signature"));
                user.setLastLogin(resultSet.getTimestamp("lastLogout"));
                return user;
            }
        } catch (SQLException e) {
            e.printStackTrace();
            throw new ChatroomException("通过姓名查找用户失败");
        } finally {
            DBUtil.close(connection, statement, resultSet);
        }
        return null;
    }
```

实现按 id 查找

```
public User selectById(int userId) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "select * from user where userId = ?";
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setInt(1, userId);
        resultSet = statement.executeQuery();
        if (resultSet.next()) {
            User user = new User();
            user.setUserId(resultSet.getInt("userId"));
            user.setName(resultSet.getString("name"));
            user.setPassword(resultSet.getString("password"));
            user.setNickName(resultSet.getString("nickName"));
            user.setIconPath(resultSet.getString("iconPath"));
            user.setSignature(resultSet.getString("signature"));
            user.setLastLogin(resultSet.getTimestamp("lastLogout"));
            return user;
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("通过 id 查找用户失败");
    } finally {
        DBUtil.close(connection, statement, resultSet);
    }
    return null;
}
```

实现更新登陆时间

实现登陆需要这个方法

```
public void updateLogoutTime(int userId) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "update user set lastLogin = now() where userId = ?";
    PreparedStatement statement = null;
```

```java
    try {
        statement = connection.prepareStatement(sql);
        statement.setInt(1, userId);
        int ret = statement.executeUpdate();
        if (ret != 1) {
            throw new ChatroomException("通过 id 更新用户登陆时间失败");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("通过 id 更新用户登陆时间失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现 ChannelDao

实现新增频道

```java
public void add(Channel channel) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "insert into channel values(null, ?)";
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setString(1, channel.getChannelName());
        int ret = statement.executeUpdate();
        if (ret != 1) {
            throw new ChatroomException("新增频道失败");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("新增频道失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现删除频道

```java
public void delete(int channelId) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "delete from channel where channelId = ?";
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setInt(1, channelId);
        int ret = statement.executeUpdate();
        if (ret != 1) {
            throw new ChatroomException("删除频道失败！ " + channelId);
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("删除频道失败！ " + channelId);
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现查看所有频道

用于在界面上显示频道列表

```
public List<Channel> selectAll() throws ChatroomException {
    List<Channel> channels = new ArrayList<>();

    Connection connection = DBUtil.getConnection();
    String sql = "select * from channel";
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        statement = connection.prepareStatement(sql);
        resultSet = statement.executeQuery();
        while (resultSet.next()) {
            Channel channel = new Channel();
            channel.setChannelId(resultSet.getInt("channelId"));
            channel.setChannelName(resultSet.getString("channelName"));
            channels.add(channel);
        }
        return channels;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("查找频道失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现 MessageDao

实现新增消息

实现消息发送

```
public void add(Message message) throws ChatroomException {
    Connection connection = DBUtil.getConnection();
    String sql = "insert into message values(null, ?, ?, ?, ?)";
    PreparedStatement statement = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setInt(1, message.getUserId());
        statement.setInt(2, message.getChannelId());
        statement.setString(3, message.getContent());
        statement.setTimestamp(4, message.getSendTime());
        int ret = statement.executeUpdate();
```

```java
        if (ret != 1) {
            throw new ChatroomException("新增消息失败");
        }
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("新增消息失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

实现按 时间 获取消息

根据指定 时间段 获取消息

获取发送时间为指定时间之前的消息 (获取历史消息)

```java
// 根据 channelId 和时间段来查找消息
public List<Message> selectByTimestamp(Timestamp from, Timestamp to) throws ChatroomException {
    List<Message> messages = new ArrayList<>();

    Connection connection = DBUtil.getConnection();
    String sql = "select * from message where sendTime >= ? and sendTime <= ?";
    PreparedStatement statement = null;
    ResultSet resultSet = null;
    try {
        statement = connection.prepareStatement(sql);
        statement.setDate(1, from);
        statement.setDate(2, to);
        resultSet = statement.executeQuery();
        while (resultSet.next()) {
            Message message = new Message();
            message.setMessageId(resultSet.getInt("messageId"));
            message.setUserId(resultSet.getInt("userId"));
            message.setChannelId(resultSet.getInt("channelId"));
            message.setContent(resultSet.getString("content"));
            message.setSendTime(resultSet.getTimestamp("sendTime"));
            messages.add(message);
        }
        return messages;
    } catch (SQLException e) {
        e.printStackTrace();
        throw new ChatroomException("查找频道失败");
    } finally {
        DBUtil.close(connection, statement, null);
    }
}
```

前后端 API 设计

用户管理

注册

```
请求:
POST /register
{
    name: xxx,
    password: xxx,
    nickName: "蔡徐坤",
    signature: "我擅长唱跳rap篮球",
}

响应:
HTTP/1.1 200 OK
{
    ok: 1,
    reason: xxx
}
```

登陆

```
请求:
POST /login
{
    name: xxx,
    password: xxx
}

响应:
HTTP/1.1 200 OK
{
    ok: 1,
    reason: xxx,
    userId: xxx,
    name: xxx,
    nickName: xxx,
    signature: xxx
}
```

检测登陆状态

```
请求:
GET /login

响应:
响应:
HTTP/1.1 200 OK
{
    ok: 1,
    userId: xxx,
    name: xxx,
    nickName: xxx,
    signature: xxx
}
```

注销

```
请求:
GET /logout

响应:
HTTP/1.1 200 OK
{
    ok: 1,
    reason: xxx
}
```

频道管理

新增频道

用户登陆了才能新增.

```
请求:
POST /channel
{
    channelName: xxx
}

响应:
HTTP/1.1 200 OK
{
    ok: 1,
    reason: xxx
}
```

查找频道信息

用户登陆了才能查找

```
请求:
GET /channel

响应:
HTTP/1.1 200 OK
[
    {
        channelId: 1,
        channelName: xxx
    },
    {
        channelId: 2,
        channelName: xxx
    }
]
```

删除频道信息

用户登陆了才能删除

```
请求:
DELETE /channel?channelId=xxx

响应:
HTTP/1.1 200 OK
{
    ok: 1,
    reason: xxx
}
```

消息管理

建立连接

```
请求:
ws://[ip]:[port]/message/{userId}
```

具体的发送接收消息由 WebSocket api 来具体实现.

登陆成功就会触发建立连接操作.

发送/接收消息格式

```
{
    "userId": 1,
    "nickName": "蔡徐坤",
    "channelId": 1,
    "content": "这是消息正文"
}
```

API 实现

创建 api 包

实现 Util 类

通过 readBody 方法把一个请求中的 body 完整读取出来.

```java
public class Util {
    public static String readBody(HttpServletRequest request) {
        int contentLength = request.getContentLength();
        byte[] buffer = new byte[contentLength];
        try (InputStream inputStream = request.getInputStream()) {
            inputStream.read(buffer, 0, contentLength);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return new String(buffer);
    }
}
```

用户管理

创建 Response 类

```java
public class Response {
    public int ok;
    public String reason;
}
```

实现注册

```java
@WebServlet("/register")
public class RegisterServlet extends HttpServlet {
    private Gson gson = new GsonBuilder().create();

    static class Request {
        public String name;
        public String password;
        public String nickName;
        public String signature;
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        // 1. 读取 body 中的数据
        // 2. 将 body 的数据从 json 字符串转成对象.
        // 3. 按用户名查找, 看该用户名是否存在.
        // 4. 将得到的用户名密码插入数据库
        // 5. 返回响应结果
    }
```

实现 doPost

```java
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    Response response = new Response();
    resp.setContentType("application/json; charset=utf-8");
    try {
        // 1. 读取 body 中的数据
        String body = Util.readBody(req);
        // 2. 将 body 的数据从 json 字符串转成对象.
        Request request = gson.fromJson(body, Request.class);
        // 3. 按用户名查找, 看该用户名是否存在.
        UserDao userDao = new UserDao();
        User existUser = userDao.selectByName(request.name);
        if (existUser != null) {
            throw new ChatroomException("用户名已经存在");
        }
        // 4. 将得到的用户名密码插入数据库
        User user = new User();
        user.setName(request.name);
        user.setPassword(request.password);
        user.setNickName(request.nickName);
        user.setSignature(request.signature);
        userDao.add(user);
        // 5. 返回响应结果
        response.ok = 1;
        response.reason = "";
    } catch (ChatroomException e) {
        // 6. 处理出错情况
        response.ok = 0;
        response.reason = e.getMessage();
    } finally {
        String jsonString = gson.toJson(response);
        resp.getWriter().write(jsonString);
    }
}
```

实现登陆

```java
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    private Gson gson = new GsonBuilder().create();

    static class Request {
        public String name;
        public String password;
    }

    static class LoginStatusResponse extends Response {
        public int userId;
        public String name;
        public String nickName;
        public String signature;
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 1. 读取 body 中的数据
        // 2. 将读到的数据解析成对象
        // 3. 按用户名进行查找
        // 4. 如果登陆失败, 则给出提示
        // 5. 如果登陆成功, 则创建 session 对象
        // 6. 结果写回给客户端
    }
}
```

实现 doPost

```java
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    LoginStatusResponse response = new LoginStatusResponse();
    resp.setContentType("application/json; charset=utf-8");
    try {
        // 1. 读取 body 中的数据
        String body = Util.readBody(req);
        // 2. 将读到的数据解析成对象
        Request request = gson.fromJson(body, Request.class);
        // 3. 按用户名进行查找
        UserDao userDao = new UserDao();
        User user = userDao.selectByName(request.name);
        // 4. 如果登陆失败, 则给出提示
        if (user == null || !request.password.equals(user.getPassword())) {
            throw new ChatroomException("用户名或密码错误");
        }
        // 5. 如果登陆成功, 则创建 session 对象
        HttpSession httpSession = req.getSession(true);
        httpSession.setAttribute("user", user);
        // 6. 结果写回给客户端
        response.ok = 1;
        response.reason = "";
        response.userId = user.getUserId();
```

```java
            response.name = user.getName();
            response.nickName = user.getNickName();
            response.signature = user.getSignature();
        } catch (ChatroomException e) {
            response.ok = 0;
            response.reason = e.getMessage();
        } finally {
            String jsonString = gson.toJson(response);
            resp.getWriter().write(jsonString);
        }
    }

```

实现检测登陆状态

先实现 Util.getSessionUser

```java
public static User getSessionUser(HttpServletRequest request) {
    HttpSession session = request.getSession(false);
    if (session == null) {
        return null;
    }
    return (User) session.getAttribute("user");
}
```

再实现 LoginServlet.doGet

```java
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    LoginStatusResponse response = new LoginStatusResponse();
    resp.setContentType("application/json; charset=utf-8");
    try {
        User user = Util.getSessionUser(req);
        if (user == null) {
            throw new ChatroomException("用户未登录");
        }
        response.ok = 1;
        response.reason = "";
        response.userId = user.getUserId();
        response.name = user.getName();
        response.nickName = user.getNickName();
        response.signature = user.getSignature();
    } catch (ChatroomException e) {
        response.ok = 0;
        response.reason = "您未登陆";
    } finally {
        String jsonString = gson.toJson(response);
        resp.getWriter().write(jsonString);
    }
}
```

实现注销

```java
@WebServlet("/logout")
public class LogoutServlet extends HttpServlet {
    private Gson gson = new GsonBuilder().create();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        Response response = new Response();
        resp.setContentType("application/json; charset=utf-8");
        try {
            // 1. 获取 Session 对象
            HttpSession session = req.getSession(false);
            if (session == null) {
                throw new ChatroomException("当前未登录");
            }
            // 2. 获取用户对象
            User user = (User) session.getAttribute("user");
            if (user == null) {
                throw new ChatroomException("当前未登录");
            }
            // 3. 删除 session 中的用户信息
            session.removeAttribute("user");
            // 4. 返回响应数据.
            response.ok = 1;
            response.reason = "";
        } catch (ChatroomException e) {
            response.ok = 0;
            response.reason = e.getMessage();
        } finally {
            String jsonString = gson.toJson(response);
            resp.getWriter().write(jsonString);
        }
    }
}
```

频道管理

创建 ChannelServlet 类

```java
@WebServlet("/channel")
public class ChannelServlet extends HttpServlet {
    private Gson gson = new GsonBuilder().create();

    static class Request {
        public String channelName;
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    }

    @Override
    protected void doDelete(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {
    }
}
```

新增频道

```
@Override
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    Response response = new Response();
    resp.setContentType("application/json; charset=utf-8");
    try {
        // 1. 检查用户登陆状态.
        User user = Util.getSessionUser(req);
        if (user == null) {
            throw new ChatroomException("您未登陆");
        }
        // 2. 读取 body
        String body = Util.readBody(req);
        // 3. 解析 body 为 json 格式
        Request request = gson.fromJson(body, Request.class);
        // 4. 插入数据到数据库中
        Channel channel = new Channel();
        channel.setChannelName(request.channelName);
        ChannelDao channelDao = new ChannelDao();
        channelDao.add(channel);
        // 5. 返回数据
        response.ok = 1;
        response.reason = "";
    } catch (ChatroomException e) {
        response.ok = 0;
        response.reason = e.getMessage();
    } finally {
        String jsonString = gson.toJson(response);
        resp.getWriter().write(jsonString);
    }
}
```

删除频道

```
@Override
protected void doDelete(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    Response response = new Response();
    resp.setContentType("application/json; charset=utf-8");
```

```java
    try {
        // 1. 检查用户登陆状态
        User user = Util.getSessionUser(req);
        if (user == null) {
            throw new ChatroomException("您未登陆");
        }
        // 2. 读取请求中的参数
        String channelIdString = req.getParameter("channelId");
        if (channelIdString == null || "".equals(channelIdString)) {
            throw new ChatroomException("channelId 参数有误");
        }
        // 3. 操作数据库
        ChannelDao channelDao = new ChannelDao();
        channelDao.delete(Integer.parseInt(channelIdString));
        response.ok = 1;
        response.reason = "";
    } catch (ChatroomException e) {
        response.ok = 0;
        response.reason = e.getMessage();
    } finally {
        String jsonString = gson.toJson(response);
        resp.getWriter().write(jsonString);
    }
}
```

查看所有频道

```java
@Override
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
    List<Channel> channels = new ArrayList<>();
    resp.setContentType("application/json; charset=utf-8");
    try {
        // 1. 检查用户登陆状态.
        User user = Util.getSessionUser(req);
        if (user == null) {
            throw new ChatroomException("您未登陆");
        }
        // 2. 查找数据库
        ChannelDao channelDao = new ChannelDao();
        channels = channelDao.selectAll();
    } catch (ChatroomException e) {
        // 失败了直接返回一个空的列表数据即可.
    } finally {
        String jsonString = gson.toJson(channels);
        resp.getWriter().write(jsonString);
    }
}
```

消息传输

MessageCenter

创建 `model.MessageCenter` 类, 用于管理消息

这是一个单例类, 主要做两件事:

1. 管理在线用户列表
2. 管理消息转发(用一个阻塞队列保存消息, 用一个专门的扫描线程来转发消息)

```java
public class MessageCenter {
    private volatile static MessageCenter instance = null;

    public static MessageCenter getInstance() {
        if (instance == null) {
            synchronized (MessageCenter.class) {
                if (instance == null) {
                    instance = new MessageCenter();
                }
            }
        }
        return instance;
    }

    // 保存当前的在线用户, 要考虑到线程安全
    private ConcurrentHashMap<Integer, Session> onlineList = new ConcurrentHashMap<>();
    // 保存当前收到了哪些消息
    private BlockingQueue<Message> messages = new LinkedBlockingQueue<>();

    public void addOnlineUser(int userId, Session session) {
        onlineList.put(userId, session);
    }

    public void delOnlineUser(int userId) {
        onlineList.remove(userId);
    }

    public void addMessage(Message message) {
        messages.add(message);
    }

    private MessageCenter() {
        // 创建一个线程, 将收到的数据源源不断的转发给所有的在线用户.
        Thread t = new Thread() {
            @Override
            public void run() {
                Gson gson = new GsonBuilder().create();
                while (true) {
                    try {
                        Message message = messages.take();
                        for (ConcurrentHashMap.Entry<Integer, Session> entry : onlineList.entrySet()) {
                            Session session = entry.getValue();
                            String jsonString = gson.toJson(message);
                            session.getBasicRemote().sendText(jsonString);
                        }
                    } catch (InterruptedException | IOException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        t.start();
    }
}
```

MessageAPI

创建 `api.MessageAPI` 类, 来处理 WebSocket 请求.

```java
// 每个连接会创建一个 MessageAPI 实例
@ServerEndpoint(value="/message/{userId}")
public class MessageAPI {
    private Gson gson = new GsonBuilder().create();
    private int userId = 0;

    @OnOpen
    public void onOpen(@PathParam("userId") String userId, Session session) throws ChatroomException, IOException {
        this.userId = Integer.parseInt(userId);
        System.out.println("打开连接: " + this.userId);
        // 1. 将建立连接的用户加入在线用户列表.
        MessageCenter.getInstance().addOnlineUser(this.userId, session);
        // 2. 获取该用户的上次下线时间.
        UserDao userDao = new UserDao();
        User user = userDao.selectById(this.userId);
        Timestamp lastLogout = user.getLastLogout();
        // 3. 从数据库拉取历史消息.
        MessageDao messageDao = new MessageDao();
        List<Message> historyMessages = messageDao.selectByTimestamp(lastLogout, new Timestamp(System.currentTimeMillis()));
        for (Message message : historyMessages) {
            String jsonString = gson.toJson(message);
            session.getBasicRemote().sendText(jsonString);
        }
    }

    @OnClose
    public void onClose() throws ChatroomException {
        System.out.println("关闭连接: " + userId);
        // 1. 将断开连接的用户加入在线用户列表.
        MessageCenter.getInstance().delOnlineUser(this.userId);
        // 2. 更新用户下线时间
        UserDao userDao = new UserDao();
        userDao.updateLogoutTime(userId);
    }

    @OnMessage
    public void onMessage(String request, Session session) throws
IOException, ChatroomException {
        System.out.println("收到消息! " + userId + ": " + request);
        // 1. 解析 message 格式.
        Message message = gson.fromJson(request, Message.class);
        // 2. 设置消息收到的时间
        message.setSendTime(new Timestamp(System.currentTimeMillis()));
        // 3. 将消息放入消息中心对象
        MessageCenter.getInstance().addMessage(message);
        // 4. 将消息对象写入数据库.
        MessageDao messageDao = new MessageDao();
        messageDao.add(message);
    }

    @OnError
    public void onError(Session session, Throwable error) {
        System.out.println("连接出现错误! " + this.userId);
        error.printStackTrace();
        // 将断开连接的用户加入在线用户列表.
        MessageCenter.getInstance().delOnlineUser(this.userId);
    }
}
```

简单测试消息转发

修改 test.html, 模拟实现多个用户的情况.

> 复制 test.html, 搞一个 test2.html, 然后硬编码写成多个不同的用户名即可
>
> **注意:**
>
> 1. 需要修改 ws 的 path.
> 2. 发送的数据是 json 格式.

参考资料

https://www.jianshu.com/p/d79bf8174196