29 聊天列表页面后端接口开发

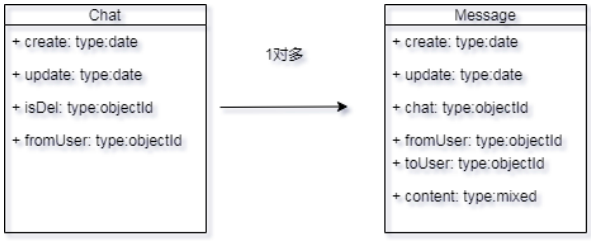
更新时间：2019-09-25 10:04:03

> 只有在那崎岖的小路上不畏艰险奋勇攀登的人,才有希望达到光辉的顶点。
>
> ——马克思

本章节完整源代码地址，大家可以事先浏览一下：

Github

聊天列表页的数据接口主要就是查询 Chat 这张表，主要提供根据关键字搜索聊天记录的功能，我们先来梳理一下 Chat 表的逻辑图，如下所示：



查询私信聊天列表

我们在之前的章节中，已经创建过了Chat的Schema，可以在这里回顾一下。

我们之间找到在后端项目的 routes 文件夹下的 message.js 路由文件中新增 getchatlist 路由方法。

下面这段代码主要是创建了一个get方法的路由，路径是/getchatlist，当浏览器请求 http://xx.xx.xx/getchatlist 就会进入这个方法，代码如下：

```
/*
* 查询私信列表接口
*/
router.get('/getchatlist', async (req, res, next)=> {


  try {
    var myId = req.user._id;
    //搜索使用的关键字字段
    var keyword = req.query.keyword || '';
    //查询接收者和发送者是当前登录用户的数据
    var list = await Chat.find({
    $or:[
      { fromUser: myId},
      { toUser: myId}
      ]
    }).populate('fromUser').populate('toUser').sort({'create':-1}).exec()

    var result = [];

    //找到chat数据后，查询相关的message数据
    for (var i = 0 ; i < list.length ; i++) {

      //todo why parse
      var chat = JSON.parse(JSON.stringify(list[i]));

      //根据chat的id，找到对应的消息列表里的第一条消息内容
      chat.msg = await getMsgByChat(list[i],keyword);

      //找到消息就push数组
      if (chat.msg) {
        var user = {};
        if (chat.toUser._id == myId) {
          user = chat.fromUser;
        } else {
          user = chat.toUser;
        }
        chat.user = user;
        result.push(chat);
      }

    }
    res.json({
      code:0,
      data:result
    });
  }catch(e){
    res.json({
      code:1,
      data:e
    });
  }

});
```

上面代码，我们来梳理一下整个逻辑，可以总结如下：

1. 首先，我们需要根据当前的用户查询出关联的`Chat`数据，我们使用的是`Chat.find()`方法，这里要注意一下，`fromUser`和`toUser`都可以进行匹配查询。
2. 然后，根据每个`Chat`的信息，找到关联的`message`，并查询出数据。我们使用`Message.find()`方法，并传入关键字keyword。
3. 将`message`数据进行组装，并返回。

根据关键字模糊查询

其中，上面代码中，根据 Chat 查 Message 的逻辑，我们封装在一个 getMsgByChat 这个方法里面，代码如下：

```
var getMsgByChat = async function(chat,keyword){
  var reg = new RegExp(keyword, 'i')

  var list = await Message.find({
    chat:chat._id,
    'content.type':'str',
    'content.value': {
      $regex: reg
    }
  }).sort({'create':-1}).exec();

  return list[0] || false
}
```

mongoose支持模糊查询

上面代码中，我们根据 keyword 来在mongoose中进行模糊查询相关的逻辑，在这里具体讲解一下这个知识点。

如果你了解过SQL查询语句，模糊查询类似：

```
NAME LIKE '%keyword%' or email LIKE '%keyword%'
```

LIKE 表示是启用模糊查询非精确查询，'%keyword%' 表示对字段模糊查询的边界，更多可以参考这里，基于SQL的查询只能是基于字符串里是否含有另一个字符串这种场景，而在mongoose将更加强大，可以直接通过正则表达式来查询：

```
{ <field>: { $regex: /pattern/, $options: '<options>' } }
{ <field>: /pattern/<options> }
```

$regex: 表示模糊查询，参数支持正则表达式， <field> 是String类型的，例如：

```
MyModel.find({ name: /john/i }, 'name friends', function () { })
MyModel.find({ name: {$regex:/john/,options:'$i'} }, 'name friends', function () { }) //  $options: '$i' 忽略大小写 $g表示全局匹配
```

Mixed字段的查询：

上面代码中的 content.value 表示content字段是一个 Mixed 类型的，

```
content:{
  type:''
  value:''
}
```

查询 Mixed 里的其中一个值可以采用 content.value：'xxx'，或者使用模糊查询 content.value：{$regex: reg}，如果是另外更深层次的 Mixed 例如：

```
content:{
  type:''
  value:{
    name:''
  }
}
```

使用 content.value.name：'xxx' 即可。

到此，聊天记录列表页面的后端接口已经开发完成，逻辑相对比较简单。

小结

本章节主要讲解了聊天列表页面的后台开发，同时讲解了在mongoose里如何使用模糊查询。

本章节完整源代码地址：
Github

}

28 聊天列表页面 UI 开发

30 页面转场动画