26 私信后端接口开发

更新时间：2019-09-18 10:44:20

> 理想的书籍是智慧的钥匙。
>
> ——列夫·托尔斯泰

私信聊天逻辑的数据建模我相信大家应该都很熟悉了，在这里我们只实现1对1的聊天，所以需要新建2张表来，一张Message代表每一条聊天消息，一张Chat代表每一个聊天，Chat和Message是1对多的关系。

本章节完整源代码地址，大家可以事先浏览一下：

Github-chat

Github-mssage

Github-mssage-route

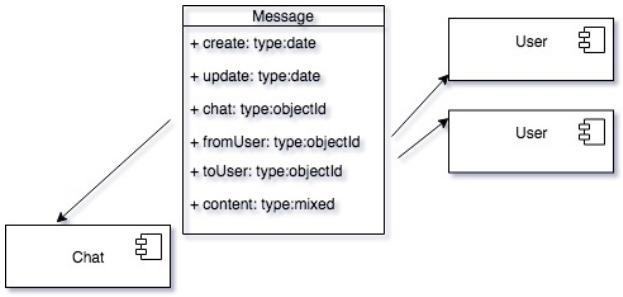
表结构分析

对于消息表和聊天表，我们首先需要分析一下：

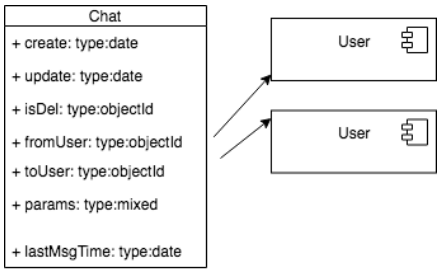
1. 消息表，很好理解，用来存储每一条消息，需要有图片消息，文字消息来区分。
2. 对于1对1聊天，消息需要有发送者和接收者，同样需要发送时间。
3. 每一条消息，都需要对应有一个聊天表。
4. 聊天表，主要有是否删除，最新一条聊天内容时间。
5. 最新一条内容时间是为了在聊天列表页来排序使用。

下面，我们就来看看表结构图：

Message表结构：



Chat表结构：



新建**ChatSchema**和**MessageSchema**

在了解了表结构之后，我们就要创建对应的Schema，在后端项目的 models 文件夹下新增 Message.js 和 Chat.js：

Chat.js:

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var ChatSchema = new mongoose.Schema({
  params: { type: Schema.Types.Mixed },//聊天相关的参数，例如背景图片等等
  isDel:{ type: Boolean,required:true,default:false },//是否删除
  lastMsgTime: { type: Date, default: Date.now },// 最近一条消息的时间
  fromUser:{ type: Schema.Types.ObjectId, ref: 'User',required:true },//聊天的发起者
  toUser:{ type: Schema.Types.ObjectId, ref: 'User',required:true },//聊天的接收者
  create: { type: Date, default: Date.now },
  update: { type: Date, default: Date.now },
}, {timestamps:{createdAt: 'create',updatedAt:'update'}});

module.exports = mongoose.model('Chat', ChatSchema);
```

Message.js:

```
var mongoose = require('mongoose');

var Schema = mongoose.Schema;

var MessageSchema = new mongoose.Schema({
  content: { type: Schema.Types.Mixed },//聊天内容
  fromUser: { type: Schema.Types.ObjectId, ref: 'User',required:true },//发送者
  chat:{ type: Schema.Types.ObjectId, ref: 'Chat',required:true },//聊天id关联
  toUser: { type: Schema.Types.ObjectId, ref: 'User',required:true },//接收者
  create: { type: Date, default: Date.now },
  update: { type: Date, default: Date.now },
},{timestamps:{createdAt: 'create',updatedAt:'update'}});

module.exports = mongoose.model('Message', MessageSchema);
```

我们在创建`Chat.js`的时候预留了`isDel`字段，一般在数据库中，做删除操作分为软删除(逻辑删除)和硬删除(物理删除)：

- **逻辑删除**：即标记删除，设置一个状态字段，判断删除，该类删除主要使用于一些，用户删除，但是可能网站还会使用到的一些数据，也包含，用户删除以后还想去恢复的一部分数据。
- **物理删除**：即直接删除该数据。这类的删除适用于使用之后，无意义的数据，比如我们现在注册发送的验证码等类型的数据。

一般情况下，对于聊天数据这种场景，后续如果有多终端登录的情况下，我们的删除还是需要采用软删除的，所以这里添加了`isDel`字段。

发送消息接口

在后端项目 routes 文件夹下新建 message.js ，作为消息模块的路由：
下面这段代码主要是创建了一个post方法的路由，路径是/addmsg，当浏览器请求`http://xx.xx.xx/addmsg`就会进入这个方法。

```
router.post('/addmsg', async (req, res, next)=> {
  //当前登录用户的id
  var myId = req.user._id;
  //发送的内容
  var content = req.body.content;
  //接收着的id
  var toUserId = req.body.toUser;
  try {
    var chatId = '';
    //首先需要查询是否已经有过聊天
    var list = await Chat.find({
      $or:[
        { $and: [{fromUser: myId}, {toUser: toUserId}]},
        { $and: [{fromUser: toUserId}, {toUser: myId}]}
        ]
      }).sort({'create':1}).exec();

    //如果有就把chatId记录下来
    if (list.length) {
      chatId = list[0]._id;
    }
    //如果没有就创建一个
    else {
      var chat = await Chat.create({
        params: {
          users:[myId,toUserId]
        },
        fromUser: myId,
        toUser: toUserId,
      });

      chatId = chat._id;
    }

    //添加一条消息
    var result = await Message.create({
      content: content,
      fromUser: req.user._id,//发送者的id，也就是当前登录用户的id
      chat:chatId,//将之前的chatId外键存入的message里
      toUser: toUserId,//接收者的ID
    });

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

});
```

上面的逻辑，我们来总结一下是：

1. 在每创建一条消息时，需要知道消息内容，和消息的发送者和接收者。
2. 在创建前，我们需要查询 Chat，是否它们之间已经有过聊天(两个人的聊天只可能出现在1个聊天对象里面)，如果有就找到 chatId，没有的话需要重新创建一个。
3. 在创建消息 Message 时需要关联聊天 chatId，这样我们在后面查询每个人的聊天列表时，就可以根据 chatId 查询到了。

mongoose条件查询

在我们查询是否曾经有过聊天时，我们使用了条件查询 $or 和 $and ，下面就来讲解一下这个知识点:

$and :参数是一个数组，表示条件的列表，顾名思义这些条件是“并且”的关系，只有当条件都满足才可以。

```
$and:[{ color: 'green' }, { status: 'ok' }]
```

$or :参数是一个数组，表示条件的列表，顾名思义这些条件是“或”的关系，只有当条件都满足1个就可以。

```
$or:[{ color: 'green' }, { status: 'not' }]
```

$or 和 $and 也可以嵌套使用：

```
$or:[
   { $and: [{color: 'green'}, {status: 'ok'}]},
   { $and: [{color: 'red'}, {status: 'ok'}]}
]
```

或者是：

```
$and:[
   { $or: [{color: 'green'}, {status: 'ok'}]},
   { $or: [{color: 'red'}, {status: 'ok'}]}
]
```

而我们代码里查询是否已经有过聊天的条件查询:

```
 $or:[
   { $and: [{fromUser: myId}, {toUser: toUserId}]},
   { $and: [{fromUser: toUserId}, {toUser: myId}]}
   ]
```

转换成熟悉的SQL就是：

where (fromUser = myId and toUser = toUserId) or (fromUser = toUserId and toUser = myId) 有木有很熟悉。

查询聊天记录接口

消息模块的另外一个重要接口就是查询个人的聊天记录，这个聊天记录就是所有和你有关的聊天内容，在后端项目 routes 文件夹下新建message.js的路由中增加方法：
下面这段代码主要是创建了一个get方法的路由，路径是/getchathistory，当浏览器请求 http://xx.xx.xx/getchathistory 就会进入这个方法。

```
/*
* 查询聊天记录接口
*/
router.get('/getchathistory', async (req, res, next)=> {


  try {
    var myId = req.user._id;
    // 根据发送者和接收着查询聊天记录
    var list = await Message.find({
      $or:[
        { $and: [{fromUser: myId}, {toUser: req.query.toUser}]},
        { $and: [{fromUser: req.query.toUser}, {toUser: myId}]}
        ]
      }).populate('fromUser').sort({'create':1}).exec()



    var result = [];
    for (var i = 0 ; i < list.length ; i++) {
      var msg = JSON.parse(JSON.stringify(list[i]));
      //如果发送者id和当前登录用户id相等，表示出主人态
      if (req.user._id == msg.fromUser._id) {
        msg.mine = true;
      } else {
        msg.mine = false;
      }
      result.push(msg);
    }

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

});
```

这里我们查询出的聊天记录是一个数组，数组的每个元素就是每条消息的内容结构，并且，我们利用`populate`特性，可以直接查出关联的用户信息。

这里查询聊天记录的数据有两个思路：

1. 根据发送者id和接收者id查询，就是我们代码里的思路。
2. 根据chatID查询，例如下面的逻辑：

```
var list = await Message.find({
  chat: chatId
  }).populate('fromUser').sort({'create':1}).exec()
```

而查询出来的数据需要标识出主人态，逻辑就是发送者的id和当前登录用户的id相同，即是同一个人。

小结

本章节讲解了聊天页面的发送消息接口和查询聊天记录接口的开发。

相关技术点：

1. 逻辑删除和物理删除的概念和区别。
2. 讲解了在mongoose使用使用 `$or` 和 `$and` 条件查询的方法和技巧。

本章节完整源代码地址，大家可以事先浏览一下：

Github-chat

Github-mssage

Github-mssage-route

}

25 底部输入框组件开发

27 使用 Scoket 改造后端接口