约-软件设计文档

技术选型及理由、架构设计、模块划分、软件设计技术

约跑

2015/7/18

约

技术选型及理由

1. 开发语言

html，css，javascript

约跑是一款搭载在 meteor 框架上的 web 应用，所以基本的网页实现由 html，css 语言开发完成，而用 javascript 写出页面的脚本，控制跳转等动作。

2. 数据库

MongoDB

Mongo DB 是目前在 IT 行业非常流行的一种非关系型数据库(NoSql),其灵活的数据存储方式备受当前 IT 从业人员的青睐。Mongo DB 很好的实现了面向对象的思想(OO 思想),在 Mongo DB 中 每一条记录都是一个 Document 对象。Mongo DB 最大的优势在于所有的数据持久操作都无需开发人员手动编写 SQL 语句,直接调用方法就可以轻松的实现 CRUD 操作。

约炮使用 MongoDB 存储每个用户的邮箱，账号，个人信息以及发布的约信，用邮箱作为用户的主键，然后各个动作直接从数据库中直接调用，各个操作也直接对数据库进行修改。

3. 框架

Meteor

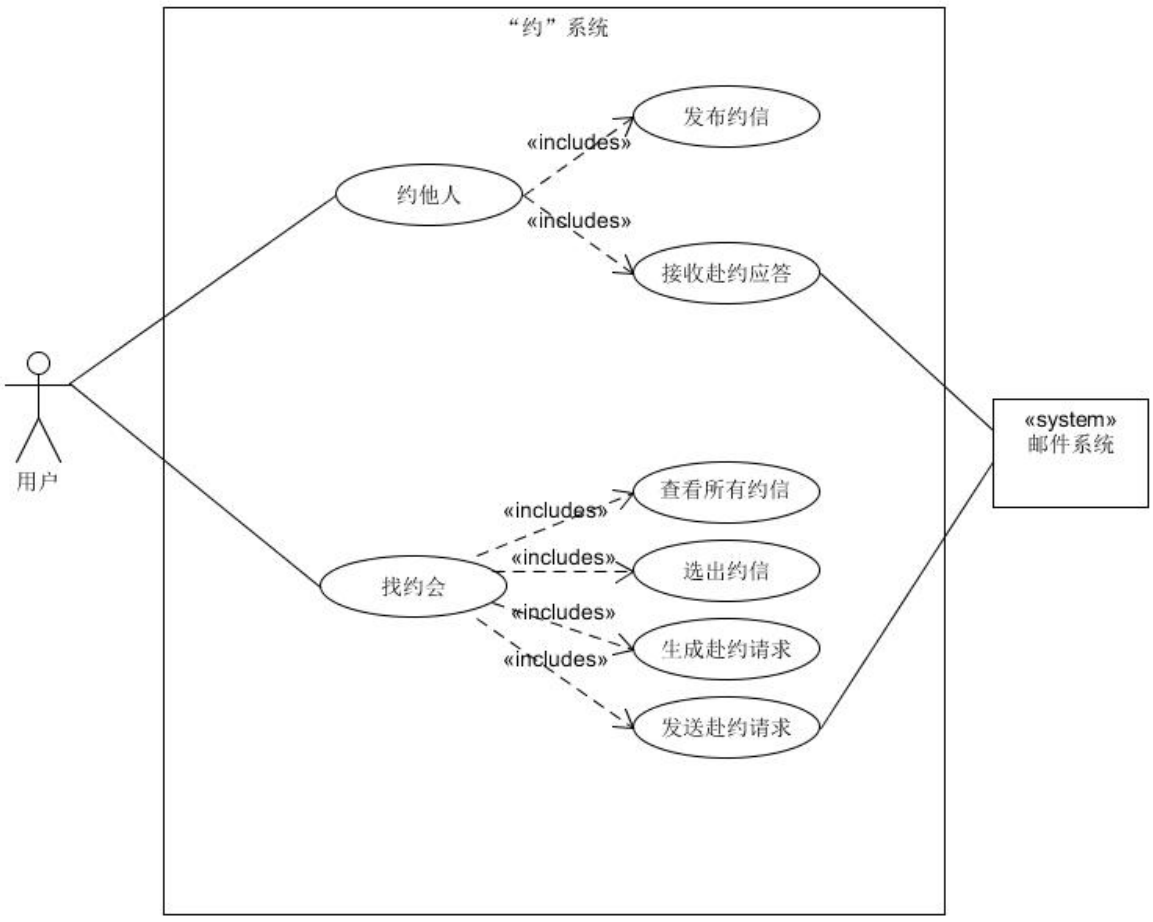
Meteor 是一种新的 JavaScript 框架，用于自动化和简化实时运行的 Web 应用程序的开发。它使用一个名为分布式数据协议 (Distributed Data Protocol, DDP) 的协议来处理实时通信，使用 WebSockets 的新浏览器以及使用 Asynchronous JavaScript + XML (Ajax) 长轮询的旧浏览器来支持这种协议。在这两种情况下，浏览器到服务器的通信是透明的。
DDP 协议旨在处理 JavaScript Serialized Object Notation (JSON) 文档集合，使 JSON 文档容易创建、更新、删除、查询和访问。因为 DDP 是一种开源协议，所以您可将它连接到任何客户端或数据存储。它为 MongoDB 提供了开箱即使用支持。事实上，Meteor 提供了两个 MongoDB 数据库：一个客户端缓存数据库和服务器上的一个 MongoDB 数据库。当一个用户更改一些数据时（例如通过单击 **Save**），在浏览器中运行的 JavaScript 代码会更新本地 MongoDB 中的相应的数据库项，然后向服务器发出一个 DDP 请求。该代码立即像操作已获得成功那样继续运行，因为它不需要等待服务器回复。与此同时，服务器在后台更新。如果服务器操作失败或返回一个意外结果，那么客户端 JavaScript 代码会依据从服务器新返回的数据立即进行调整。这种调整称为*延迟补偿*，向用户提供了更高的认知速度。

约

架构设计

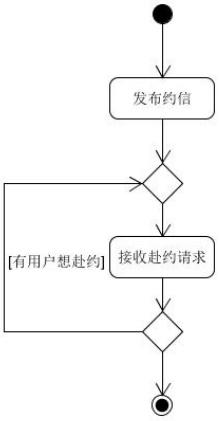
使用系统分析与设计 UML 构图绘制，其中包括用例图，活动图（发布约信和约他人），状态图（发布约信合约他人），领域模型，顺序图，部署图。

用例图如下：

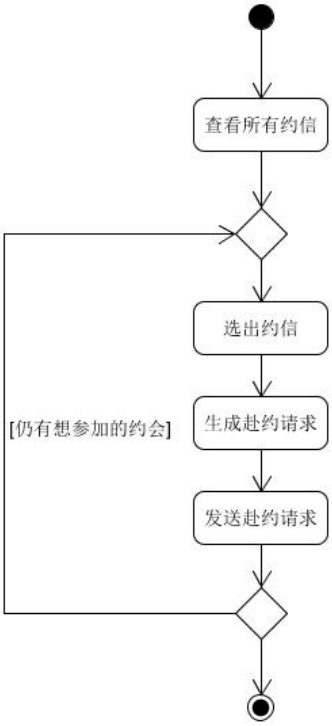


约

发布约信活动图：

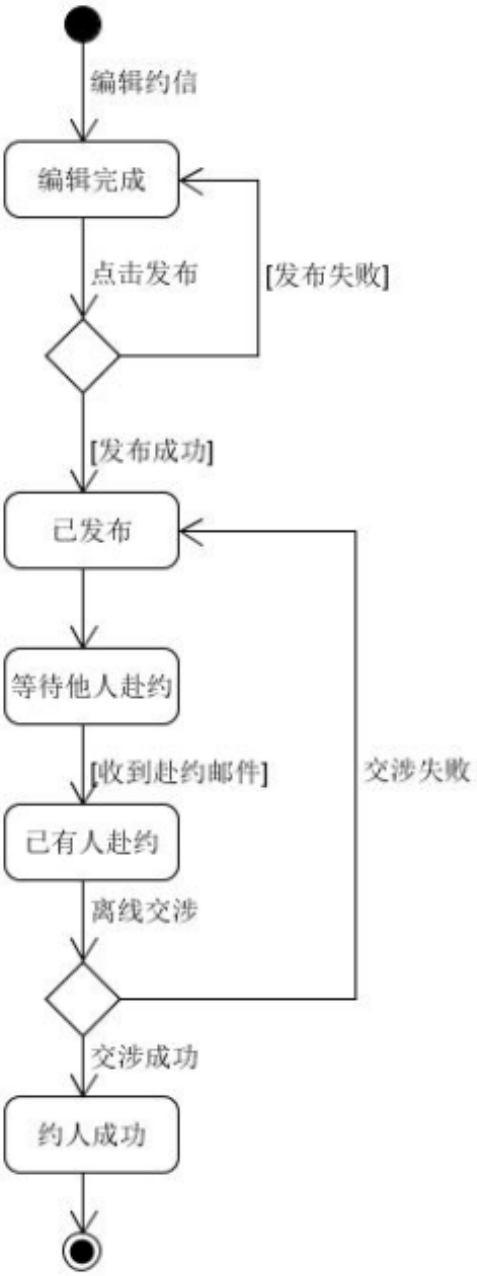


想要约会的活动图：



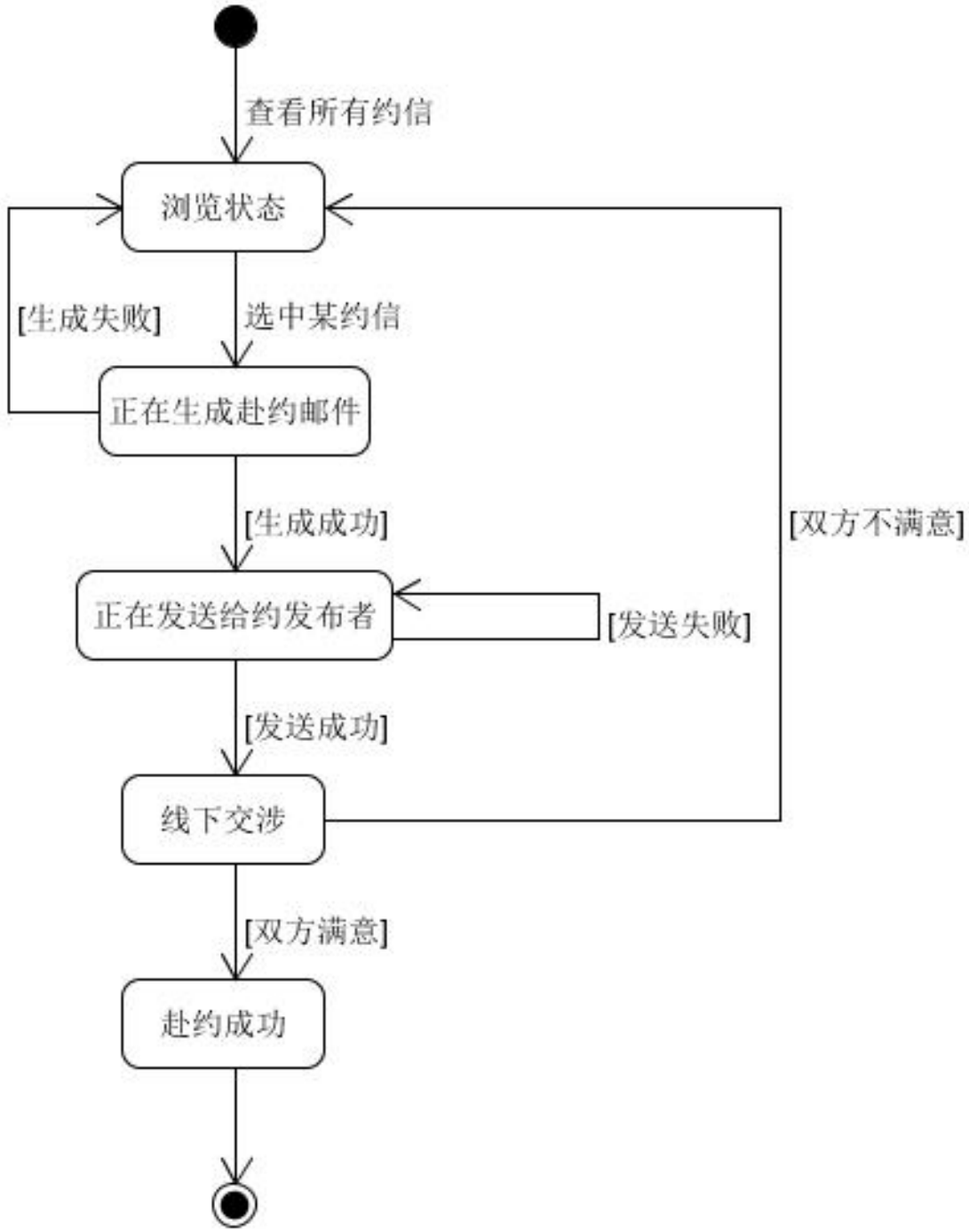
约

发布约信状态图


编辑约信
编辑完成
点击发布
[发布失败]
[发布成功]
已发布
等待他人赴约
[收到赴约邮件]
交涉失败
已有人赴约
离线交涉
交涉成功
约人成功

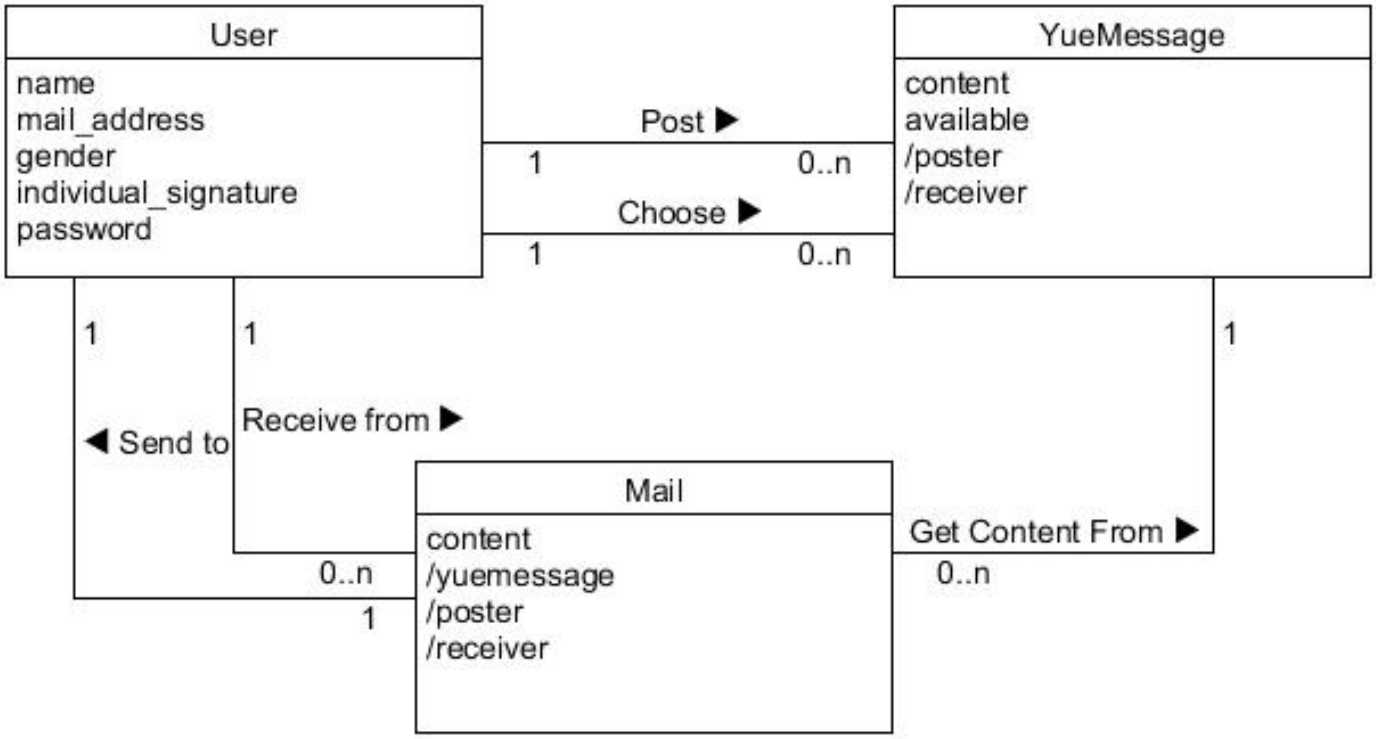
约

约别人状态图：


查看所有约信
浏览状态
[生成失败]
选中某约信
正在生成赴约邮件
[生成成功]
[双方不满意]
正在发送给约发布者
[发送失败]
[发送成功]
线下交涉
[双方满意]
赴约成功

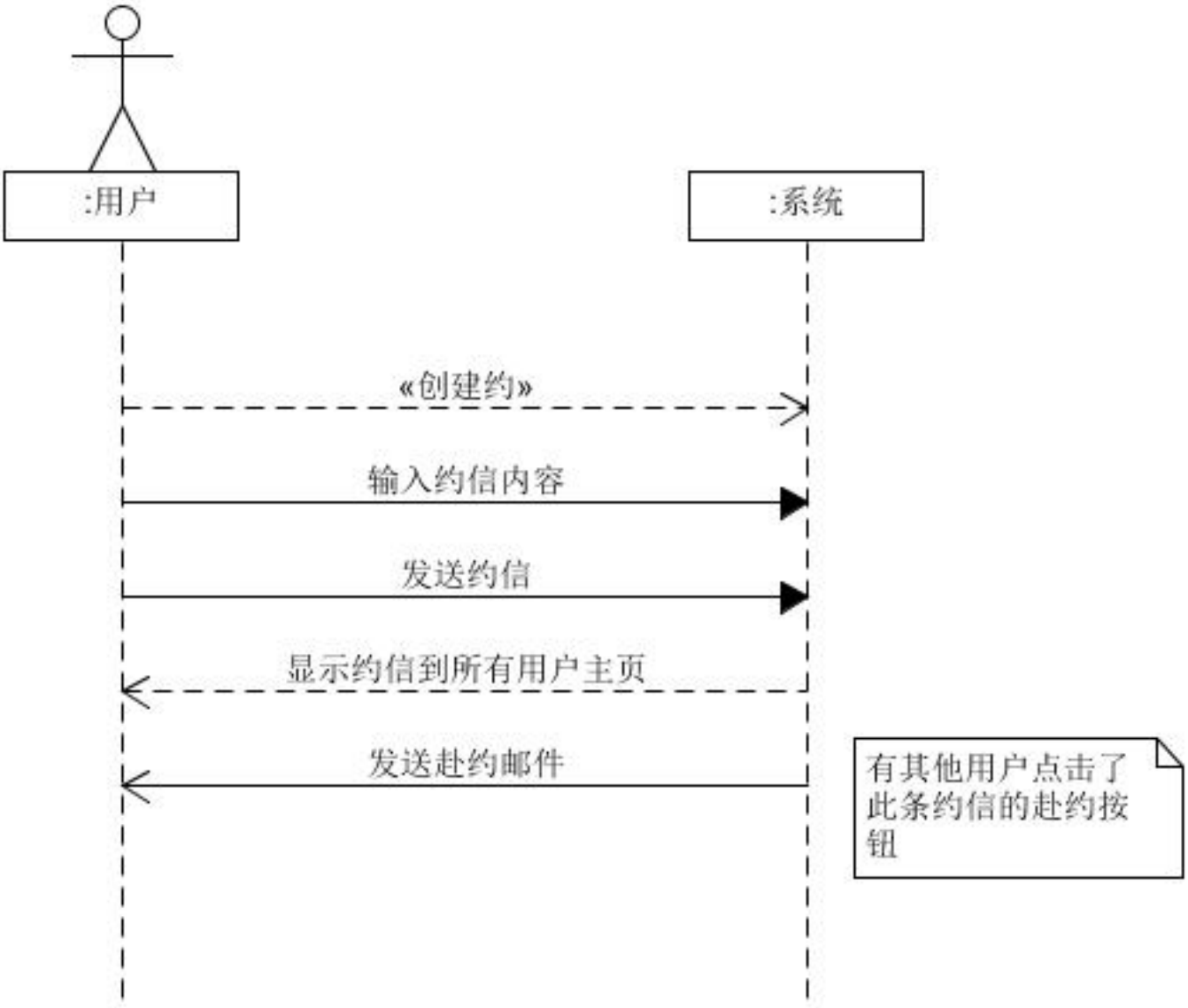
约

领域模型：


User
name
mail_address
gender
individual_signature
password
YueMessage
content
available
/poster
/receiver
Post ▶
1
0..n
Choose ▶
1
0..n
1
1
1
◀ Send to
Receive from ▶
Mail
content
/yuemessage
/poster
/receiver
Get Content From ▶
0..n
0..n
1

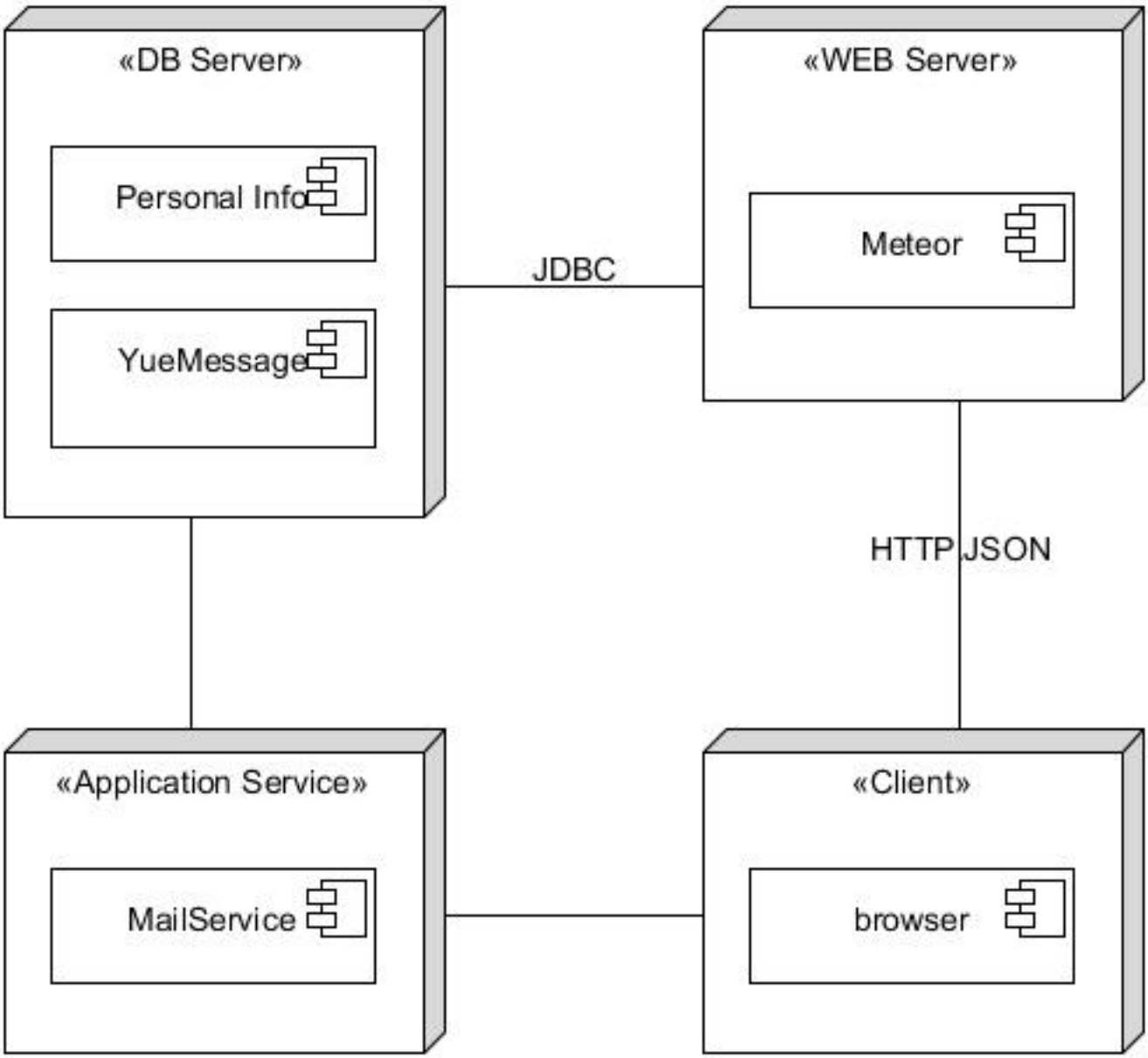
约

系统顺序图：


:用户
:系统
«创建约»
输入约信内容
发送约信
显示约信到所有用户主页
发送赴约邮件
有其他用户点击了此条约信的赴约按钮

约

部署图：


«DB Server»
Personal Info
YueMessage
JDBC
«WEB Server»
Meteor
HTTP JSON
«Application Service»
MailService
«Client»
browser

约

模块划分

所谓软件的模块划分是指在软件设计过程中，为了能够对系统开发流程进行管理，保证系统的稳定性以及后期的可维护性，从而对软件开发按照一定的准则进行模块的划分。根据模块来进行系统开发，可提高系统的开发进度，明确系统的需求，保证系统的稳定性。

在系统设计的过程中，由于每个系统实现的功能不同，所以每个系统的需求也将会不同。也就导致了系统的设计方案不同。在系统的开发过程中，有些需求在属性上往往会有一定的关联性，而有些需求之间的联系很少。如果在设计的时候，不对需求进行归类划分的话，在后期的过程中往往会造成混乱。

软件设计过程中通过对软件进行模块划分可以达到一下的好处：

(1) 使程序实现的逻辑更加清晰，可读性强。
(2) 使多人合作开发的分工更加明确，容易控制。
(3) 能充分利用可以重用的代码。
(4) 抽象出可公用的模块，可维护性强，以避免同一处修改在多个地方出现。
(5) 系统运行可方便地选择不同的流程。
(6) 可基于模块化设计优秀的遗留系统，方便的组装开发新的相似系统，甚至一个全新的系统。

按任务需求进行模块划分的主要步骤如下：

(1) 分析系统的需求，得出需求列表；
(2) 对需求进行归类，并划分出优先级；
(3) 根据需求对系统进行模块分析，抽取出核心模块；
(4) 将核心模块进行细化扩展，逐层得到各个子模块，完成模块划分。

由于我们的约跑应用是打在在 Meteor 上的 web 应用，是一个小型的项目，所以我们在模块划分上，分为 3 个模块

1. html 与 css 界面的设计与编码，将简单的界面，界面上的每个 div 进行功能的归类。

2. MongoDB 数据库的搭建，将我们所需要的用户的账号，密码，个人信息，发布的约信进行数据库的建立，以及接口的实现，方便 javascript 直接调用。

3. 将界面的交互用 javascript 实现，并且实现发布邮件的主要功能。

约

软件设计技术

Structure Programming、

Object-Oriented Programming

Design Patterns

数据库代码截图：

collection.js

```
User = new Meteor.Collection("User");//global variable

post=new Meteor.Collection("post");

Mess=new Meteor.Collection("Message");
```

yue.js

```
post.insert({Mail:mail,Content:content,Time:date,ID:id});
 Template.page.helpers({
        'client': function() {
                return post.find({},{sort:{ID:-1}});
        }
 });
```

login.js

```
 if(User.find({Mail:mail,Pass:pass}).count()==1) {
```

约

personal.js

```
else
{
   post.insert({Mail:mail,Content:content,Time:date,ID:id});
}

 return post.find({Mail:mail},{sort:{ID:-1}});
```

register.js

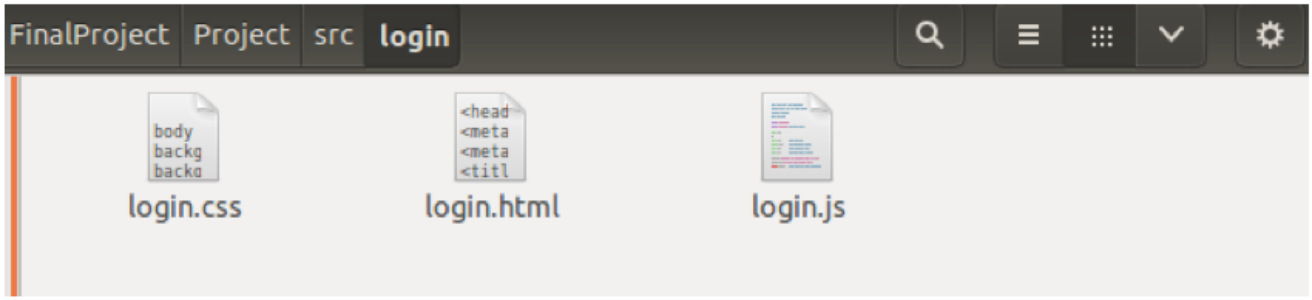
```
User.insert({Mail:mail,Pass:pass});
var url="http://p3.gexing.com/touxiang/20121101/1742/509243ee3badd_200x200_3.jpg";
var id=Mess.find().count()+1;
Mess.insert({Mail:mail,image:url,name:"昵称",self:"编辑你的个性签名^o^",ID:id});
```

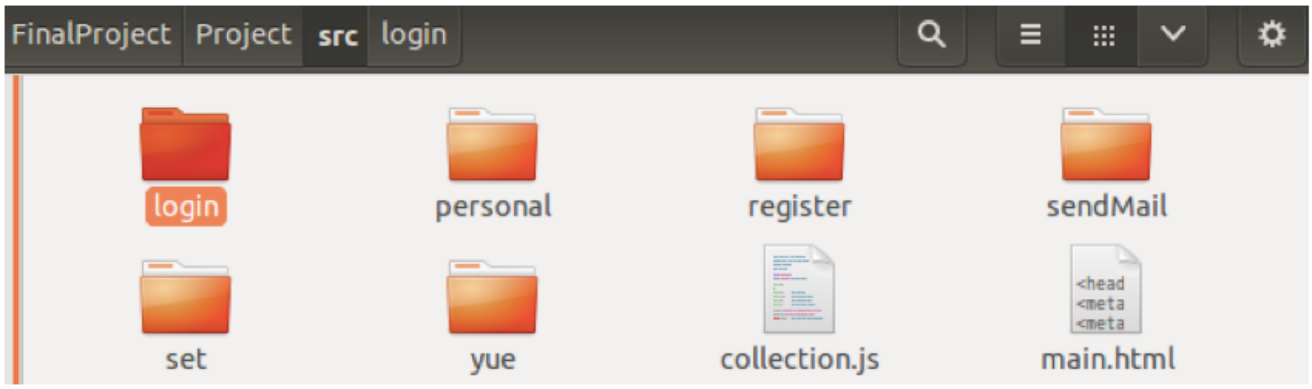
set.js

```
var id=Mess.find().count()+1;
Mess.insert({Mail:mail,image:url1,name:name1,self:self1,ID:id});
```

约

4.项目是把每个页面分为一个 template,然后通过路由确定要跳转到哪个页面，每个页面对应一个 js 文件，一个 html 文件和一个 css 文件





5.路由部分的代码在 collection.js 文件中:

约

```
var urlRouter = Backbone.Router.extend({
  initialize:function(){
    this.route(/^page/,"page");
    this.route("","login");
    this.route("reg","reg");
    this.route(/^sendMail/,"sendMail");
    this.route(/^setting/,"setting");
    this.route(/^personal/,"personal");
},
  login: function () {
    Session.set("currentUrl", {login: "active", page: "", reg: "",sendMail:"",setting:"",personal:""});
  },
  page: function () {

    Session.set("currentUrl", {login: "", page: "active", reg: "",sendMail:"",setting:"",personal:""});
  },
  reg: function () {

    Session.set("currentUrl", {login: "", page: "", reg: "active",sendMail:"",setting:"",personal:""});
  },
  sendMail: function () {

    Session.set("currentUrl", {login: "", page: "", reg: "",sendMail:"active",setting:"",personal:""});
  },
  setting: function () {

    Session.set("currentUrl", {login: "", page: "", reg: "",sendMail:"",setting:"active",personal:""});
  },
  personal: function () {

    Session.set("currentUrl", {login: "", page: "", reg: "",sendMail:"",setting:"",personal:"active"});
  },
```

6、template 的分块可以在 main.html 中看到

约

```
<body id="main">
{{> container}}
</body>

<template name="container">
  <div id="container" class="container">



    {{#if currentUrl.reg}}
      {{> reg}}

    {{/if}}

    {{#if currentUrl.login}}
      {{> login}}

    {{/if}}


    {{#if currentUrl.page}}
      {{> page}}
    {{/if}}

    {{#if currentUrl.sendMail}}
      {{> sendMail}}
    {{/if}}

    {{#if currentUrl.setting}}
      {{> setting}}
    {{/if}}

    {{#if currentUrl.personal}}
      {{> personal}}
    {{/if}}
```