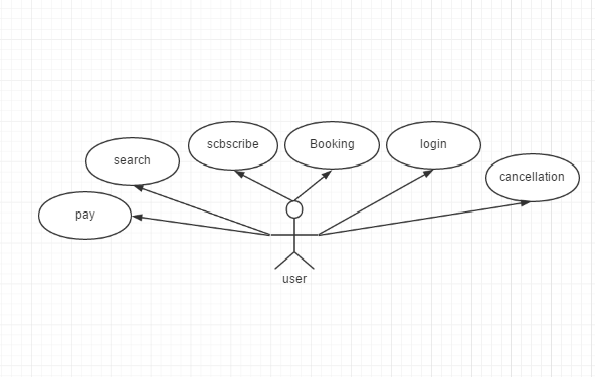
一需求

1.1功能需求

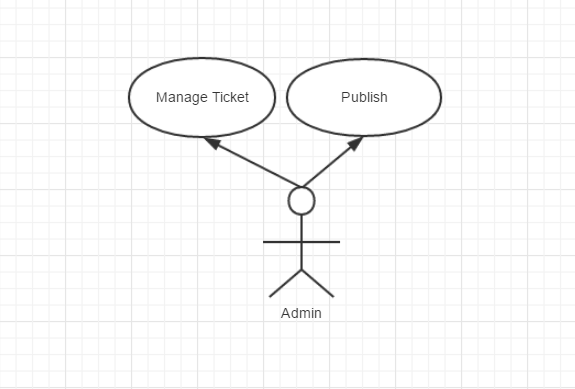
旅行管理系统为用户提供在线选票购票，退票的服务，并且维护用户的个人信息。功能需求表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 功能需求 | 描述 | 输入 | 输出 |
| 1 | 查询票 | 用户输入搜索条件来查询票务的信息 | 地点，时间，价格区间 | 符合条件的票列表 |
| 2 | 订票 | 用户选择票，预订该票 | 所选择的票，绑定购票人信息 | 预订成功或失败 |
| 3 | 退订 | 用户选择自己已经预订或者购买的票，点击退订 | 已经订购的某些票 | 退订成功，将资金返还给用户的账户，退订失败，提示用户 |
| 4 | 订阅 | 用户选择廉价票推送，订阅推送 | 选择一个订阅点（现在只有一个订阅点），订阅廉价票信息 | 订阅成功或者失败 |
| 5 | 支付 | 用户选择已预订的票，并且是未支付状态的进行支付 | 预订未支付的票 | 支付成功，或者失败 |
| 6 | 用户登录 | 用户登录系统 | 用户名和密码 | 登录成功或者失败，提示错误信息 |
| 7 | 管理票务信息 | 管理员管理票务信息 | 设置票的库存和价格 | 设置成功或者失败 |
| 8 | 发布 | 管理员发布廉价票的信息 | 发布廉价票信息到topic ，会将这些廉价票的信息推送至已订阅该信息的用户那里 | 发布成功 |

用例图如下：

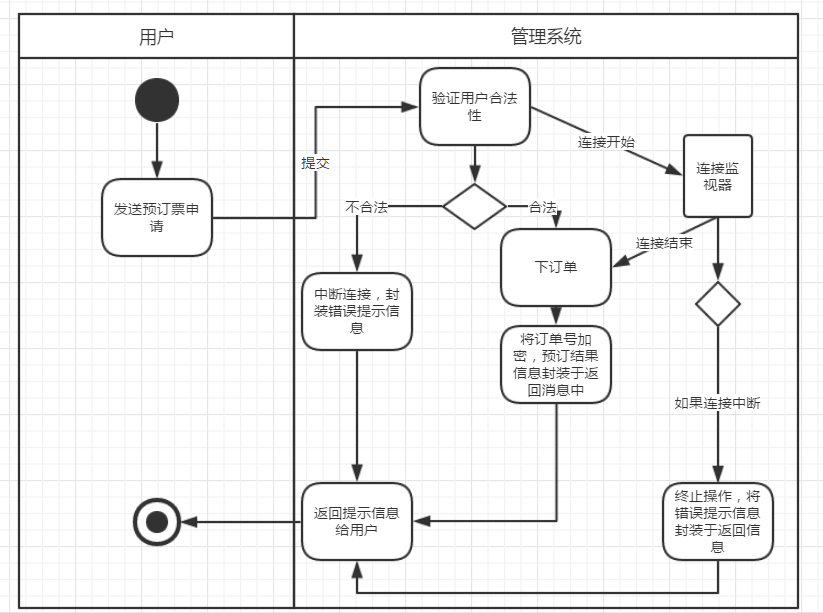


用户用例

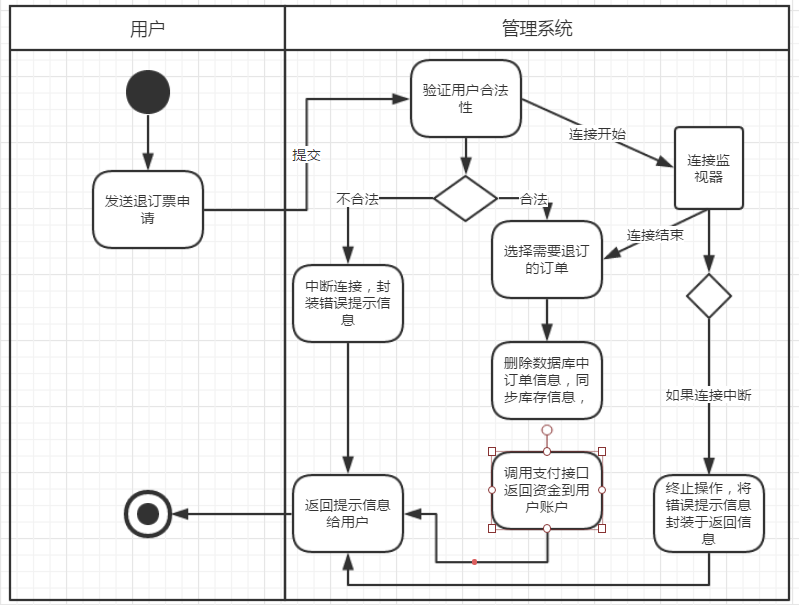


管理员用例图

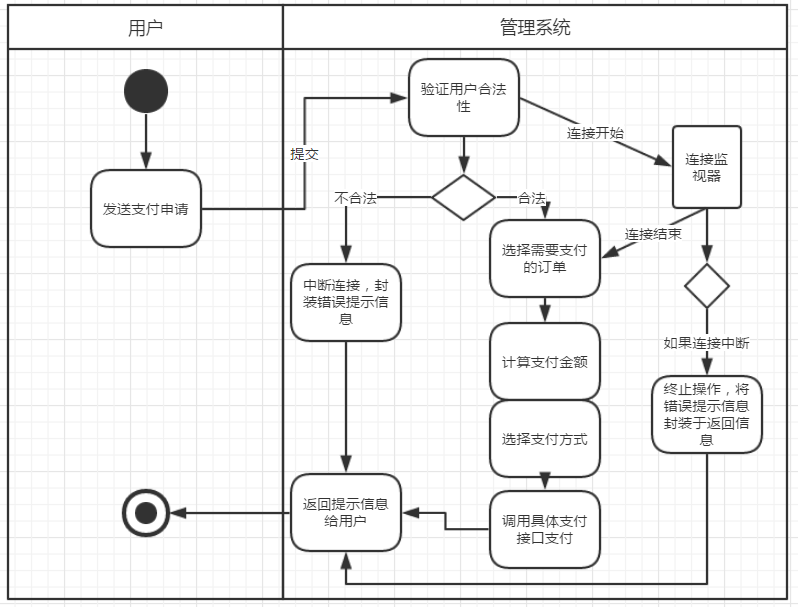
1.2各用例活动图



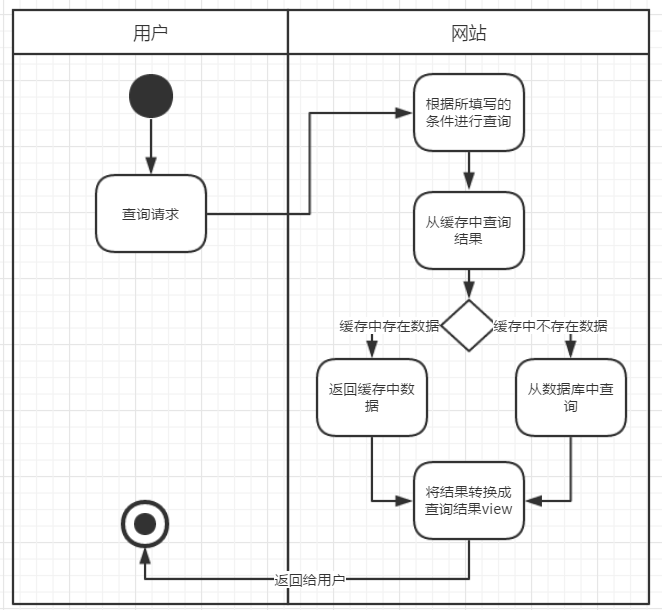
订票流程



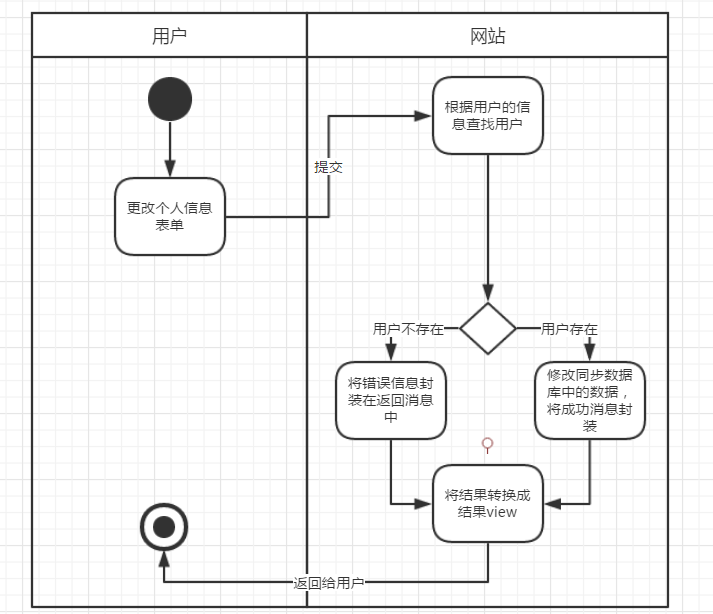
退订流程



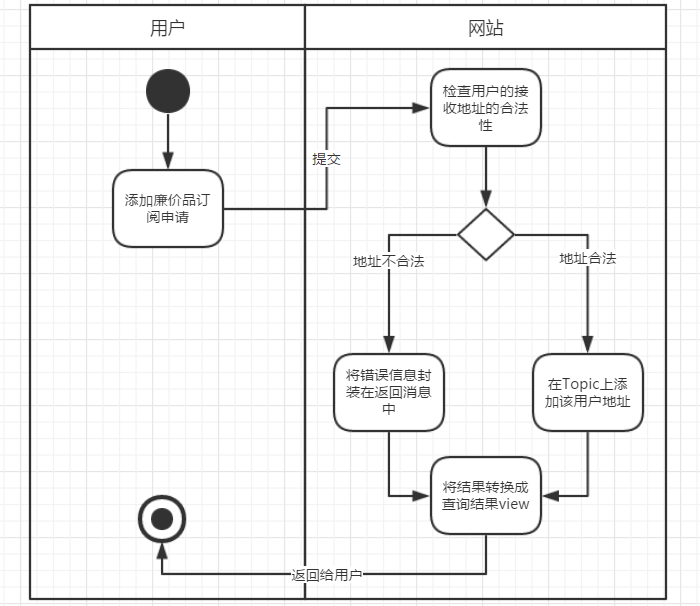
支付流程



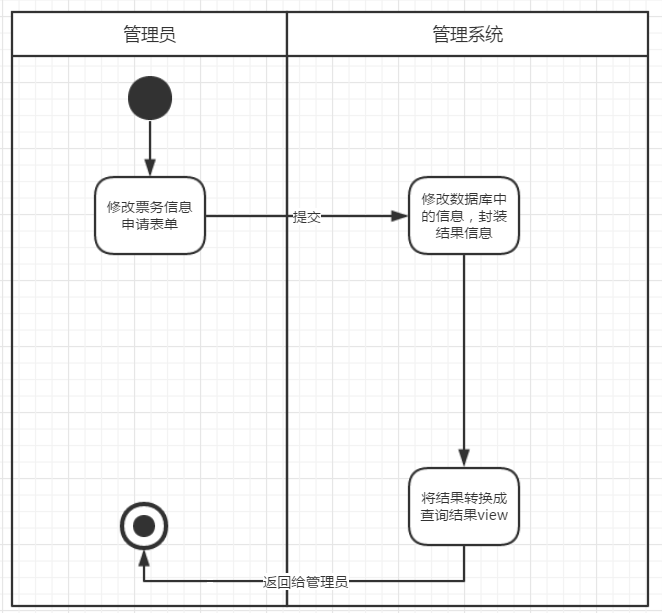
查询流程



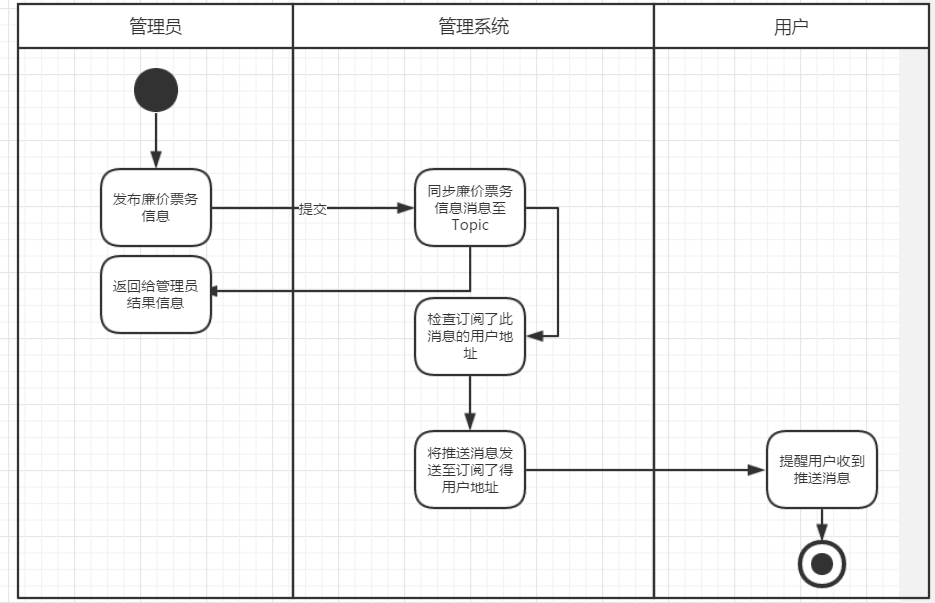
更改个人信息流程



订阅信息流程

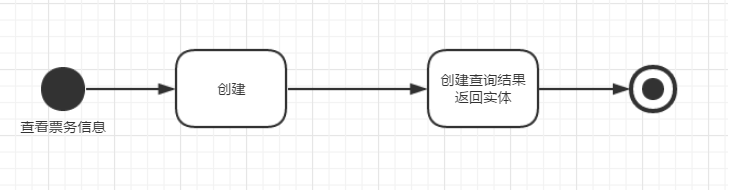


管理票务流程

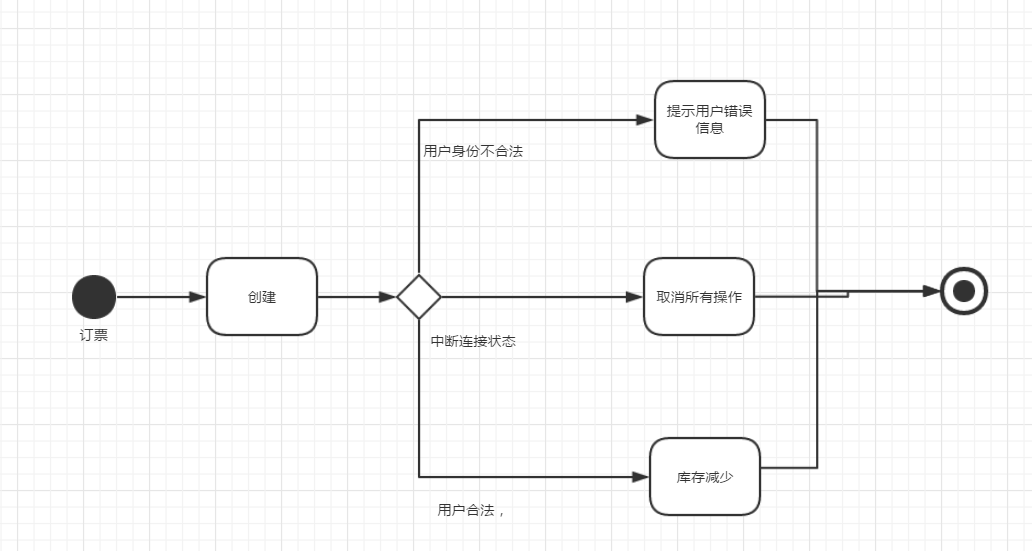


发布推送流程

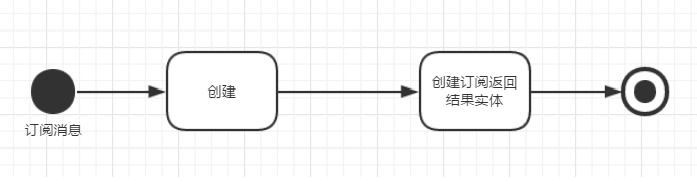
1.3用例状态图



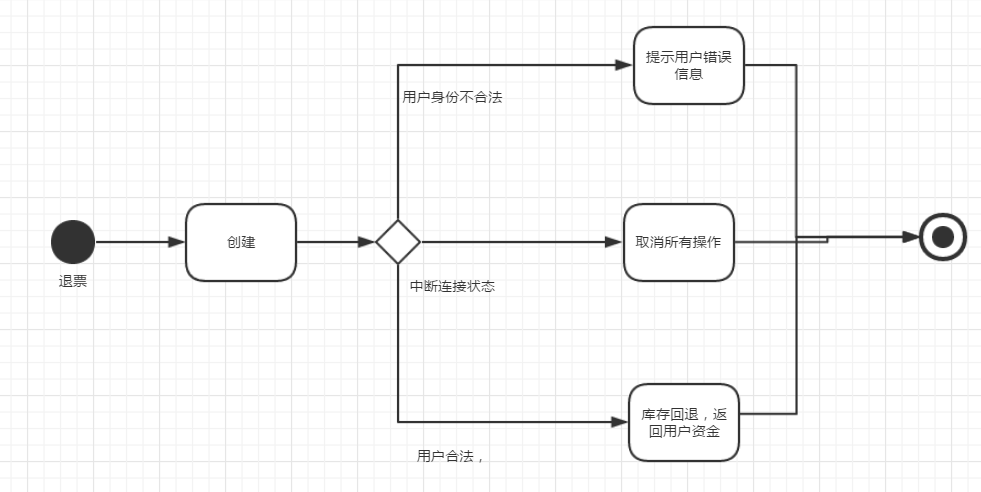
查询状态



订票状态

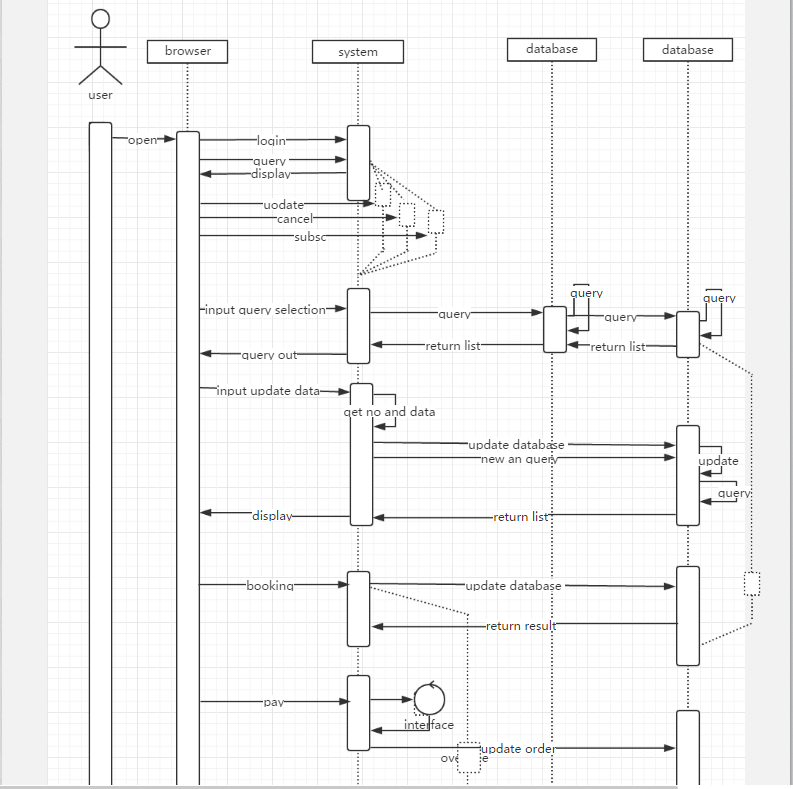


订阅状态



退票状态

1.4用例的时序图

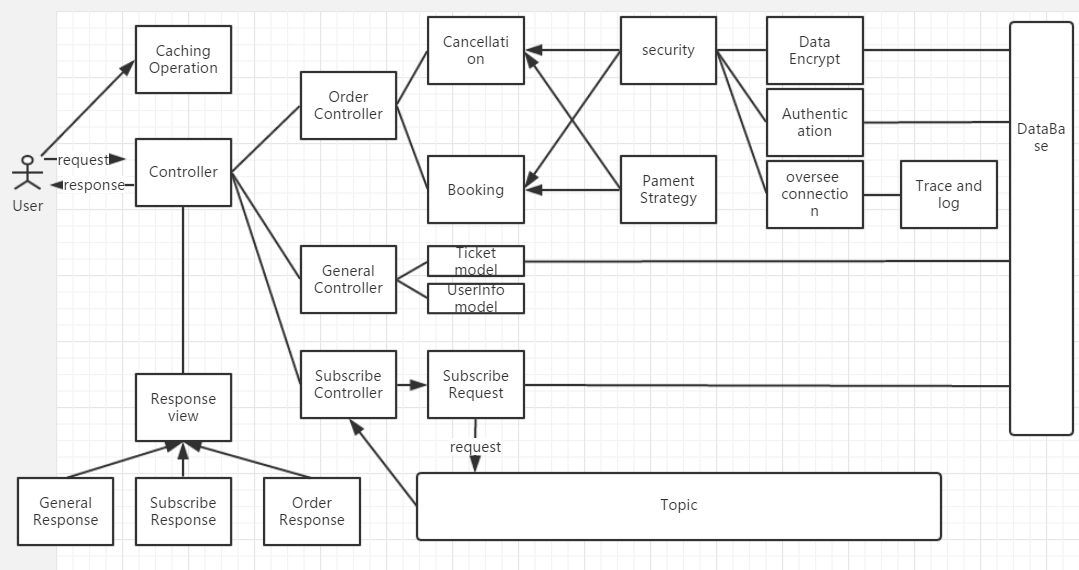


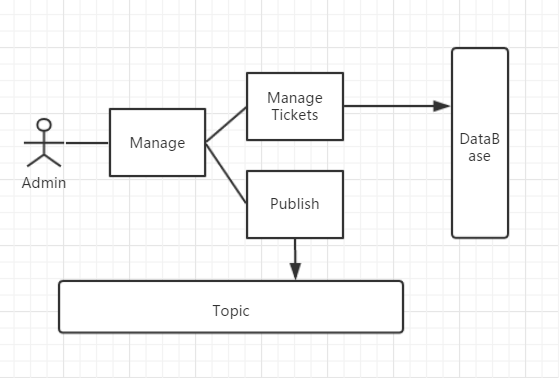
用例时序图

1.5系统的质量需求

应该主要满足安全性，和可修改性，其次，性能也比较重要，不能让用户等待太久。

二 设计

2.1系统整体架构



首先用户所使用的客户端应为浏览器，进入系统之后，可以通过浏览器向服务器发送请求，主要有查询票，订票，退订，管理个人信息，订阅推送的功能，另外，为了用户使用方便，添加了一个缓存用户操作的组件，防止用户进行误操作之后不能返回上一步。在预订和退订的时候需要保障安全，添加了安全组件。主要有数据加密，身份认证和监视连接的功能。一旦用户在进行订票和退票操作的途中断开了与服务器的连接，则视为交易不安全，中断所有操作。记录log，提示用户。因为考虑到用户可能会用很多不同的方式支付，所以这里使用策略模式来做用户支付的策略。方便以后增加和修改新的支付方式。用户可以订阅推送廉价票的信息。将用户的订阅绑定在话题上，如果有新的推送的话，可以发送到用户的地址告知用户。

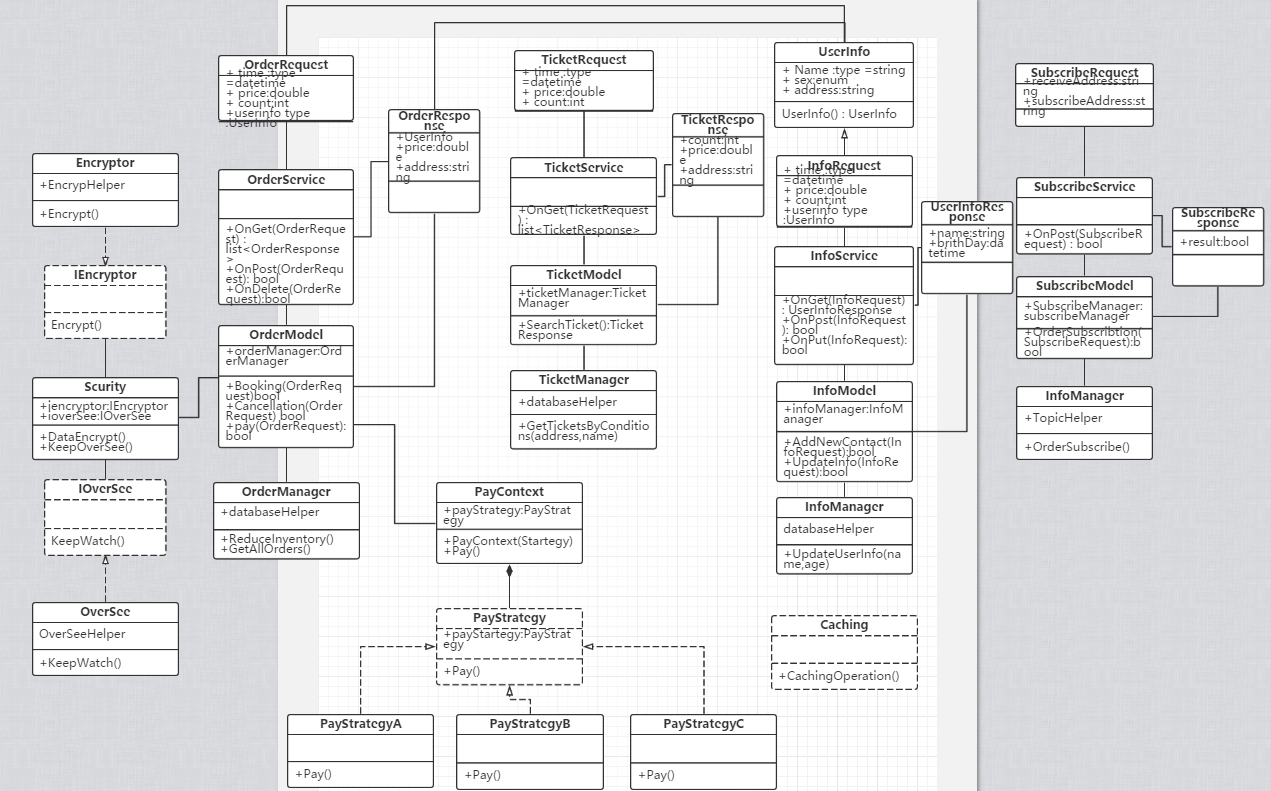
管理员的模块主要有发布廉价票推送消息到话题。还有管理票务信息。设置票的价格和库存以及一些基本的信息。

2.2系统数据设计

这次设计我想用REST风格来划分资源。所有的请求都是围绕资源的操作。

主要的资源有User ,Ticket,Order, subscription四个资源。

2.3UML类图设计



针对数据设计的四个资源对API进行划分。定义WEB API 入口，OrderRequest ，TicketRequest，InfoRequest，SubscribeRequest，这里需要定义这些请求实体里需要有的属性，比如InfoRequest里需要有用户名，联系方式等字段。接下来定义了API具体的四个动作，GET ,POST,PUT,DELETE在Service中，同样对应于四个资源。再定义需要返回给客户端的数据类型。Response。比如票的信息，需要指定数据的Content-type为applications/json。票价，地址等。

接下来是业务层，Model。这里需要写具体的业务操作，预订和退订的时候需要有安全模块的支撑。安全模块用到了两个接口，数据加密接口和监视连接的接口。OrderModel实现了订票退票，支付功能。在查询订单或者添加订单，取消订单的时候会更新数据库中的信息。单独用一个类来实现数据库的操作方法，降低业务和具体数据操作的耦合性，提高可修改性。由于支付的手段可能会有很多，在未来也会不断地增加，一般是调用三方支付系统的接口进行实现。这里维护了一个支付的策略接口，用具体的策略来实现。在以后如果有其他的支付方式加入就可以再新加策略类就可以了。其他两个资源的业务层类似，订阅的业务层主要交互的是话题管理器而不是数据库。

加入一个缓存操作的类使用操作缓存接口来帮助缓存用户的操作，方便用户返回上一步和撤销动作。

数据库操作的部分可以使用ORMapping 的组件来帮助完成数据库连接以及sql。

2.4API设计

GetTickets

|  |  |
| --- | --- |
| URI | /travel/tickets |
| Method | GET |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | N/A |
| Response | **{**  **"Inventory"**: 3**,**  **"Price"**: 3000 ,  **"Destination"**: **"Tokyo"**  **}** |
|  |  |

SetTickets

|  |  |
| --- | --- |
| URI | /travel/tickets |
| Method | POST |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"Inventory"**: 3**,**  **"Price"**: 3000 ,  **"Destination"**: **"Tokyo"**  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

UpdateTickets

|  |  |
| --- | --- |
| URI | /travel/tickets |
| Method | PUT |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"Inventory"**: 3**,**  **"Price"**: 3000 ,  **"Destination"**: **"Tokyo"**  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

ChangeUserInfo

|  |  |
| --- | --- |
| URI | /travel/user |
| Method | PUT |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"name"**: **"Lucy",**  **"age"**: 30 ,  **"sex"**: **"1"**  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

GetOrders

|  |  |
| --- | --- |
| URI | /travel/order/{sellerid} |
| Method | GET |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | N/A |
| Response | **{**  **"OrderNumber"**: "20012315234326321"  **"TotalPrice"**: 5000,  **}** |
|  |  |

AddOrders

|  |  |
| --- | --- |
| URI | /travel/order/{sellerid} |
| Method | POST |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"Action"**: "Add",  **"Destination"**: "Tokyo",  **"TotalPrice"**: 5000,  **"Count"**: 2,  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

PayOrders

|  |  |
| --- | --- |
| URI | /travel/order/{sellerid} |
| Method | POST |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"Action"**: "Pay",  **"OrderNumber"**: 5646516315674899,  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

DeleteOrders

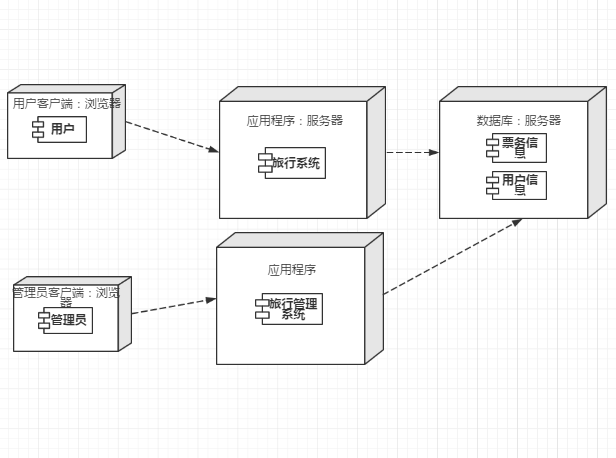
|  |  |
| --- | --- |
| URI | /travel/order/{sellerid} |
| Method | DELETE |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"OrderNumber"**: 5646516315674899,  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

AddScrib

|  |  |
| --- | --- |
| URI | /travel/scrib/ |
| Method | POST |
| Header | Accept:Application/json  Content-type:Application/json |
| Body | **{**  **"ScribAddress"**:**"sgfsa56461235"**  **}** |
| Response | **{**  **"ResponseCode"**: "200"  **"HasError"**: false  **"ErrorMessage"**: ""  **}** |
|  |  |

2.5部署图

系统的部署图为

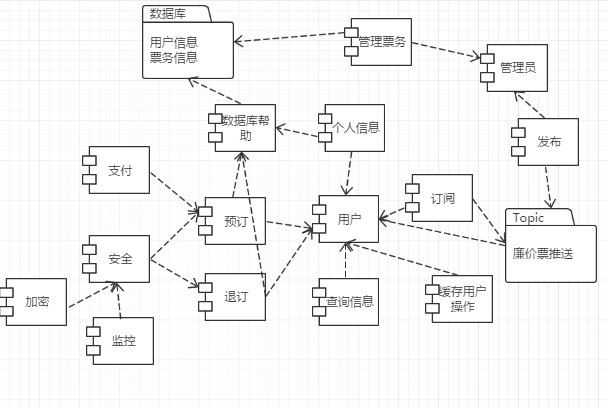


部署图

用户使用浏览器发送请求到服务器，服务器连接后台数据库。管理员使用浏览器登录系统进行管理。

2.6组件图

系统组件图为



系统组件图

使用数据库帮助组件完成各个业务中数据库操作的部分。在预订的时候需要用到支付的组件，退订和预订模块为了保障安全性，使用到安全组件，加密组件和监控组件。另外使用到一个缓存用户操作的组件来保存用户的操作。用到订阅组件在话题上订阅推送。

管理员使用发布组件在话题上发布廉价票信息推送。票务管理组件来设置票的信息。