## 一 vue部分

### 1 对于MVVM的理解？

MVVM 是 Model-View-ViewModel 的缩写。

Model代表数据模型，也可以在Model中定义数据修改和操作的业务逻辑。

View 代表UI 组件，它负责将数据模型转化成UI 展现出来。

ViewModel 监听模型数据的改变和控制视图行为、处理用户交互，简单理解就是一个同步View 和 Model的对象，连接Model和View。

在MVVM架构下，View 和 Model 之间并没有直接的联系，而是通过ViewModel进行交互，Model 和 ViewModel 之间的交互是双向的， 因此View 数据的变化会同步到Model中，而Model 数据的变化也会立即反应到View 上。

ViewModel通过双向数据绑定把 View 层和 Model 层连接了起来，而View 和 Model 之间的同步工作完全是自动的，无需人为干涉，因此开发者只需关注业务逻辑，不需要手动操作DOM, 不需要关注数据状态的同步问题，复杂的数据状态维护完全由 MVVM 来统一管理。

### 2 vue几种常用的指令。

v-for 循环

v-if 判断

v-bind 为标签绑定值

v-on 注册事件

v-show 根据表达式之真假值，切换元素的 display CSS 属性

v-else 当v-if表达式不成立时, v-else执行，与v-if结合起来用

### 3 什么是组件？

组件是 Vue.js 最强大的功能之一。组件是对特定功能代码(html,css,js)的封装, 通过组件的名字可以重复利用该组件中的代码.

组件分为全局组件和局部组件

全局组件:在所有vue实例中(.在它所挂载元素下面有效)有效

局部组件:在自己vue实例中(.在它所挂载元素下面有效)有效

### 4 vue的生命周期。

Vue实例从创建到销毁的过程，就是生命周期。从开始创建、初始化数据、编译模板、挂载Dom→渲染、更新→渲染、销毁等一系列过程，称之为 Vue 的生命周期。

Vue的生命周期中有多个事件钩子，让我们在控制整个Vue实例的过程时更容易形成好的逻辑。

Vue生命周期可以总共分为8个阶段：创建前/后, 载入前/后,更新前/后,销毁前/销毁后。

### 5 什么是路由？

路由是负责将进入的浏览器请求映射到特定的组件代码中。即决定了由谁(组件)去响应客户端请求。简单说路由就是url地址和对应的资源的映射，通过一个路径的url地址，可以唯一找到一个资源。路由不包含在vue中，是一个插件，需要单独下载。

## 二 前后端分离

### 1 前后端分离的理解

在传统的web应用开发中，大多数的程序员会将浏览器作为前后端的分界线。将浏览器中为用户进行页面展示的部分称之为前端，而将运行在服务器，为前端提供业务逻辑和数据准备的所有代码统称为后端。

前后端分离不只是一种web应用开发模式，只要在web应用的开发期进行了前后端开发工作的分工就是前后端分离。这个只能叫前后端分工。

其实前后端分离并不只是开发模式，而是web应用的一种架构模式。在开发阶段，前后端工程师约定好数据交互接口，实现并行架构，开发和测试；在运行阶段前后端分离模式需要对web应用进行分别部署，前后端之间使用HTTP或者其他协议进行交互请求。

前后端分离的优缺点：

优点：

分工明确，专业的人做专业的事情。

专业可以做出炫丽优美界面

人员培养更加专业化，精确化

缺点：

成本提高，人员多。

项目管理难度也提高，前后端都要管，协调他们之间交流。

### 2 前端怎么独立开发？

我们采用的是Element，element是一套为开发者、设计师和产品经理准备的基于 Vue 2.0 的桌面端组件库，提供了很多的ui组件。我们可以使用这些组件快速开发前端项目。

### 3 swagger是什么？

swagger是接口规范。接口测试人员和前段开发人员要通过接口描述测试接口，所以需要定义一个接口规范。

swagger可以直接罗列所有接口,每个接口有访问地址(访问方式),参数及返回值.可以直接通过后端代码产生能够让前台开发或测试人员能够看懂的接口文档。

### 4 交互方式

在前后端分离架构中，后端只需要负责按照约定的数据格式向前端提供可调用的API服务即可。前后端之间通过HTTP请求进行交互，前端获取到数据后，进行页面的组装和渲染，页面跳转，最终返回给浏览器。

### 5 后端如何进行接口测试?

采用了一个postman工具。postman就是一个http协议接口测试工具,可以来发送各种http请求,可以用它来测试http协议接口.

## 三 Redis

### 1 为什么使用redis？

Redis 是一个使用 C 语言开发的key-value形式的高速缓存nosql数据库。它先存到内存中，然后会根据一定的策略持久化到磁盘,即使断电也不会丢失数据。另外它支持较多的数据类型，可以应用到多个场景。比如；

记录帖子点赞数、点击数、评论数；

缓存近期热帖；

缓存文章详情信息；

记录用户会话信息。

### 2 redis的持久化机制。

redis的持久化其实就是把数据放到磁盘里面，让我们的数据可以一直存在，即使断电也不会造成数据丢失。

Redis 的持久化有两种方式，或者说有两种策略：

RDB（Redis Database）：指定的时间间隔能对你的数据进行快照存储。

AOF（Append Only File）：每一个收到的写命令都通过write函数追加到文件中。

### 3 redis支持的数据类型。

string（字符串）、list（列表）、hash（字典）、set（集合）、zset（有序集合）。

## 四 注册

### 1 注册是怎么做的？

注册包括手机注册和邮箱注册，逻辑都是一样的，只是激活方式不一样。

大致流程都是：

①用户在页面输入手机号/邮箱之后，点击获取验证码。

②后台生成验证码，并将验证码存入redis。（如果问为什么用redis?回答redis的优点就可以了）

③然后调用发送短信/邮件接口，将验证码发送到收件/邮箱。

④用户收到验证码，输入页面，点击注册。后端注册接口要比较用户输入的验证码与redis中存入的验证码比较。如果一致继续下一步，如果失败返回验证码失败，让用户重新输入。

### 2 密码是怎么处理的？

密码采用了md5加密的方式再存入数据库，因为md5是不可逆的，更安全。

## 五 登录

### 1 用户和管理员登录是分开写的吗？

用户和管理员登录是同一个接口，通过前端传过来的type判断是用户还是管理员，然后跳转不同的页面。

### 2 登录这块的表设计是怎么样的？

用户是一张表，管理员是一张表，然后我们还做了一个登录表，登录表里面是所有用户和管理员的登录账号和密码，因为管理员也可以登录我们的网站享受服务，所以有可能一个手机或邮箱既是用户又是管理员。为了区分这种登录账号到底是要登录到后台还是网站，所以我们登录表里面有一个type字段，区分该账号到底是用户还是管理员。

### 3 怎么判断用户是否登录？

登录成功，用uuid生成了一个随机数，将该随机数作为key，用户信息作为value存入redis，然后传到前端，前台取名为token会将它放到localStorage里面，并且我们在每次发起请求到后台时，都会在请求头中放入该token，在后台的拦截器中，我们会获取每一个请求的token，如果没有token或者通过该token在redis中没有获取到相应的用户信息，我们都认为它没有登录，然后返回前台，否则就放行该请求。

### 4 前端发起的所有的请求都会被拦截吗？

除了登录和注册接口，其他的会被拦截，如果登录和注册也被拦截的话，那么程序也就进行不下去了。

### 5 是如何让所有的请求都携带token的？

通过axios前置拦截器，在所有请求发起前，判断token是否存在，如果存在就在请求的header中加入token。

## 六 其他业务

### 1 寻主是什么意思？

用户因为各种原因不想继续抚养自己的宠物。通过系统的寻主功能发布通告，平台根据用户所在的市区查询相关的合作门店，将寻主信息推送给附近的门店。

大概流程：

1 用户发布寻主消息

2 平台接收消息

3 找到最近门店，发送短信消息给，并且把这个消息划归他门下

4 收取宠物，如果要钱的，还要创建订单并完成支付

### 2 是怎么找到最近的合作门店的？

我们集成了百度地图的接口，因为寻主功能需要推算离用户最近的线下门店，系统需要将用户的输入的地址解析为经纬度。再通过经纬度计算哪一家店铺离用户地址最近。

### 3 订单表设计是怎样的?

订单包括：领养订单、商品订单、服务订单、充值订单，因为业务不同，所以这4种订单是分表存放的。另外还有一个用户地址表，因为用户的地址可能不止一个，所以我们有一个专门的地址表存放用户的地址，另外还有一个订单地址表，关联某订单是哪一个收货地址。

### 4 支付是怎么做的？

我们是对接的支付宝的接口。像大平台一般钱先到平台，用户确认后，平台才划账到店家。如果用户长时间不确认，自动确认。我们小平台钱是直接到店家，我们没有支付牌照。

### 5 订单超时未支付会取消吗？是怎么做的？

我们订单超过30分钟未支付就会取消，是通过定时器来实现的。

## 七 docker

### 1 什么是docker

Docker 是一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的镜像中，然后发布到任何流行的 Linux或Windows 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口

### 2 容器与虚拟机比较

（1）本质上的区别-共享操作系统，不需要安装额外操作系统

（2）使用上的区别，虚拟机已死，容器才是未来。

### 3 docker特点（为什么用docker）

（1）上手快。

用户只需要几分钟，就可以把自己的程序“Docker化”。Docker依赖于“写时复制”（copy-on-write）模型，使修改应用程序也非常迅速，可以说达到“随心所致，代码即改”的境界。

随后，就可以创建容器来运行应用程序了。大多数Docker容器只需要不到1秒中即可启动。由于去除了管理程序的开销，Docker容器拥有很高的性能，同时同一台宿主机中也可以运行更多的容器，使用户尽可能的充分利用系统资源。

（2）职责的逻辑分类

使用Docker，开发人员只需要关心容器中运行的应用程序，而运维人员只需要关心如何管理容器。Docker设计的目的就是要加强开发人员写代码的开发环境与应用程序要部署的生产环境一致性。从而降低那种“开发时一切正常，肯定是运维的问题（测试环境都是正常的，上线后出了问题就归结为肯定是运维的问题）” 解决环境差异

（3）快速高效的开发生命周期

Docker的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让你的应用程序具备可移植性，易于构建，并易于协作。（通俗一点说，Docker就像一个盒子，里面可以装很多物件，如果需要这些物件的可以直接将该大盒子拿走，而不需要从该盒子中一件件的取。）

（4）鼓励使用面向服务的架构

Docker还鼓励面向服务的体系结构和微服务架构。Docker推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或者服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的内省性。（当然，可以在一个容器中运行多个应用程序）

### 4 docker组件

（1）Docker客户端和服务器

（2）Docker镜像

（3）Registry（注册中心）

（4）docker容器

### 5 什么是docker镜像

Docker镜像是由文件系统叠加而成（是一种文件的存储形式）。最底端是一个文件引导系统，即bootfs，这很像典型的Linux/Unix的引导文件系统。Docker用户几乎永远不会和引导系统有什么交互。实际上，当一个容器启动后，它将会被移动到内存中，而引导文件系统则会被卸载，以留出更多的内存供磁盘镜像使用。Docker容器启动是需要的一些文件，而这些文件就可以称为Docker镜像。

### 6 常用的docker命令

systemctl命令是系统服务管理器指令，它是 service 和 chkconfig 两个命令组合。

启动docker：systemctl start docker

停止docker：systemctl stop docker

重启docker：systemctl restart docker

查看docker状态：systemctl status docker

开机启动：systemctl enable docker

查看docker概要信息：docker info

查看docker帮助文档：docker --help

列出docker下的所有镜像：docker images

搜索镜像 docker search 镜像名称

拉取镜像 docker pull centos:7

删除指定镜像 docker rmi $IMAGE\_ID

删除所有镜像 docker rmi `docker images -q`

查看正在运行的容器 docker ps

查看最后一次运行的容器 docker ps –l

创建容器命令：docker run

### 7 什么是dockerfile

Dockerfile 是一个用来构建镜像的文本文件，文本内容包含了一条条构建镜像所需的指令和说明。

对于开发人员，可以为开发团队提供一个完全一致的开发环境

对于测试人员，可以直接拿开发时所构建的镜像或者通过Dockerfile文件构建一个新的镜像开始工作了。

对于运维人员，在部署时可以实现应用的无缝移植。

### 8 什么是docker-compse

一个一个容器的创建或启动比较麻烦，期望一键操作。这时需要编排工具.而compose是最简单的一个。

D ocker Compose 是 Docker 官方编排（Orchestration）项目之一，负责快速的部署分布式应用(多个应用)。

Compose 中有两个重要的概念：

服务 (service)：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例。

项目 (project)：由一组关联的应用容器组成的一个完整业务单元，在 docker-compose.yml 文件中定义。