项目前导

ju7ran

目　录

Redis

1-Redis介绍

Redis内存数据库

背景

随着互联网+大数据时代的来临，传统的关系型数据库已经不能满足中大型网站日益增长的访问量和数据量。这个时候就需要一种能够快速存取数据的组件来缓解数据库服务I/O的压力，来解决系统性能上的瓶颈。

数据库的发展历史

1.在互联网+大数据时代来临之前，企业的一些内部信息管理系统，一个单个数据库实例就能满足系统的需求
单数据库实例

2.随着系统访问用户的增多，数据量的增大，单个数据库实例已经满足不了系统的读取需求
缓存（memcache）+单数据库实例

3.缓存可以缓解系统的读取压力，但是数据量的写入压力持续增大，
缓存+主从数据库+读写分离

4.数据量再次增大，读写分离以后，主数据库的写库压力出现瓶颈、
缓存+主从数据库集群+读写分离+分库分表

5.互联网+大数据时代来临，关系型数据库不能很好的存取一些并发性高，实时性高的，并且数据格式不固定的数据。
nosql+主从数据库集群+读写分离+分库分表

Redis是什么？

Redis是一个高性能的，开源的，C语言开发的，键值对存储数据的nosql数据库。

NoSQL：not only sql，泛指非关系型数据库 Redis/MongoDB/Hbase Hadoop

关系型数据库：MySQL、oracle、SqlServer

Redis特性

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用
- Redis不仅仅支持简单的key-value类型的数据，同时还提供List，set等数据类型
- Redis支持数据的备份

Redis有什么用？

Redis的主要作用：快速存取

Redis应用场景

点赞/秒杀/直播平台的在线好友列表/商品排行榜/单点登录

Redis怎么用？

官网地址：https://redis.io/

命令地址：http://doc.redisfans.com/

Redis的五大数据类型以及应用场景

```
string/list/set/hash/zset
```

Redis的安装及启动

```
sudo apt-get install redis-server

查看帮助命令
redis-server --help

编辑Redis配置文件
sudo vim /etc/redis/redis.conf
将daemonize no改为 daemonize yes保存退出

启动
redis-server

打开服务
sudo service redis start

关闭服务
```

```
sudo service redis stop
```

Redis的配置文件

```
/etc/redis/redis.conf

当redis作为守护进程运行的时候，它会写一个 pid 到 /var/run/redis.pid 文件里面。
daemonize no

监听端口号，默认为 6379，如果你设为 0 ，redis 将不在 socket 上监听任何客户端连接。
port 6379

设置数据库的数目。
databases 16

根据给定的时间间隔和写入次数将数据保存到磁盘
下面的例子的意思是：
900 秒内如果至少有 1 个 key 的值变化，则保存
300 秒内如果至少有 10 个 key 的值变化，则保存
60 秒内如果至少有 10000 个 key 的值变化，则保存

save 900 1
save 300 10
save 60 10000

监听端口号，默认为 6379，如果你设为 0 ，redis 将不在 socket 上监听任何客户端连接。
port 6379

Redis默认只允许本地连接，不允许其他机器连接
bind 127.0.0.1
```

更多配置文件:https://www.cnblogs.com/kreo/p/4423362.html

Redis数据库简单使用

```
DBSIZE       查看当前数据库的key数量
keys *       查看key的内容
FLUSHDB      清空当前数据库的key的数量
FLUSHALL     清空所有库的key(慎用)
exists key    判断key是否存在
```

redis常用五大数据类型

1.redis-string

string是redis最基本的类型，一个key对应一个value

string可以包含任何数据，最大不能超过512M

1.set/get/del/append/strlen

```
set  ---- 设置值
get  ---- 获取值
mset  ---- 设置多个值
mget  ---- 获取多个值
append ---- 添加字段
del ---- 删除
strlen ---- 返回字符串长度
```

2.incr/decr/incrby/decrby

```
incr ---- 增加
decr ---- 减少
incrby  ----- 制定增加多少
decrby  ----- 制定减少多少
```

3.getrange/setrange

```
getrange ---- 获取指定区间范围内的值,类似between....and的关系
setrange ---- 代表从第几位开始替换,下脚本从零开始
从0 -1表示全部
```

2.redis-list(单值多value)

List(列表)

列表是简单的字符串列表，按照插入顺序排序，可以添加一个元素列表的头部（左边）或者尾部（右边）

它的底层实际是个链表

1.lpush/rpush/lrange

```
lpush/rpush/lrange ---- 从左/从右/获取指定长度
lpush list01  1 2 3 4 5  倒序排列
rpush list02  1 2 3 4 5  正序排列
lrange  list01  0  -1  获取list01 中的所有值
```

2.lpop/rpop

```
lpop/rpop ---- 移除最左/最右
lpop list01 删除元素5
rpop list01 删除元素1
```

3.lindex，按照索引下标获得元素（从上到下）

```
lrange list01 0 -1
lindex list01 1
```

4.llen,求列表长度

```
llen list01
```

5.lrem key

```
删N个value
lrem list01 2 1   在list01中删除2个1
```

6.ltrim key

```
ltrim ---- 开始index结束index,截取指定范围的值后在赋值给key
ltrim list01 0 2    截取list01 从0到2的数据在赋值给list01
```

7.rpoplpush list1 list2 将list1中最后一个压入list2中第一位

```
lrange list01 0 -1
lrange list02 0 -1
rpoplpush list1 list2
```

8.lset key index value

```
lset list01 0 x     将list02中第一位换成x
```

9.linsert key before/after

```
linsert list01b  before x php  在x之前加字段php
```

3.redis-Hash

hash是一个键值对集合

hash是一个string类型的field和value的映射表，hash特别适合存储对象

1.hset/hget/hmset/hmget/hgetall/hdel

```
设值/取值/设值多个值/取多个值/取全部值/删除值
hset user id 11
hget user id
hmset customer id 11 name juran age 26
hmget customer id name age      只返回相应的值
hgetall   customer              返回全部
hdel user id   删除id
```

2.hlen

```
求哈希长度
hlen customer
```

3.hexists key

```
hexists ---- 在key里面的某个值
存在返回1 , 不存在返回0
```

4.hkeys/hvals

```
hkeys students
hvals students
```

4.redis-set(不重复的)

Set(集合)

set是string类型的无序集合

1.sadd/smembers/sismember

```
sadd/smembers/sismember ---- 添加/查看集合/查看是否存在
sadd set01 1 2 2 3 3  去掉重复添加
smembers set01   得到set01
sismember set01 1  如果存在返回1  不存在返回0
```

2.scard

```
scard ---- 获取集合里面的元素个数
scard set01
```

3.srem key value

```
srem ---- 删除集合中元素
srem set01 3
SMEMBERS set01   3已经被删除掉
```

4.srandmember key

```
srandmembe ---- 随机出几个数
sadd set02  1 2 3 4 5 6 7 8
srandmember set02  2
```

5.spop key

```
spop ---- 随机出栈
spop set01
```

6.smove key1 key2

```
sadd set03 x y z
smove set01 set03 2  将set01中的2 移动到set03中
```

7.数学集合类

```
sadd set01 1 2 3 4 5
sadd set02 1 2 3 a b
差集
SDIFF set01 set02   返回 4 5 在第一个set中不在第二个set中
交集
SINTER set01 set02   返回 1 2 3
并集
SUNION set01 set02  返回set01 set02 中的值  去掉重复
```

5.redis-Zset

Zset(有序集合)

1.zadd/zrange

```
zadd zset01 60 v1 70 v2 80 v3 90 v4 100 v5
zrange zset01 0 -1
带分数返回   withscores
```

2.zrangebyscore key start end

```
zrangebyscore key start end----根据开始结束来取值
zrangebyscore zset01 60 70

zrangebyscore zset01 60 (90   表示不包含90

zrangebyscore zset01  60 90  limit  1 2 从第一条开始截取2条
```

3.zrem key

```
zrem key value---- 某score下对应的value值,作用是删除元素
zrem zset01 v1
```

4.zcard/zcount key score 区间/zrank key values

```
zcard   求zset01 总条数
zcount  zset01 60 90  求60-90个数
zrank   zset01  v2   返回1  返回对应下角标，从0开始
```

2-Redis与Python交互

Python操作Redis

redispy安装及连接

安装

```
pip install redis
```

连接

```
r = redis.StrictRedis(host='localhost',port=6379,db=0)
```

字符串相关操作

```
import redis

class TestString(object):
    def __init__(self):
        self.r = redis.StrictRedis(host='192.168.75.130',port=6379)
    设置值
    def test_set(self):
        res = self.r.set('user1','juran-1')
        print(res)
    取值
    def test_get(self):
        res = self.r.get('user1')
        print(res)
    设置多个值
    def test_mset(self):
        d = {
            'user2':'juran-2',
            'user3':'juran-3'
```

```
        }
        res = self.r.mset(d)
    取多个值
    def test_mget(self):
        l = ['user2','user3']
        res = self.r.mget(l)
        print(res)
    删除
    def test_del(self):
        self.r.delete('user2')
```

列表相关操作

```
class TestList(object):
    def __init__(self):
        self.r = redis.StrictRedis(host='192.168.75.130',port=6379)
    插入记录
    def test_push(self):
        res = self.r.lpush('common','1')
        res = self.r.rpush('common','2')
        # res = self.r.rpush('jr','123')
    弹出记录
    def test_pop(self):
        res = self.r.lpop('common')
        res = self.r.rpop('common')
    范围取值
    def test_range(self):
        res = self.r.lrange('common',0,-1)
        print(res)
```

集合相关操作

```
class TestSet(object):
    def __init__(self):
        self.r = redis.StrictRedis(host='192.168.75.130', port=6379)
    添加数据
    def test_sadd(self):
        res = self.r.sadd('set01','1','2')
        lis = ['Cat','Dog']
        res = self.r.sadd('set02',lis)
    删除数据
    def test_del(self):
        res = self.r.srem('set01',1)
    随机删除数据
    def test_pop(self):
        res = self.r.spop('set02')
```

哈希相关操作

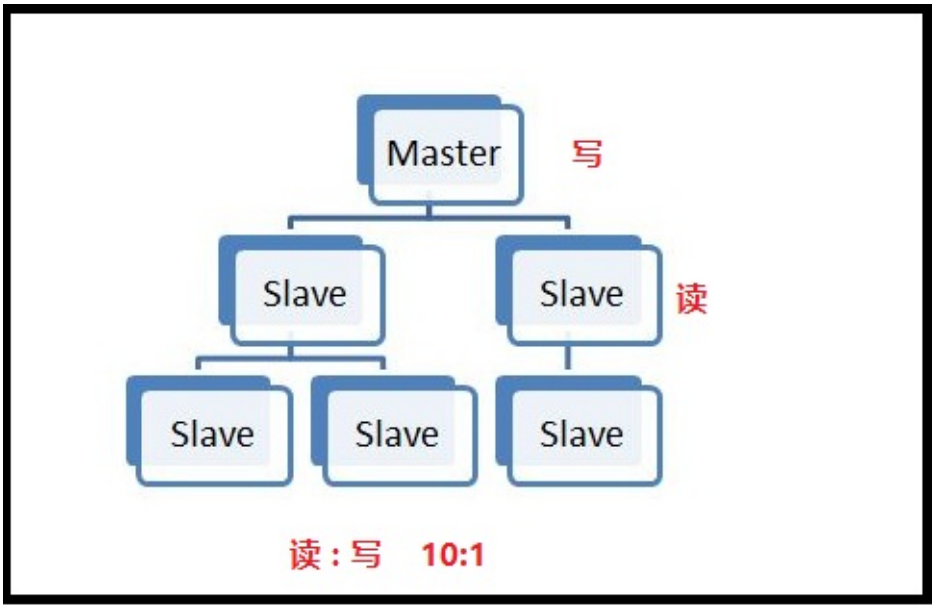
```
class TestHash(object):
    def __init__(self):
        self.r = redis.StrictRedis(host='192.168.75.130', port=6379)

    批量设值
    def test_hset(self):
        dic = {
            'id':1,
            'name':'huawei'
        }
        res = self.r.hmset('mobile',dic)
    批量取值
    def test_hgetall(self):
        res = self.r.hgetall('mobile')
    判断是否存在  存在返回1  不存在返回0
    def test_hexists(self):
        res = self.r.hexists('mobile','id')
        print(res)
```

3-Redis主从配置

主从概念

- 一个master可以拥有多个slave，一个slave又可以拥有多个slave，如此下去，形成了强大的多级服务器集群架构
- master用来写数据，slave用来读数据，经统计：网站的读写比率是10:1
- 通过主从配置可以实现读写分离
- master和slave都是一个redis实例(redis服务)



主从配置

配置主

- 修改etc/redis/redis.conf文件

```
bind 0.0.0.0  或者改成本机IP
```

开启主机服务

```
src/redis-server redis.conf
```

配置从

- 复制etc/redis/redis.conf文件

```
cp redis.conf slave.conf
```

- 修改redis/slave.conf文件

```
vim slave.conf

bind 192.168.154.131(主机IP)
slaveof 192.168.154.131(主机IP) 6379(主机端口)
port 6378(从机端口)
```

开启主机服务

```
src/redis-server slave.conf
```

数据操作

- 在master和slave分别执行info命令，查看输出信息 进入主客户端

```
src/redis-cli -h 192.168.154.131 -p 6379
```

进入从的客户端

```
src/redis-cli -h 192.168.154.131 -p 6378
```

- 在master上写数据

```
set name juran
```

- 在slave上读数据

```
get name
```

Git

4-初始Git

Git实战

什么是Git

Git是一个分布式的版本控制软件。

- 软件,类似于QQ、office等安装到电脑上才能使用的工具
- 版本控制，类似于毕业论文、写文案、视频剪辑等，需要反复修改和保留原历史数据
- 分布式
  - 文件夹拷贝
  - 本地版本控制
  - 集中式版本控制
  - 分布式版本控制

为什么要做版本控制

要保留之前所有的版本，以便回滚和修改。

安装Git

Git地址：https://git-scm.com/book/zh/v2/%E8%B5%B7%E6%AD%A5-%E5%AE%89%E8%A3%85-Git

屌丝创业故事

一个浪迹于北京的屌丝程序猿的终极梦想

第一阶段:自己写代码

版本控制

- 进入要管理的文件夹
  - 右键 Git Bash Here

- 初始化
  - git init
- 管理目录下的文件状态
  - git status
- 管理指定文件
  - git add index.html
  - git add .
- 个人信息配置：用户名、邮箱

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

- 生成版本
  - git status
  - git commit -m '描述信息'
- 查看版本

```
git log
```

第二阶段：拓展新功能

```
git add
git commit -m "短视频"
```

第三阶段：约饭功能

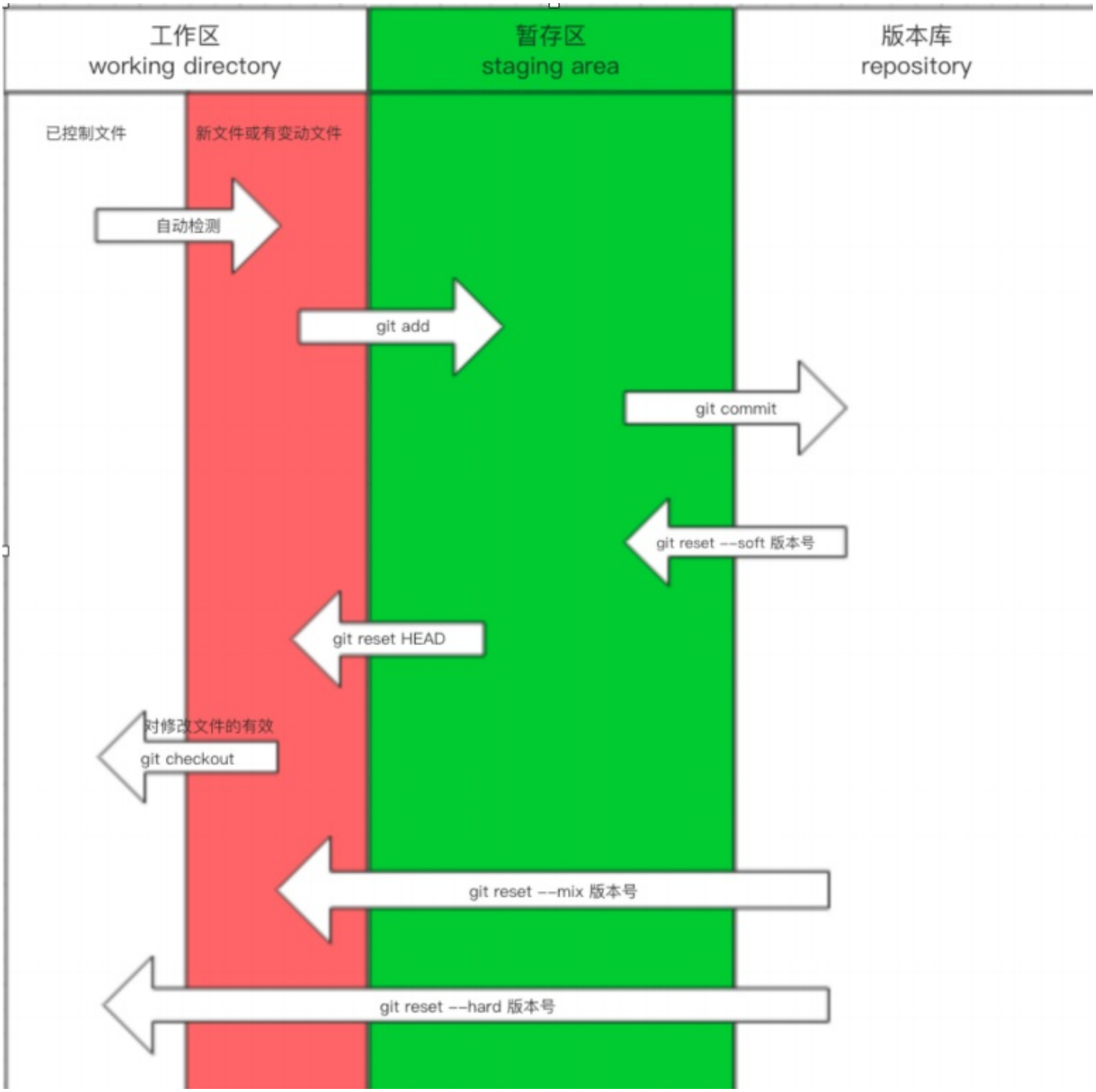
- 回滚至之前版本

```
git log
git reset --hard 版本号
```

- 回滚之后版本

```
git reflog
git reset --hard 版本号
```

总结


工作区
working directory
暂存区
staging area
版本库
repository
已控制文件
新文件或有变动文件
自动检测
git add
git commit
git reset --soft 版本号
git reset HEAD
对修改文件的有效
git checkout
git reset --mix 版本号
git reset --hard 版本号

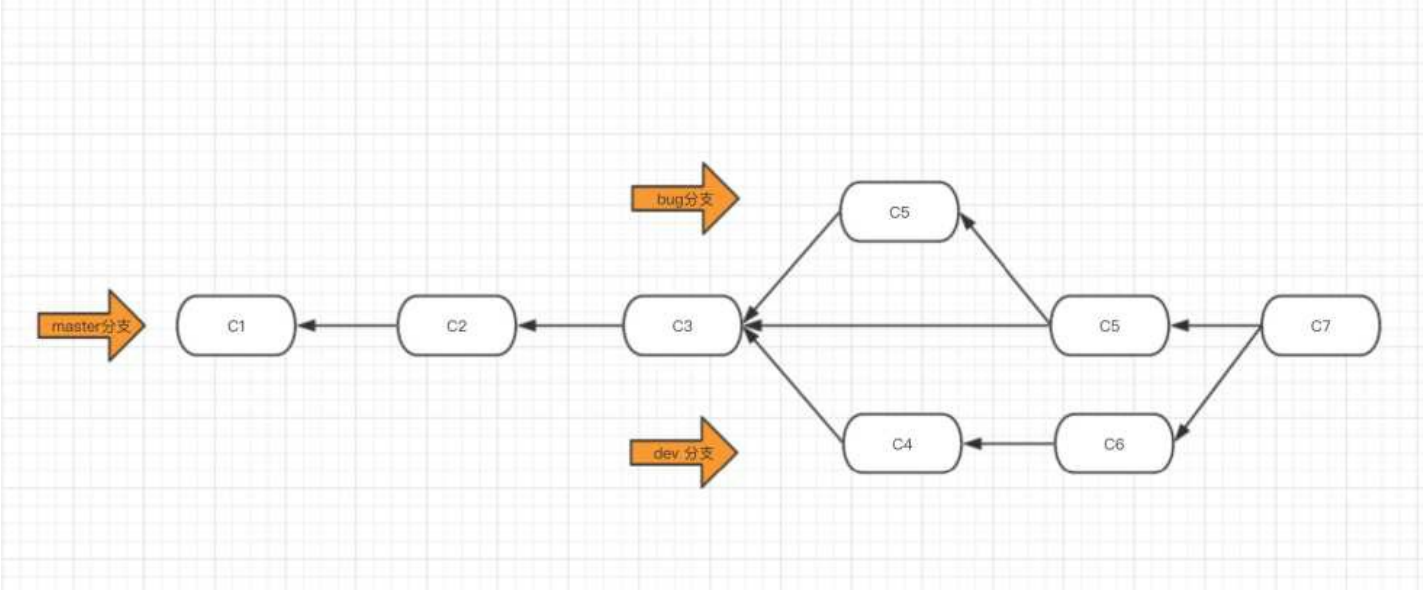
5-Git中的分支

初识分支

分支可以给使用者提供多个环境，意味着你可以把你的工作从开发主线上分离开来，以免影响开发主线。

第四阶段：商城&紧急修复bug

紧急修复线上bug的思路



基于分支修复线上bug

目前你所处在的分支

```
git branch
```

创建分支

```
git branch 分支名字
```

切换分支

```
git checkout 分支名称
```

分支合并（可能产生冲突）

```
git merge 要合并的分支
```

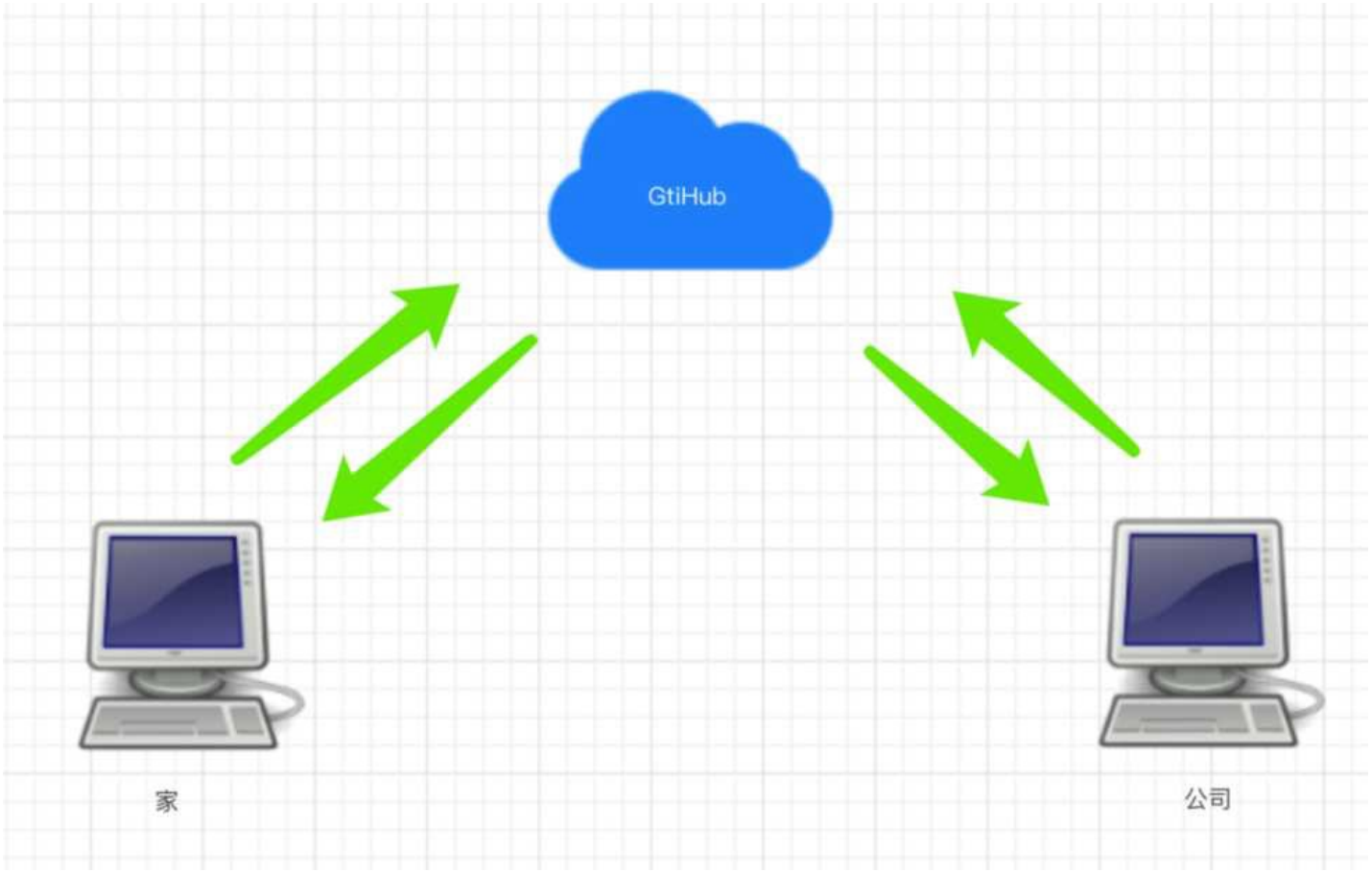
删除分支

```
git branch -d 分支名称
```

6-GitHub

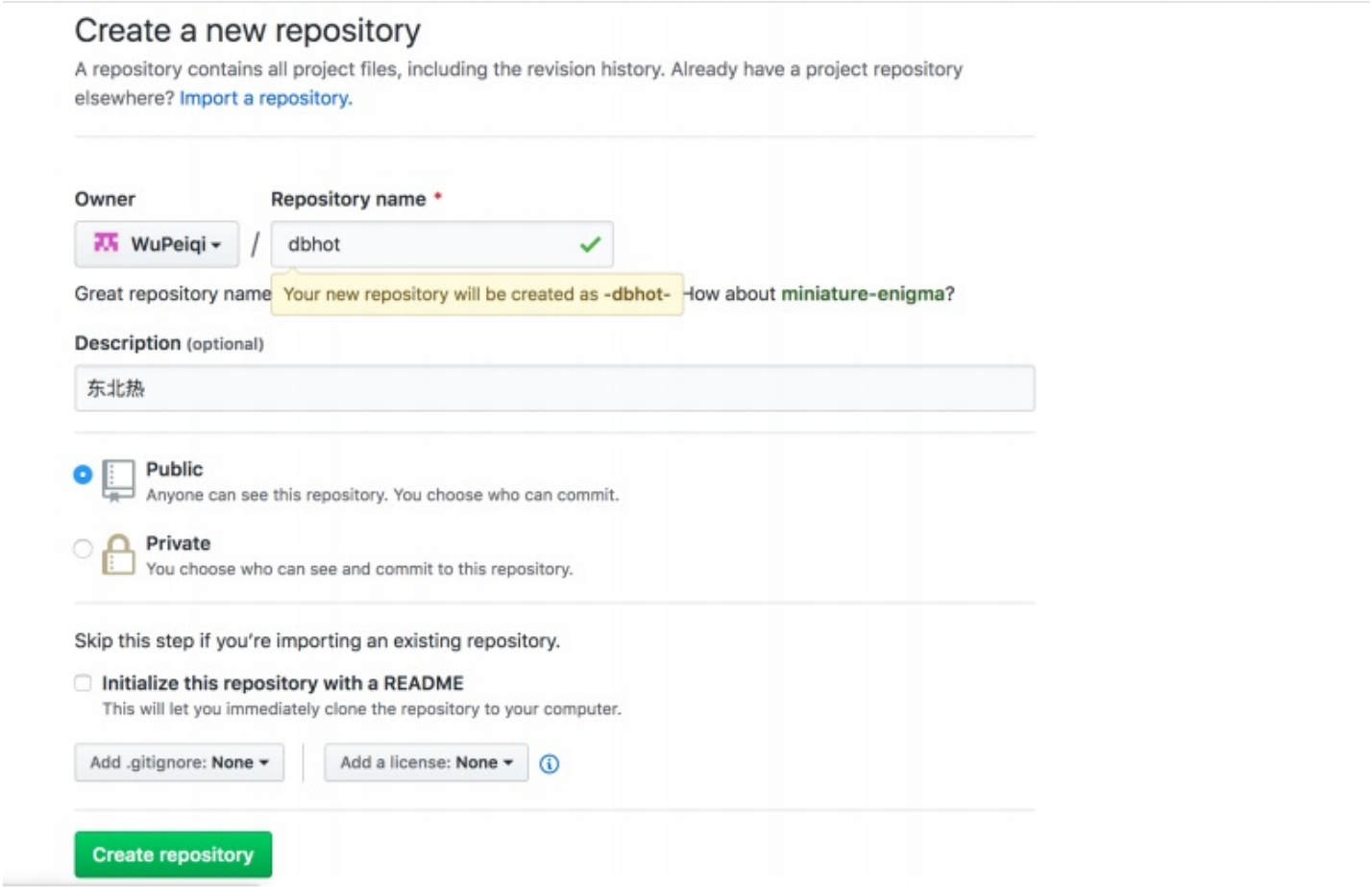
GitHub

第五阶段->进军三里屯



第一天上班前在家上传代码

首先，需要注册github账号，并创建远程仓库，然后再执行如下命令，将代码上传到github。



给远程仓库起别名

```
git remote add origin 远程仓库地址
```

向远程推送代码

```
git push -u origin 分支
```

初次在公司新电脑下载代码

克隆远程仓库代码

```
git clone 远程仓库地址
```

切换分支

```
git checkout 分支
```

在公司下载完代码后，继续开发

切换到dev分支进行开发

```
git checkout dev
```

把master分支合并到dev

```
git merge master
```

提交代码

```
git add .
git commit -m "xxx"
git push origin dev
```

开发完毕，要上线

将dev分支合并到master，进行上线

```
git checkout master
git merge dev
git push origin master
```

把dev分支也推送到远程

```
git checkout dev
git merge master
git push origin dev
```

约妹子忘记提交代码

拉代码

```
git pull origin dev
```

提交代码

```
git add .
git commit -m "xxx"
```

没有提交到GitHub托管

回家继续写代码

拉代码，发现并没有公司的代码

```
git pull origin dev
```

无奈，继续开发其他功能

把dev分支也推送到远程

```
git add .
git commit -m "xxx"
git push origin dev
```

到公司继续写代码

拉代码，把昨天的代码拉到本地(可能存在冲突)

```
git pull origin dev
```

解决冲突，继续开发

把dev分支也推送到远程

```
git add .
git commit -m "xxx"
git push origin dev
```

7-rebase使用

rebase

rebase可以保持提交记录简洁，不分叉。

8-多人协同开发工作流

多人协同开发工作流

创建项目&邀请成员

协同开发时，需要所有成员都可以对同一个项目进行操作，需要邀请成员并赋予权限，否则无法开发。 github支持两种创建项目的方式（供多人协同开发）。

- 1.合作者，将用户添加到仓库合作者中之后，该用户就可以向当前仓库提交代码。
- 2.组织，将成员邀请进入组织，组织下可以创建多个仓库，组织成员可以向组织下仓库提交代码。

成员开发

注册Github或Gitlab账号

- 邀请成员进入组织（默认对组织中的项目具有读权限）
- 邀请成员成为某项目的合作者

code review

- 配置，代码review之后才能合并到dev分支
- 成员提交code review申请
- 组长做 code review

提测上线（预发布）

由专门团队或团队leader执行以下步骤

1.基于dev分值创建release分值

```
git checkout dev

git checkout -b release
```

2.测试等

3.合并到master

```
使用 pull request

本地将release合并到master分支
```

4.在master分支打tag

```
git tag -a v2 -m '第二版 斗地主功能'

git push origin --tags
```

5.运维人员就可以去下载代码做上线了

```
git clone -b v2 地址
```

给开源项目贡献代码

- 1.fork源代码将别人源代码拷贝到我自己的远程仓库。
- 2.在自己仓库进行修改代码
- 3.给源代码的作者提交修复bug的申请(pull request)

9-Git补充

配置文件存放三个位置

- 项目配置文件：项目/.git/config

```
git config --local user.name 'juran'
git config --local user.email 'juran@xx.com
```

- 全局配置文件:~/.gitconfig

```
git config --global user.name 'juran'
git config --global user.name 'juran@xx.com
```

- 系统配置文件：/etc/.gitconfig

```
git config --system user.name 'juran'
git config --system user.name 'juran@xx.com'
注意：需要有root权限
```

免密码登录

- URL中体现

```
原来的地址：https://github.com/WuPeiqi/dbhot.git
修改的地址：https://用户名:密码@github.com/WuPeiqi/dbhot.git
git remote add origin https://用户名：密码@github.com/WuPeiqi/dbhot.git
git push origin master
```

- SSH实现

```
1.生成公钥和私钥（默认放在~/.ssh目录下,id_rsa.pub公钥、id_rsa私钥） ssh-keygen
```

```
2.拷贝公钥的内容,并设置到github中。

3.在git本地中配置ssh地址
git remote add origin git@github.com:WuPeiqi/dbhot.git

4.以后使用
git push origin master
```

git忽略文件

让Git不再管理当前目录下的某些文件。

```
*.h
!a.h
files/
*.py[c|a|d]
```

更多参考：https://github.com/github/gitignore

github任务管理相关

- issues,文档以及任务管理。
- wiki，项目文档。

Vue基本使用

10-Vue介绍

VSCode插件安装

后面开发 `Vue` 项目，使用 `.vue` 的单文件开发，就需要一些插件来帮我们识别 `.vue` 文件。插件安装在 `Extension` 中，点开即可看到一个搜索按钮，可以输入关键字搜索自己想要的插件。
这里我们开发 `Vue` 推荐的几个插件：

1. `jshint` ： `js` 代码规范检查。
2. `Beautify` ：一键美化代码的插件。
3. `Vetur` ： `.vue` 文件识别插件。
4. `Javascript(ES6) code snippets` ： `ES6` 语法提示。
5. `Auto Rename Tag` ：自动重命名标签。标签都是成对出现的，开始标签修改了，结束标签也会跟着修改。
6. `Auto Close Tag` ：自动闭合标签。针对一些非标准的标签，这个插件还是很有用的。
7. `vue helper` ：一些 `Vue` 代码的快捷代码。
8. `vscode-icons` ：可选。提供了很多类型的文件夹 `icon` ，不同类型的文件夹使用不同的 `icon` ，会让文件查找更直观。
9. `open in browser` ：可选。右键可以选择在默认浏览器中打开当前网页。

Vue介绍

Vue(读音/vju:/，类似于view) 是一套用于构建前后端分离的框架。刚开始是由国内优秀选手尤雨溪开发出来的，目前是全球“最”流行的前端框架。使用vue开发网页很简单，并且技术生态环境完善，社区活跃，是前后端找工作必备技能！

Vue安装和使用

vue的安装大体上分成三种方式，第一种是通过script标签引用的，第二种是通过npm(node package manager)来安装，第三种是通过vue-cli命令行来安装。作为初学者，建议直接通过第一种方式来安装，把心思集中在vue

的学习上，而不是安装上。

```
# 开发环境
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
# 或者是指定版本号
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.js"></script>
# 或者是下载代码保存到本地
<script src="lib/vue.js"></script>


# 生产环境，使用压缩后的文件
<script src="https://cdn.jsdelivr.net/npm/vue@2.6.10/dist/vue.min.js"></script>
```

基本使用

要使用Vue，首先需要创建一个Vue对象，并且在这个对象中传递el参数，el参数全称是element，用来找到一个给vue渲染的根元素。并且我们可以传递一个data参数，data中的所有值都可以直接在根元素下使用{{}}来使用。示例代码如下：

```
<div id="app">
    <p>{{username}}</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "username": "逻辑教育"
        }
    });
</script>
```

其中data中的数据，只能在vue的根元素下使用，在其他地方是不能被vue识别和渲染的

```
<div id="app">
</div>
<!-- 这里渲染不了 -->
<p>{{username}}</p>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "username": "逻辑教育"
        }
    });
</script>
```

另外也可以在vue对象中添加methods，这个属性中专门用来存储自己的函数。methods中的函数也可以在模板中使用，并且在methods中的函数来访问data中的值，只需要通过this.属性名就可以访问到了，不需要额外加this.data.属性名来访问

```
<div id="app">
    <p>{{greet()}}</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "username": "逻辑教育"
        },
        methods: {
            greet: function(){
                return "晚上好！" + this.username
            }
        }
    });
</script>
```

11-Vue模板语法

文本

在html中通过{{}}（双大括号）中可以把Vue对象中的数据插入到网页中。并且只要Vue对象上对应的值发生改变了，那么html中双大括号中的值也会立马改变。

```
<div id="app">
    <p>{{username}}</p>
    <button v-on:click="change">点击修改</button>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "username": "逻辑教育"
        },
        methods: {
            change: function(){
                this.username = 'China';
            }
        }
    });
</script>
```

当然，如果在html的{{}}中，第一次渲染完成后，不想要跟着后期数据的更改而更改，那么可以使用v-once指令来实现。

```
<p v-once>{{username}}</p>
```

显示原生的HTML

有时候我们的Vue对象中，或者是后台返回给我们一个段原生的html代码，我们需要渲染出来，那么如果直接通

过{{}}渲染，会把这个html代码当做字符串。这时候我们就可以通过v-html指令来实现

```
<div id="app">
    <div v-html="code"></div>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "code": "<a href='https://www.baidu.com/'>百度一下，你就知道！</a>"
        }
    });
</script>
```

属性绑定

如果我们想要在html元素的属性上绑定我们Vue对象中的变量，那么需要通过v-bind来实现

```
<div id="app">
    <img class="{{classname}}">你好，世界</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "classname": "pclass"
        },
    });
</script>
```

需要使用v-bind才能生效

```
<div id="app">
    <img v-bind:src="imgSrc" alt="">
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            "imgSrc": "https://i.ytimg.com/vi/5HKZ6bU6Zg0/maxresdefault.jpg"
        }
    });
</script>
```

属性绑定Class和Style

在绑定class或者style的时候，可以通过以下方式来实现。

绑定Class

通过数组的方式来实现

```
<div id="app">
    <p v-bind:class="[classname1,classname2]">你好，世界</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            classname1: "pcolor",
            classname2: "pfont"
        }
    });
</script>
```

通过对象的方式来实现

```
<div id="app">
    <p v-bind:class="{pcolor:isColor,pfont:isFont}">你好，世界</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            isColor: true,
            isFont: true
        }
    });
</script>
```

绑定Style

用对象的方式

```
# 读取对应样式的值
<li :style="{backgroundColor:pBgColor,fontSize:pFontSize}">首页</li>
# 或者是直接读取整个字符串
<li :style="liStyle">首页</li>
```

但是样式如果有横线，则都需要变成驼峰命名的方式

用数组的方式

```
<li :style="[liStyle1,liStyle2]">首页</li>
<script>
    new Vue({
        el: "#app",
        data: {
            liStyle: {
            backgroundColor: "red",
            fontSize: "14px"
        },
        liStyle2: {
            fontWeight: 800
        }
        }
    })
</script>
```

使用JavaScript表达式

在使用了v-bind的html属性，或者使用了{{}}的文本。我们还可以执行一个JavaScript表达式

```
<div id="app">
  <p v-bind:style="{color:color?'red':'blue'}">{{username.split(" ").reverse().
join(" ")}}</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            username: "luoji ketang",
            color: false
        }
    });
</script>
```

注意，只能是JavaScript表达式，不能是语句，比如var a=1;a=2;这样的是js语句，不是表达式了

条件判断

在模板中，可以根据条件进行渲染。条件用到的是v-if、v-else-if以及v-else来组合实现的

```
<div id="app">
    <p v-if="weather == 'sun'">今天去公园玩！</p>
    <p v-else-if="weather == 'rain'">今天去看电影！</p>
    <p v-else>今天哪儿也不去！</p>
</div>
<script>
    let vm = new Vue({
        el: "#app",
```

```
        data: {
            weather: 'sun'
        }
    });
</script>
```

有时候我们想要在一个条件中加载多个html元素，那么我们可以通过template元素上实现

```
<div id="app">
    <template v-if="age<18">
        <p>数学多少分？</p>
        <p>英语多少分？</p>
    </template>
    <template v-else-if="age>=18 && age<25">
        <p>女朋友找到了吗？</p>
        <p>考研究生了吗？</p>
    </template>
    <template v-else>
        <p>二胎生了吗？</p>
        <p>工资多少？</p>
    </template>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            age: 24
        }
    });
</script>
```

另外，在模板中，Vue会尽量重用已有的元素，而不是重新渲染，这样可以变得更加高效。如果你允许用户在不同的登录方式之间切换

```
<div id="app">
    <template v-if="loginType=='username'">
        <label for="username">用户名：</label>
        <input type="text" id="username" name="username" placeholder="用户名">
    </template>
    <template v-else-if="loginType=='email'">
        <label for="email">邮箱：</label>
        <input type="text" id="email" name="email" placeholder="邮箱">
    </template>
    <div>
        <button v-on:click="changeLoginType">切换登录类型</button>
    </div>
</div>
```

```
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            loginType: "username"
        },
        methods: {
            changeLoginType: function(event){
                this.loginType = this.loginType=="username"?"email":"username";
            }
        }
    });
</script>
```

这个里面会有一个问题，就是如果我在username的输入框中输入完信息，切换到邮箱中，之前的信息还是保留下来，这样肯定不符合需求的，如果我们想要让html元素每次切换的时候都重新渲染一遍，可以在需要重新渲染的元素上加上唯一的key属性，其中key属性推荐使用整形，字符串类型。

```
<div id="app">
    <template v-if="loginType=='username'">
        <label for="username">用户名：</label>
        <input type="text" id="username" name="username" placeholder="用户名" key
="username">
    </template>
    <template v-else-if="loginType=='email'">
        <label for="email">邮箱：</label>
        <input type="text" id="email" name="email" placeholder="邮箱" key="emai
l">
    </template>
    <div>
        <button v-on:click="changeLoginType">切换登录类型</button>
    </div>
</div>
```

注意，元素仍然会被高效地复用，因为它们没有添加key属性

v-show和v-if

v-if是“真正”的条件渲染，因为它会确保在切换过程中条件块内的事件监听器和子组件适当地被销毁和重建。v-if也是惰性的：如果在初始渲染时条件为假，则什么也不做——直到条件第一次变为真时，才会开始渲染条件块。 相比之下，v-show就简单得多——不管初始条件是什么，元素总是会被渲染，并且只是简单地基于CSS进行切换。 一般来说， v-if有更高的切换开销，而v-show有更高的初始渲染开销。因此，如果需要非常频繁地切换，则使用v-show较好；如果在运行时条件很少改变，则使用v-if较好。

循环

在模板中可以用 `v-for` 指令来循环数组，对象等

循环数组

```
<div id="app">
    <table>
            <tr>
                <th>序号</th>
                <th>标题</th>
                <th>作者</th>
            </tr>
            <tr v-for="(book,index) in books">
                <td>{{index}}</td>
                <td>{{book.title}}</td>
                <td>{{book.author}}</td>
            </tr>
    </table>
</div>
```

循环对象

循环对象跟循环数组是一样的。并且都可以在循环的时候使用接收多个参数

```
<div id="app">
    <div v-for="(value,key) in person">
        {{key}}:{{value}}
    </div>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            person: {
                "username": "逻辑教育",
                "age": 18,
                "homepage": "https://www.baidu.com/"
            }
        }
    });
</script>
```

保持状态

循环出来的元素，如果没有使用 `key` 元素来唯一标识，如果后期的数据发生了更改，默认是会重用的，并且元素的顺序不会跟着数据的顺序更改而更改

```
<div id="app">
    <div v-for="(book,index) in books">
        <label for="book">{{book}}</label>
        <input type="text" v-bind:placeholder="book">
    </div>
    <button v-on:click="changeBooks">更换图书</button>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            books: ['三国演义','水浒传','红楼梦','西游记']
        },
        methods: {
            changeBooks: function(event){
                this.books.sort((x,y) => {
                    return 5 - parseInt(Math.random()*10)
                });
            }
        }
    });
</script>
```

触发视图更新

`Vue` 对一些方法进行了包装和变异，以后数组通过这些方法进行数组更新，会出发视图的更新。

1. `push()` ：添加元素的方法。

```
this.books.push("居然")
```

2. `pop()` ：删除数组最后一个元素。

```
this.books.pop()
```

3. `shift()` ：删除数组的第一个元素。

```
 this.books.shift()
```

4. `unshift(item)` ：在数组的开头位置添加一个元素。

```
this.books.unshift("居然")
```

5. `splice(index,howmany,item1,...,itemX)` ：向数组中添加或者删除或者替换元素。

```
// 向books第0个位置添加元素
this.books.splice(0,0,"金瓶梅")
// 下标从0开始，删除2个元素
this.books.splice(0,2)
// 下标从0开始，替换2个元素
this.books.splice(0,2,'金瓶梅','骆驼祥子')
```

6. `sort(function)` ：排序。

```
this.books.sort(function(x,y){
    // 取两个随机数排序
    a = Math.random();
    b = Math.random();
    return a-b;
});
```

7. `reverse()` ：将数组元素进行反转。

```
this.books.reverse();
```

视图更新注意事项

1. 直接修改数组中的某个值是不会出发视图更新的。比如：

```
this.books[0] = 'Python';
```

这种情况应该改成用 `splice` 或者是用 `Vue.set` 方法来实现：

```
Vue.set(this.books,0,'Python');
```

2. 如果动态的给对象添加属性，也不会触发视图更新。只能通过 `Vue.set` 来添加。比如：

```
<div id="app">
    <ul>
        <li v-for="(value,name) in person">{{name}}:{{value}}</li>
    </ul>
    <script>
        let vm = new Vue({
            el: "#app",
            data: {
                person: {"username": '逻辑教育'}
```

```
            },
            methods: {
                changePerson: function(event){
                    Vue.set(this.person,'age',18)
                }
            }
        });
    </script>
</div>
```

事件绑定

事件绑定就是在 `HTML` 元素中，通过 `v-on` 绑定事件的。事件代码可以直接放到 `v-on` 后面，也可以写成一个函数。

```
<div id="app">
    <p>{{count}}</p>
    <button v-on:click="count+=1">加</button>
    <button v-on:click="subtract(10)">减10</button>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            count: 0
        },
        methods: {
            subtract: function(value){
                this.count -= value;
            }
        }
    });
</script>
```

传入 `event` 参数

如果在事件处理函数中，想要获取原生的 `DOM` 事件，那么在 `html` 代码中，调用的时候，可以传递一个 `$event` 参数。

```
<button v-on:click="subtract(10,$event)">减10</button>
...
<script>
...
methods: {
    subtract: function(value,event){
        this.count -= value;
```

```
        console.log(event);
    }
}
...
</script>
```

12-计算属性和监听器

计算属性和监听器

一般情况下属性都是放到 `data` 中的，但是有些属性可能是需要经过一些逻辑计算后才能得出来，那么我们可以把这类属性变成计算属性。比如以下：

```
<div id="app">
    <label for="length">长：</label>
    <input type="number" name="length" v-model:value="length">
    <label for="width">宽：</label>
    <input type="number" name="width" v-model:value="width">
    <label for="area">面积：</label>
    <input type="number" name="area" v-bind:value="area" readonly>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            length: 0,
            width: 0,
        },
        computed: {
            area: function(){
                return this.length*this.width;
            }
        }
    });
</script>
```

可能有的小伙伴会觉得这个计算属性跟我们之前学过的函数好像有点重复。实际上，计算属性更加智能，他是基于它们的响应式依赖进行缓存的。也就是说只要相关依赖（比如以上例子中的 `area` ）没有发生改变，那么这个计算属性的函数不会重新执行，而是直接返回之前的值。这个缓存功能让计算属性访问更加高效。

计算属性的 `set` ：

计算属性默认只有 `get` ，不过在需要时你也可以提供一个 `set` ，但是提供了 `set` 就必须要提供 `get` 方法。示例代码如下：

```
<div id="app">
    <div>
        <label>省：</label>
        <input type="text" name="province" v-model:value="province">
    </div>
    <div>
        <label>市：</label>
        <input type="text" name="city" v-model:value="city">
    </div>
    <div>
        <label>区：</label>
        <input type="text" name="district" v-model:value="district">
    </div>
    <div>
        <label>详细地址：</label>
        <input type="text" name="address" v-model:value="address">
    </div>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            district: "",
            city: "",
            province: ""
        },
        computed: {
            address: {
                get: function(){
                    let result = "";
                    if(this.province){
                        result = this.province + "省";
                    }
                    if(this.city){
                        result += this.city + "市";
                    }
                    if(this.district){
                        result += this.district + "区";
                    }
                    return result;
                },
                set: function(newValue){
                    let result = newValue.split(/省|市|区/)
```

```
                    if(result && result.length > 0){
                        this.province = result[0];
                    }
                    if (result && result.length > 1){
                        this.city = result[1];
                    }
                    if(result && result.length > 2){
                        this.district = result[2];
                    }
                }
            }
        }
    });
</script>
```

监听属性：

监听属性可以针对某个属性进行监听，只要这个属性的值发生改变了，那么就会执行相应的函数。示例代码如下：

```
<div id="app">
    <div>
        <label>搜索：</label>
        <input type="text" name="keyword" v-model:value="keyword">
    </div>
    <div>
        <p>结果：</p>
        <p>{{answer}}</p>
    </div>
</div>
<script>
    let vm = new Vue({
        el: "#app",
        data: {
            keyword: "",
            answer: ""
        },
        watch: {
            keyword: function(newKeyword,oldKeyword){
                this.answer = '加载中...';
                let that = this;
                setTimeout(function(){
                    that.answer = that.keyword;
                },Math.random()*5*1000);
            }
        }
    });
```

```
</script>
```

13-表单输入绑定

表单输入绑定

v-model指定可以实现表单值与属性的双向绑定。即表单元素中更改了值会自动的更新属性中的值，属性中的值更新了会自动更新表单中的值。

绑定的属性和事件：

v-model在内部为不同的输入元素使用不同的属性并抛出不同的事件：

1. text和textarea元素使用value属性和input事件。
2. checkbox和radio使用checked属性和change事件。
3. select字段将value作为prop并将change作为事件。

表单元素绑定

input绑定

```
v-model是v-model:value的缩写,改变 input标签中的值 可以改变下面的属性
<input v-model="message" placeholder="请输入...">
<input v-model:value="message" placeholder="请输入...">
<p>输入的内容是：{{ message }}</p>


new Vue({
  el: '#example-3',
  data: {
    message: ""
  }
})
```

textarea绑定

```
<span>输入的内容是：</span>
```

```
<p style="white-space: pre-line;">{{ message }}</p>
<br>
<textarea v-model="message" placeholder="请输入多行内容..."></textarea>
```

checkbox绑定

```
<div id='example-3'>
  <input type="checkbox"  value="Jack" v-model="checkedNames">
  <label for="jack">Jack</label>
  <input type="checkbox"  value="John" v-model="checkedNames">
  <label for="john">John</label>
  <input type="checkbox"  value="Mike" v-model="checkedNames">
  <label for="mike">Mike</label>
  <br>
  <span>Checked names: {{ checkedNames }}</span>
</div>
new Vue({
  el: '#example-3',
  data: {
    checkedNames: []
  }
})
```

radio绑定

```
<div id="example-4">
    <input type="radio" value="男" v-model="gender">
    <label>男</label>
    <br>
    <input type="radio" value="女" v-model="gender">
    <label>女</label>
    <br>
    <span>Picked: {{ gender }}</span>
</div>
new Vue({
  el: '#example-4',
  data: {
    gender: ''
  }
})
```

select绑定

```
<div id="example-5">
    <select v-model="selected">
        <option disabled value="">请选择</option>
```

```
        # 如果有value值 选择的就是value值
        <option value="1">A</option>
        <option>B</option>
        <option>C</option>
    </select>
    <span>Selected: {{ selected }}</span>
</div>
new Vue({
  el: '...',
  data: {
    selected: ''
  }
})
```

修饰符

.lazy

在默认情况下，v-model在每次input事件触发后将输入框的值与数据进行同步 (除了上述输入法组合文字时)。你可以添加lazy修饰符，从而转变为使用change事件进行同步：

```
<!-- 在"change"时而非"input"时更新 光标移除input输入框的时候 -->
<input type="text" v-model:value.lazy="message">
<input v-model.lazy="message" >

new Vue({
  el: '#app',
  data: {
    message: ''
  }
})
```

.number

如果想自动将用户的输入值转为数值类型，可以给v-model添加number修饰符

```
<input v-model.number="age" type="number">
```

这通常很有用，因为即使在type="number"时，HTML输入元素的值也总会返回字符串。如果这个值无法被parseFloat()解析，则会返回原始的值。

.trim

如果要自动过滤用户输入的首尾空白字符，可以给 v-model 添加 trim 修饰符

```
<input v-model.trim="msg">
```

14-自定义组件

自定义组件

有时候有一组html结构的代码，并且这个上面可能还绑定了事件。然后这段代码可能有多个地方都被使用到了，如果都是拷贝来拷贝去，很多代码都是重复的，包括事件部分的代码都是重复的。那么这时候我们就可以把这些代码封装成一个组件，以后在使用的时候就跟使用普通的html元素一样，拿过来用就可以了。

基本使用

```
<div id="app">
    <button-counter></button-counter>
    <button-counter></button-counter>
    <button-counter></button-counter>
</div>
<script>
    Vue.component('button-counter', {
        template: '<button v-on:click="count+=1">点击了{{ count }}次</button>'
        data: function(){
            return {
                count: 0
            }
        },
    });
    let vm = new Vue({
        el: "#app",
        data: {}
    });
</script>
```

以上我们创建了一个叫做button-counter的组件，这个组件实现了能够记录点击了多少次按钮的功能。后期如果我们想要使用，就直接通过button-counter使用就可以了。然后因为组件是可复用的Vue实例，所以它们与new Vue接收相同的选项，例如data、computed、watch、methods以及生命周期钩子等。仅有的例外是像el这样根实例特有的选项。另外需要注意的是：组件中的data必须为一个函数！

给组件添加属性

像原始的html元素都有自己的一些属性，而我们自己创建的组件，也可以通过prop来添加自己的属性。这样别人在使用你创建的组件的时候就可以传递不同的参数了

```
<div id="app">
    <blog-post v-for="blog in blogs" :title="blog.title"></blog-post>
    <blog-post v-bind:blogs="blogs"></blog-post>
</div>
<script>
    Vue.component('blog-post', {
        props: ['blogs'],
        template: `
        <table>
            <tr>
                <th>序号</th>
                <th>标题</th>
            </tr>
            <tr v-for="(blog,index) in blogs">
                <td>{{index+1}}</td>
                <td>{{blog.title}}</td>
            </tr>
        </table>
        `
    })
    new Vue({
        el: "#app",
        data: {
            blogs: [
                {"title":"钢铁是怎样练成的？","id":1},
                {"title":"AI会毁灭人类吗？","id":2},
                {"title":"如何学好Vue！","id":3},
            ]
        }
    });
</script>
```

单一根元素

如果自定义的组件中，会出现很多html元素，那么根元素必须只能有一个，其余的元素必须包含在这个根元素中。比如以下是一个组件中的代码，会报错：

```
<h3>{{ title }}</h3>
<div v-html="content"></div>
```

我们应该改成：

```
<div class="blog-post">
  <h3>{{ title }}</h3>
  <div v-html="content"></div>
</div>
```

子组件事件和传递事件到父组件

子组件中添加事件跟之前的方式是一样的，然后如果发生某个事件后想要通知父组件，那么可以使用this.$emit函数来实现。

```
<div id="app">
    <blog-item v-for="blog in blogs" v-bind:blog="blog" @check-changed="checks">
</blog-item>

    <div v-for="blog in componentblog">
        {{blog.title}}
    </div>
</div>
<script>
    Vue.component('blog-item',{
        props:['blog'],
        template:`
        <div>
            <span>{{blog.title}}</span>
            <input type="checkbox" @click="onCheck">
        </div>
        `,
        methods:{
            onCheck:function(){
                // console.log(123)
                this.$emit('check-changed',this.blog)
            }
        }
    })

    new Vue({
        el: '#app',
        data: {
            blogs:[
                {"title":"钢铁是怎样练成的？","id":1},
                {"title":"AI会毁灭人类吗？","id":2},
                {"title":"如何学好Vue！","id":3},
            ],
            componentblog:[]
        },

        methods:{
            checks:function(blog){
```

```
                // indexOf 判断某个元素在数组中的位置,返回下标
                var index = this.componentblog.indexOf(blog)
                if(index >= 0){
                    this.componentblog.splice(index,1)
                }else{
                    this.componentblog.push(blog)
                }
                console.log(blog)
            }
        }
    })
</script>
```

需要注意的是，因为html中大小写是不敏感的，所以在定义子组件传给父组件事件名称的时候，不要使用myEvent这种驼峰命名法，而是使用my-event这种规则。

自定义组件v-model

一个组件上的v-model默认会利用名为value的prop(属性)和名为input的事件，但是像单选框、复选框等类型的输入控件可能会将value特性用于不同的目的。这时候我们可以在定义组件的时候，通过设置model选项可以用来实现不同的处理方式

```
<div id="app">
    <stepper v-model:value="goods_count"></stepper>
</div>

<script>
     Vue.component('stepper',{
        props:['count'],
        model:{
            event: 'count-changed',
            prop: "count"
        },
        template:`
        <div>
            <button @click="sub">-</button>
            <span>{{count}}</span>
            <button @click="add">+</button>
        </div>
        `,
        methods:{
            sub:function(){
                this.$emit("count-changed", this.count-1)
            },
            add:function(){
                this.$emit("count-changed", this.count+1)
            }
```

```
        }
    });

    new Vue({
        el: "#app",
        data:{
            "goods_count":0
        }
    })
</script>
```

其中的props定义的属性分别是给外面调用组件的时候使用的。model中定义的prop:'count'是告诉后面使用v-model的时候，要修改哪个属性；event:'count-changed'是告诉v-model，后面触发哪个事件的时候要修改属性。

插槽

我们定义完一个组件后，可能在使用的时候还需要往这个组件中插入新的元素或者文本。这时候就可以使用插槽来实现。

```
<div id="app">
    <navigation-link url="/profile/">
        个人中心
    </navigation-link>
</div>
<script>
    Vue.component('navigation-link', {
        props: ['url'],
        template: `
        <a v-bind:href="url" class="nav-link">
            <slot></slot>
        </a>
        `
    })
    new Vue({
        el: "#app"
    });
</script>
```

当组件渲染的时候，将会被替换为“个人中心”。插槽内可以包含任何模板代码，包括HTML：

```
<navigation-link url="/profile">
  <!-- 添加一个 Font Awesome 图标 -->
  <span class="fa fa-user"></span>
  个人中心
</navigation-link>
```

如果没有包含一个元素，则该组件起始标签和结束标签之间的任何内容都会被抛弃。

15-生命周期函数&过滤器

生命周期函数

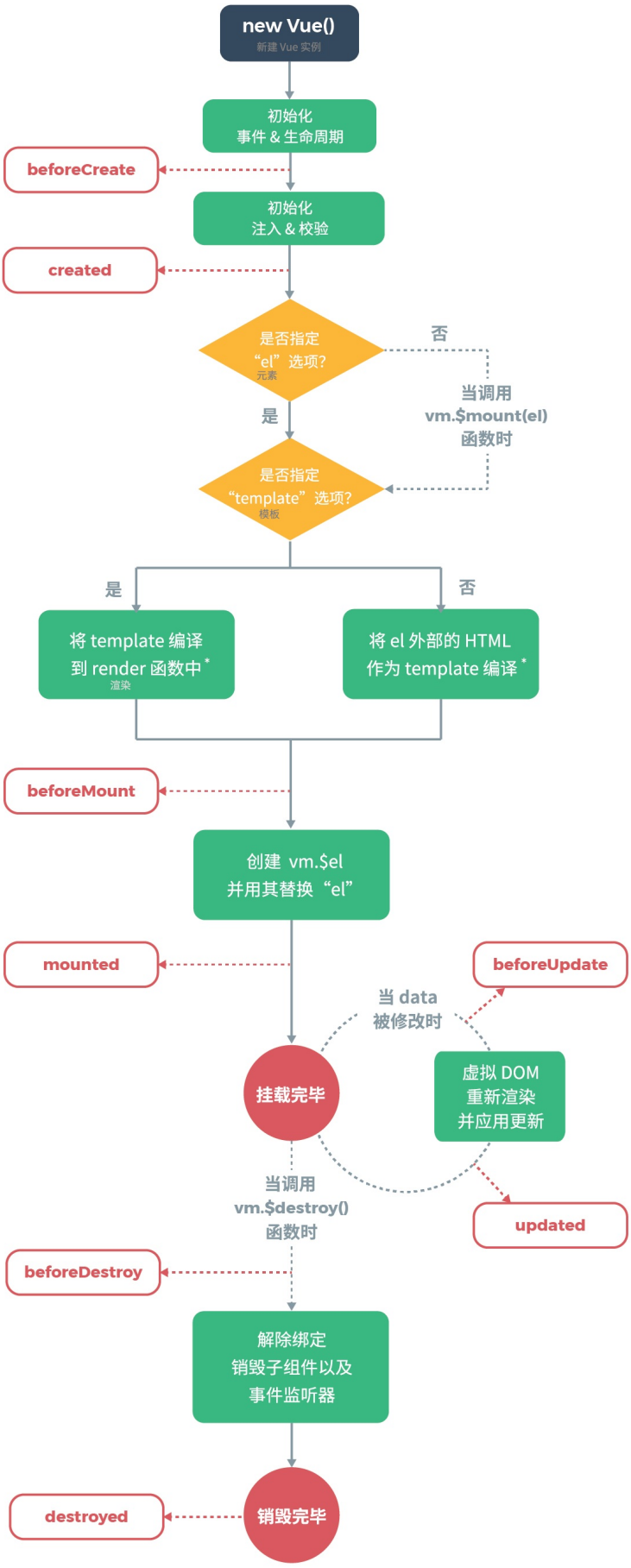
生命周期函数代表的是 `Vue` 实例，或者是 `Vue` 组件，在网页中各个生命阶段所执行的函数。生命周期函数可以分为创建阶段和运行期间以及销毁期间。

其中创建期间的函数有 `beforeCreate` 、 `created` 、 `beforeMount` 、 `mounted` ；

运行期间的函数有 `beforeUpdate` 、 `updated` ；

销毁期间有 `beforeDestroy` 、 `destroyed` 。

以下是官方文档给到的一张图，从这种图中我们可以了解到每个部分执行的函数。



* 如果使用构造生成文件（例如构造单文件组件），
模板编译将提前执行

创建期间：

beforeCreate：

`Vue` 或者组件刚刚实例化， `data` 、 `methods` 都还没有被创建。

created：

此时 `data` 和 `methods` 已经被创建，可以使用了。模板还没有被编译。

beforeMount：

`created` 的下一阶段。此时模板已经被编译了，但是并没有被挂在到网页中。

mounted：

模板代码已经被加载到网页中了。此时创建期间所有事情都已经准备好了，网页开始运行了。

运行期间：

beforeUpdate：

在网页网页运行期间， `data` 中的数据可能会进行更新。在这个阶段，数据只是在 `data` 中更新了，但是并没有在模板中进行更新，因此网页中显示的还是之前的。

updated：

数据在 `data` 中更新了，也在网页中更新了。

销毁期间：

beforeDestroy：

`Vue` 实例或者是组件在被销毁之前执行的函数。在这一个函数中 `Vue` 或者组件中所有的属性都是可以使用的。

destroyed：

`Vue` 实例或者是组件被销毁后执行的。此时 `Vue` 实例上所有东西都会解绑，所有事件都会被移除，所有子元素都会被销毁。

过滤器

过滤器就是数据在真正渲染到页面中的时候，可以使用这个过滤器进行一些处理，把最终处理的结果渲染到网页中。

过滤器使用：

过滤器可以用在两个地方：双花括号插值**和 `v-bind` 表达式 (后者从2.1.0+开始支持)。过滤器应该被添加在 `JavaScript` 表达式的尾部，由“管道”符号指示：

```
<!-- 在双花括号中 -->
{{ message|capitalize }}
<!-- 在 `v-bind` 中 -->
<div v-bind:id="rawId|formatId"></div>
```

过滤器定义：

你可以在一个组件的选项中定义本地的过滤器：

```
filters: {
  capitalize: function (value) {
    if (!value) return ''
    value = value.toString()
    return value.charAt(0).toUpperCase() + value.slice(1)
  }
}
```

或者在创建 Vue 实例之前全局定义过滤器：

```
Vue.filter('capitalize', function (value) {
  if (!value) return ''
  value = value.toString()
  return value.charAt(0).toUpperCase() + value.slice(1)
})

new Vue({
  // ...
})
```

Vue-Router

Vue-Router

`Vue-Router` 是用来将一个 `Vue` 程序的多个页面进行路由的。比如一个 `Vue` 程序（或者说一个网站）有 `登录` 、 `注册` 、 `首页` 等模块，那么我们就可以定义 `/login` 、 `/register` 、 `/` 来映射每个模块。

安装：

- 通过 `script` 加载进来：
  `<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>` 。
- 通过 `npm` 安装： `npm install vue-router` 。

16-vue-router基础

路由基本

在网页中，经常需要发生页面更新或者跳转。这时候我们就可以使用 `Vue-Router` 来帮我们实现。

`Vue-Router` 是用来做路由的，也就是定义 `url规则` 与具体的 `View` 映射的关系。可以在一个单页面中实现数据的更新。

安装：

1. 使用 `CDN` ：
    - 加载最新版的：
      `<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>` 。
    - 加载指定版本的：
      `<script src="https://unpkg.com/vue-router@3.0.7/dist/vue-router.js"></scrip`
      。
2. 下载到本地： `<script src="../../lib/vue-router.js"></script>` 。
3. 使用 `npm` 安装： `npm install vue-router` 。

基本使用：

```
<div id="app">
    <div class="container-fluid">
        <div class="row">
            <div class="col-md-8 col-md-offset-2">
                <ul class="nav nav-tabs">
                    <li role="presentation" class="active"><router-link to="/fi
nd">发现音乐</router-link></li>
                    <li role="presentation"><router-link to="/my">我的音乐</rout
er-link></li>
                    <li role="presentation"><router-link to="/friend">朋友</rou
ter-link></li>
```

```
                </ul>
                <!-- 路由出口 -->
                <!-- 路由匹配到的组件将渲染在这里 -->
                <router-view></router-view>
            </div>
        </div>
    </div>
</div>

<script>
    var find = Vue.extend({template: "<h1>发现音乐</h1>"});
    var my = Vue.extend({template: "<h1>我的音乐</h1>"});
    var friend = Vue.extend({template:"<h1>朋友</h1>"});
    var routes = [
        {path: "/find",component: find},
        {path: "/my",component: my},
        {path: "/friend",component: friend},
        {path: "/",component:find}
    ]
    const router = new VueRouter({routes});
    new Vue({router}).$mount("#app");
</script>
```

解释：

1. 在 `vue-router` 中，使用 `<router-link>` 来加载链接，然后使用 `to` 表示跳转的链接。`vue-router` 最终会把 `<router-link>` 渲染成 `<a>` 标签。
2. `<router-view>` 是路由的出口，也就是相应 `url` 下的代码会被渲染到这个地方来。
3. `Vue.extend` 是用来加载模板的。
4. `routes` 是定义一个 `url` 与组件的映射，这个就是路由。
5. `VueRouter` 创建一个路由对象。

动态路由：

在路由中有一些参数是会变化的，比如查看某个用户的个人中心，那肯定需要在 `url` 中加载这个人的 `id` ，这时候就需要用到动态路由了。

```
<div id="app">
    <router-link to="/user/123">个人中心</router-link>
    <router-view></router-view>
</div>

<script>
    let UserProfile = {template:"<h1>个人中心：{{$route.params.userid}}</h1>"}
    var routes = [
```

```
        {path: "/user/:userid",component: UserProfile}
    ]
    const router = new VueRouter({routes});
    new Vue({router}).$mount("#app");
</script>
```

解释：

1. `:userid` ：动态的参数。
2. `this.$route.params` ：这个里面记录了路由中的参数。

组件复用：

当使用路由参数时，例如从 `/user/foo` 导航到 `/user/bar` ，原来的组件实例会被复用。因为两个路由都渲染同个组件，比起销毁再创建，复用则显得更加高效。不过，这也意味着组件的生命周期钩子不会再被调用。复用组件时，想对路由参数的变化作出响应的话，你可以简单地 `watch(监测变化)``$route` 对象：

```
const User = {
  template: '...',
  watch: {
    '$route' (to, from) {
      // 对路由变化作出响应...
    }
  }
}
```

或者是使用后面跟大家讲到的导航守卫：

```
const User = {
  template: '...',
  beforeRouteUpdate (to, from, next) {
    // react to route changes...
    // don't forget to call next()
  }
}
```

匹配404错误：

在路由规则中， `*` 代表的是任意字符。所以只要在路由的最后添加一个 `*` 路由，那么以后没有匹配到的 `url` 都会被导入到这个视图中。

```
let UserProfile = {template:"<h1>个人中心：{{$route.params.userid}}</h1>"};
let NotFound = {template: "<h1>您找的页面已经到火星啦！</h1>"}
var routes = [
    {path: "/user/:userid",component: UserProfile},
    {path: "*",component: NotFound},
]
```

嵌套路由：

有时候在路由中，主要的部分是相同的，但是下面可能是不同的。比如访问用户的个人中心是 `/user/111/profile/` ，查看用户发的贴子是 `/user/111/posts/` 等。这时候就需要使用到嵌套路由。

```
const router = new VueRouter({
  routes: [
    { path: '/user/:id', component: User,
      children: [
        {
          // 当 /user/:id/profile 匹配成功,
          // UserProfile 会被渲染在 User 的 <router-view> 中
          path: 'profile',
          component: UserProfile
        },
        {
          // 当 /user/:id/posts 匹配成功
          // UserPosts 会被渲染在 User 的 <router-view> 中
          path: 'posts',
          component: UserPosts
        }
      ]
    }
  ]
});
```

编程式导航：

之前我们学习了使用 `<router-link>` 可以在用户点击的情况下进行页面更新。但有时候我们想要在 `js` 中手动的修改页面的跳转，这时候就需要使用编程式导航了。

`$router.push` 跳转：

想要导航到不同的 `URL` ，则使用 `router.push` 方法。这个方法会向 `history` 栈添加一个新的记录，所以，当用户点击浏览器后退按钮时，则回到之前的 `URL` 。

当你点击 `<router-link>` 时，这个方法会在内部调用，所以说，点击 `<router-link :to="...">` 等同于调用 `router.push(...)` 。

声明式	编程式
`<router-link :to="...">`	`router.push(...)`

```
// 字符串
router.push('home')

// 对象
router.push({ path: 'home' })

// 命名的路由
router.push({ name: 'user', params: { userId: '123' }})

// 带查询参数，变成 /register?plan=private
router.push({ path: 'register', query: { plan: 'private' }})
```

注意：如果提供了 `path` ， `params` 会被忽略，上述例子中的 `query` 并不属于这种情况。取而代之的是下面例子的做法，你需要提供路由的 `name` 或手写完整的带有参数的 `path` ：

```
const userId = '123'
router.push({ name: 'user', params: { userId }}) // -> /user/123
router.push({ path: `/user/${userId}` }) // -> /user/123
// 这里的 params 不生效
router.push({ path: '/user', params: { userId }}) // -> /user
```

`router.replace(location, onComplete?, onAbort?)` ：

跟 router.push 很像，唯一的不同就是，它不会向 history 添加新记录，而是跟它的方法名一样 —— 替换掉当前的 history 记录。

声明式	编程式
`<router-link :to="...">`	`router.push(...)`

`router.go(n)` ：

这个方法的参数是一个整数，意思是在 `history` 记录中向前或者后退多少步，类似 `window.history.go(n)` 。

```
// 在浏览器记录中前进一步，等同于 history.forward()
```

```
router.go(1)

// 后退一步记录，等同于 history.back()
router.go(-1)

// 前进 3 步记录
router.go(3)

// 如果 history 记录不够用，那就默默地失败呗
router.go(-100)
router.go(100)
```

命名路由：

有时候，通过一个名称来标识一个路由显得更方便一些，特别是在链接一个路由，或者是执行一些跳转的时候。你可以在创建 `Router` 实例的时候，在 `routes` 配置中给某个路由设置名称。

```
const router = new VueRouter({
  routes: [
    {
      path: '/user/:userId',
      name: 'user',
      component: User
    }
  ]
})
```

要链接到一个命名路由，可以给 `router-link` 的 `to` 属性传一个对象：

```
<router-link :to="{ name: 'user', params: { userId: 123 }}">User</router-link>
```

这跟代码调用 router.push() 是一回事：

```
router.push({ name: 'user', params: { userId: 123 }})
```

命名视图：

有时候想同时 (同级) 展示多个视图，而不是嵌套展示，例如创建一个布局，有 `sidebar` (侧导航) 和 `main` (主内容) 两个视图，这个时候命名视图就派上用场了。你可以在界面中拥有多个单独命名的视图，而不是只有一个单独的出口。如果 `router-view` 没有设置名字，那么默认为 `default` 。

```
<router-view class="view one"></router-view>
<router-view class="view two" name="a"></router-view>
<router-view class="view three" name="b"></router-view>
```

一个视图使用一个组件渲染，因此对于同个路由，多个视图就需要多个组件。确保正确使用 `components` 配置 (带上 `s`)：

```
const router = new VueRouter({
  routes: [
    {
      path: '/',
      components: {
        default: Foo,
        a: Bar,
        b: Baz
      }
    }
  ]
})
```

重定向和别名：

重定向也是通过 `routes` 配置来完成，下面例子是从 `/a` 重定向到 `/b` ：

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: '/b' }
  ]
})
```

重定向的目标也可以是一个命名的路由：

```
const router = new VueRouter({
  routes: [
    { path: '/a', redirect: { name: 'foo' }}
  ]
})
```

“重定向”的意思是，当用户访问 `/a` 时， `URL` 将会被替换成 `/b` ，然后匹配路由为 `/b` ，那么“别

名”又是什么呢？

`/a` 的别名是 `/b` ，意味着，当用户访问 `/b` 时， `URL` 会保持为 `/b` ，但是路由匹配则为 `/a` ，就像用户访问 `/a` 一样。

上面对应的路由配置为：

```
const router = new VueRouter({
  routes: [
    { path: '/a', component: A, alias: '/b' }
  ]
})
```

官方文档：

更多内容请参考 `vue-router` 官方文档： `https://router.vuejs.org/zh/`

Vue-Cli

17-node环境配置

node环境配置：

nvm安装：

`nvm（Node Version Manager）` 是一个用来管理 `node` 版本的工具。我们之所以需要使用 `node` ，是因为我们需要使用 `node` 中的 `npm(Node Package Manager)` ，使用 `npm` 的目的是为了能够方便的管理一些前端开发的包！ `nvm` 的安装非常简单，步骤如下：

1. 到这个链接下载 `nvm` 的安装包： `https://github.com/coreybutler/nvm-windows/releases` 。
2. 然后点击一顿下一步，安装即可！
3. 安装完成后，还需要配置环境变量。在 `我的电脑->属性->高级系统设置->环境变量->系统环境变量->Path` 下新建一个，把 `nvm` 所处的路径填入进去即可！
4. 打开 `cmd` ，然后输入 `nvm` ，如果没有提示没有找不到这个命令。说明已经安装成功！
5. `Mac` 或者 `Linux` 安装 `nvm` 请看这里： `https://github.com/creationix/nvm` 。也要记得配置环境变量。

`nvm` 常用命令：

1. `nvm install [version]` ：安装指定版本的 `node.js` 。
2. `nvm use [version]` ：使用某个版本的 `node` 。
3. `nvm list` ：列出当前安装了哪些版本的 `node` 。
4. `nvm uninstall [version]` ：卸载指定版本的 `node` 。
5. `nvm node_mirror [url]` ：设置 `nvm` 的镜像。
6. `nvm npm_mirror [url]` ：设置 `npm` 的镜像。

node安装：

安装完 `nvm` 后，我们就可以通过 `nvm` 来安装 `node` 了。这里我们安装 `10.16.0` 版本的的`node.js。

```
nvm install 10.16.0
```

如果你的网络够快，那以上命令在稍等片刻之后会安装成功。如果你的网速很慢，那以上命令可能会发生超时。因为 `node` 的服务器地址是 `https://nodejs.org/dist/` ，这个域名的服务器是在国外。因此会比较慢。因此我们可以设置一下 `nvm` 的源。

```
nvm node_mirror https://npm.taobao.org/mirrors/node/
nvm npm_mirror https://npm.taobao.org/mirrors/npm/
```

npm：

`npm(Node Package Manager)` 在安装 `node` 的时候就会自动的安装了。当时前提条件是你需要设置当前的 `node` 的版本： `nvm use 10.16.0` 。然后就可以使用 `npm` 了.

初始化：

在新的项目中，需要先执行 `npm init` 初始化，创建一个 `package.json` 文件用来保存本项目中用到的包。

安装包：

安装包分为全局安装和本地安装。全局安装是安装在当前 `node` 环境中，在可以在cmd中当作命令使用。而本地安装是安装在当前项目中，只有当前这个项目能使用，并且可以通过require引用。安装的方式只有 `-g` 参数的区别：

```
npm install vue   # 本地安装
npm install vue --save   # 本地安装，并且保存到package.json的dependice中
npm install vue --save-dev # 本地安装，并且保存到package.json的dependice-dev中
npm install vue -g   #全局安装
npm install -g @vue/cli  #全局安装vue-cli
```

本地安装

1. 将安装包放在 `./node_modules` 下（运行 npm 命令时所在的目录），如果没有 `node_modules` 目录，会在当前执行 `npm` 命令的目录下生成 `node_modules` 目录。
2. 可以通过 `require()` 来引入本地安装的包。

全局安装

1. 将安装包放在 `/usr/local` 下或者你 `node` 的安装目录。
2. 可以直接在命令行里使用。

卸载包：

```
npm uninstall [package]
```

更新包：

```
npm update [package]
```

搜索包：

```
npm search [package]
```

使用淘宝镜像：

`npm` 的服务器在国外。那么可以安装一下 `cnpm` ，并且指定镜像为淘宝的镜像：

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

那么以后就可以使用cnpm来安装包了！

手动安装npm：

有时候使用 `nvm` 安装完 `node` 后， `npm` 没有跟着安装，这时候可以到 `https://github.com/npm/cli/releases` 下载 `6.10.1` 的版本。然后下载完成后，解压开来，放到 `v10.16.0/node_modules` 下，然后修改名字为 `npm` ，并且把 `npm/bin` 中的 `npm` 和 `npm.cmd` 两个文件放到 `v10.16.0` 根目录下。

18-vue-cli

vue-cli

`vue-cli` 是和 `vue` 进行深度组合的工具，可以快速帮我们创建 `vue` 项目，并且把一些脚手架相关的代码给我们创建好。真正使用 `vue` 开发项目，都是用 `vue-cli` 来创建项目的。

安装：

`Vue CLI` 需要 `Node.js 8.9` 或更高版本 (推荐 `8.11.0+`)。 `node` 环境安装后，直接通过 `npm install -g @vue/cli` 即可安装。安装完成后，输入 `vue --version` ，如果出现了版本号，说明已经下载完成。

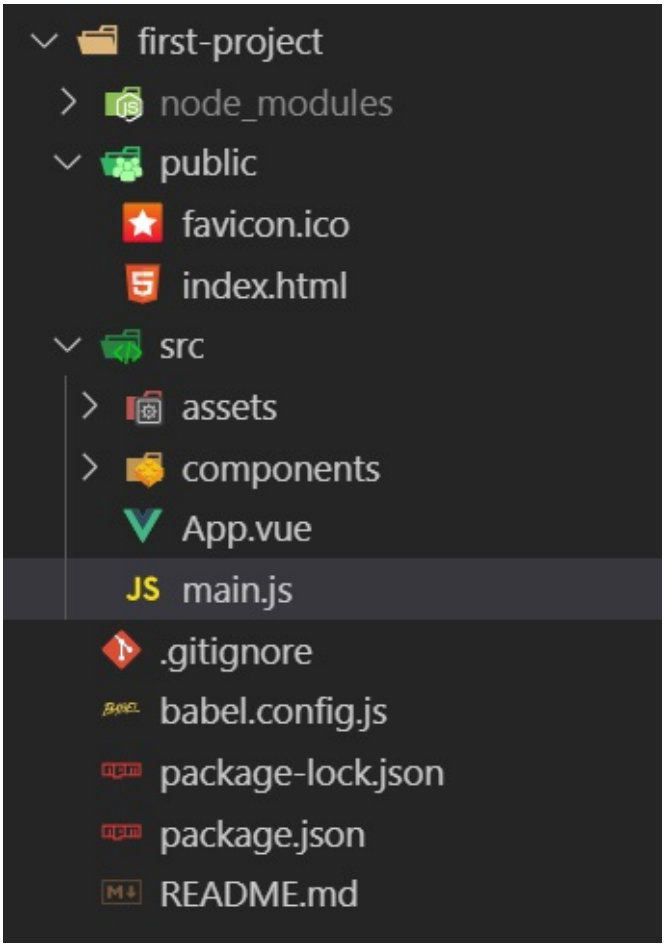
用命令行创建项目：

1. 在指定路径下使用 `vue create [项目名称]` 创建项目。
2. 会让你选择要安装哪些包（默认是 `Babel` 和 `ESLint` ），也可以手动选择。
3. 如果在安装的时候比较慢，可以在安装的时候使用淘宝的链接：
   `vue create -r https://registry.npm.taobao.org [项目名称]` 。
4. 如果实在不想在创建项目的时候都指定淘宝链接，可以在当前用户目录下，找到 `.npmrc` ，然后设置 `registry=https://registry.npm.taobao.org` 。

用界面创建项目：

1. 打开 `cmd` ，进入到你项目存储的路径下。然后执行 `vue ui` ，就会自动打开一个浏览器界面。
2. 按照指引进行选择，然后一顿下一步即可创建。

项目结构介绍：



1. `node_modules` ：本地安装的包的文件夹。
2. `public` ：项目出口文件。
3. `src` ：项目源文件：
    - `assets` ：资源文件，包括字体，图片等。
    - `components` ：组件文件。
    - `App.vue` ：入口组件。
    - `main.js` ： `webpack` 在打包的时候的入口文件。
4. `babel.config.js` ： `es*` 转低级 `js` 语言的配置文件。
5. `package.json` ：项目包管理文件。

组件定义和导入：

1. 定义：组件定义跟之前的方式是一模一样的。只不过现在模板代码专门放到 `.vue` 的 `template` 标签中，所以不再需要在定义组件的时候传入 `template` 参数。另外，因为需要让别的组件使用本组件，因此需要用 `export default` 将组件对象进行导出。
2. 导入：因为现在组件是在不同的文件中。所以如果需要引用，那么必须进行导入。用 `ES6` 语法的 `import XXX from XXX` 。

局部样式：

默认情况下在 `.vue` 文件中的样式一旦写了，那么会变成全局的。如果只是想要在当前的组件中起作用，那么可以在 `style` 中加上一个 `scoped` 属性即可。示例代码如下：

```
<style scoped>
.info{
   background-color: red;
}
</style>
```

使用 `sass` 语法：

很多小伙伴在写样式代码的时候，不喜欢用原生 `css` ，比较喜欢用比如 `sass` 或者 `less` ，这里我们以 `sass` 为例，我们可以通过以下两个步骤来实现：

1. 安装 `loader` ： `webpack` 在解析 `scss` 文件的时候，会去加载 `sass-loader` 以及 `node-loader` ，因此我们首先需要通过 `npm` 来安装一下：

```
npm install node-sass@4.12.0 --save-dev
npm install sass-loader@7.0.3 --save-dev
```

2. 指定 `sass` 语言：在指定 `style` 的时候，添加 `lang="scss"` 属性，这样就会将 `style` 中的代码识别为 `scss` 语法。