python3_Application Documentation

发布 *1.0*

HangHang

2023 年 02 月 26 日

Contents

注解:

- 大家好，我是一名软件开发工程师，崇尚用技术解决实际问题，提高效率，致力于用技术创造无限价值。在这里主要整理日常的编程开发、数据库、数据结构、算法、机器智能等笔记，很高兴和各位小伙伴交流、探讨，分享。
- 更新时间：2023 年 2 月 26 日.

CHAPTER 1

目录:

1.1 公共资源

1.1.1 关于我

注解: 更新日期：2023-02-26

基本情况

- 本科专业：软件工程 2013.9-2017.6
  - 主修云计算相关课程。如，KVM、OpenStack 平台等
  - 辅修大数据相关课程。如数据挖掘、Mahout 等
- 大学生活
  - 挺丰富充实的吧，拿到学位的同时，也不断拓展自己的边界。
  - 如果聊大学生活或如何度过大学，我们私信聊吧，很乐意交流沟通，感觉自己也挺有发言权的。
- 长期研究领域:
  - 机器学习、自然语言处理、知识图谱的应用
  - 云计算、大数据与机器学习的结合应用

- 兴趣岗位：后端开发及机器智能系统开发
- 编程语言
  - 擅长：Java、Python
  - 一般：C#、PHP、C++
- 数据库
  - 关系型：MySQL、SqlServer

工作经历

- 工作年限：5.5 年
- 工作情况：
  1. 经历一：2017.7-2019.6 中国科学院大学计算机学院 CCIP 实验室技术工程师
     - 承担完成北京市科委类脑智能推理系统（金融领域）前后端开发工作。主要使用 Java 及借助 MATLAB 语言、Python 及 TensorFlow 等开发完成。
       * 系统前后端实现。
       * 财务风险预测、文本挖掘模型建模与实现。
       * 数据可视化。
     - 调研智能医疗领域（医疗影像分析）发展情况。
     - 学习机器学习及深度学习的理论知识并应用到推理系统的相关算法模型。
     - 学习自然语言处理基础理论并研究知识图谱方向。
  2. 经历二：2019.7-至今北京文因互联科技有限公司软件开发工程师
     - 服务领域：金融科技，科技监管等
     - 主流语言：Python，JAVA
     - 主流技术：NLP，知识图谱
     - 涉及场景：信息抽取系统，金融科技监管系统，数据平台，AI 中台等
     - **阶段 1 2019.7.1–2020.7.1**
       * 职业角色：后端开发
       * 主要工作内容：文本信息抽取及平台建设
     - 阶段 2 2020.7.1–2022.4.1
       * 职业角色：项目管理与执行、后端开发。
       * **主要工作内容：**

1. 项目管理（部分）及总体执行、项目各资源统筹、需求对接、分解、落地。
2. 文本信息抽取系统总体架构及功能完善与维护。
3. 系统交付及部署。
4. 项目文档编写。

- 阶段 3 2022.4.1–2022.11
  * 职业角色：teamleader, 后端开发。
  * **主要工作内容：产品研发工作**
    1. 制定小组开发计划并跟进完成情况。
    2. 和其小组 teamleader 协调开发计划。
    3. 定期向上汇报开发进展。
    4. 参与系统框架设计及编码工作，codeReview。
- **阶段 4 2022.12–至今**
  * 职业角色：技术研发管理负责人，产品研发负责人。
  * **主要工作内容：**
    1. 技术研发日常管理。人才梯度建设，跟进区域内项目执行情况，日常培训，考核等。
    2. 公司核心产品研发管理。研发任务拆解，迭代，跟进日常开发，测试，部署，测试，复盘等。
    3. 参与公司相关商务，售前打单，技术评估，沟通汇报等工作。

- 更多了解，见我的流水文章Mason
- E-mail: hanghangli@aliyun.com

1.1.2 关于技术屋

主要整理下自己日常学习、研究、工作中的内容。

- 包含两大系列
  - 后端编程开发系列（全流程）
    * 语言、数据结构、网络、算法、服务器、运维等
  - 机器智能系列（认知智能）
    * 机器学习、自然语言处理、知识图谱等
    * 规则引擎
- 其他

- 项目管理及技术团队管理
- 技术学习及实践思考及经验总结（道与术）
- 技术输出思考

1.1.3 基于 ReadtheDocs 托管在线知识库的步骤

注解： 更新日期：2022-08-01

Abuout Hang’s Tec Room

本文主要为想使用 ReadtheDocs 构建在线知识库的小伙伴提供一个参考步骤，我在用这个工具之前尝试了比较多的网站来记录一些笔记，但是感觉还是不到位，Hang’s Tec Room 是基于目前的工具构建的，用起来虽然有些门槛，但只要多练习下就能很快掌握。具体有以下特点：

1. 工具：Sphinx
2. 托管地址：ReadtheDocs
3. 支持 rst 和 md 格式的文档，md 还是很常用，编写起来也容易。
4. 基于 github 或 gitlab 进行代码管理，推送后 ReadtheDocs 自动进行构建和发布。
5. 支持导出 PDF，自动排版，易阅读。

基本实践步骤 (QuickStart)

注解： 做一个基本的操作步骤说明，希望对大家有帮助。

1. 新建目录，这里为：my_books 并进入目录

```
$ mkdir my_books
$ cd my_books
```

2. 安装工具 Sphinx

   ```
   $ pip install -i https://pypi.tuna.tsinghua.edu.cn/simple sphinx
   ```

3. 使用 Sphinx 初始化文档目录

   $ sphinx-quickstart

```
$ sphinx-quickstart
Welcome to the Sphinx 1.7.9 quickstart utility.

Please enter values for the following settings (just press Enter to
accept a default value, if one is given in brackets).

Selected root path: .

You have two options for placing the build directory for Sphinx output.
Either, you use a directory "_build" within the root path, or you separate
"source" and "build" directories within the root path.
> Separate source and build directories (y/n) [n]: y

Inside the root directory, two more directories will be created; "_templates"
for custom HTML templates and "_static" for custom stylesheets and other stat
files. You can enter another prefix (such as ".") to replace the underscore.
> Name prefix for templates and static dir [_]: book

The project name will occur in several places in the built documentation.
> Project name: my_books
> Author name(s): lihanghang
> Project release []: 1.0

If the documents are to be written in a language other than English,
you can select a language here by its language code. Sphinx will then
translate text that it generates into that language.

For a list of supported codes, see
http://sphinx-doc.org/config.html#confval-language.
> Project language [en]: zh_CN

The file name suffix for source files. Commonly, this is either ".txt"
or ".rst".  Only files with this suffix are considered documents.
> Source file suffix [.rst]: .rst
```

初始化后的内容

```
>>> ls
Makefile
build    # 存放编译后的文件
make.bat # win 下编译脚本
source # 文件目录
```

4. **每次发生文档变动后进行编译，生成后的文件放在 build/html 目录下**

   - macos:

     ```
     $ make html
     ```

   - win:

     ```
     $ ./make.bat html
     ```

5. 选择发布的网站主题 - 主题网站 ， 我这里选择比较常见的 Read the Docs - 效果图

   - **install theme** `$ pip install sphinx-rtd-theme`
   - **set configure *conf.py* file , 位置：my_books/source/conf.py # line 81** $ html_theme

= 'sphinx_rtd_theme'

6. **生成依赖文件，服务于在 readthedoc 托管文档时自动构建，在根目录下生成** `$ python -m pip freeze > requirements.txt`

7. 将代码仓库在 github 管理

```
$ git init
$ git add .
$ git commit -m "init my books code"
$ git push
```

8. readthedoc 网站托管–建立 GitHub 代码仓库与 readthedocs 的关联
   - 大家可直接 参考官方步骤
9. 至此，从 sphinx 的安装，主题选择，再到推送到 github，建立与 readthedocs 的关联就完成了。
10. 后续就是具体的文档编写，大家可以多看看 rst 的语法内容，也可以使用 markdown 语法去编写文档。

小技巧: 在 source 目录下进行文档编写，编写完成后进行 make html 编译操作，再推送 github，构建由 readthedocs 自动完成。

谈谈题外话

:: 可以将此网站作为自己的日常生活或工作总结的地方，梳理总结的有价值的地方作为经验沉淀下来。对于工作多说一些，建议大家建立自己完整的知识体系，然后不断完善这个体系，这样有助于在我们工作，学习中不容易迷失方向。最后，如果自己的输出能有一些帮助到别人的部分也是很有价值的!

- 更多了解Mason
- E-mail: hanghangli@aliyun.com

1.2 语言及框架

主要包含 Java、C++ 可编译性语言及脚本语言 Python 的技术笔记整理；同时还涉及一些常用框架的整理。更新时间：2021 年 01 月 03 日.

1.2.1 C++

-

1.2.2 Java 基础技术体系

过一下 Java 的基础知识及相关组件的原理。如：JVM、类装载、线程、并发、IO 资源管理、网络等。

一、编程范式

面向过程

面向对象

函数式编程

响应式编程：(Reactive programing)

1. 关心数据流和流的变化。
2. 和传统方式的区别：有数据即刻响应
3. 设计模式：观察者模式
4. 关键机制：背压。数据流的生产端能够知道消费端的处理能力，并以此调整生产量。
5. 核心项目：Reactor。flux(publisher) 和 mono(publisher)
6. 一些实现框架：Spring-webflux.

二、JVM

1.2.3 Spring

注解： 了解一些 Spring 框架的设计思想和原理，本文记录下学习的笔记。至于应用层面可以：一个项目、一台电脑、一双手去实践吧，框架就是工具贵在和具体业务结合应用，工具的目的很简单就是来提高效率（开发、协作、运维等）的。

参考资料

1. 查看 Spring 框架的设计理念与设计模式分析。

Spring 的设计思想

1. **框架基石：Core（道具）、Context（舞台）、Bean（演员）（面向 BOP 编程,Bean Oriented Programming）**

    - BOP 在 Spring 框架中的地位相当于 OOP 在编程中的地位。
    - Bean 封装的是对象，而对象本身的数据、行为的由 Context 承载，Context 会对 Bean 之间的关系进行建立和维护，Context 是 Bean 关系的一个集合。
    - IOC 容器就是 Context 中 Bean 的关系集合。
    - Core 为维护 Bean 之间关系提供一系列的工具。

2. **关键是对象之间依赖关系的管理，Spring 提供了配置文件形式来管理，即依赖注入机制。**

    - 注入关系是在 IOC（IOC，Inversion of Controll）容器中进行管理。

1.2.4 SpringBoot 基础

注解:

- 开始日期：2020-03-16
- 更新日期：2020-03-16
- 结束日期：2020-06-01

入门

- **背景**
    - **目的是简化 Spring 的应用开发，约定大于配置。**
        * 传统 J2EE 开发配置繁琐、开发效率低、部署复杂，集成难度大
    - Spring 的技术大整合
    - 更多可参考官网：https://spring.io/projects/spring-boot
- **优点**
    - 快速创建独立运行的 Spring 项目
    - 方便发布，直接打 Jar 包
    - starts 自动依赖和版本控制
    - 配置自动化程度高，开发上手容易，不用配置 XML
    - 监控方便

– 与云计算技术更好兼容

配置

日志

Web 开发

Docker

数据访问

启动配置原理

自定义 starters

1.2.5 SpringBoot 进阶

缓存

消息

检索

任务

安全

分布式

开发热部署

监控管理

1.2.6 Python 基础知识

开发口头禅：“Don’t Repeat Yourself”

coding env: python3.10+

书籍推荐

1. 《流畅的 Python》
   - 能够解决大部分疑难杂症

2. 《python3 cookbook》
3. 《python 进阶》深入 python 语法等
4. 《编写高质量代码改善 Python 程序的 91 个建议》
5. 《python Cookbook》
6. 《Python 工匠-案例，技巧与工程实践》
    1. 讲的也比较深入，也有具体的示例，兼顾了了理论和实践，如面向对象的设计原则等。

第一部分编码规范

良好的习惯我们就从最开始刻意练习，让我们的代码也要同翻译的三字标准一样：**信、雅、达**。
这里记录我认为比较好的习惯吧，供大家参考使用。

1. 缩进规范
    - 使用**纯空格**缩进，不要 Tab、不要空格与 Tab 混用。
2. **导包规范**
    - 尽量避免 import *，如果要导入一个命名空间下的多个方法，可使用 import test_class，再使用 test_class.func 的方式
    - 顺序问题。标准库》第三方库》本地库
3. 命名规范
    - 命名要尽量达意！长一点倒不怕，就怕他人看到名字不能很快了解其含义。比如表示订单数量：num、order_num 你说哪个好！
4. 对于嵌套处理尽量不要超过 4 层。比如 for 与 if 的嵌套，按照我的经验这类代码肯定是有优化空间，可以从 Python 自身的一些好用特性和业务逻辑本身两个角度与优化。

第二部分基础知识

基础不牢，地动山摇。

第三部分进阶知识

攀登高峰的脚步　不能停下来，向前进……。主要包含 python 的高阶函数使用及技巧。

1. 产生随机字典、集合（一般在测试代码时使用）
    - 主要使用 random 包中的 randint 模块

```
#!/usr/bin/env python3
from random import randint

# generate dict
d = {"name%d" % n :randint(10, 30) for n in range(10)}
# generate set
s = {randint(10, 30) for _ in range(10)}
```

2. 集合的求交集操作

   - key points: keys() of dict、func map, reduce
   - 把函数作为结果返回的函数是高阶函数：map、filter、reduce（py3 已不再内置）、sorted、all etc.
   - map 函数：把函数应用到各个元素上，生成一个新序列。
   - reduce 函数：把一系列值归约成单个值。第一个参数是接受两个参数的函数，第二个参数是一个可迭代的对象。看下一个例子：

```
from functools import reduce

# example:  reduce
reduce(func, iterable): reduce(lambda a, b: a * b, range(1, 5))
>>> 24
```

```
from random import randint, sample
from functools import reduce

# random generate three dict
s = 'china'
d1 = {k: randint(2, 5) for k in sample(s, randint(4, 7))}
d2 = {k: randint(2, 5) for k in sample(s, randint(4, 7))}
d3 = {k: randint(2, 5) for k in sample(s, randint(4, 7))}
# 求以上三个字典的公共键：使用 map、all
d1 = [d1, d2, d3]
[k for k in d1[0] if all(map(lambda d: k in d, d1[1:]))]
>>> ['o']

# 使用集合运算求交集：reduce。使用场景：存在多个自典时。
reduce(lambda dict1, dict2: dict1 & dict2, map(dict.keys, d1))
>>> {'o'}
```

3. 枚举、命名元组模块

第四部分高阶知识

会当凌绝顶，一览众山小。

一切皆对象（包括类和函数）

```
```python
    def func_1(name='zhang'):
        print(name)

def func_2():
        return func_1
    # return func
    new_inst = func_2() # 返回一个函数
    new_inst('li')
```
```

类型、对象和类

元编程

还记得咱们的开发口头禅吗？对就是“don’t repeat yourself”，Python 自身也为我们开发人员提供了一种实现方式，即元编程，目的就是减少重复性代码，主要技术是使用装饰器、类装饰器和元类。

1. 装饰器
   - 场景 1: 在系统中大多数的模块都有日志打印、时间统计等功能。
   - 类方法。@classmethod 装饰器，表示属于类但无需实例化也可调用
   - 静态方法。@staticmethod 装饰器，这个方法和实例内容没有关联，就相当于一个普通的函数，但是静态方法能被子类继承和重写。
   - @property 。模糊属性和类方法的调用差异。当某个方法使用了该关键字后，该方法就变为一个虚拟的属性，比如获取学生的信息方法，stu.stu_info() –> stu.stu_info。其次，对于学生信息的增加和删除也可以基于 @stu_info.setter 或 @stu_info.deleter 装饰器实现。
   - @abstractmethod。来源于 collections.abc 模块，把某个方法标记为抽象类方法，如果该方法所在类作为父类被继承则需要在子类中实现该方法。相当于 Java 的接口实现。

多线程

1. 先要知道 GIL（Global Interpreter Lock），Python 的一把全局解释锁

协程

协程（Coroutine），又称微线程

第五部分性能优化

1. 性能优化参考之一

第六部分思考和总结

不忘初心，方得始终，回望过去，是为了更好走向远方。

python 小应用

总结工作中可能用到的业务功能。

update: 2020-03-04

1.2.7 Django

Django 框架是使用非常广泛的优秀框架（源自国外一个新闻系统，自带后台），无论其架构设计及构建系统的便利，我们都有必要去学习研究，对于基于其开发系统或者自己设计项目架构都有借鉴意义！

这是我大学时的一门课程，在这里再回顾下，将笔记记录于此，与大家一起学习探讨。

设计哲学（理念）

1. 基本目标
   - 低耦合高内聚。Django 本身也是一个技术栈，其构建目标就是各模块，如视图、数据库、模板等部分最大化相互解耦，这就增加了框架的灵活性和可扩展性。
2. 基本原则
   - 精简。借助 Python 语言自身的特性：动态性，如自身机制使框架代码尽可能地精简。
   - 构建快。能够让开发者快速基于框架进行 web 开发，把一些通用的、复用的功能代码抽象，使我们专注业务实现。
   - 不重复造轮子。各功能模块各就其位，减少冗余，提供扩展性强的接口。
   - 遵循 PEP20（翻译为：Python 之禅）的原则。
   - 一致性。

各组件设计原则

模型（Model）

1. 明确优于隐式
   - 字段不应该仅仅根据字段的名称来假定某些行为
   - 多使用关键字参数

     什么是关键字参数？如：*args, **kwargs。使用关键字参数可增加函数的灵活性，健壮性。
2. 包括所有相关领域逻辑
   - 遵循Active Record模式

     表现为类似 ORM 的作用
   - 在模型中将数据库表定义为一个 object, 并尽可能全面的定义表的所需信息

数据库 API

1. 简洁数据操作代码
   - 避免再写一些繁琐的数据库操作代码，快速实现业务逻辑。
2. 可选择原始 SQL
   - 框架支持自定义纯 SQL 语句，可根据实际业务需求实现。

URL 设计

1. 松耦合
   - URL 不用实际的处理函数名耦合
2. 灵活性
3. 鼓励
   - 在 URL 中应避免出现文件后缀名。

模板系统

1. 逻辑分类方案
   - view 层和 Control 层分离
2. 避免冗余

- 把一些的通用的视图放在一起。

3. 从 HTML 中解耦
   - 模板系统不应该被设计成只能输出 HTML。它应该同样擅长生成其他基于文本的格式，或者仅仅是纯文本。
4. 设计能力交给开发
   - 开发直接编辑相应 HTML
5. 更加直接的处理空格
6. 安全机制
   - 内置了开箱即用的模板系统禁止包含恶意代码
7. 可扩展性
   - 自定义的模板标签和过滤器背后的理念.

视图

1. 简洁
   - 传承 Python 的优良风格
2. 使用请求对象
   - 存储当前请求的元数据的对象，直接传给视图函数，避免从全局访问请求数据的视图函数
3. 松耦合
   - 不受开发人员使用任意模板的影响
4. GET 方法和 POST 方法的区别
   - 框架是的 GET 和 POST 数据很容器区分

缓存框架

目的：减少请求系统的资源开销

```
given a URL, try finding that page in the cache
if the page is in the cache:
    return the cached page
else:
    generate the page
    save the generated page in the cache (for next time)
    return the generated page
```

1. 更少的代码
2. 一致性
   - 缓存 API 应该为不同的缓存后端提供一致的接口。
3. 可扩展性
   - 我不想用自带的缓存，我想用第三方的缓存框架

注解: 规则引擎主要完成的就是将业务规则从代码中分离出来。Drools 是常用的开源 JAVA 规则引擎之一，本笔记也以此走进规则引擎的世界。我们将从其基本用法、业务场景落地、具体实践方面讲解。

1.2.8 Drools 应用场景

1. 银行、支付、保险领域等
2. 风控场景。

1.2.9 Drools 基本概念

1. 事实（Fact）：对象之间及对象属性之间的关系
2. 规则（rule）：是由条件和结论构成的推理语句，一般表示为 if…Then。一个规则的 if 部分称为 LHS，then 部分称为 RHS。
3. 模式（module）：就是指 IF 语句的条件。这里 IF 条件可能是有几个更小的条件组成的大条件。模式就是指的不能在继续分割下去的最小的原子条件。

1.2.10 Drools 应用经验总结

1. 设置全局变量 global。
2. 设置全局公共 function。
3. 执行规则文件可设定规则前缀。
4. 尽量拆分规则，避免 then 中再出现 if else 语句。
5. 使用一些关键函数会使多线程失效，这一点需要注意。

1.3 接口（Interface）开发

注解:

- 更新时间：2020-06-30
- 整理一下日常接口开发规范和经验，主要包含基本概念、设计项目接口、接口开发原则、接口定义规范等。仅供参考！

1.3.1 参考

- 《理解 RESTful 架构》作者：阮一峰
- 《RESTful API 设计指南》作者：阮一峰

1.3.2 几个概念

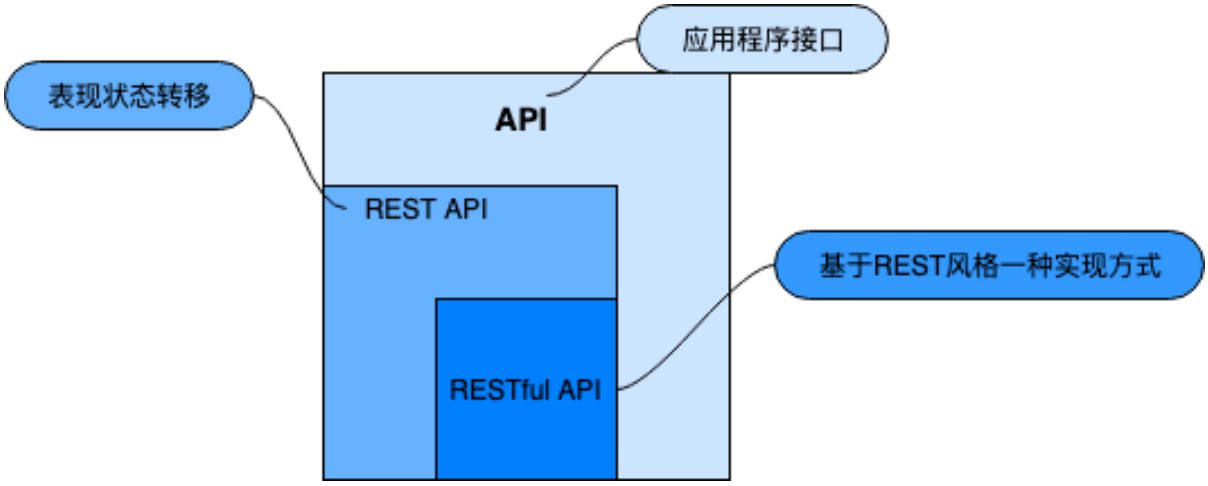
平时开发基本都会听到这些高大上的词：API、REST-API、RESTful-API，对于追求事物本质的我就集中梳理总结下。

1. **API**
    - Application Programming Interface（API）应用程序（编程）接口。(有点蒙、有点蒙……)
    - 打个比方你现在用的电脑操作系统 windows 就是一个接口，这个接口连接了你和电脑的硬件。类似的，API 就是连接程序和程序之间的接口。
    - 现在人工智能产品越来越多，好多公司也开放了功能接口，赋能各行各业，包括个人开发这。这就是接口的价值，你无需了解底层的原理，就可以使用开放的接口进行应用的开发。
2. **REST API**
    - Representational State Transfer(REST) 表现层状态转化。
    - **关键的几个概念：资源、表现层、状态转移。**
        - 资源：文本、图像、音频等实体；这里涉及两个概念 URI 和 URL，URI 统一资源 * 标识符 *，就是标识前面提到资源的，一般用名词定义；URL 统一资源 * 定位符 *，URL 是一种 URI，标识一个资源，并指定对其进行 * 操作或获取该资源 * 的方法，一般用动词定义。
        - 表现层：将资源按照怎样的数据结构来表示。如 XML、JSON、txt 等；
        - 状态转移：指的是请求接口的操作，如 GET、PUT、DELETE、POST。
3. **RESTful API**
    - 理解了 REST，再回头看 RESTful 就简单多了，只要符合 REST 的设计风格和原则就成为 RESTful 架构。
    - REST 不完全等价于 RESTful.REST API 是 Web API 设计的一种规范或者指导原则，而 RESTful API 则是这中架构设计原则或者规范的一种 * 具体实现 * 方式之一。

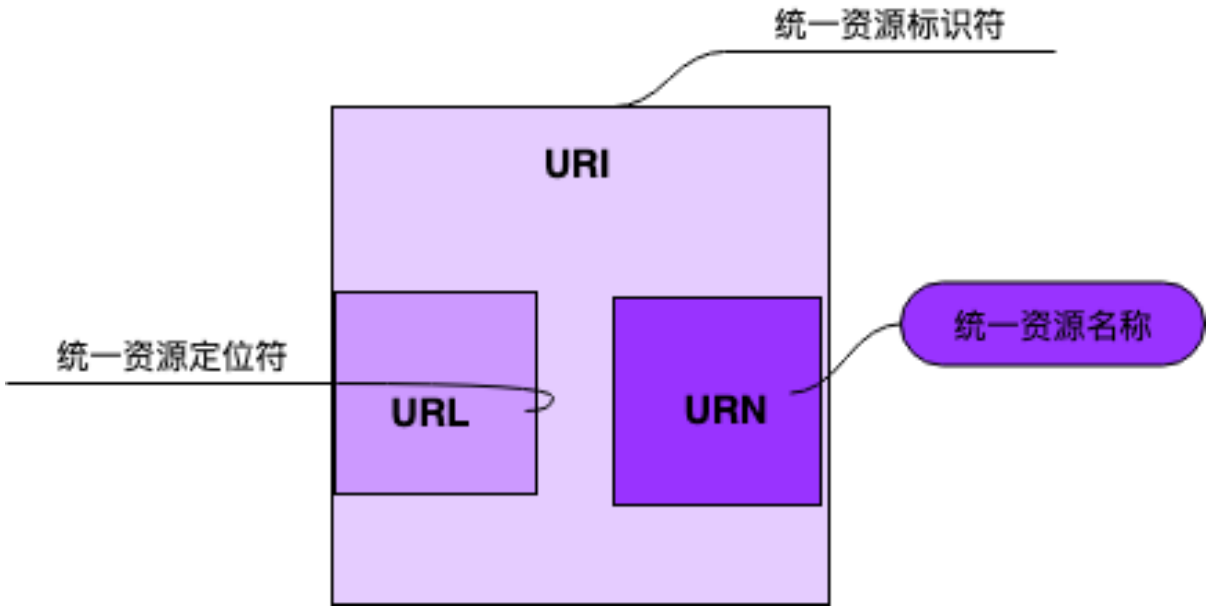
图述概念

画两张图直观感受下这几个概念的关系。

- **API**



- **URI**



1.3.3 接口的好处

- 提高开发效率。前后端分类，通过约定来解藕前后端的开发。
- 便于维护。
- 便于功能扩展。

1.3.4 接口定义原则及规范

接口的好处，离不开清晰的原则和规范。只有接口设计的合理才能更好的满足系统的使用和管理。

注解： 本文就采用 Backend For Frontend(BFF 层)，称为前端的后台，前端（网页、手机、平板等）的业务操作基于接口完成。

原则

常用规范（参考）

基本状态码

注解： 一般根据实际业务前后端约定码就行，大多数是一个区间，如：1000-1999，2000-2999 等

- 200 - 请求成功
- 301 - 资源（网页等）被永久转移到其它 URL
- 404 - 请求的资源（网页等）不存在
- 500 - 内部服务器错误

1. **错误码设计**
    - 核心是沟通：系统与系统、人与人、人与系统间的沟通。
    - 回答问题：谁的错？错在哪？
2. **错误码编码方式**
    1. 纯数字（大多数采用，腾讯、微博、Google 等）
    2. 纯英文
    3. **字母 + 数字（阿里巴巴推荐方式）**
        - 模式：错误产生来源（表示错误来源）+ 四位数字编号（表示具体错误）
        - **如，A0001-9999，便于快速问题溯源和记忆。**
            - A 来自用户、B 来自系统.（提取标准化约定好系统各个服务的字母编号）
            - 如 A 来用户端。A0001 用户端错误（一级）、A0100 用户注册错误（二级）A0101 未同意隐私条约（三级，具体细节错误）
        - 全部正常可选择错误码：00000
3. **错误码输出路径**
    1. 面向日志输出
    2. **面向外部传递**

- 服务器端传向前端
- OpenAPI 错误码
- 内部不同域之间

4. **错误码组成**
    1. HttpCode HTTP 响应码，符合 HTTP 协议语义
    2. Code 错误标识
    3. Message 错误消息
    4. Advice 建议处理方式

路由名称

1. 这里的路由名称就是请求接口时使用的，命名规则都是 *名词*，即前面讲到的 URI，也就是说定位资源。
2. 至于为何非要用名词，本质是定位资源（弄清概念），因此名称描述的也就是资源。
3. 我们很容易用动词，比如/get_results、/del_item 等等，其实这是多此一举的，因为 RESTful 方法名（GET、HEAD、POST、PUT、DELETE、OPTIONS、TRACE、PATCH）已经表达了对资源进行的动作了，所以接口名应定义为：/resluts、/item 就行了（即资源的名称，如学生、肥皂、汽车等）。

数据格式

小技巧: 考虑到接口的可维护性和可扩展性，返回的 msg、code 最好集中定义，一一对应约定好。

```
{
    "code": 0,        // 0 位正常，其他数字则为异常，这里不须强制使用 http 的状态码
    "msg": ""  // 附加信息
    "data": "",  // 返回数据
}
```

1.3.5 项目结构

1.3.6 接口开发经验

1.4 常见接口框架

注解:

- 更新时间：2020-05-22
- 常用接口开发框架整理

1.4.1 Flask

1.4.2 FastAPI

1.5 项目开发杂谈

注解： 记录一些项目开发中琐碎但是必然思考和使用的方法、规范等。本内容不限项目类型、不限编程语言都是一些通用的问题，也会分类进行梳理总结。

1.5.1 软件迭代的环境问题

1.5.2 软件开发的流程

1.5.3 工程结构问题

1.5.4 本地开发环境

1.6 数据结构 (Data Structure)

..note::

- 写代码时有必要考虑复杂度（空间及时间）。
- 主定理（Master Theorem）.
- 开始时间：2020-06-02
- 更新时间：2020-06-02

1.6.1 方法论

1. 坚持、刻意练习。
2. 练习缺陷（会的总会，不会总不会）、多练自己薄弱的、不熟悉的。
3. 刚开始 coding 是不爽、不舒服、枯燥的。

4. **练题平台：leetcode(力扣)，可以看国外版的网站 Discuss（有世界范围的大神解题思路，你说看不看……）**

    - 举个力扣的 例子
    - 一题多解（最优（时间空间小）解法）
    - 寻求反馈（成长的关键）：leetcode 的 solution 和 discussion.

5. 现在就开始动手！

1.6.2 数据结构

1. 两大类：线性结构（顺序表、链表、栈、队列）和非线性结构（树、图、二（多）维数据、广义表）

1.6.3 Array

1.6.4 Stack and Queue

1.6.5 PriorityQueue

1.6.6 LinkedList(single/double)

1.6.7 Tree/Binary Tree

树

二叉树

- 二叉搜索树：左子树小于根结点，右子树大于根结点。

1.6.8 HashTable

1.6.9 Disjoint Set

1.6.10 Trie

1.6.11 BloomFilter

1.6.12 LRU Cache

1.7 算法（Algorithm）

1.7.1 General Coding

1.7.2 In-order/pre-order/Post-order traversal

1.7.3 Greedy

1.7.4 Recursion/Backtrace

1.7.5 Breadth-first search

1.7.6 Depth-first search

1.7.7 Divide and Conquer

1.7.8 Dynamic Programming

1.7.9 Binary search

1.7.10 Graph

1.8 设计模式

update: 2022-08-13

注解:

- 设计模式比较抽象，都是思想层面的，仅仅是归纳总结的软件设计思路，不是万灵药，切不可生搬硬套。设计模式的目的就是为了是软件系统各功能更灵活、架构清晰、协同性好、可维护性、可扩展性等只要能达到这些目的你的设计模式（抽象类）就是值得点赞的!

- 还是要有一定的实际的工程的经验再看设计模式，这样容易产生共鸣，不然它可能不是心目的样子，看着它没感觉。

1.8.1 资源推荐

1. 《大话设计模式》
2. 《设计模式-可复用 ** 面向对象 ** 软件的基础》（经典）
3. 设计模式 视频课程

** Gong of Four 23(GOF 23) 四个人总结了 23 种设计原则模式 **

注解： 推荐两本书，可以看看。第二本书比较全面讲了 23 种模式，可作为日常完善所需，总体划分了三大类型：创建型、结构型、行为型。

1.8.2 知识框架图

1.8.3 初识设计模式

什么是设计模式？

1. **定义**
    - 对用来在特定场景下解决一般设计的问题的类和相互通信的对象的描述。
2. **模式四要素**
    1. 模式名称 (P)。
    2. 问题 (I)。
    3. 解决方案（S)。
    4. 效果（C)。

OOP（面向对象）七大原则

注解： 作为架构设计的七大原则

---

1. 开闭原则：对扩展开放，对修改关闭。
2. 里氏替换原则：继承必须确保超类所拥有的性质在子类中仍然成立。
3. 依赖倒置原则：要面向接口编程，不要面向实现编程。
4. 单一职责原则：控制类的粒度大小，将对象解耦，提高其内聚性。
5. 接口隔离原则：要为各个类建立它们需要的专用接口。
6. 迪米特法则：只与你的直接朋友交谈，不跟“陌生人”说话。
7. 合成复用原则：尽量先使用组合或者聚合等关联关系来实现，其次才考虑使用继承关系来实现。

1.8.4 创建型（5）

工厂方法

抽象工厂

1. 超级工厂

生成器

原型

单件

1.8.5 结构型（7）

适配器

桥接

组合

装饰

外观

享元

代理

1.8.6 行为型（11）

职责链

命令

解释器

迭代器

中介者

备忘录

观察者

状态

策略

模板方法

访问者

1.9 计算机网络系列

-

1.10 关系数据库-MySQL

注解： 更新日期：2022-08-13 对数据库的基本概念、深层次知识点做个笔记，如索引及其原理、分布式数据库等。

1.10.1 书籍推荐

1. **《数据库系统教程 (第 2 版上)/下次》电子工业出版社王能斌老师**
    - 全书共分 6 篇 23 章，分上、下两册。2 篇分在上册，共有 12 章；第 3 ~ 6 篇分在下册，共有 11 章。其中，上册于 2008 年全面修订。

1.10.2 基础知识

数据模型（基础核心）

小技巧： 不同的形式来描述现实世界数据的方法，如层次、结构等

1. 数据
2. 数据模型
3. **数据模式**
    - 对表结构的描述：DDL，如 schema。3 级模式：物理（磁盘）、概念（逻辑）、视图（外，最终展示）模式
    - 视图，可经过映射计算查看，如：查看公司的证券代码（基本信息表）和公司对应的非财务信息点（非财务表）
    - 具备了数据独立性，应用和数据分离

1.10.3 设计规约

建表

1. 表，字段命名。
    1. 必须使用小写字母，禁用保留字（如 desc order group 等），是否概念可用 is_[field_name] 方式。
    2. 表名命名。业务名称 _ 表的用途
    3. 库名。尽量和应用名保持一致。

2. 数据类型。
3. 分库分表。单表行数据量大于 500 万行，大小超过 2G。

索引

注解： 提高查询效率。

---

1. 索引的特性：持久性，有序性。
2. 索引分类：
    1. 存储方式：聚簇和非聚簇
    2. 数据约束：主键索引，唯一索引，非唯一索引
    3. 索引列数量：单列索引，组合索引
    4. innodb 可以创建的索引：主键索引，唯一索引，普通索引

SQL 与 ORM 的映射

1.11 非关系数据库–Redis

1.12 Docker

注解： 更新日期：2020-03-23

1.12.1 书籍推荐

- 访问 docker_practice 下载。
- 《Docker 开发实践》
- **其他**
    - 访问 菜鸟教程 查看。
    - 访问 Docker 学习路线图 。

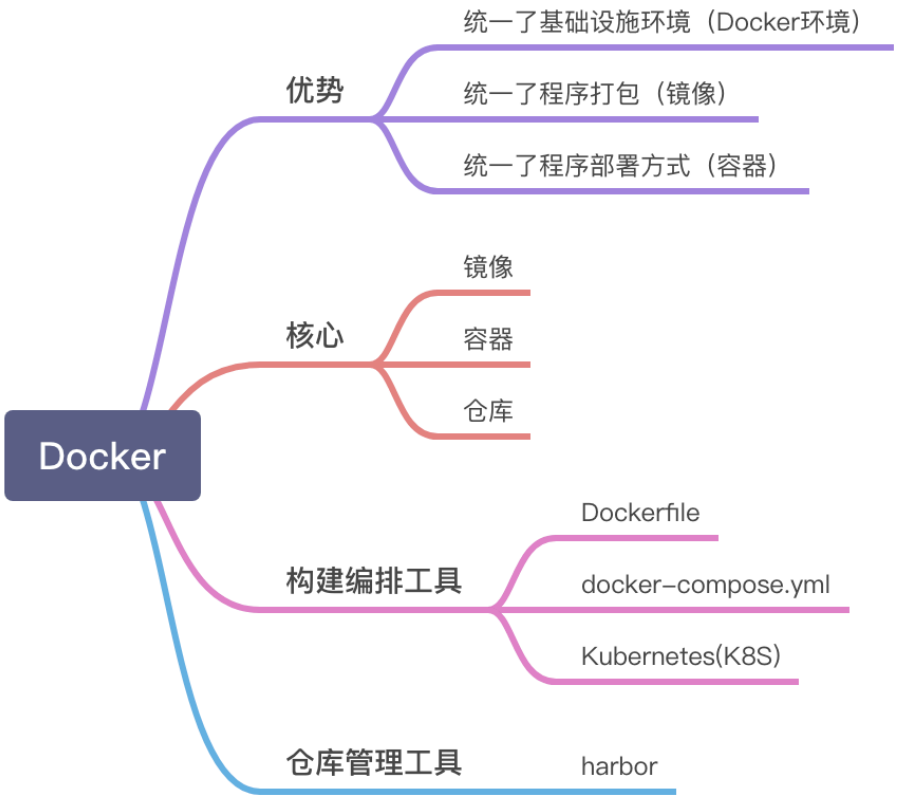
1.12.2 初识 Docker

注解： 总的来说，Docker 属于运维部署领域。如果是应用者就是熟练的在合适的场景下操作相应的命令；如果是进一步研究其底层原理，那需要的基础就要多些了，比如虚拟化技术。

1. Docker 就是开源的容器引擎，诞生于 2013 年。最形象的比喻就是集装箱，各自自成体系，互不影响，可任意放在不同的平台。
2. Docker 是容器的一种，容器是一种 * 轻量级 (占用空间小) 的虚拟技术 *，相对应的，虚拟机（VMware、VirtualBox 等）就是重量级的虚拟技术。
3. Docker 是虚拟化的是操作系统，而虚拟机虚拟化的是硬件。
4. **最切实的价值是什么？**
    - 项目开发有三大基础环境：开发环境（程序员）、测试环境（测试人员）、生产环境（运维人员）。
    - 要保持三者环境的统一 Docker 的优势不明觉厉。举个例子，小王，程序员一枚，在自己电脑上优雅的完成了业务开发。怀着如释重负的心情将代码推送至测试环境，然后休息，进行吃鸡游戏……。突然，测试人员小李发来消息，小王呀，你昨天推的代码不 work 呀，小王说：不可能！我本地非常 work 的，你再仔细看看吧……；小李说：我这真没问题。
    - 你看看是不是吃鸡的心情不好了！问题究竟在哪里呢？原来呀小王本地开发环境使用的 Python3.7，但是小李所用的测试环境是 Python3.6，造成了有些功能不 work。如果我们有了 Docker，就可以代码 + 环境保持一致，不再造成上面的尴尬！
5. 可移植。MAC、Win、Linux 等平台。
6. 相互隔离。
7. 开销低。

Docker 是一种容器技术，解决软件跨环境迁移的问题！

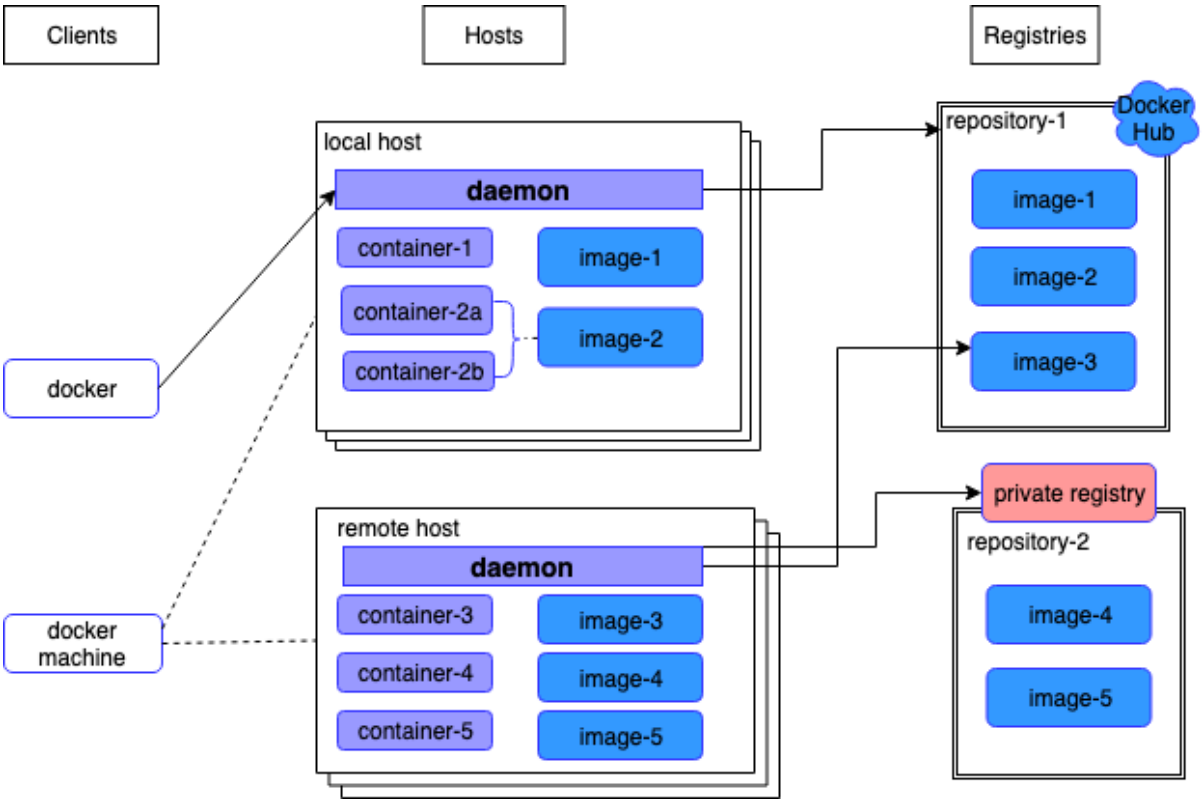
1.12.3 图述 Docker



1.12.4 Docker 架构（组件）

重要： Docker 采用 C/S 架构。Docker 客户端（Client）可以通过命令行形式或 API 来与提供 Docker 服务（Server）的守护进程进行通信。

基本架构图



核心组件

重要： Docker 共包含三大核心组件：**镜像（image）**、**容器（container）**、**仓库（Repository）**。镜像和容器可以类比面向对象编程中类和实例的关系，image->class、container->instance。仓库可类别代码控制中心，负责存储和共享用户的镜像。

---

1. **Image**
    - 一个只读的静态模板。存储容器所需要的环境和应用的执行代码。相当于是一个 root 文件系统，比如官方镜像 Python:3.9 就包含了完整的一套 Python3.9 最小系统的 root 文件系统。
    - 采用分层机制。
2. **Container**
    - 一个运行时的环境。镜像是静态的定义，容器是镜像运行时的实体。
    - 容器就相当于集装箱，不关心里面装什么，所有应用都有统一的生命周期：创建、启动、删除、暂停、重启等。

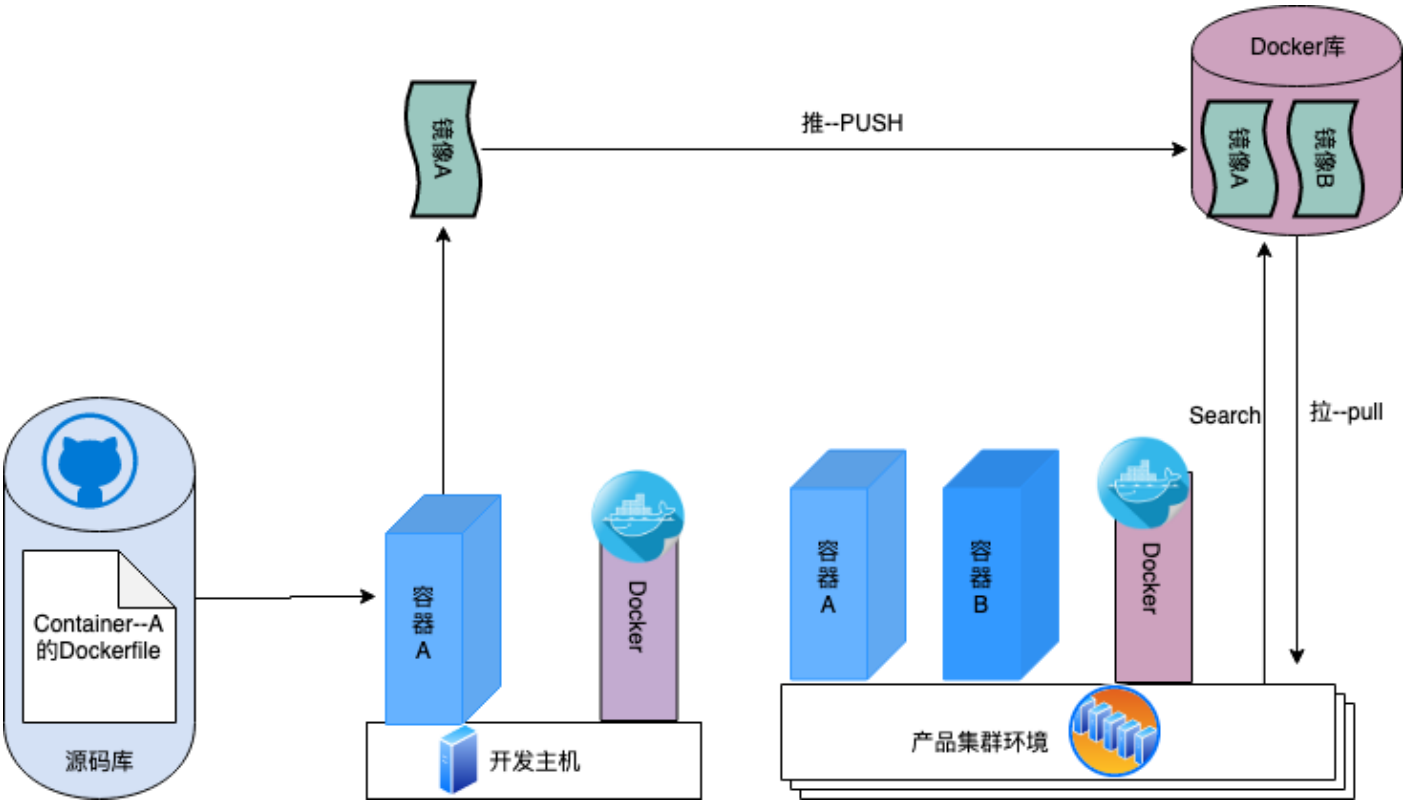
- 容器也不在乎自己所处的平台。本机、虚拟机、服务器等都可相互移植，对于前面提到的部署都是非常适合的。

3. **Repository**

- Docker 采用注册服务器来存储和共享用户的镜像。
- 注册服务器分为公共和私有两种。公共就是官方的 Docker Hub，私有就是自己注册一个 Docker Hub 账号建立自己的私有仓库，便于小范围内的共享。

通过 Docker 开发和部署的流程图

注解： 利用下图能更好的理解 Docker 在日常开发、部署中的应用流程和三大组件。



流程概述

- **开发主机上**

1. 创建容器 A，创建方法可以手动也可通过 Dockerfile 文件自动构建。

小技巧： Dockerfile 文件后续会讲，这也是最常用的一种构建容器方式。

---

2. 容器 A 必须基于镜像来创建。镜像 A 就是容器的静态形式，容器是镜像的动态形式。

3. 将容器 A 保存为镜像 A，然后推送到 Docker 库进行共享

- **集群环境上**

1. 在 Docker 库中搜索所需镜像 A，并将其拉取到本地。
2. 拉取后在本地就可以运行容器 A 了。
3. 在集群环境中可以运行很多容器，彼此相互独立、互不影响。

1.12.5 安装 Docker（MAC）并注册国内镜像加速器

小技巧: MAC 系统可以直接安装桌面版 Docker，社区版就够用了。关于镜像加速器推荐使用国内阿里云镜像加速器，配置也比较容易，配置后再使用 docker 镜像就比较快了。官网下载太慢吗？推荐前往：http://get.daocloud.io/。

安装

1. 点击 下载 docker 桌面版。
2. 查看 docker 版本，验证是否安装成功

```
$ docker -v
Docker version 19.03.5, build 633a0ea
```

配置阿里云镜像加速器

step-1 使用阿里云或支付宝等账号登录 阿里云镜像加速器 网站。

step-2 登录后就能看到针对不同操作系统的操作步骤了。如下图所示：

1.12.6 Docker 常用命令

注解： 本章是 docker 知识的重点，基本都是命令。跟着命令敲起你的小键盘吧。

Docker 服务（Daemon）相关命令

注解： mac 系统下直接点击客户端就启动了 docker 服务，非常简单。使用 Mac 系统，就可以跳过这部分内容了。为了使本笔记不失一般性，这里使用 CentOS 进行相关命令演示。

休息一下：你们公司更倾向于选择什么操作系统作为服务器呢？centos、RH、Linux？why？知乎 上有一篇帖子讨论了这个问题。

1. **启动 docker 服务**

```
$ systemctl start docker
```

2. **停止 docker 服务**

```
$ systemctl stop docker
```

3. **重启 docker 服务**

```
$ systemctl restart docker
```

4. **查看 docker 服务状态**

```
$ systemctl status docker
  docker.service - Docker Application Container Engine
Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor␣
↪preset: disabled)
Active: active (running) since 四 2019-12-12 10:06:56 CST; 2 months 24 days ago
Docs: https://docs.docker.com
```

5. **开机启动 docker 服务**

```
$ systemctl enable docker
```

Docker 镜像（Image）相关命令

1. **查看**

小技巧:

- docker images -q 查看所有镜像 ID
- docker iamges 查看所有镜像信息

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             ␣
↪ SIZE
python              3.8                 f88b2f81f83a        9 days ago          ␣
↪ 933MB
nginx               latest              2073e0bcb60e        4 weeks ago         ␣
↪ 127MB
ubuntu              14.04               6e4f1fe62ff1        2 months ago        ␣
↪ 197MB
```

注解: 可以看到，执行命令后列出了已创建（可能你还没有镜像，列表就为空）的镜像。下面针对表头做一个说明。

- REPOSITORY：仓库名称
- TAG：版本号，默认为 latest
- IMAGE ID：镜像唯一标识
- CREATED ： 创建时间
- SIZE ： 镜像所占的虚拟大小

2. **搜索**

小技巧:

- docker search [name]

```
$ docker search mysql
NAME                                DESCRIPTION                                     ␣
↪    STARS               OFFICIAL  (是否官方)            AUTOMATED
mysql                               MySQL is a widely used, open-source relation…
   9196                [OK]
mariadb                             MariaDB is a community-developed fork of MyS…
   3274                [OK]
mysql/mysql-server                  Optimized MySQL Server Docker images. Create…
   679                 [OK]
centos/mysql-57-centos7             MySQL 5.7 SQL database server                   ␣
↪    70
```

搜索是联网进行的，列出可用的镜像。官方镜像搜索网站，可以查看下有没有自己想要的版本。

3. **拉取（下载）**

小技巧:

- docker pull [name]:[tag]
- 不写 tag，则默认为 latest
- 访问 Docker Hub 镜像网站，可以了解更多关于的版本信息。

```
$ docker pull mysql:5.6
5.6: Pulling from library/mysql
6d28e14ab8c8: Pull complete
dda15103a86a: Pull complete
55971d75ab8c: Pull complete
f1d4ea32020b: Pull complete
61420072af91: Pull complete
30862a48418b: Pull complete
c6c2ee3a9a57: Pull complete
0f4efadb31df: Pull complete
dd931017b211: Pull complete
488a86083079: Pull complete
921d4bdabca2: Pull complete
Digest: sha256:a72a05bcf3914c902070765a506b1c8c17c06400258e7b574965763099dee9e1
Status: Downloaded newer image for mysql:5.6
docker.io/library/mysql:5.6
```

上面的拉取镜像过程就体现了分层。

4. **删除**

小技巧:

- **单个删除 docker rmi image-id/[name]:[tag]**
  - rmi。rm 就是删除，i 参数指的就是镜像。可以指定一个或多个镜像名称或者镜像的 ID，多个镜像之间可以使用空格隔开。
- **删除本次所有镜像: docker rmi *docker images -q***
  - docker images -q 列出所有镜像的 ID

```
$ docker rmi c8078e
Untagged: mysql:5.6
Untagged:␣
↪mysql@sha256:a72a05bcf3914c902070765a506b1c8c17c06400258e7b574965763099dee9e1
Deleted:␣
↪sha256:c8078e8ab06d8dabd6c30cffb03951fa035d85f75c19a83ace29b01cb3ecd272
```

警告:

- **如果不能删除成功，可能是因为这个镜像正在被容器使用。**
  - 可以使用 -f 参数强制删除。
  - 也可以先移除正在使用该镜像的容器后再删除。

docker 容器（container）相关命令

1. 查看

小技巧:

- **docker ps**
  - 查看正在开启的容器
- **docker ps -a**
  - 查看所有创建的容器列表

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ␣
↪      STATUS              PORTS               NAMES

$ docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED          ␣
↪           STATUS                      PORTS                                     ␣
↪            NAMES
3c7e127ff4ae        nginx:v3            "/bin/bash"              29␣
↪minutes ago      Exited (0) 25 minutes ago                                        ␣
↪                   web_server
```

2. 创建

小技巧:

- docker run -i -t –name=container_name image_name:tag /bin/bash
- **docker run -i -d –name=container_name image_name:tag /bin/bash**
  - -i -d[t] 可以合并为-id[t]。d 标志位表示创建后台容器。
- **退出容器：执行 exit 命令。**

– 退出后容器将关闭

```
$ docker run -it --name=web_server nginx:v3 /bin/bash

root@3c7e127ff4ae:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr

$ docker run -id --name=app_server  nginx:v3  /bin/bash

4b19f6042d9739a3dba3eccd93d4404259883ecf0f6402232124357914835b30
```

3. 进入

小技巧: docker exec -it [容器名称] /bin/bash

```
$ docker exec -it app_server /bin/bash
root@4b19f6042d97:/#
$ exit
$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             ␣
↪     STATUS              PORTS               NAMES
4b19f6042d97        nginx:v3            "/bin/bash"         4 minutes ago       ␣
↪     Up 4 minutes        80/tcp              app_server
```

4. **启动**
   - docker start [app_server]
5. **停止**
   - docker stop [app_server]
6. **删除**
   - **docker rm app_server**
     – 删除单个
   - **docker rm ‘docker ps -aq‘**
     – 删除所有
     – q 标志表示只列出容器 ID，不列出其他信息。

注解: ‘符号是键盘 table 上方的键位

7. **查看容器信息**
    - docker inspect [app_server]

1.12.7 Docker 容器的数据卷

小技巧: 主要探讨容器中的应用数据管理相关话题。如数据如何保存、外部如何使用数据等。

概念及作用

概念

1. 数据卷是宿主机中的一个目录或文件。
2. 容器目录（文件）和数据卷目录（文件）绑定后，双方修改会立即同步。
3. 一个数据卷可被多个容器挂载；一个容器也可挂载多个数据卷。

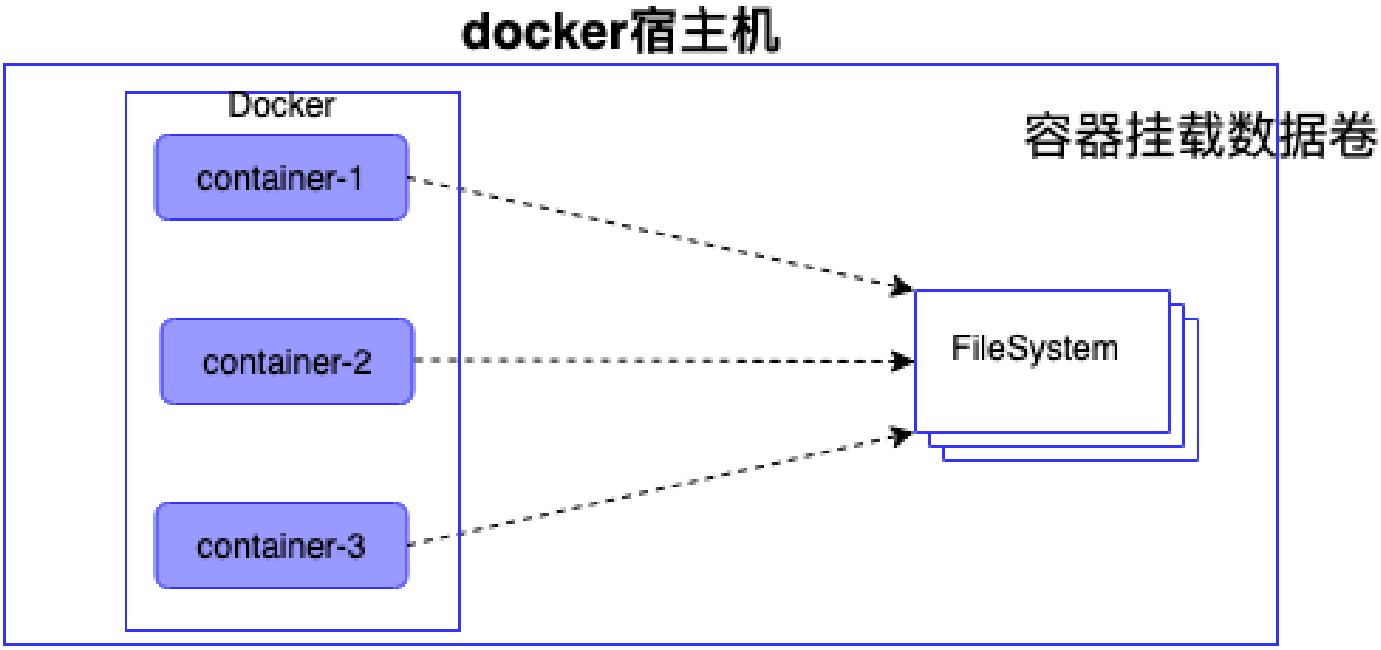
作用

1. 可持久化保存容器数据。
2. 实现外部机器和容器间接通信。
3. 容器之间进行数据交换。

配置数据卷

小技巧:

1. **创建容器时，使用-v 参数**
    - docker run ……-v 宿主机目录（文件）：容器内部目录（文件）
    - 目录不存在时，会自动创建。
    - 目录是绝对路径。
    - 可以挂载多个数据卷。



1. 挂载单个数据卷：将本机的 host_data 目录挂载到容器的 container_data 下

```
$ docker run -it --name=c1 -v /Users/hanghangli/Desktop/host_data:/root/
↪container_data  nginx:v3
# 进入容器
$docker exec -it c1 /bin/bash
$root@2c651df94731:/# cd root/
# 可以看到在容器内已经有了挂载目录
$root@2c651df94731:~# ls
container_data
```

2. 一个容器挂载多个数据卷：将本机的 data_1.txt、data_2.txt 文件挂载到容器的 container_data_1.txt、container_data_2.txt

```
$ docker run -it --name=c2 \
-v /Users/hanghangli/Desktop/data_1.txt:/root/container_data_1.txt \
-v /Users/hanghangli/Desktop/data_1.txt:/root/container_data_2.txt \
nginx:v3
# 进入容器
$ docker exec -it c2 /bin/bash
$ ls root/
# 可以看到在容器内已经有了挂载的两个文件
container_data_1.txt  container_data_2.txt
$ cat container_data_1.txt
```

3. 多个容器挂载一个数据卷。c3 与 c4 容器挂载一个 config.ini.txt 文件

```
$ docker run -it --name=c3 \
-v /Users/hanghangli/Desktop/config.ini.txt:/root/container_config.ini.txt.
↪txt \
nginx:v3
$ docker run -it --name=c4 \
-v /Users/hanghangli/Desktop/config.ini.txt:/root/container_config.ini.txt␣
↪\
nginx:v3
# 现在修改一下 config.ini.txt 文件内容并查看下容器的数据卷是否同步了修改。
# 先看下 c3 容器
$ docker exec -it c3 /bin/bash
$ root@d8b63fe631cb:~# ls
    container_config.ini.txt.txt
$ root@d8b63fe631cb:~# cat container_config.ini.txt.txt
    我修改了宿主机的配置文件。
$ root@d8b63fe631cb:~# exit
# 再看下 c4 容器
$ docker exec -it c4 /bin/bash
$ root@cfb85d4cb3c4:/# cat root/container_config.ini.txt
    我修改了宿主机的配置文件。
```

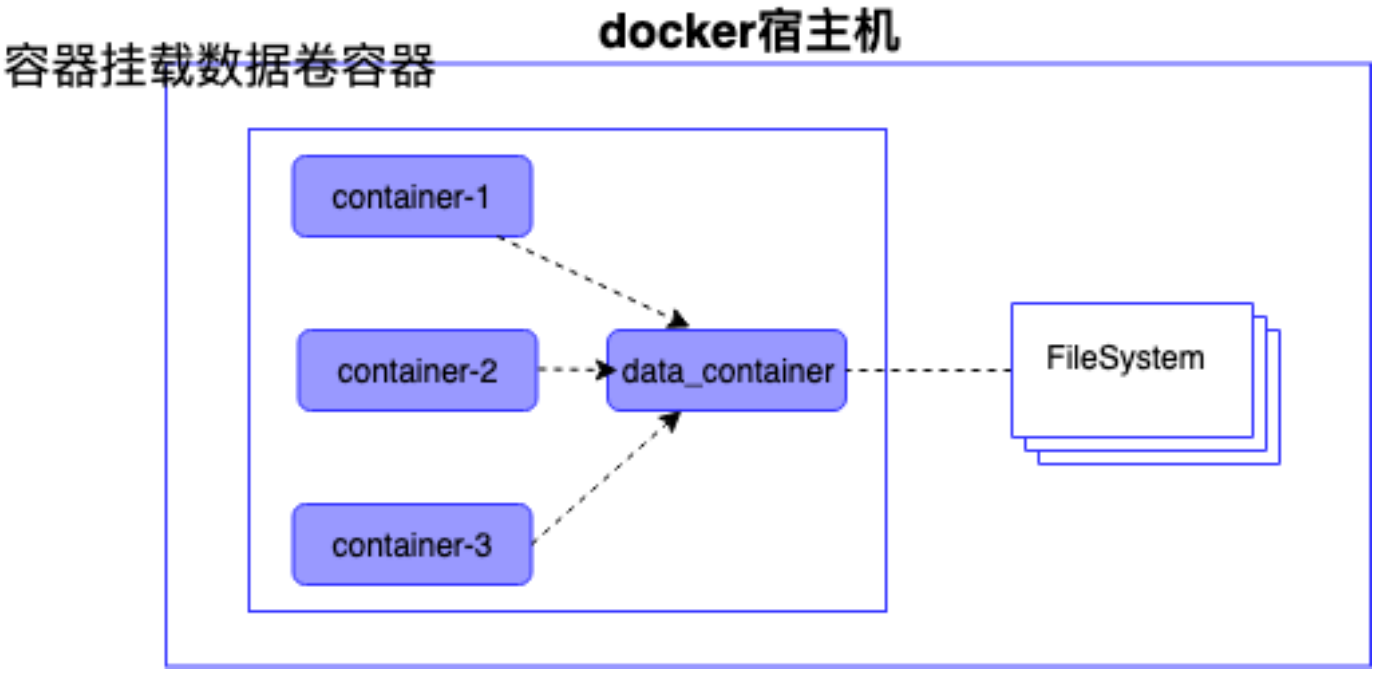
配置数据卷容器

小技巧:

- 使用场景：的时，并不想指定挂载的宿主机的目录，只想实现容器与容器之间的数据共享。
- 上面的方法是给每个容器挂载本地数据卷，这样在容器比较少的情况下是一个好方法。但当我们的容器很多且都有挂载数据卷的需求，上面的方式就显得不够高效和友好。
- 我们可以考虑专门做个挂载数据卷的容器，让它专门负责数据卷挂载，其他容器直接挂载这个数据卷容器即可。这样就增加了可扩展行和可维护性!

无论数据卷容器停止还是删除都不会影响其他容器对于数据卷的使用!

- 容器之间共享一些持续更新的数据，最简单的方式是使用数据卷容器



- **创建数据卷容器 test_1**

    1. docker run -it –name=test_1 -v /volume image:tag /bin/bash

- **挂载数据卷 test_1 给容器 test_2、test_3**

    2. docker run -it –name=test_2 –volumes-from test_1 image:tag /bin/bash
    3. docker run -it –name=test_3 –volumes-from test_1 image:tag /bin/bash

```
# 创建数据卷容器 会自动分配一个目录
$ docker run -it --name=test_1 -v /volume nginx  /bin/bash

# 挂载 test_1 到 test_2
$ docker run -it --name=test_2 --volumes-from test_1 nginx  /bin/bash
# 挂载 test_1 到 test_3
$ docker run -it --name=test_3 --volumes-from test_1 nginx  /bin/bash

# 我们可以测试数据同步情况，我在 test_1 容器/volume 目录新建一个 config.ini，看下 test_2 和
test_3 下是否会出现呢？
$ root@75fb3393fb19:/volume# echo "hello,Docker" >> config.ini
# 在 test_2 下的 volume 目录中查看写入内容
$ docker exec -it test_2 /bin/bash
$ root@95025edc8a00:/# cat volume/config.ini
    hello,Docker
# 类似的 test_3 下也会出现的，自己看下吧，聪明的你看到了吗？
```

1.12.8 Docker 使用案例（应用部署实战）

注解:

- 下面就进入 Docker 的在我们开发中的实际应用了，让我们一点点感受它带来的便利吧。加油，老铁们!
- **如果抽象出来部署操，可分为以下几步:**
  - 搜索需要安装的软件（镜像）。如 mysql 的版本。
  - 从仓库获取镜像。从私有或公共仓库获取。
  - 创建容器。创建容器的方式可以是命令行也可以使用 Dockerfile 文件来 build。
  - 完成。

MySQL 部署

1. **目标** 实现在 Docker 中部署 MySQL，并通过外部客户端操作该容器中的数据库。

   小技巧: 思考：外部如何访问容器内的数据库呢？解决方案：引入端口映射方法。

2. **过程**

   - **搜索 mysql（可省略步骤）** docker search mysql:5.6
   - **拉取 mysql** docker pull mysql:5.6
   - **创建容器**

```
# 在本地创建一个数据库目录并进入。
$ mkdir mysql
$ cd ~/mysql
# $PWD 表示当前目录路径
$ docker run -id \
-p 3307:3306 \
--name mysql_container \
-v $PWD/conf:/etc/mysql/conf.d \
-v $PWD/logs:/logs \
-v $PWD/data:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=pass \
mysql:5.6
e39f78f46f1585225bab52499ad4d81032bc35d52972341503f47bdd1992d277
$ docker ps
```

(下页继续)

(续上页)

```
CONTAINER ID        IMAGE               COMMAND                  ␣
↪CREATED             STATUS              PORTS                     NAMES
e39f78f46f15        mysql:5.6           "docker-entrypoint.sh"   7␣
↪seconds ago        Up 5 seconds        0.0.0.0:3307->3306/tcp   mysql_
↪container
```

- **参数说明**
  - -p 3307:3306 端口映射。将容器 mysql 的 3306 映射到主机的 3307。
  - -v $PWD/conf:/etc/mysql/conf.d 挂载数据库配置数据卷。将本地（刚才创建的 mysql 目录）当前目录的 conf 挂载到容器/etc/mysql/conf.d
  - -v $PWD/logs:/logs 挂载日志数据卷。将本地当前目录的 logs 挂载到容器/logs。
  - -v $PWD/data:/var/lib/mysql 挂载数据数据卷。将本地当前目录的 data 挂载到容器/var/lib/mysql
  - -e MYSQL_ROOT_PASSWORD=pass 初始化 root 用户的密码

- **操作容器中的 mysql**

```
# 进入容器
$ docker exec -it mysql_container /bin/bash
# 在容器中登录 mysql
root@e39f78f46f15:/#  mysql -uroot -ppass
mysql>
# 下面我们可以在本地用任意客户端登录 mysql，注意端口填写 3307 就行。登录后可以常见一个数据库和表，再进入容器就会看到刚才创建的表了。
# 到此，mysql 的容器化就完成啦。

# 其他命令：mysql 容器端口的映射信息
$ docker port mysql_container 3306
    0.0.0.0:3307
```

Tomcat 部署

1. **目标**
   - 实现在 Docker 中部署 Tomcat，并通过本地浏览器访问网页，确定服务器是否正常工作。
2. **创建过程**
   - **拉取镜像**

```
# 在本地创建一个数据库目录并进入。
$ mkdir tomcat
$ cd tomcat
# $PWD 表示当前目录路径
$ docker pull tomcat
Using default tag: latest
latest: Pulling from library/tomcat
50e431f79093: Pull complete
dd8c6d374ea5: Pull complete
c85513200d84: Pull complete
55769680e827: Pull complete
e27ce2095ec2: Pull complete
5943eea6cb7c: Pull complete
3ed8ceae72a6: Pull complete
91d1e510d72b: Pull complete
98ce65c663bc: Pull complete
27d4ac9d012a: Pull complete
Digest:␣
↪sha256:2c90303e910d7d5323935b6dc4f8ba59cc1ec99cf1b71fd6ca5158835cffdc9c
Status: Downloaded newer image for tomcat:latest
```

- **创建 Tomcat 容器**

```
# 在本地创建一个数据库目录并进入。
$ mkdir tomcat
$ cd tomcat
# $PWD 表示当前目录路径

$ docker run -id --name=tomcat \
    -p 8080:8080 \
    -v $PWD:/usr/local/tomcat/webapps \
    tomcat

    5949c2cfd5fe1d4d3395996d22804d08e7e5debc8255d032fd12ab1f1d54be4f
```

- **使用容器**
  - **在本地机器创建的 tomcat 目录下**
    * mkdir my_app
    * echo '<h1>hello,Docker!</h1>' > my_app/index.html
    * 访问：http://0.0.0.0:8080/my_app/index.html

* 大功告成!

Nginx 部署

- **拉取镜像**

```
$ docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
68ced04f60ab: Pull complete
28252775b295: Pull complete
a616aa3b0bf2: Pull complete
Digest: sha256:2539d4344dd18e1df02be842ffc435f8e1f699cfc55516e2cf2cb16b7a9aea0b
Status: Downloaded newer image for nginx:latest
```

- **创建容器并测试**

```
# 准备工作
$ mkdir nginx
$ cd nginx
$ mkdir conf html logs
$ vim conf/nginx.conf
    #user  nobody;
    worker_processes  1;

    #error_log  logs/error.log;
    #error_log  logs/error.log  notice;
    #error_log  logs/error.log  info;

    #pid        logs/nginx.pid;

    events {
        worker_connections  1024;
    }

    http {
        include       mime.types;
        default_type  application/octet-stream;

        #log_format  main  '$remote_addr - $remote_user [$time_local] "$request
↪" '
        # '$status $body_bytes_sent "$http_referer" '
```

(下页继续)

(续上页)

```
        # '"$http_user_agent" "$http_x_forwarded_for"';
        #access_log  logs/access.log  main;
        sendfile        on;
        #tcp_nopush     on;
        #keepalive_timeout  0;
        keepalive_timeout  65;
        #gzip  on;
    include /etc/nginx/conf.d/*.conf;
    }
$ echo '<h1>Hello, Nginx. </h1>' > html/index.html
$ docker run -id --name=my_nginx \
    -p 80:80 \
    -v $PWD/conf/nginx.conf:/etc/nginx/nginx.conf \
    -v $PWD/logs:/var/log/nginx \
    -v $PWD/html:/usr/share/nginx/html \
    nginx
$ 访问 0.0.0.0
    Hello, Nginx.
```

Redis 部署（一个 key-value 存储系统）

- **目标**
  - 创建 Redis 容器，并使用本地机器进行访问。
- **拉取镜像**
  - docker pull reids
- 创建容器

```
$ docker run -id --name=my_redis \
-p 6379:6379 \
redis
# 内部先测试下
$ docker exec -it my_redis /bin/bash
$ root@679f5de7ab12:/data# redis-cli
$ 27.0.0.1:6379> set name 'test'
    OK
$ 127.0.0.1:6379> get name
    "test"
```

- **外部连接测试**

- **mac 系统下安装 Redis**
  - brew install redis

> **小技巧:** 如果下载太慢可更换 brew 的仓库源，可参考：https://www.jianshu.com/p/8a2ac505ff3e

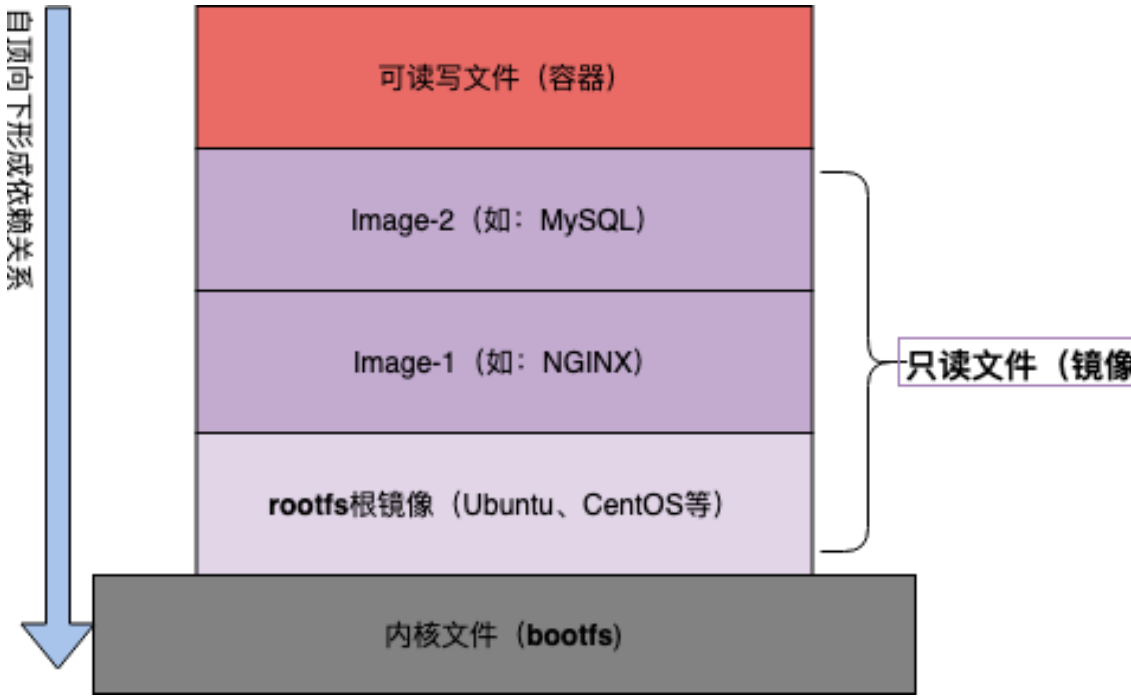
- 连接测试

```
$ redis-cli -h 0.0.0.0 -p 6379
$ 0.0.0.0:6379> get name
    "test"
```

1.12.9 使用 Dockerfile 制作镜像

镜像原理

- 镜像是由特殊的文件系统叠加而成。
- 采用分层的文件系统，通过在只读文件（镜像）上增加可读写层（容器）的形式来改变镜像。
- **Docker 镜像结构图**



  - **bootfs。启动文件系统镜像，复用了宿主机的文件系统。**
    - 这也就解释了我们单独下载 Ubuntu 可能就是好几个 G 大小，但是利用 docker 就是几百兆的大小。

  - rootfs。根文件系统，也成为根镜像，一般就是一个操作系统。
  - **Image-1、Image-2。这些就是我们用户的镜像，可以不断叠加，下层为父镜像。**
    * 这也就解释了如果单独下载 MySQL 可能也就几十兆的大小，但使用 docker 就要几百兆的大小，反而变大了。究其原因，就是只读文件存在依赖的关系，叠加后变大了。
  - 可读写文件。这一层就是容器了，当我们基于镜像进行容器启动时，就会在最顶层加载一个可读写的文件系统作为容器。
  - 修改完后就可以提交新的镜像（制作一个新镜像）了。

-
- 创建新的镜像本质上也就是对已有的镜像文件集合进行增、删、改的操作。
- 这种叠加的方式有利于实现镜像共享、增加可扩展性、减少磁盘空间使用。

Dockerfile 的概念和作用

镜像制作方法

1. 容器转镜像（不常用）

```
docker commit [id] [Image-name]:[tag]
```

```
docker save -o [压缩文件名称] [Image-name]:[tag]
```

```
docker load -i [压缩文件名称]
```

> **注意：** 若原有镜像含有挂载文件，则 commit 时不会将其挂载到新制作的镜像。

- **操作步骤**

```
$ docker ps -a

CONTAINER ID        IMAGE               COMMAND                  ␣
↪CREATED             STATUS                  PORTS                ␣
↪      NAMES
679f5de7ab12        redis               "docker-entrypoint.sh"   ␣
↪12 hours ago        Up 2 hours              0.0.0.0:6379->6379/
↪tcp   my_redis
```

（下页继续）

(续上页)

```
# 制作一个 Redis 镜像名为 make_redis
$ docker commit 679f5de7 make_redis

sha256:c5f603178b0cc95aeb04d3e674060d1268541d361748852d73d7eba652f0c6d3
# 查看镜像
$ docker images

REPOSITORY          TAG                 IMAGE ID            ␣
↪CREATED             SIZE
make_redis          latest              c5f603178b0c        8␣
↪hours ago          98.21 MB
redis               latest              f0453552d7f2        32␣
↪hours ago         98.21 MB
# 打包镜像
$ docker save -o make-redis.tar make_redis
# 为了还原镜像，我们先删除存在的
$ docker rmi c5f6031
Untagged: make_redis:latest
Deleted:␣
↪sha256:c5f603178b0cc95aeb04d3e674060d1268541d361748852d73d7eba652f0c6d3
Deleted:␣
↪sha256:88106bcdc3c35ca6ea7bdb8e7dd06d91c921a328587a0f72b87628ffea654945
# 加载制作的新镜像
$ docker load -i make-redis.tar
    936d71f61caa: Loading layer 3.584 kB/3.584 kB
    Loaded image: make_redis:latest
# 查看是否成功还原
$ docker images
    REPOSITORY          TAG                 IMAGE ID            ␣
↪CREATED             SIZE
    make_redis          latest              c5f603178b0c        8␣
↪hours ago          98.21 MB
    redis               latest              f0453552d7f2        32␣
↪hours ago         98.21 MB
```

2. **Dockerfile（常用方法）**

   - Dockerfile 是一个文本文件，包含了一行行的指令。
   - 不一定叫 Dockerfile 名称，可根据实际需求来。如 nginx_dockerfile
   - 每一行指令构建一层镜像，基于基础镜像，最终构建出一个新的镜像。

- **Dockerfile 关键字，可参考：https://docs.docker.com/develop/develop-images/dockerfile_best-**

  - FROM：父镜像
  - RUN：执行命令，[“command-1” ， “command-2”]
  - CMD: 容器启动命令
  - COPY：复制文件
  - WORKDIR：工作目录。指定容器内的工作目录
  - ADD：添加文件

案例

案例一之 CentOS 安装 Vim

- **任务**

  制作一个 centos7 镜像。保证：

  - 默认登录路径为/usr;
  - 可以使用 vim

- **步骤**

```
$ docker pull centos:7
$ docker run -it --name=c1 centos:7
# 编写 Dockerfile 文件
$ vim Dockerfile
    # 定义基础镜像
    FROM centos:7
    # 作者信息
    MAINTAINER Mason
    # 执行操作
    RUN yum install -y vim
    # 设置工作目录
    WORKDIR /usr
    # 设置启动命令
    CMD ["/bin/bash"]
# . 表示的是当前目录，不要忘记写了
$ docker build -f Dockerfile -t test_centos:1 .
    Sending build context to Docker daemon 222.6 MB
    Step 1 : FROM centos:7
```

(下页继续)

(续上页)

```
    ---> 5e35e350aded
    Step 2 : MAINTAINER Mason
    ---> Using cache
    ---> efe73e688fd0
    Step 3 : RUN yum install -y vim
    ---> Using cache
    ---> d79eb09c16dc
    Step 4 : WORKDIR /usr
    ---> Using cache
    ---> 0da2b1bca082
    Step 5 : CMD /bin/bash
    ---> Running in 697bd23b374a
    ---> a71548cf9f8e
$ docker run -it --name c2 test_centos:1
$ [root@aa24050fdab5 usr]# vim test.txt

# 完成所有任务
```

案例二之发布 Flask 应用

- **任务** 定义 Dockerfile，发布一个 Hello,Flask 版的 Flask-Web 项目
- 步骤

```
# 创建并进入 flask-web 目录
$ cd flask-web
# 写一个启动脚本 app.py
$ vim app.py
    #app.py
    from flask import Flask

    app = Flask(__name__)

    @app.route('/')
    def index():
        return '<h1> hello, Flask </h1>'

    if __name__ == '__main__':
        app.run(host='0.0.0.0', port=5000, debug=True)
# 编写 Dockerfile 文件，名称为 flask_dockerfile
```

(下页继续)

(续上页)

```
$ vim flask_dockefile
    FROM centos:7
    # 作者信息
    MAINTAINER Mason
    # 执行操作
    RUN yum install -y python3 && yum -y install epel-release && yum␣
↪install -y python-pip
    RUN pip install flask -i https://pypi.tuna.tsinghua.edu.cn/simple
    # 设置工作目录
    WORKDIR /web_app
    # 设置默认命令
    ENTRYPOINT [ "python" ]
    # 设置启动命令
    CMD ["app.py"]
# 开始构建镜像
    $ docker build -f flask_dockerfile -t flask-web:1.0 .
    # 创建应用容器
    $ docker run -id \
    -p 5001:5000 \
    --name web_app \
    -v $PWD/flask-web:/web_app \
    flask-web:1.0

# 浏览器访问容器应用
0.0.0.0:5001
# 完成
```

1.12.10 docker-compose（服务编排技术）

服务编排概念

- 目的：将一系列的容器创建自动化
- docker-compose 是一个工具。能够编排多容器分布式部署，以指令集的形式管理容器化应用的开发周期，如构建、启动、停止。
- **基本过程**
  - 利用 Dockerfile 文件定义运行环境镜像。
  - **使用 docker-compose.yml 定义组成应用的各服务。**
    * 前后依赖关系

        * 容器数量等

    - 运行 docker-compose up 启动应用。

日常命令

1. 修改 docker-compose.yml 文件后重启容器

```
$ docker-compose down
# -d 参数为后台运行
$ docker-compose up -d
```

安装（Mac 系统）

- Mac 系统安装 Docker 后会包含这个工具，直接查看下版本

```
$ docker-compose version
docker-compose version 1.8.1, build 878cff1
```

实例-发布简单 Nginx+Flask-web 应用

注解:

- 要发布一个完整的项目就存在很多组件服务，如数据库、反向代理等，逐个启动太过麻烦了，我们可以使用 docker-compose 工具统一进行定义并启动。
- 这个案例就要编排反向代理及 flask 两个服务应用。

```
# 创建工作目录
$ mkdir ~/docker-compose
$ cd ~/docker-compose
# 编写 yml 文件。名称是固定的。注意 yaml 缩进
$ vim docker-compose.yml
```

```
# 这个版本号需要与 docker 版本对应: https://docs.docker.com/compose/compose-file/
    version: '2'
    services:
        # 服务名称可随便取
        Nginx:
            image: nginx  # 镜像名称
```

(下页继续)

(续上页)

```
            ports: # 端口映射，启动服务后，直接访问 80 端口即可（监听 6000，6000
会映射为 5000）
                - 80:6000
            links: # 和另一个服务关联
                - flask-web
            volumes: # 配置文件
                - ./nginx/conf.d:/etc/nginx/conf.d
        flask-web: # 服务应用名称，可随便取
            image: flask-web:1.0 # 镜像名称，采用上一个实例的
            volumes: # 应用目录
                - ../flask-web/:/web_app
            ports: # 端口映射
                - "6000:5000"
```

```
$ mkdir -p ./nginx/conf.d
# Nginx 相关配置
$ vim ./nginx/conf.d/server.conf
```

```
server {
    listen 6000; # 监听 6000 端口
    access_log off;



    location / {
    proxy_pass http://flask-web:5000
    }
}
```

```
# 回到 docker-compose 目录，开始启动服务。
$ docker-compose up # 后面-d 参数将会后台运行，这里就不加了
    Starting dockercompose_flask-web_1
    Starting dockercompose_Nginx_1
    Attaching to dockercompose_flask-web_1, dockercompose_Nginx_1
```

```
若正常启动，则直接在浏览器访问
    http://0.0.0.0
```

1.12.11 搭建私有仓库

注解: 搭建自己的仓库。上传本地镜像并从私有拉取镜像

创建私有仓库

- **拉取私有仓库镜像**

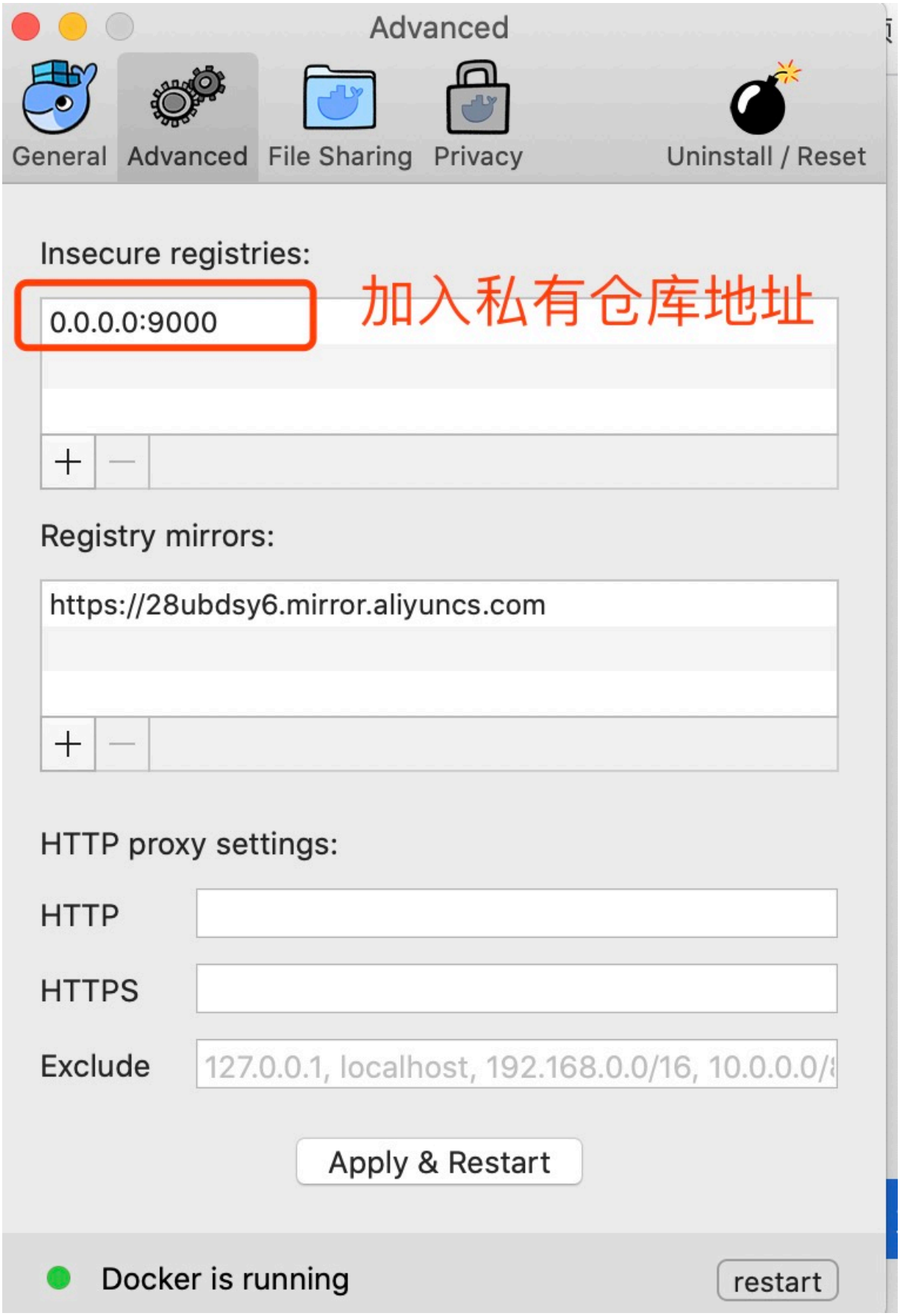
```
$ docker pull registry:2
```

- **启动仓库**

```
$ docker run -id --name=my_hub -p 9000:5000 registry:2
```

- **浏览器访问** http://127.0.0.1:9000/v2/_catalog 可以看到是空的
- **修改 daemon.json**

小技巧: Mac 系统可直接客户端的配置


Advanced
General
Advanced
File Sharing
Privacy
Uninstall / Reset
Insecure registries:
0.0.0.0:9000
加入私有仓库地址
Registry mirrors:
https://28ubdsy6.mirror.aliyuncs.com
HTTP proxy settings:
HTTP
HTTPS
Exclude
127.0.0.1, localhost, 192.168.0.0/16, 10.0.0.0/
Apply & Restart
Docker is running
restart

传并拉镜像

- 传 CentOs

```
# 先重新打个标记
$ docker tag centos:7 0.0.0.0:9000/centos:7.1
$ docker images
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
0.0.0.0:9000/centos   7.1                 5e35e350aded        4 months ago        203 MB
# 开始传入仓库
$ docker push 0.0.0.0:9000/centos:7.1
    The push refers to a repository [0.0.0.0:9000/centos]
    77b174a6a187: Pushed
    7.1: digest: sha256:934aed62ee9ee05733d233c679a576a1d21aee98ef809493260686aad2bd3e0a␣
↪size: 529
```

```
# 浏览器访问，即可看到上传的镜像
http://127.0.0.1:9000/v2/_catalog
{
    repositories: ["centos"]
}
```

注意:

- **报错 1:**
  - The push refers to a repository [0.0.0.0:9000/centos]
  - Get https://0.0.0.0:9000/v1/_ping: http: server gave HTTP response to HTTPS client
- **解决**
  - 首先，确保添加了安全组 insecure-registries：0.0.0.0:9000
  - 重启 shell 终端及 docker 服务
- **报错 2:**
  - Get https://0.0.0.0:9000/v2/: dial tcp https://0.0.0.0:9000: connect: no route to host
- **解决**
  - 重新启动容器：docker start my_hub

- 拉取镜像

```
$ docker pull 0.0.0.0:9000/centos:7.1
Digest: sha256:934aed62ee9ee05733d233c679a576a1d21aee98ef809493260686aad2bd3e0a
Status: Image is up to date for 0.0.0.0:9000/centos:7.1
```

1.12.12 结束

至此，关于 docker 的基础知识及命令已完成，后续会继续网络、容器集群管理、终端运维管理相关的学习。

1.13 Kubernetes（K8S）

1.13.1 推荐

1. 访问 Kubernetes 中文社区 | 中文文档 查看。

1.14 服务器系列

注解:

- 创建时间：2020-11-07
- 更新时间：2020-11-07
- 服务器环境下的基础知识、结合工作场景的常用命令

1.14.1 Linux 基础知识

1.14.2 Linux 常用命令

- 使用命令本地上传代码/文件到服务器?
- 将服务器的文件传输到本地的命令?
- 在服务器上测试模块代码的步骤?

1.15 分布式与微服务

1.15.1 中间件集合

注解:

- 创建于 2021 年 1 月 24 日
- 更新于 2022 年 8 月 13 日

消息队列

基本知识

1. **消息队列主要应用的业务场景：解耦、错峰和流控、广播、最终一致性。**
    - 消息队列（MQ）是一种能实现生产者到消费者单向通信的通信模型，一般来说是指实现这个模型的中间件。
    - 如：RabbitMQ、ActiveMQ、Kafka 等

2. **为什么需要消息队列？** 比如下面的场景 A->B，A 请求端、B 是后端处理端，如果没有消息队列，A 的每一次操作都要同时触发 B 端，意味着 A 的直接对接者是 B，对于大流量的场景这样会对接口造成很大的压力。A->[队列]->B/C/D……，A 间接服务于 B，B 何时处理 A 的请求就由 B 自身来决定了。而且接入者可以是 C/D 等。
3. **RPC 和 MQ 的区别？**
    - **RPC（Remote Procedure Call）远程过程调用，主要解决远程通信间的问题，不需要了解底层网络的通**
        - 如阿里的 dubbo。
    - 区别：对于实时性高的场景优先考虑 RPC，对于流量控制、解耦等优先考虑 MQ。

RabbitMQ

1.15.2 分布式与微服务系列

1.16 机器学习基础

注解:

- 创建日期：2020-03-11
- 更新日期：2020-09-12
- 完成日期：进行中……

1.16.1 书籍推荐

1. 《机器学习》周志华图文并茂，提纲挈领
2. 《统计学习方法第二版》李航偏数学理论，硬核干货

1.17 机器学习应用

1.17.1 常见应用领域

1.17.2 常见算法回顾

1.17.3 模型评价指标

1. 准确率
2. 精确率
3. 召回率
4. **精确率 + 召回率 =F1 score**

$$F1 - measure = \frac{2 * precesion * recall}{(precesion + recall)}$$

1.17.4 常用建模工具

1. scikit-learn

1.17.5 实际问题建模思路流程

1. 数据处理
2. 特征工程
3. 模型选择
4. 调参。寻找最优超参数，交叉验证
5. 模型分析与模型融合

1.18 深度学习

注解： 深度学习是机器学习的一个分支，是实现机器学习的技术之一。所以能学深度学习肯定是具备一定的机器学习基础知识了。

1.18.1 书籍推荐

1. **《神经网络与深度学习》邱锡鹏著**
    - https://nndl.github.io
2. **《动手学深度学习》李沐**
    - 特点：通俗易懂，理论 + 实践、具备可操作性、对于了解原理及应用有帮助
    - 是基于不同框架写的书，可根据自己需求选择
3. **《深度学习》（花书）**
    - 特点：书厚、关于数学基础、机器学习、深度学习都有涉及、还分了深度学习不同的领域，如 NLP 等
    - 可当做工具书

1.18.2 深度学习基本点

1. **深度？学习？**
    - 深度就是模型复杂。
    - 学习学的是特征表达（表示）。也就是学习的目的是什么。
    - 总结：用相对深的模型来学习特征表示。
2. **属于的人工智能学派：连接主义**。
    - 连接主义是认知科学领域的信息处理方式。相对应的还有符号主义学派。
    - 连接主义主要由神经元（人工神经网络）进行信息处理，其缺点是可解释性差（黑箱）。

    注解： Q1：人工神经网络的本质是什么？

    A：数学角度：基于的理论的函数逼近论。表达不同维数空间之间的一个函数映射。（总之它不神秘，也不是真正的智能）

    Q2：人工神经网络和深度学习的关系？

A：人工神经网络是实现深度学习的手段或者方法之一。TA 俩不等价！

3. 属于机器学习的一个分支。

注解： Q：深度学习、机器学习的最大区别是什么？

A：深度学习可以基于大数据自动进行特征学习，完成端到端的学习。相反，机器学习大多数需要人为进行特征标记，学习结果受人为因素影响大。

4. 深度学习相对于机器学习模型更为复杂。其理论上个世纪就出现了，随着算力、算法、数据三要素的具备深度学习再次被唤醒。

1.19 自然语言理解

注解： 更新日期：2020 年 04 月 01 日

1.19.1 书籍推荐

1. **《Speech and Language Processing, 2nd Edition》**
    - 内容全面，覆盖面广
2. 《统计自然语言处理》宗成庆
3. 《信息检索导论》
4. 《语言本能-人类语言进化的奥秘》

1.19.2 什么是自然语言理解？

自然语言？

1. 和编程语言相对的语言称为自然语言。

自然语言理解？

1. 目的：使计算机理解人类的自然语言。
2. 本质：结构预测的过程。如输出一句话的句法结构或者语义结构。
3. **从无结构化序列预测有结构的语义**。

- 词性标注、命名实体识别、依存分析、句法分析、指代消解等任务。

4. **数据驱动的自然语言理解**
    - 深度学习技术的突破
5. **语义表示（核心）**
    - **one-hot （0/1）也称独热编码**
        - 表示简单，但局限性很大，如相似度计算。
    - **分布式语义表示（空间表示）**
        - 目前最主要的表示形式。
        - 1986 年提出的方法。

自然语言的特点

1. **歧义性多**。 ` 南京市/长江大桥。南京市长/江大桥。` - 关键目标：消除歧义性。
2. **递归性** ` 你好/不好意思。你/好不好意思。`
3. **主观性** ` 仔细体会这两句：别回了。我没事，你忙你的。`
4. **社会性**
    - 敬语、语言协调等

难点

1. **语言的语义表示**
    - 世界（社会/客观）、心智（人/主观）、语言三者相互影响。
2. **将人类的知识融入到语义表示的过程中**。
    - 人类知识相当于给足了上下文信息
3. **多模态复杂语境的理解（说话的表情、手势动作、场景等）**
    - 欢迎/新老/师生/前来/参观！
    - 欢迎/新老师/生前/来参观！

未来

1. 句子消歧。
2. 引入知识。如知识图谱。
3. 多级的跨句子建模。

4. 生成句子更符合当下对话场景。
5. 理解并创作。

自然语言理解交叉科学

1. 计算机科学
2. 脑科学
3. 语言学
4. 语言哲学
5. 心理学、社会学、认知学
6. 神经语言学、汉语言学

1.19.3 自然语言技术方向

- 基于规则驱动
- **基于数据驱动**
  - 统计学语言模型
  - 深度语言模型

1.19.4 NLP 国内外优秀学者及实验室

- Christopher Manning

1.20 CS224n-2019-课程笔记 by Chris Manning

小技巧:

- CS224n-2019. 课程源自斯坦福 CS course，2019 年发布的自然语言处理，算是 NLP 的经典吧，老爷子讲的也很风趣幽默。Ok, Hello, everyone! 一起来追剧吧。
- 题外话：课时并不多，所以暗示自己要尽量耐心地把每一节的知识搞懂（慢工出细活，理论知识很重要），自己思考的同时也要动手推导公式甚至编写代码（我就是这么弄得）来刺激大脑理解，难受一阵会发现知识理解很深刻，再回顾此前的知识，豁然开朗！

1.20.1 一些说明和资源

- 课程主页
- 本笔记涉及代码部分将使用 pipenv 构建虚拟环境。(推荐)
- **课后作业使用 PyTocrh 框架使用**。
    - HW1-HW5.
    - FP,which is QA .
- B 站资源
- 笔记参考 1
- 笔记参考 2

1.20.2 词向量 (Word Vectors)

Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 1: Introduction and Word Vectors

自然语言和词义

1. **自然语言**

- 你永远无法确定任何单词对他人意味着什么。(中文这个情况就更普遍啦)
- 写作是另一件让人类变得强大的事情，这实现了知识的传播和共享。

2. **语言的意义**

- 通过一个词或句子等来表达概念
- 人们通过文字或声音信号等来表达思想、想法
- 在写作、艺术中表达含义

```
一般通过下面这种语言方式进行有意义的思考:
   signifier(symbol) signified(idea or thing) =denotational semantics
```

3. **语义计算**

- **常见方案的不足**
  - **类似 WordNet 一个面向语义的英语词典，包含上义词 (hypernyms)、同义词 (synonym sets)**
    * 没有考虑上下文，忽略一个词的细微差别
    * 不能及时更新。
    * Can't compute accurate word similarity
  - **传统 NLP 的做法。离散符号表示。one-hot，0-1 进行编码: Means one 1, the rest 0s**
    * 向量大小就是词汇表的大小 (很多无用的信息)
    * 无法计算相似度。如下例两个词向量是正交的，点积为 0.

```
motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
```

- **提取新方案**
  - Could try to rely on WordNet's list of synonyms to get similarity?
  - learn to encode similarity in the vectors themselves (学习词自身的编码信息)

4. **通过上下文表示词**

- 分布式语义: 一个词的含义往往是由附近高频出现的词决定的。
- word 出现在文本中，这个 Word 周围会有由词的集合组成的 Context 出现。这个上下文是固定一个窗口 size 的。
- **我们可以使用存在 Word 的大量 语料 来学习其向量表示。比如学习 "中国科学院" 词 (实际中会学习每个**

1. 先向获得 2009 年度国家最高科学技术奖的 **中国科学院**院士、复旦大学数学研究所名誉所长谷超豪和
2. 院士、复旦大学数学研究所名誉所长谷超豪和 **中国科学院**院士、中国航天科技集团公司高级技术顾
3. 大国”向“造船强国”迈进。由 **中国科学院**和上海市政府共同建设的上海同步辐射光源工
4. 丽；河南卓越工程管理有限公司董事长邬敏 **中国科学院**研究生院教授杨佳十人“全国三八红旗手

Word Vectors(词向量)

- 根据一个的词的上下文，来为词构建密集的向量，以使得该向量与出现在类似上下文中的词相似
- **引出词向量，也称词嵌入或词表示**。
  - word vectors are sometimes called **word embeddings** or word representations.
  - They are a **distributed representation**.
  - 例如“中国”这个词经过训练后的词向量为：

$$= \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word2Vector 介绍

注解: Word2vec (Mikolov et al. 2013) 是一种学习词向量的 框架。

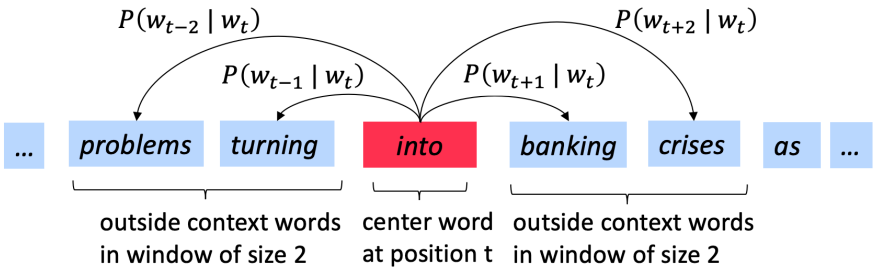
主要思想

1. 我们有个比较大的文本数据集。
2. 文本中的每个词通过一个固定长度的词向量表示。
3. 扫描文本中每一个位置 **t** 所表示的词, 其中有一个中心词 **c** 和上下文词 **o** 。
4. 通过 c 和 o 的词向量的相似性，计算在给定 c, 即中心词来计算 o, 即上下文的概率。反之亦然。

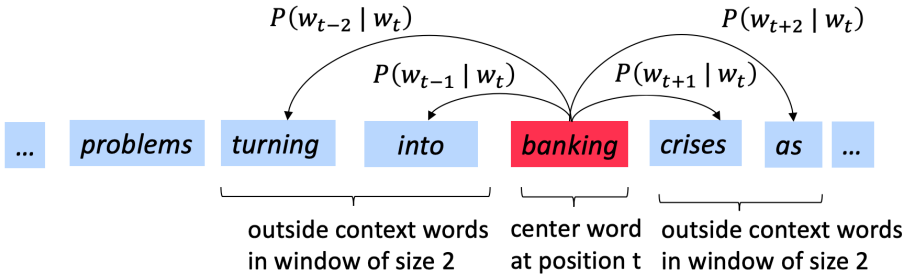
5. 不断调整词向量来最大化上面提到的概率。

- 举例如下

- Example windows and process for computing $P(w_{t+j} \mid w_t)$



- Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2Vector 目标函数

1. 思路（后面要说的 Skip-grams 模型）

   在每个位置 t （t = 1，……，T），给定一个中心词 w_j 和一段固定长度的窗口 m，预测上下文中每个单词的概率。

$$Likelihood = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} | w_t; \theta)$$

$$\theta$$

- 目标函数 $J(\theta)$ （也称为 **代价或损失函数**），是一个负对数似然：

$$J(\theta) = -\frac{1}{T}logL(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} P(w_{t+j}|w_t;\theta)$$

Q1：如何计算 $P(w_{t+j}|w_t;\theta)$?

A1：每个词 w 用两个向量表示

- 当 w 是中心词时用向量 v_w 表示
- 当 w 是上下文词时用向量 u_w 表示

那么对于一个中心词 c 和上下文词 o 可用如下形式表示

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)}$$

其中，$u_o^T v_c$ 目的是为了对整个词汇表进行标准化。

- **softmax function** softmax 函数作用是将任意标量 x_i 。

$$softmax(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(u_w^T v_c)} = p_i$$

  - “max” 对比较大的 x_i 映射比较大的概率
  - ” soft” 对那些小的 x_i 也会给予一定概率
  - 这是一种常见的操作，如深度学习

小技巧:

- 利用对数的特性将目标函数转换为对数求和，减少计算的复杂度。
- 最小化目标函数　最大化预测的准确率

- 通过不断的优化参数最小化误差来训练模型。
- **为了训练模型，需要计算所有向量的梯度**
  - θ 用一个很长的向量表示所有模型的参数。
  - **每个单词有个两个向量**。
    * Why two vectors? àEasier optimization. Average both at the end.
  - 利用不断移动的梯度来优化这些模型的参数。

梯度计算推导

下面开始推导 $P(w_{t+j}|w_t;\theta)$: 对 v_c 求偏微分

$$\begin{aligned}
\frac{\partial}{\partial v_c} logP(o|c) &= \frac{\partial}{\partial v_c} log\frac{exp(u_o^T v_c)}{\sum_{w\in V} exp(u_w^T v_c)} \\
&= \frac{\partial}{\partial v_c}\left(\log\exp(u_o^T v_c) - \log\sum_{w\in V}\exp(u_w^T v_c)\right) \\
&= \frac{\partial}{\partial v_c}\left(u_o^T v_c - \log\sum_{w\in V}\exp(u_w^T v_c)\right) \\
&= u_o - \frac{\sum_{w\in V}\exp(u_w^T v_c)u_w}{\sum_{w\in V}\exp(u_w^T v_c)} \\
&= u_o - \sum_{w\in V}\frac{\exp(u_w^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}u_w \\
&= u_o - \sum_{w\in V} P(w|c)u_w
\end{aligned}$$

u_o $\sum_{w\in V} P(w|c)u_w$

0.

还有对 u_o 的偏微过程，大家动手推导下，比较简单的。

小技巧: 补充一点边角知识，在上面的推导过程中用的到:

- 向量函数与其导数 $\frac{\partial Ax}{\partial x} = A^T, \frac{\partial x^T A}{\partial x} = A$
- 链式法则: $log' f[g(x)] = \frac{1}{f[g(x)]}g'(x)$

---

word2vector 的概览

前面提到的 Word2Vector 是一种学习词向量的框架（模型），它包含两个实现算法:

1. Skip-grams (SG) （课上讲的就类似这种）
   - 根据中心词周围的上下文单词来预测该词的词向量
2. Continuous Bag of Words (CBOW)
   - 根据中心词预测周围上下文的词的概率分布。

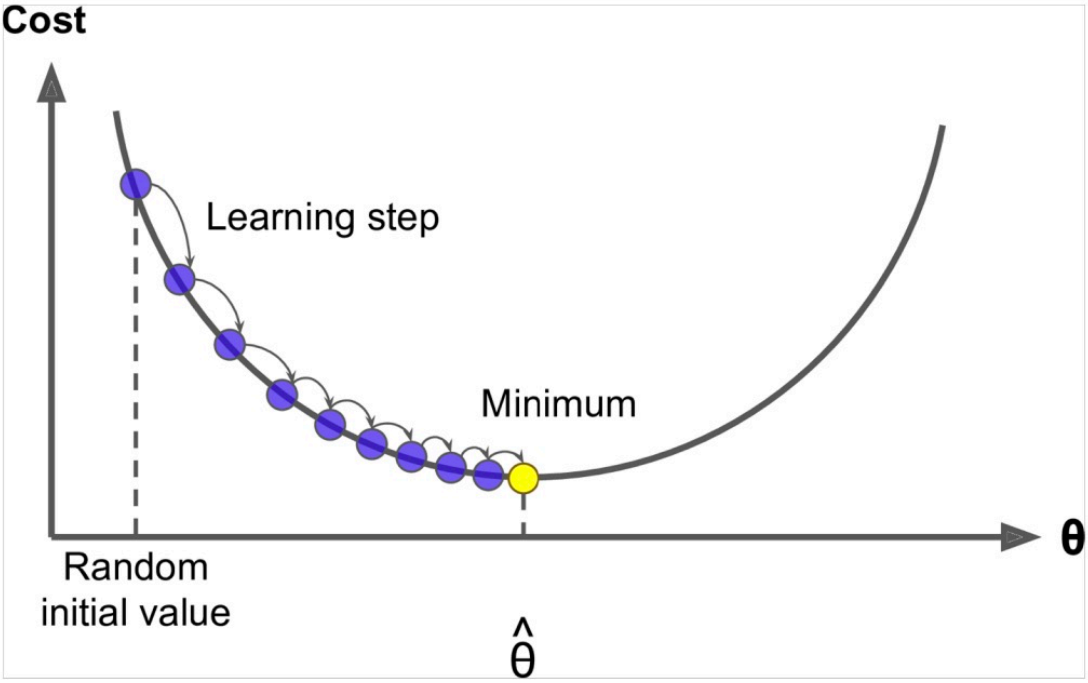
另外提到两个训练的方式:

1. negative sampling (比较简单的方式)
   - 通过抽取负样本来定义目标

2. hierarchical softmax
    - 通过使用一个树来计算所有词的概率来定义目标。

优化：梯度下降与随机梯度下降算法的要点

- **梯度下降（Gradient Descent，GD）**
    1. 最小化的目标（代价）函数 $J(\theta$
    2. 使用梯度下降算法去优化 $J(\theta$
    3. 对于当前 θ 采用一个合适的步长（学习率）不断重复计算 $J(\theta$ 的梯度，朝着负向梯度方向。



    4. 更新等式（矩阵）

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

$$\theta =$$

    5. **更新等式（单个参数）**

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- **随机梯度下降（Stochastic Gradient Descen, SGD）**
    - **目的** 进一步解决 $J(\theta$ 的训练效率（因为目标函数包含所有的参数，而且数据集一般都是很大的）问题：太慢了。
    - Repeatedly sample windows, and update after each one

小结

- 本节首先从语言的语义问题开始讲起，然后为了表示语义，引出了词向量概念，接着着重讲了 Word2Vector 框架、原理、算法推导等，最后简单提了下目标函数的优化的方式。
- **看完并梳理完本节知识，我产生了几个问题：**
  - 词向量提了好多次，那么每个词的词向量究竟是如何产生（计算）的呢？存在哪些方法？
  - **有几个点的原理还需进一步深入理解：**
    * 负采样、层次采样；区别和前后的优势在哪里？
    * SG、CBOW 算法的细节；本质区别和各自优势是什么？

1.20.3 词向量和语义

注解： 本节课后，我们就能够开始读一些关于词嵌入方面的论文了。

Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 2: Word Vectors,Word Senses, and Classifier Review

Review：word2vec

主要思想

- **这里以 skip-gram）模型为例**
    1. 遍历整个语料库的每个词，通过中心词向量预测周围的词向量
    2. 算法学到的词向量能用来计算词的相似度或语义等相关需求。

关于梯度计算

- GD。计算效率低，每次对所有样本进行梯度计算
- SGD。每次只对一个固定大小的样本窗口进行更新，效率较高。
- **梯度计算存在稀疏性（0 比较多）**
    - But in each window, we only have at most 2m + 1 words, so it is very sparse!
    - **解决方案：**
        * only update certain rows of full embedding matrices U and V. (使用稀疏矩阵仅更新稀疏性低的词向量矩阵 U 和 V)
        * you need to keep around a hash for word vectors（使用 hash 来更新，即 k-v，k 表示 word,v 表示其词向量）

基于负采样 (negative sample) 方法计算

1. **计算下列式子：**

$$P(o|c) = \frac{exp(u_o^T v_c)}{\sum_{w \in V} exp(u_w^T v_c)}$$

其中:math:‘ {sum_{win V} exp(u_w^Tv_c)}‘ 计算代价非常大（整个语料库计算），如何降低这一块的计算复杂度就是需要考虑的问题。

2. **负采样方法介绍**
    - 主要思想：train binary logistic regressions。除了对中心词窗口大小附近的上下文词取样以外 (即 true pairs)，还会随机抽取一些噪声和中心词配对（即 noise pairs）进行计算，而不是遍历整个词库。
    - 这个 **负**指的是噪声数据（无关的语料词 noise pairs）
3. 负采样计算细节

小技巧： 可参考论文 Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013) 第 3 页。

- 最大化下面的目标函数

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$$

$$J_t(\theta) = \log\sigma(u_o^T v_c) + \sum_{i=1}^{k} \mathbb{E}_{j\sim P(w)}[\log\sigma(-u_j^T v_c)]$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

  - 公式第一项表示最大化真实的中心词和其上下文词的概率；第二项是最小化负采样的噪声值（中心词及其上下文）的概率，j 表示负采样的样本，并以 P(w) 大小进行随机采样。

注解:

- P(w)，这里使用了 N 元统计模型且 N 取 1，即一元统计模型（unigram），表示每个词都和其它词独立，和它的上下文无关。每个位置上的词都是从多项分布独立生成的。
- 补充 N 元统计模型，N=2 时就为二元统计模型，即每个词和其前 1 个词有关。一般的，假设每个词 x_t 只依赖于其前面的 n−1 个词（n 阶马尔可夫性质）。
- 通过 N 元统计模型，我们可以计算一个序列的概率，从而判断该序列是否符合自然语言的语法和语义规则。
- 这个方法在构建词向量时最大的问题就是 **数据稀疏性**，大家可以思考下为什么？还能想到改进或其他更好的方法?

基于共现矩阵生成词向量

But why not capture co-occurrence counts directly? 1. 主要思想

- **有一个共现矩阵 x，其可选的粒度有两种：固定窗口大小的 window、文档级的 document**
  - **window 级别的共现矩阵：类似有 word2vector，利用每个词使用固定的窗口大小来获取其语法或语义信**
    - **例如，window_len = 1 的单词与单词同时出现的次数来产生基于窗口的 co-occurrence matrix**
      - 语料：I like deep learning. I like NLP. I enjoy flying. c

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

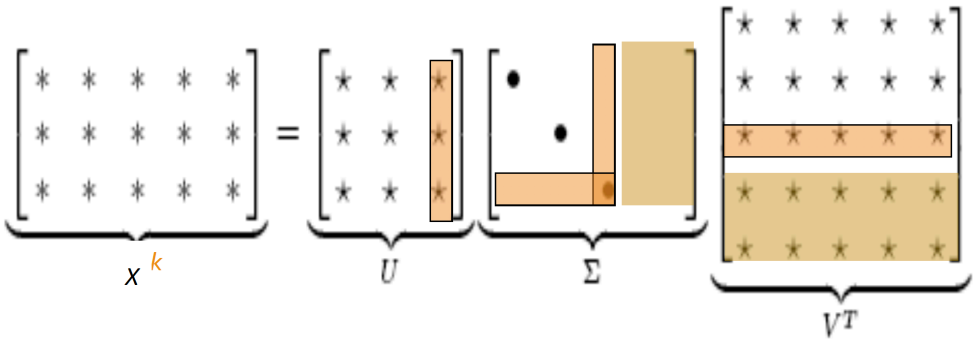
17

* 解释下上述矩阵的含义：按窗口为 1，同时出现的原则，I I 语料没有这样的表达即同时出现的次数为 0，I like 前两条都出现了，即为 2。
* 简单观察矩阵，就会发现很稀疏，存在大量的 0，而且随着语料库的增大，维数也会增大，这显然不是最佳方法，得想办法增加空间的利用率，目标就是稀疏变稠密，以免因稀疏性对下游任务造成影响。

– document 级别的共现矩阵：基本假设是文档若存在相关联，则其会出现同样的单词。一般使用 “Latent Semantic Analysis”（LSA）潜在语义分析方法进行矩阵的生成

2. 问题：怎么对共现矩阵进行降维呢?

- Singular Value Decomposition（SVD）奇异值分解
  – 将矩阵 X 分解为 $U\Sigma V^{\top}\Sigma$ 是对角阵，其主对角线的每个值都是奇异值；U 和 $V^{\top}$ 两个正交矩阵



注解:

1. SVD 就是将一个线性变换分解为两个线性变换，一个线性变换代表旋转，一个线

性变换代表拉伸。

2. 正交矩阵对应的变换是旋转变换，对角矩阵对应的变换是伸缩变换。
3. 这里我们可以再联想对比另外一个经典的降维算法 PCA。

3. Hacks to X

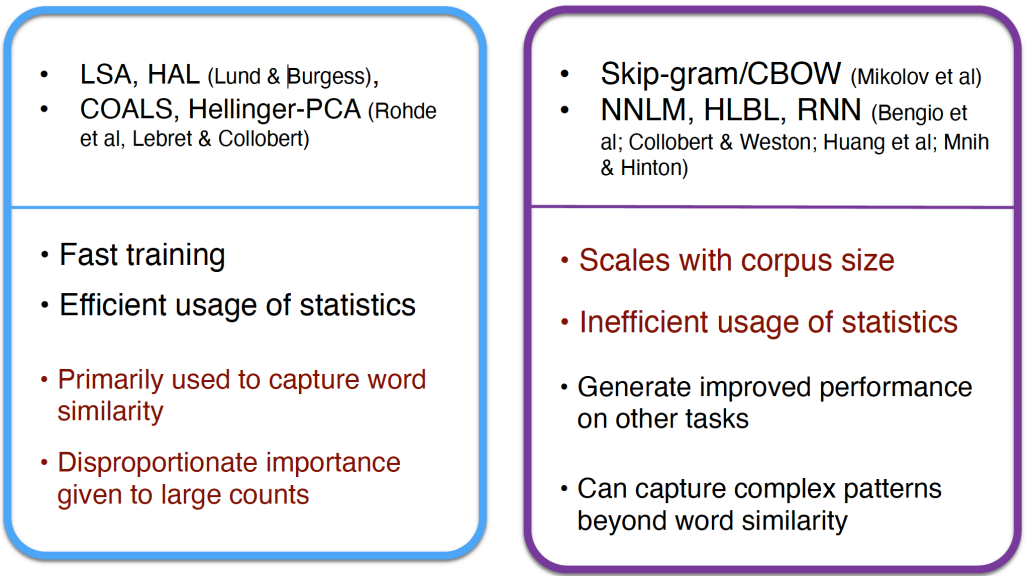
- **按比例缩放计数会有很大帮助。（怎么讲？）哦，就是对一些高频的功能性（has、the 等）词进行缩放，缩**
  - min(X,t), with t 100
  - 忽略这些词
- 定义一个 Ramped windows，统计距离很小（关联度高）的的词。（距离越小，代表关联度越高）
- 使用皮尔逊相关系数来替换直接计数方式，并设无关联值为 0。

注解： Pearson 相关系数是衡量向量相似度的一种方法。输出范围为-1 到 +1, 0 代表无相关性，负值为负相关，正值为正相关。

Glove 词向量模型

小技巧： 可参考阅读 Glove 主页

1. 基于计数和直接预测的比较



- 基于计数（统计理论）
- 直接预测（概率模型或神经网络）

到底哪一个方法是正法呢，还是说走向合作呢？答案是合作，相互借鉴才能发挥更大价值 (1+1>2)

2. **探索在向量间的差异中挖掘语义**

- 方向：能在共现矩阵的概率的比值中看出语义相关

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

- **解释下表所传递的信息：当需知道 ice 冰和 steam 气的关系时，可借助词 k：**
  - 当 k=solid，k 和 ice 近似，这时 ratio>>1；==> solid 与 ice 有关
  - 当 k=gas，k 和 steam 接近时，ratio<<1；==> gas 与 steam 有关
  - 当 k=water/fashion 等与 2 个词都不相关时，ratio 1。==> 之间无关

通俗的讲下这个比值：1 为阈值，当远大于 1 或远小于 1 就说明词之间有相关度的；当接近于 1 时证明无关

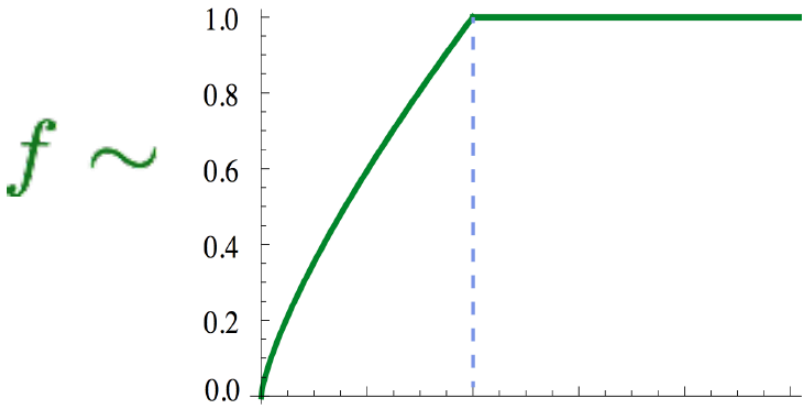
3. 如何表示共现矩阵的概率的比值呢?

- Log-bilinear model：$w_i \cdot w_j = logP(i|j)$
- 使用向量差值表示：$w_x \cdot (w_a - w_b) = log\frac{P(x|a)}{P(x|b)}$

4. **Glove 目标函数为：**

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- 其中 f(x) 如下图所示：



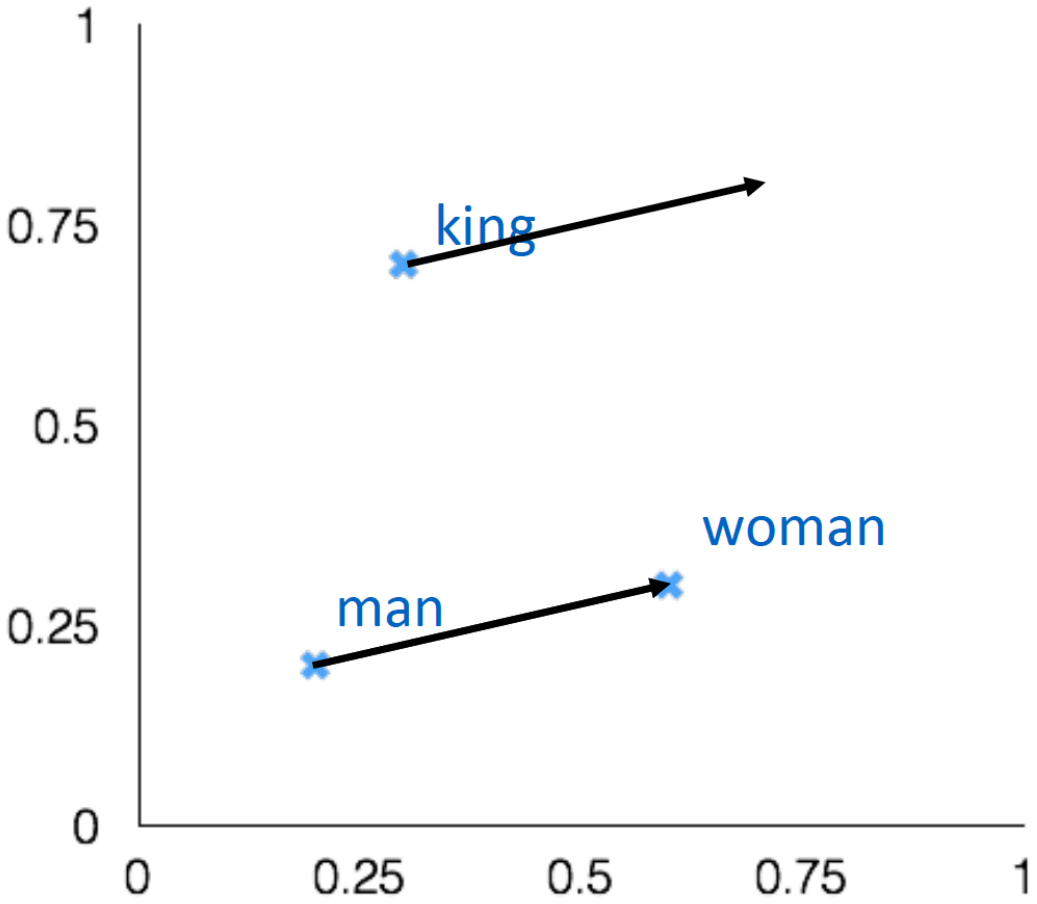
- X_{ij} 表示词 j 在词 i 的上下文中出现的次数。

5. **Glove 的优势**
    - 训练速度快
    - 可扩展到大语料库
    - 即时小预料库，其效果也不错

怎样评估词向量？

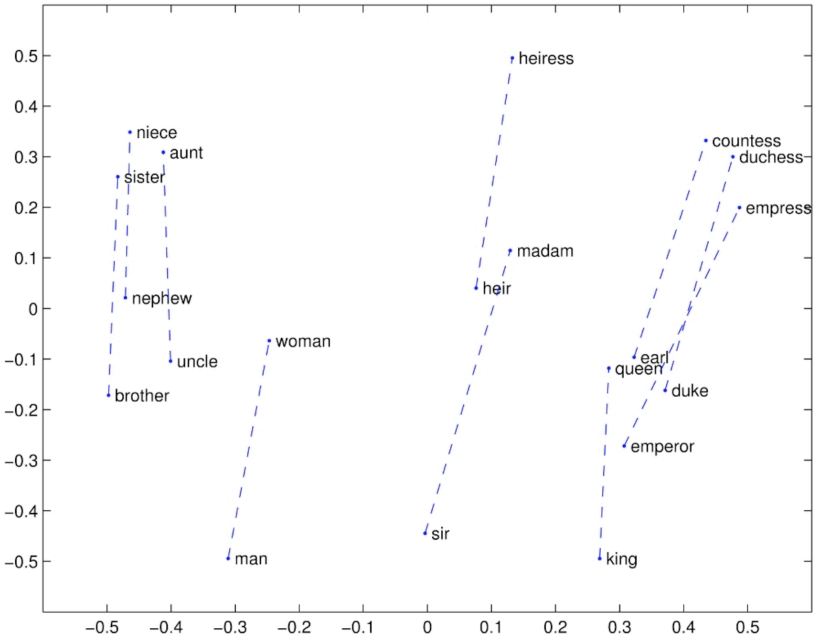
1. 问题：如何评估 NLP 的模型。
2. **主要从两个方面：内部和外部**
    - **内部（自身评估）**
        - 评估特定或中间子过程：速度快、有利于理解整个模型
    - **外部（将词向量应用到下游任务，如推荐、搜索、对话等系统中）**
        - 在真实场景下进行评估：消耗时间可能过长、不明确子系统是否有交互问题。
        - 如果用一个子系统替换另外一个子系统后能提高准确率，那这个模型就很棒了
        - 词向量应用到搜索、问答等领域来进行效果评估
3. **评估：词向量**
    - 通过计算余弦距离后并求和来获取语义和相似的句法。
    - 技巧：丢弃输入的几个关键词，以此来验证词向量的相似度计算性能
    - **例：a:b :: c:? man:woman :: king:? man–>king 那么 woman–>?**
        - 在语料中找到一点 x_i，即为和 woman 最为相近的词

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$



- 下面是 glove 的某些词向量相似度可视化的结果

Glove Visualizations



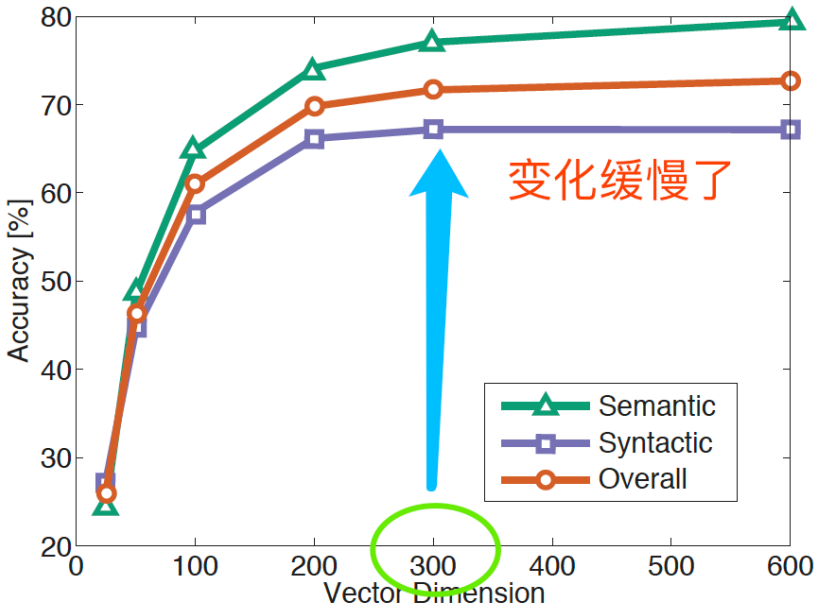
4. **评估：相似性和参数**

- Glove 的语义或句法相似度表现的更好，如下表：

Glove word vectors evaluation

Model	Dim.	Size	Sem.	Syn.	Tot.
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7

- 特点：语料的规模要大、词向量维数为 300 更合适



5. 相关性评估（距离）

- Word vector distances and their correlation with human judgments

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

6. **外部评估**

- 词向量好不好最直接的方法就是应用到实际场景，比如最常用的 NER（命名实体识别）任务中

recognition: finding a person, organization or location

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

- glove 对于 NER 任务表现理论上还说的过去，但凭上表中的准确率在工业领域中还是很难拿来用的。那么还有什么好的模型或者思路呢?
- 从下一小节就开始尝试将词向量输入到神经网络中，来进一步提升下游任务的性能。

语义及其歧义性

注解: 实际上，许多词都是一词多义的，特别是咱们的汉语，更甚有如今曾层出不穷的网络流行语，有时你不懂点八卦还真的猜不来词要表达的真实含义！这就是 NLP 绕不开的一个问题：歧义性，对应的任务就是：消歧。

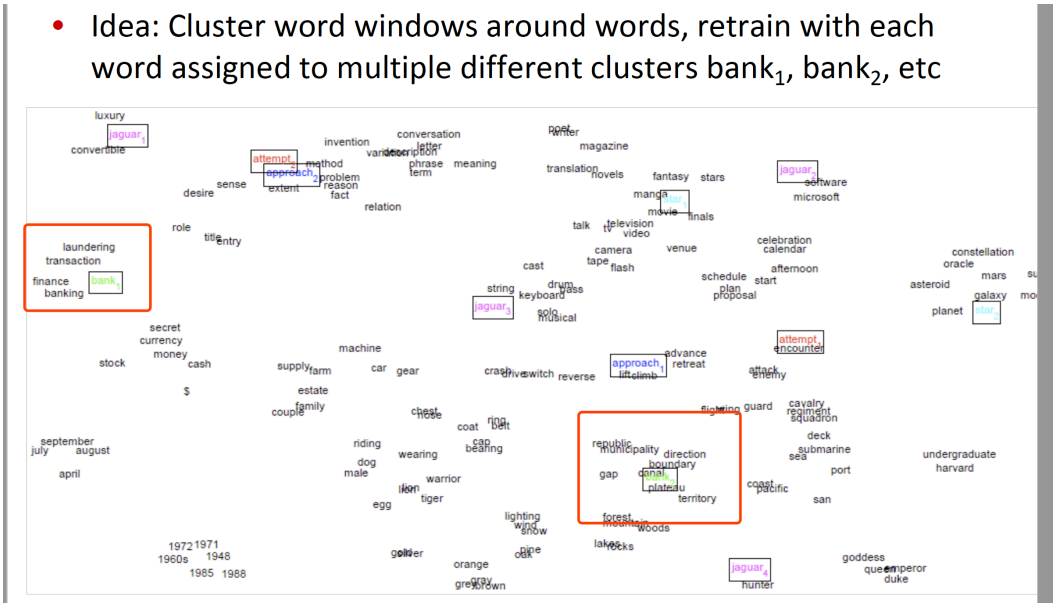
- 看下单词 **pike** 的含义

pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something: *I reckon he could have climbed that cliff, but he piked!*

  – 看上图易知词的含义还是相当丰富的，如何相对准确的捕捉到当前场景下（上下文）的真实含义就是值得思考和研究的一个问题。

1. 论文 1：Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)



- 使用了聚类的思路：通过一些关键词来聚类，但常常出现重叠（误分）的现象

2. 论文 2：Linear Algebraic Structure of Word Senses, with Applications to Polysemy 将同一词的不同语义

进行线性叠加

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$$

$$\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$$

$$f$$

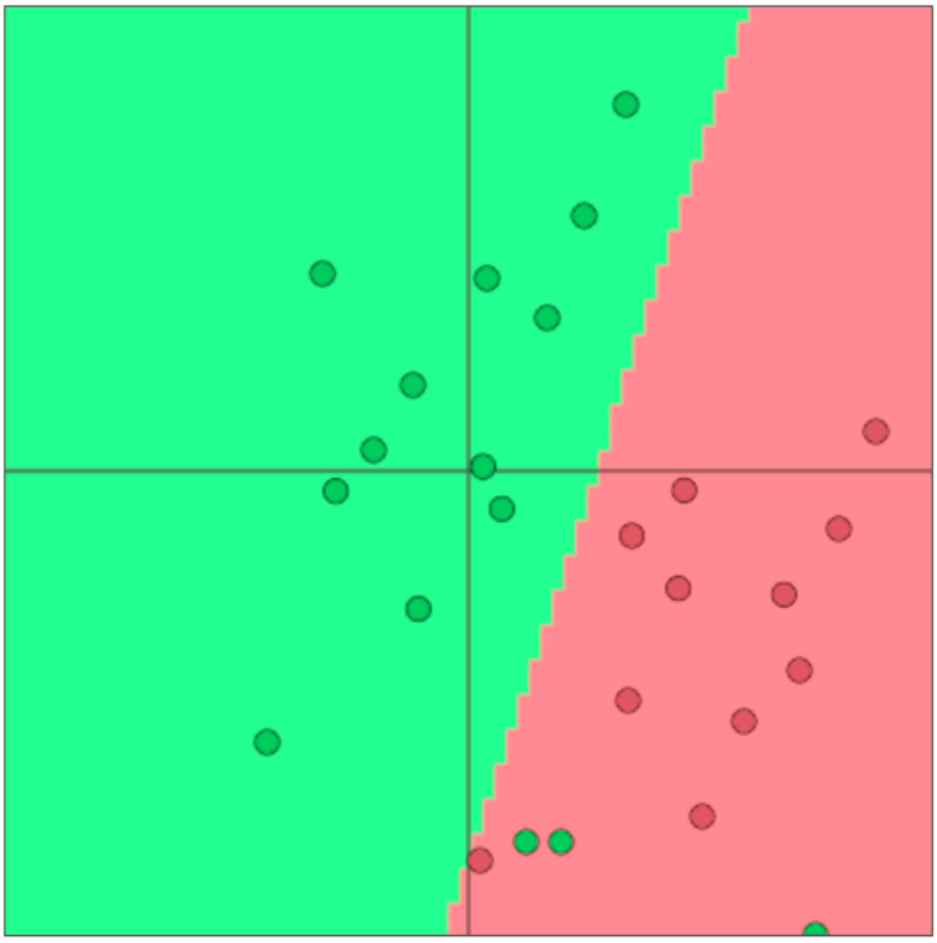
- 论文的思路来自词向量的稀疏编码，

分类（Classification）模型知识点回顾

1. 样本（数据集）：$\{x^{(i)}, y^{(i)}\}_1^N$，其中 x_i 为输入（词、句、文档等）y_i 就是标签，预测的分类目标（正负向、文档主题等）‘

2. 例如简单的二分类

   - 可视化 地址



3. 一般的机器学习方法或统计方法：假定 x_i 为固定大小，我们使用 sofmax 或逻辑回归算法训练权重参数 W，以此来寻找一个决策边界。+ 例如 softmax 算法，对于固定的 x 预测 y：

$$p(y \mid x) = \frac{exp(W_{j.}x)}{\sum_{c=1}^{C} exp(W_{c.}x)}$$

We can tease apart the prediction function into three steps:

1. Take the y^{th} row of W and multiply that row with x:

$$W_{y.}x = \sum_{i=1}^{d} W_{yi}x_i = f_y$$

Compute all f_c for $c = 1, \ldots, C$

2. Apply softmax function to get normalized probability:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y)$$

3. Choose the y with maximum probability

- 我们的目标就是最大化正确类别 y 的概率，不过我们一般为了降低运算的复杂度，会转为下式进行计算：

$$-logP(y|x) = -log(\frac{exp(f_y)}{\sum_{c=1}^{C} exp(f_c)})$$

4. **交叉熵损失**

1. 交叉熵的概念源自信息论中的知识，对于样本实际的概率分布 P 与模型产生的结果概率分布 q，其交叉熵可表示为下式：

$$H(p,q) = -\sum_{c=1}^{C} p(c)logq(c)$$

2. **整个数据集 $\{x^{(i)}, y^{(i)}\}_1^N$ 的交叉熵可表示为：**

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} -log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

$$f_y = f_y(x) = W_y \cdot x = \sum_{j=1} dW_{y_j} \cdot x_j$$

5. 优化

- 传统机器学习方法优化

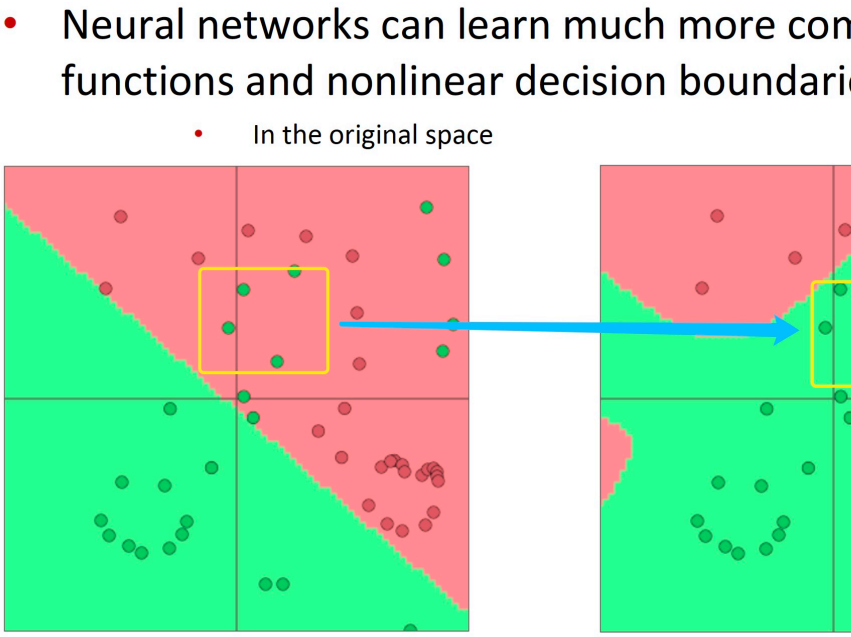
对于 θ 的数量一般和权重 W 的维数一致，线性决策模型至少需要一个 d 维的词向量输入和生成一个 C 个类别的分布。因此更新模型的权值，我们需要 C d 个参数

$$\theta = \begin{bmatrix} W_{\cdot 1} \\ \vdots \\ W_{\cdot d} \end{bmatrix} = W(::) \in \mathbb{R}^{Cd}$$

梯度优化：更新决策边界的参数

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W_{\cdot 1}} \\ \vdots \\ \nabla_{W_{\cdot d}} \end{bmatrix} \in \mathbb{R}^{Cd}$$

6. 使用神经网络 + 词向量分类一般的分类算法其通常解决的是一些线性问题，表达能力有限，对一些非线性的决策边界无法更好的建模，借用神经网络模型可进一步提升模型的能力。



- **基于神经网络分类（模型的预测能力更强）**
- **词向量与神经网络模型结合**
  - 同时学习权重 W 和词向量 x，参数量为：cd+vd，非常大的参数量，C 表示分类大数量，d 表示每个词向量维数，V 表示词汇表的大小。

$$to$$

$$\nabla_\theta J(\theta) = \begin{bmatrix} \nabla_{W.1} \\ \vdots \\ \nabla_{W.d} \\ \nabla_{aardvark} \\ \vdots \\ \nabla_{zebra} \end{bmatrix}$$

- 参数量大意味着表达（拟合数据）能力很强（这里可以联系数学中的多项式，参数多了意味着多项式越长，图像也就越复杂），但过犹不及，容易造成过拟合，模型泛化能力不足，怎么办呢？常见做法就是加入 **正则项**

小结

本节首先讲了词向量的训练基本过程和常用方法：解决维数高、复杂度高的问题；其次，讲了 Golve 的主要思想；最后通过分类引出神经网络的优势：非线性能力，能够更好地获取语义信息。后续小节将开始神经网络的篇章。

1.20.4 神经网络

Natural Language Processing with Deep Learning
CS224N/Ling284

Matthew Lamm

Lecture 3: Word Window Classification, Neural Networks, and PyTorch

---

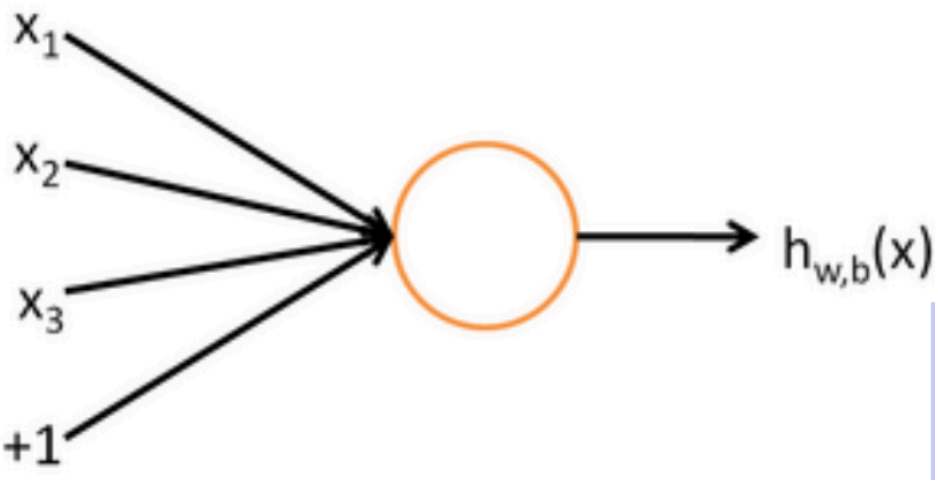
注解： 本小节主要讲神经网络的基础知识和 NLP 中的命名实体识别（NER）任务。

Neural NetWork 基础

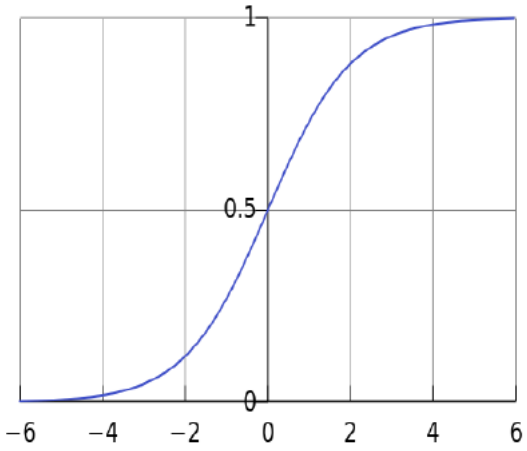
1. 神经网络是指由很多人工神经元构成的网络结构模型，这些人工神经元之间的连接强度是可学习的参数。
2. 人工神经网络（Artificial Neural Network，ANN）是一种模拟人脑神经网络而设计的数据模型或计算模型，它从结构、实现机理和功能上模拟人脑神经网络。
3. **神经元（Neuron），是构成神经网络的基本单元，其主要是模拟生物神经元的结构和特性，接受一组输入信号并产出**

    - 例如神经元可以是一个二元的逻辑回归单元，典型的结构如下图：

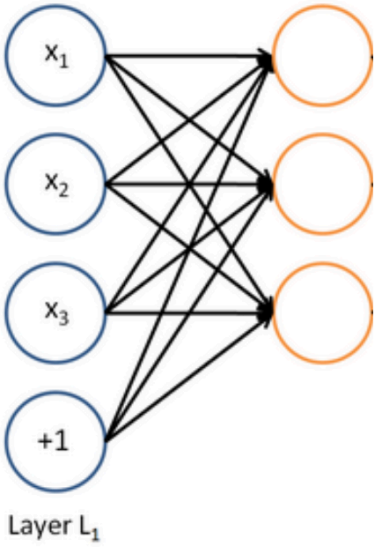
    $$h_{w,b}(x) = f(w^T x + b)$$



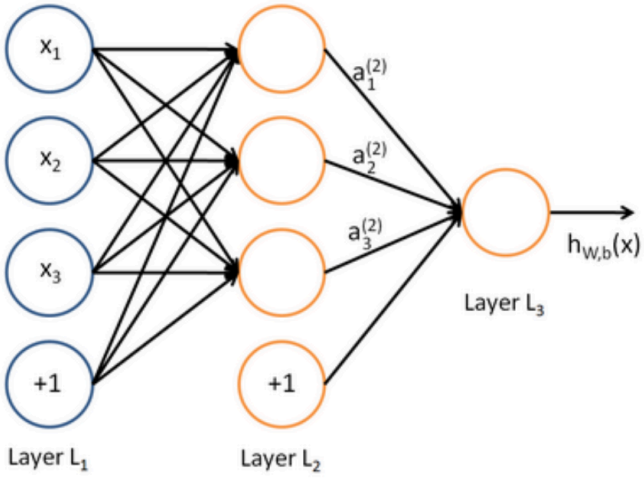
其中，$f(z) = \frac{1}{1+e^{-z}}$ 。f 称为 **激活函数**例如 sigmoid 函数：Logistic、Tanh；w 称为 **权重**，表示信号的强弱（特征的重要程度）；b 称为 **偏置**；h 称为 **隐藏层**；x 表示 **输入的特征值**。在本例的逻辑回归模型中，w,b 就是这个神经元的参数。



4. 神经网络结构还是上面的神经元，但是是同时运行多个逻辑回归单元。比如有一下几种形式：

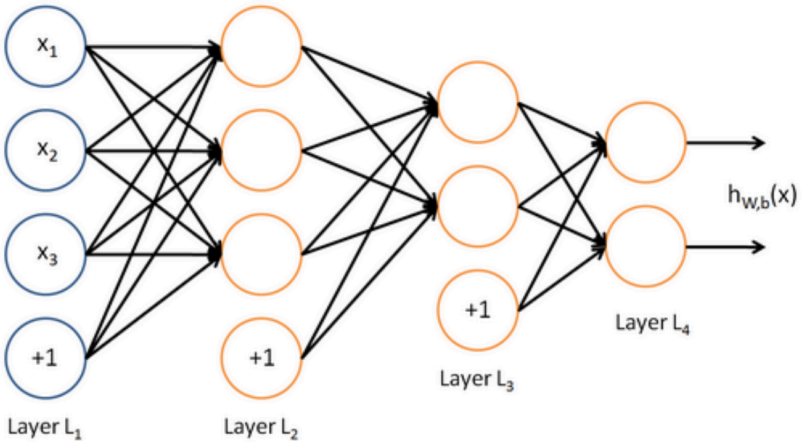


But we don't have to decide ahead of time what variables these logistic regressions are trying to predict!



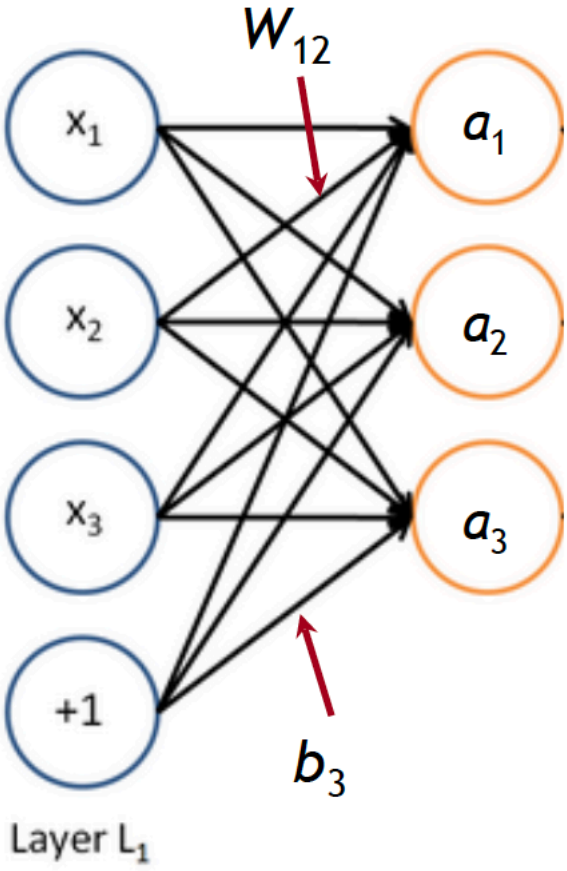
It is the loss function that will direct what the intermediate hidden variables should be, so as to do a good job at predicting the targets for the next layer, etc.

Before we know it, we have a multilayer neural network....



20

- 每一次神经元用矩阵符号表示



$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$
$$\dots$$

可用矩阵符号表示：

$$z = WX + b$$
$$a = f(z)$$

激活函数 f(x) 逐元素进行相乘。$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$

5. **为什么需要非线性 f 激活函数**

   - 一句话总结：提高模型的表示能力或者学习能力。为什么呢?
   - 因为没有非线性变化，我们就无需选择神经网络模型了，因为只能进行线性变换和传统的机器学习模型类似了
   - 因为线性变换组合后还是线性变换，不能进行深层次的特征学习
   - 非线性变换的层数（隐藏层）越多，那么模型的就能拟合更为复杂的数据（联想下多项式函数，一次、二次、三次甚至更高次函数拟合样本点程度是完全不同的，当然复杂度也是递增的），不过也很容易造成过拟合，这个度要想拿捏的好，是一门玄学（运气 + 实力）

- 这里补充点关于激活函数的知识：

注解:

- **激活函数：**
  - 连续并可导（允许少数点上不可导）的非线性函数。可导的激活函数可以直接利用数值优化的方法来学习网络参数。
  - 激活函数及其导函数要尽可能的简单，有利于提高网络计算效率。
  - 激活函数的导函数的值域要在一个合适的区间内，不能太大也不能太小，否则会影响训练的效率和稳定性。
- **常见激活函数**
  - Sigmoid 型激活函数：是指一类 S 型曲线函数，为两端饱和函数。常用的 Sigmoid 型函数有 Logistic 函数和 Tanh 函数。
  - 修正线性单元（Rectified Linear Unit，ReLU）。常用的激活函数之一，图像类似一个斜坡。
  - 指数线性单元（Exponential Linear Unit，ELU）。是一个近似的零中心化的非线性函数。

命名实体识别介绍

注解: 命名实体识别（Named Entity Recognition，NER）。NLP 领域的一个基础型子任务。

---

1. **一些用途**
   - 获取文档中的实体名称。
   - 问答、对话等系统需要实体。
   - 从实体与实体之间的关联中获取信息。
   - 也可扩展到填槽（slot-filling）分类的任务中。在对话系统中 **填槽**指的是为了让用户意图转化为用户明确的指令而补全信息的过程。
   - 在构建知识库/知识图谱的过程中，获取命名实体也是重要的一环。
2. **思路**
   - **首先通过上下词对单词进行 分类，然后将实体提取为词 序列来预测实体**。
     - 我们可发现 NER 不仅是分类问题，还是一个序列问题，也称为序列标注问题
   - BIO 标注体系：B-实体起始位置、I-实体结束位置、O-非实体

- 补充一点 NER 的知识：

注解:

- NER 的目的是从一段文本中找出实体同时也需要标注出实体的位置，实体一般包含人名、地名、组织名等。
- NER 标注是使用的标签体系包括：IO、BIO（常用）、BMEWO、BMEWO+，一般地，标签体系越复杂其标注结果也更准确。
- NER 常用算法：BiLSTM+CRF、BiLSTM-LAN、也可结合词典进行实体识别，具备扩展性
- 标注工具：brat

---

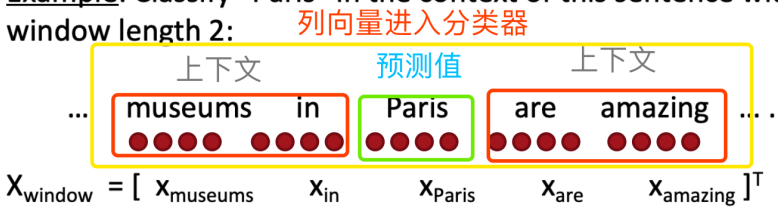
3. **NER 的难点何在？**

   - 边界问题。NER 任务在文本的处理中是很常用的一个工具，但其本身会对一些文本中的实体边界识别有偏差。这一点在我的实际业务中也遇到过：“股东 ** 李杨楠 ** 近日……”，对人名会识别为：“杨楠”，而实际是：“李杨楠”，我的处理策略之一就是加一个人名词典。
   - 实体似是而非问题。“Future School” 是一个学校（ORG）名称，还是其本意未来的学校。
   - 不确定是人还是机构或地点。
   - 实体识别依赖上下文。我遇到的实际业务问题：公司 xxx 先生，由于公司干扰会将 PER: xxx 识别为 ORG。

个人结合实际对于上述难点问题有两个解决思路：一是模型本身的调整（语料、结合实体上下文进行分类、引入知识等）；二是维护一个词典，将识别有误的实体添加到字典，这个方法感觉比较常用。

4. **尝试：词基于窗口（上下文）分类**

- 思路：classify a word in its context window of neighboring words.
- 例如：在实体的上下文中判断是人名、地名、机构名或者什么也不是。
- **实现策略**
  - **之一：每个词的上下文存在差别，所以可对上下文窗口的词向量进行平均化，然后再进行单词分类。**
    * 缺点：丢失位置信息。（为什么呢？我想是 word embedding 被运算的造成的）
  - **之二：Softmax**
    * 训练一个 softmax 分类器对实体以及其窗口词（上下文）整体进行分类
      - Example: Classify "Paris" in the context of this sentence with window length 2:



      - Resulting vector $x_{window} = x \in R^{5d}$, a column vector!

    * **数学含义** softmax 分类器：

$$\hat{y}_y = p(y|x) = \frac{exp(W_y \cdot x)}{\sum_{c=1}^{C} exp(W_c \cdot x)}$$

      交叉熵损失函数:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -log(\frac{e^{f_{yi}}}{\sum_{c=1}^{C} e^{f_c}})$$

  - **进阶之三：多层感知机**
    * 参考论文 (2011)Natural Language Processing (Almost) from Scratch
    * 思路：在 softmax 分类器加入中间层（引入非线性）提升分类器的分类能力（复杂性）。根据是否是需要分类的实体进行权重的分配。
    * 神经网络前向计算

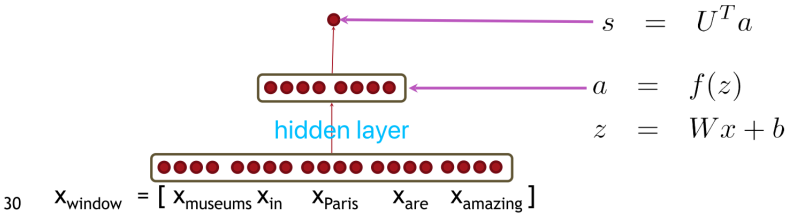
返回一个实数

$$score(x) = U^T a \in \mathbb{R}$$

We compute a window's score with a 3-layer neural net:

- s = score("museums in Paris are amazing")

$$s = U^T f(Wx + b)$$
$$x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8x20}, U \in \mathbb{R}^{8x2}$$



* 中间层可与输入的词向量进行非线性的变换（赋予不同的权重）

* **Objective Function 我们可使用 Hinge 损失函数（Max-margin loss）。目的是使目标窗**

  - $s = score(\text{"}museums\ in\ Paris\ are\ amazing\text{"})$
  - $s_c = score(\text{"}Not\ all\ museums\ in\ Paris)$
  - $Minimize(J) = \max(0, 1 - s + s_c)$
  - 函数为不连续可微，因此可计算梯度。

注解: Hinge 损失函数（又称，max-margin objective）对于两类分类问题，假设 y 和 f(x, θ) 的取值为 $\{-1, +1\}$。Hinge 损失函数（Hinge Loss Function）为:

$$L(y, f(x, \theta)) = \max(0, 1 - yf(x, \theta))$$

小练习: 理解并使用 Python 实现 Softmax 分类算法。

1.20.5 矩阵计算与 BP（反向传播）算法

矩阵的梯度计算

- **都是基础的高数计算，可直接查看 幻灯**
  - 求导链式法则
  - 复合函数求导
  - Jacobian matrix（Jacobian: Vector in, Vector out）

  - 推荐补充一份关于微分及 BP 的 文稿

BP 算法（重点）

甜点 Yes you should understand backprop

注解:

- 进行微分并使用链式法则。共享参数以减少计算量。

---

1. 计算图和 BP
2. BP 算法核心知识
3. 计算效率

总结

1.21 知识图谱 (Knowledge Graph)

1.21.1 资源推荐

1. **据我了解截止到 2020 年市面上关于知识图谱的系统性书籍并不多，但是这方面零零散散的资料还是比较多。推荐一本**

   - **《知识图谱：方法、实践与应用 》**
     - 从知识图谱的各个环节到相关应用都有涉及，属于入门级书籍。
     - 如果想快一点了解 KG 领域，推荐可以快速看看这本书。
2. 本笔记主要通过学习 2020 年春 Stanford University CS520 知识图谱课程来记录，很有爱的一个名字。知识图谱是人工智能在认知领域目前最佳的实践方向，一起来学习吧
3. Stanford CS520 - 知识图谱 Knowledge Graph (Spring 2020) B 站资源
4. The Stanford AI Lab Blog

1.21.2 什么是知识图谱？

注解:

1. KG 是一个存在多环节、结合多技术的 **工程**。

---

1. store: Resource Description Framework(RDF)

2. Query language for RDF: SPARQL
3. Application: searching the Web-Google,Bing, Maps、Smart Assistants-Siri, Cortanta, Echo.
4. science application: search, discovery,data exploration.

1.21.3 如何应用知识图谱?

1.21.4 如何构建知识图谱？

1.21.5 知识图谱案例介绍

1.22 Pytorch

1.22.1 书籍推荐

1. 不具体推荐哪本书了（太多了），直接官网文档吧，详细准确。

1.23 Tensorflow

1.23.1 书籍推荐

1. 不具体推荐哪本书了（太多了），直接官网文档吧，详细准确。
2. 框架也开源了，了解底层可以去 GitHub 找。

1.24 计算语言学

注解:

- 创建日期：2022-01-23
- 更新日期：2022-01-23

1.24.1 前沿跟进

1. 刘群博士个人网站

1.24.2 问题引入【问题驱动学习】

1. 自动翻译：最古老的问题之一、是推动计算机语言学的永恒动力、计算语言学的终极目标。
   - Google 翻译
2. 自动问答：IBM 沃森、图灵测试
   - https://home.pandorabots.com/home.html
3. 其他问题：信息抽取、文本摘要、信息检索……

1.24.3 定义和特点

1. 定义: 以计算为手段进行自然语言的研究和处理的科学。
2. 自然语言是一种符号系统。
3. 语言、思维、客观世界
4. 研究的层面：语音、语法（词汇和句法）、语义（这句话说了什么）、语用（为什么要说这句话）
5. 语法层面：词法歧义（词性兼类、词语切分）、句法歧义（结构歧义、组合关系歧义）- 张三和李四的朋友 - 观赏鱼

注解： ETL，是英文 Extract-Transform-Load 的缩写，用来描述将资料从来源端经过抽取、转置、加载至目的端的过程。

1.25 ETL 工具

1. Kettle 2.

1.26 Kettle 工具使用笔记

1.26.1 基本概念

kettle 是一个 ETL（Extract,Transform and Load 抽取、转换、载入）工具，ETL 工具在数据仓库项目使用非常频繁。

1.26.2 应用场景

1. 在不同应用或数据库之间整合数据。如 XBRL 数据导入关系型数据库。

2. 把数据库中的数据导出到文本文件。
3. 大批量数据装载入数据库。
4. 数据清洗。
5. 集成应用相关项目是个使用。

1.27 漫谈数学

注解:

- 数学一门严谨的科学，数学本身也很美，在这里记录一些自己关于数学平日的学习和总结。
- **讲座列表:**
  - UCAS：席南华老师讲座

1.27.1 怎么理解数学

1. 数学的美感，对象之间的逻辑联系
2. 基本的对象，如素数、结构、形
3. **典型的定理，优美的证明**
   - 举例：素数有无穷个、黎曼函数、计数（有限集、无限集）
   - 不完备性定理（希尔伯特）、P 与 NP 问题（计算机中的经典问题）

P与NP

A: 有限集, 由一些整数组成
S_A: A中所有数的和.

问题: 能否找到多项式时间的算法
以确定是否存在A的子集B使得
$S_B = 0$.

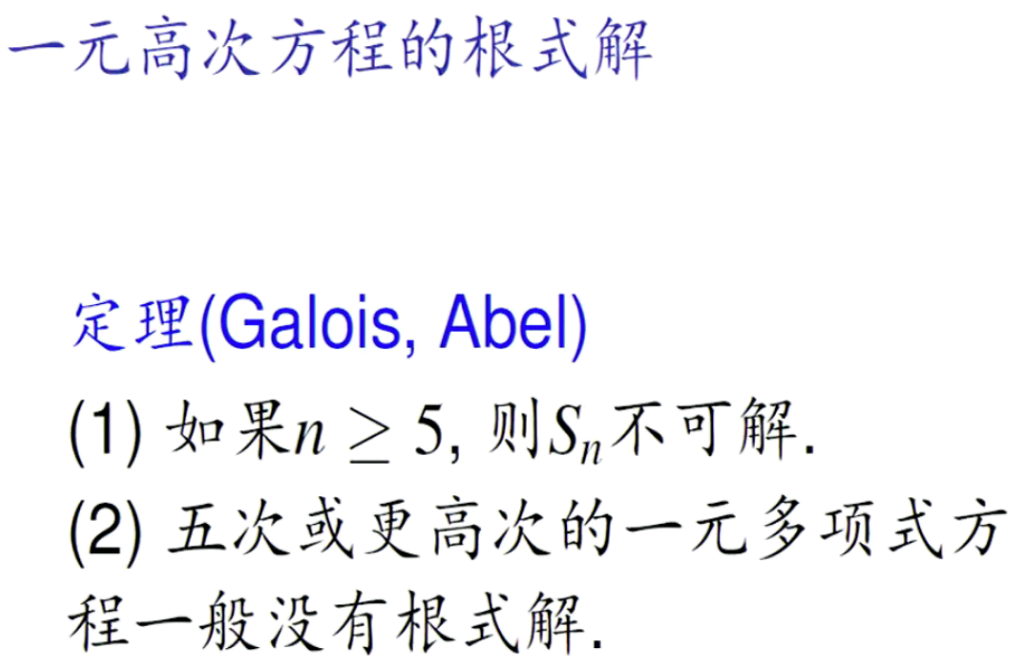
- 排队。排队之中有结构：排队是一种映射，如数据结构中的数组下标和存储值得关系；所有的排队方式就成了群，群论。

群

在这个运算下, 1, 2, 3的六种不同
的排队成为一个群, 记作S_3

群的定义:

集合G称为一个群如果它有一个二
元运算, 满足结合律, 有一个单位
元素, 每个元素都有逆元. 即:



- 群论在很多地方都有应用，在证明费马大定理时群论也发挥了很大的作用

4. 陶哲轩、张益唐华裔等数学家的作出了很多有价值的工作。
5. ” 概念 “的威力，引入一个恰当的概念往往能够解释事物的本质，我们国内研究的绝大数问题都是来自西方的，这一点值得思考！提出一个概念是很难的，但这是思维方式的问题，西方 19 世纪末 20 世纪初自然科学发展的速度是惊人的，如爱因斯坦、普朗克、莱布尼茨等，背后可能与哲学相关，这些人提出的概念背后都有雄厚的哲学知识，比如康德的关于时间的问题就对爱因斯坦的相关问题思考有启发。我们有机会可以多听听关于西方的哲学，如休谟、康德等。
6. ” 形 “。极小曲面、分形几何、动力系统（气象学应用）。提到了丘成桐先生的工作：卡拉比-丘流形猜想，这个证明第一次是出错的（数学家出错是很正常的），发现后又证明了几年才完成。

    - 关注：大小（长度、面积等）、形状（直、曲等）、性质（切线、弯曲程度等）
    - 求切线、加速度等就有了微分；求长度、面积、路程等就有了积分。
    - 纽结：一根首尾相连但不打结的形状各种各样

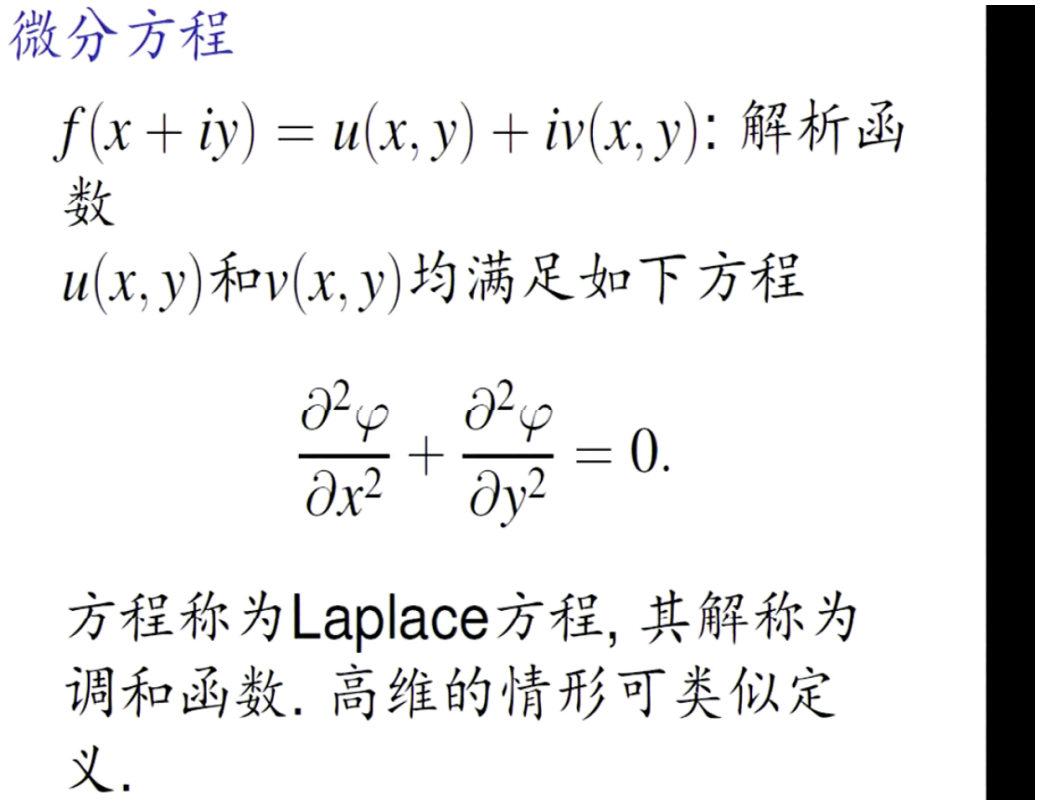
解方程

一元高次方程$\longrightarrow$群论

多元一次方程$\longrightarrow$矩阵理论

解方程

费马大定理(Wiles):
如果$n \geq 3$, 则方程$x^n + y^n = z^n$ 无非平凡的整数解. 即如果x, y, z是整数解, 则$xyz = 0$.



- 只有一个独立变量的微分方程就称为常微分方程；混沌理论来自微分方程的研究；常微分方程的定性研究和动力系统密切相关。

7. 数学三分天下：分析、几何、代数。陈省身先生认为几何将会统治数学的天下，但可能不会发生。

1.27.2 怎么掌握数学的思维方式？

1.27.3 什么是数学最基本的？

1.27.4 什么问题是好问题？

1.28 团队建设与项目管理

注解:

- 创建时间：2020-1-1
- 更新时间：2022-07-31
- 开发与项目管理是不可分家的（自我定位要准），对于个人也是同样的。虽然你我可能还处于所谓的“码农”阶段，但也要有 big picture!，了解团队协作、项目从立项到实施再到交付、维护全流程等，这样才能在工作中多角度思考所遇到的问题，同时也有利于协作能力的提升。

- 本章就会集中记录下我在日常工作的思考、学习、实践的心得和经验。

1.28.1 项目管理（Project Management）

注解:

- 几个名词：PM、项目管理办公室 (Project Management Office，PMO)

- **技术思考**
  - 对于小团队，应该仅最大可能使用现有成熟的技术和工具，快速工程化产品，面向客户，快速迭代！
  - 如何提高工程效率。在实际开发中注重工具的打造是很有必要的！去快速标准化、流程化一些开发过程进而思考开发一些小工具去提升团队的开发效率，压缩开发周期能达到事半功倍的效果，另一方面也可让开发人员将注意力更多集中于解决真正的业务问题。
- **项目开发文档**
  - 项目过程各种文档，都应在项目各个阶段进行及时的总结管理，避免后期因文档工作影响项目整体进度等。
- 闭环思路（有始有终）。
- 善用管理工具，如看板，可以很直观追踪项目的进展。
- 一切任务安排都要让团队成员知道 DDL 及目标，并在执行过程中让成员积极反馈，保证任务有效执行，达到预期结果。

经验

1. 确定任务完成的标准
2. 任务分解：交付物、责任人、完成效果评价
3. 确定效能指标。
4. Product Market Fit，PMF。产品和市场达到最佳的契合点。

项目经理

注解: 具备服务意识、清晰的逻辑思考、总结、表达能力。2020 年 7 月 1 日，来公司 1 周年啦，也是正式职业（之前在高校实验室）生涯 1 周年，按照公司安排，我开始尝试完整执行一个项目！下面总结下实践经

验和所思所想。目标：具备计划、跟踪、总结的完整能力。

1. **基本素质：**
    - 理解业务的能力，分解需求及安排优先级，不懂的及时沟通确认。
    - 能够积极跟踪，以天、周、月或季度进行项目执行的跟进。
    - 预判项目风险，积极调整优先级，确保项目能如期交付。
    - 里程碑意识，利用里程碑来掌控阶段性目标完成情况。
    - 组内日报、项目周报、月报的编写。
    - 多思考、总结、优化管理的方式和工作流模式，极大提升组内、项目的的执行效率。
    - 负责任的态度、善于倾听和引导、总协调能力、写（各种文档、PPT 等）、时间管理能力、清晰的目标和强有力的执行能力。
2. 核心职责：协调项目资源、梳理项目组织架构，整合项目计划；全生命周期监控管理整个项目，重点不是管人，是管项目。
3. 注意各种文档的归档、做好会议记录，留底溯源、设定明确的任务目标、任务分解及执行时间计划。
4. 合理评估项目工作量，做好相关预案。
5. 项目问题分解成一个个小问题，做好和甲方的沟通和同步工作，让对方了解团队及项目的执行情况，有哪些风险点等等。
6. 对于超越合同的需求（一般刚开始定需求时，可能因为考虑不周，一些隐性需求没有提出），一定要做好评估，并和甲方沟通确认是否进行需求变更等方式。以免因沟通不到位，造成项目整体时间紧张。

1.28.2 团队协作

1. 借助功能智能化管理和协作（钉钉、腾讯会议等）
2. 清晰的目标和快速执行力、问题定位能力要强。
3. 定期组会和必要的代码评审。
4. 技术交流会，分享开发中的经验。
5. 和团队成员同步项目的背景信息，这样有助于团队成员理解日常工作，也有利于日常工作的推进，因为组员有时不明白为何要这样做，就会造成效率比较低或达不到预期的目标。
6. 适度向组员传导压力，不用过多阻塞在项目经理或执行经理这里。

1.28.3 敏捷（Agile）开发

背景

从 2022 年 4 月开始，公司产品研发迭代基于敏捷开发进行，每个迭代周期基本持续 4 周，每个周期的工作包含了需求梳理，设计，编码，测试，发布，验收。在这个过程中我以 teamleader 的角色全程参与，也是初次完整实践敏捷开发流程，下文主要将敏捷开发的基本动作和个人的实践做一总结。在 Scrum 团队我担任的角色是 Scrum Master。

1. 看板。小组内的迭代目标及每个人的任务进度，未完成，进行中，已完成。
2. 每日站会。每天固定时间花费在 15 分钟内同步下任务完成情况，昨天的任务完成情况，今天的计划，遇到的问题，需要哪些配合等。
3. 迭代完成后的总结，复盘。

什么是敏捷开发？

- 作用：

  应对开发中的不确定性，周期性，增量式交付成果，最终交互按时，保质的产品。

- 方法：

  1. scrum
  2. 极限编程 (Extreme programming, xp)

Scrum

scrum 是敏捷开发的一种框架，流程化具体做法。涉及到一些名词概念：

1. sprint。冲刺，也可在 scrum 中译为迭代。
2. backlog。任务池，还未开始做的工作。
3. Daily Scrum Meeting。每日站会。

Scrum 涉及到的三个角色：

1. SScrum Master。敏捷专家，简称 SM
2. Product Owner。产品负责人，简称 PO
3. Development Team。开发团队

实践中的真实情况

1. 沟通是关键，面对面的沟通尤为重要，因为它更有效。
2. 任务分解和跟踪是保证。

实践总结及建议

1. 产品的总体规划和系统总体设计尤为重要，这会为后续的迭代提供最根本的遵循，以团队防止迷失方向。

1.29 项目管理（Project Manager，PM）的理论与实践

注解： post time: 2021-10-21

update time: 2022-01-04

core content: 项目管理的理论、工具、实践

1.29.1 比较重要的几点

注解： 以下内容摘自 2021 年 12 月份公司内部的项目管理培训，个人人为比较重要的几方面内容，记录在这里，以示提醒，不断实践、提升管理水平。

---

1. 项目经理的素质
    1. 沟通谈判的能力
    2. 执行力
    3. 专业知识 + 业务知识
    4. 技术流程
    5. 项目计划能力【积极推动项目完成】
    6. 项目跟踪和控制能力
    7. 团队影响力
2. 项目成功的定义
    1. 包含 3 部分内容：组织成功 <–团队成功 <–个人成功 (过程合规、多快好省完成目标)
3. 技术项目经理
    1. 一个技术人员能力比较好被提拔为技术项目经理，由于缺乏项目管理经验，往往会在项目中产生一定的风险甚至失控。
4. 缺乏监控和控制

    1. 好的项目经理是需要记录项目开始时间、实施时间、预估剩余时间，考虑成本、产出、时间等监督落实推动项目进展。

5. 项目管理过程

    1. 计划、沟通、执行三者项目依赖、相互影响、不断循环直至达成项目目标。

1.29.2 what PM ?

1. 项目生命周期是通过一系列项目管理活动进行的，即项目管理过程。
2. 过程：输入–> 工具、技术—> 输出
3. 启动–> 计划–> 执行–> 监控（质量/范围）–> 收尾

小技巧:

- 《项目管理知识体系指南》
- 《中国项目管理二十年发展报告》

1.29.3 基本概念

1. WBS （work Breakdown Structure, 工作分解结构）- 把整个项目要做的事情，划定范围并层级分解。- 团队工作的指南 - 后续计划的前提 - 监督控制的前提
2. OBS - 组织分解结构
3. RAM - 责任分配矩阵

1.29.4 项目管理工具

注解:

- 记录工作中的点点滴滴很重要，做了什么、收获了什么、不足在哪里。

---

1. WBS - 按照生命周期分解 - 按照子项目分解
2. RAM
3. 问题日志: 记录并跟进所有问题的项目文件
4. 关键路径法（CPM）：关键路径要时间、非关键路径要资源
5. 风险登记册：风险识别、风险分析、应对规划的结果的文件
6. 冲突管理技术

7. 滚动式规划 - 详细规划近期要完成的工作，在较高层级上粗略规划远期工作。
8. 核对表：抽检工作完成情况
9. 资源日历
10. 挣值管理技术（EVM）

1.29.5 沟通

1. 如何向上沟通？- 多交流沟通，凡事多想一步。- 多做一些额外工作。- 把老板变成老师。- 循环：有反馈、有行动

1.30 技术负责人画像

注解:

- 创建时间：2022-08-13
- 更新时间：2022-08-31
- 本专题主要回答什么是技术负责人？技术负责人的职责？如何做好技术负责人？技术负责人如何持续成长？技术负责人如何赋能团队？
- 技术负责人下文简称为：TL（Technology Leader, TL）

1.30.1 什么是 TL？

1.30.2 TL 的职责？

1.30.3 如何做好 TL？

1.30.4 TL 如何持续成长？

1.30.5 TL 如何赋能团队

1.31 道与术

注解:

- 将工作，职场中的的所学、所见、所感记录在这块田地里，不断成长，林荫满园。

- 创建时间：2020-05-23
- 更新时间：2022-08-13

1.31.1 工作

关于 coding

工具/调试

1. 一般来说，使用 Pycharm 进行开发还是比较方便的，有较为完善的调试工具和交互界面，比较高效。善于使用工具可以大大提高工程开发的效率，要有意识去尝试方便的开发工具。也可以使用 VSCode 工具（插件化生态做的可以，界面简洁功能同样强大）
2. 对于一个专业的以开发为营生的人来说，开发前准备好两个基本工具是很有必要的。一是个人喜欢且具备高效率的 IDE、二是保证随时能够 Google!
3. 公司的日常开发，必然是协作。这时必须熟练使用 GIT 命令及创建分支与 commit code 的规范，开发环境最好使用虚拟环境，推荐使用的工具有：pipenv: 虚拟环境管理工具、pyenv:Python 版本管理工具。以上都是作为一个有开发经验人所应具备的。
4. 有一个能力必须在日常实践中有意识的训练：debuge、debuge、debuge!!!
5. Docker 作为一个工具，在部署环节可大大减少工作量，以配置文件代替大量的复杂环节，对后续的可扩展起到一定帮助作用。
6. pycharm 官方学习文档包含工程方面的讲解比较全面，想快速学习相关调试等可以进入官方文档学习。
7. 掌握常用 vim 命令也能在服务器操作中起到事半功倍的效果。

精益代码

1. 看到公司对于代码的要求：简洁优美、重要的是代码的可扩展能力和易用性，需要在开发中去思考代码如何能够最大化扩展性能。主要有几个特点：封装、高内聚，低耦合、兼容性。
2. 对于新进公司的人才，当接触到公司庞大的代码时，可以在具体的任务背景下去运行并调试代码，需要带着问题去看代码，如：参数是什么格式或类型、传参过程是怎样的、最终返回了什么等等。还有一个技巧就是参考并手敲（经典的代码 2 遍起吧）并能运行前人写的代码，注意人家的命名规范和解决问题的思路等等。这个过程可能会有些痛苦（目前（201907）的我）但是成为一个优秀程序员的必经之路，没有什么捷径可走!
3. 如何写出” 有味道” 的代码，主要从两个方面入手：一是标准的代码风格；二是优秀的具有逻辑的实现。

Code Review

1. CR 原则梳理 脑图
2. 待提升代码 + 优质代码解析。

架构设计

1. KISS 原则。大道至简，解决问题，保持微笑，迎接各种否定。
    1. 可扩展性和可维护性。
    2. 恰到好处地解决问题。
    3. 系统能够运行 3-5 年不重构。
2. 七大设计原则
    1.

1.31.2 软实力方法论集合

写作方法论

以下内容摘自刘润老师的公众号文章 请笑纳：我这 28 年的写作心法，全部都在这里了。

写作时可以借鉴的几个心态或者要点，我根据自己的经验和感受排了个序，看看你觉得哪些点更重要呢?

1. 逻辑。推荐书籍《金字塔原理》。文章的背景、冲突、问题、答案。
2. 同理心。
3. 对象感。对象要具体，如用”你“代替“大家”
4. 讲故事-关键在于细节。因为没有人喜欢被强加的观点，我们喜欢的，是自己得出结论。
5. 举例子-降低认知成本。
6. 幽默感。目的是让读者更加读懂和理解要讲的内容。开自己的玩笑，是一种幽默感。
7. 开门见山。直接给出结论，然后再说背景和冲突。个人体悟：这一点你在工作肯定用的到，比如给主管或领导汇报工作，一般结论现行都是很正确的做法。
8. 有能力解决冲突，能够提出问题也有能力解决问题，让读者能从文字中获取信心。
9. 打比方-窥探核心本质。
10. 商派。场景导入-打破认知-核心逻辑-举一反三-回顾总结。
11. 结构。刘润老师的公众号文章一般有 3 段式、10 点式、32 条。
12. 观点。

13. 突出忧虑。直接跑出一个引起人忧虑的问题？让人进入情境。一般骗子就喜欢制造焦虑。

思考方法论

1. 理想的学习方法：师其意，不师其辞。学习的时候要去理解搞懂老师的思考逻辑和假设前提，至于如何说和做只要借鉴即可，不用生搬硬套，要结合自身的实际采用相应的方式去实践。
2. 学习要抓本质，才能应对外在形式的变化。一个追求成长的人，肯定是要搞清一个产品或一个技术的内在逻辑才能形成自身的判断。
3. 先生存再发展。生存期突出实用性。
4. 保持空杯心态。
5. 学习思维认知偏差。

学习方法论

1. 如何表达？答：遵循 START 原则。
2. 如何学习？
    1. 思维能力
    2. 融会贯通的能力
    3. 总结能力：可以按 4 个文档进行演化：收集，整理，专题，哲学。
3. 如何快速学习？
    1. 抓住主要信息的能力
    2. 提升输入的质量
    3. 内化于心