
架构师速成 1-前言 - for5million 的专栏 - 博客频道 - CSDN.NET

从事 it 工作 10 余年，痛并快乐着。

忠告以下人员远离 it：

1. 不能吃苦
2. 耐不住寂寞
3. 想赚大钱

如果你不是上面的人，而且非常想成为架构师话，请继续看下去。

1. 需要3年时间
2. 需要超强自制力
3. 需要极强计划能力
4. 需要吃苦

如果你能满足以上 4 条，那肯定就可以速成。可能有人会说"3 年也算速成，这也太龟速了"，我回答你，如果没人系统指导，想成为一个合格的架构师需要 10 几年甚至更长，我只是帮你少走一些弯路而已。本人成为一个合格的架构师花了 10 年，而我的 2 个兄弟只用了 3 年。如果你更聪明更勤奋，甚至可以缩短到 2 年，那你将是同龄人中的 top 100。

列举一下我的成功案例：

- 马同学 山东top3的软件公司技术总监
- 侯同学 新加坡国立大学计算机学院产学研中心技术总监
- 庄同学 某上市公司子创业公司CTO
- 李同学 某即将上市的营收10亿美金创业公司前端技术架构师
- 赵同学 我买网技术经理
- 王同学 先去搜狐，再去京东，现在是某银行子公司CTO
- 于同学 某top3数据公司技术经理

庄同学和李同学是我提到的 2 位用了 3 年就成为架构师的案例，后面希望他们也能给我写一些段子。

希望在后面的日子，我的文章能帮你迅速的成为你所在行业的 top100 架构师。

顺便推广一下，我的个人域名：http://www.100j.top 就是希望大家成为 top100 的架构师

架构师速成 2-概述 - for5million 的专栏 - 博客频道 - CSDN.NET

成为一名合格的架构师，需要经历菜鸟、码农、资深码农、项目经理、技术经理、架构师等一系列的过程。为了让大家通俗易懂，我把整个过程按照大家熟知的上学的顺序排了一下，从幼儿园-小学-中学-...一直到博士，至于博士后需要大家自己去实践和想象了。每个过程我都会进行统一的描述：

- 阶段：例如 幼儿园
- 需要做的事情：例如 学会一门编程语言
- 完成任务耗时：例如 2-5个月
- 升级标准：例如：能写出简单的计算器，接受用户输入的+-x/运算
- 风险：例如 有人打断

当然在正式开始之前，我还是要提示一下相关的风险：

1. 任何行业想走到顶端都要付出超出常人的努力，it尤为明显，因为人多，而且新技术层出不穷。解决方案：
    1. 尽量不要搞it
    2. 做好准备，乐观面对，以苦为乐
    3. 大道至简，把握基础理论，新技术自然就懂
2. 经常加班，导致个人提升和工作关系很难协调，解决方案：
    1. 早起的2小时时间永远留给自己
    2. 尽量将个人提升和工作融为一体
    3. 开发自己的工具，节省时间
3. 经常加班，导致个人提升和家庭关系很难协调，解决方案：
    1. 单身的时候早努力，就不需要担心
    2. 学会时间管理，和另一半一起提升
4. 没人带，一个人学起来很苦，很难，解决方案：
    1. 三人行必有我师
    2. 学会时间管理、知识管理等软技能非常重要

ok，肯定还有更多的风险，我会再继续补充，其实风险根本不可怕，可怕的是不知道如何控制风险。

http://www.100j.top/category/jg/ 架构师速成目录

架构师速成 3-开发者境界 - for5million 的专栏 - 博客频道 - CSDN.NET

修炼的境界自下而上分为：

筑基、开光、融合、心动、金丹、元婴、出窍、分神、合体、洞虚、大乘、渡劫

其实开发者也可以按照修炼的境界进行划分：

入门、对象、模式、框架、架构、成神

1.入门 初学者就是为了把功能实现，不考虑其他，此时根本不考虑可读或者可修改性。
2.对象 以面向对象方式进行编码，把代码分开写到不同的对象中，能够进行跨对象的交互。
3.模式 关键点可以使用设计模式进行设计
4.框架 某一语言内部进行高度封装，使常用的功能开发步骤极度简化，提升开发效率，并极大降低对开发人员的要求。例如使用 spring mvc 进行封装，对错误、事务、日志等进行统一处理，或者更进一步对前后端交互进行封装。
5.架构 针对不同场景，进行跨语言、跨系统、跨容器进行设计及高度封装，使系统高可用，并能支持高流量，高并发。

架构师速成 2.1-论成功 - for5million 的专栏 - 博客频道 - CSDN.NET

为什么在这里插入这么一个看起来无比宏大的问题？

是想告诉你成功是什么。知识在整个人生中的作用。

成功是什么？这是一个很难定义的问题。有人说知足，有人说有钱，有人说有权。我自己对成功的定义是自我情感的满足和自我状态的认同，其实就是知足，而成功是通过自身的提升的达到的。

自身实力包含什么呢？

```
1.
2. 实力Map={
3.
4.        素质list:行动、坚持、自信、坚强、感恩、谦卑;
5.
6.        能力list：学习能力、思考能力、沟通能力、管理能力、解决问题能力;
7.
8.        知识list：it、生活、.....
9.
10. }
11.
12.  你的目标Map={感情和谐：falsel;身体健康:false;事业成功:false;生活富足:false}
```

```
13.
14.  while(!你的目标Map.each(isTrue)){
15.
16.      增强你的实力Map={..}
17.
18.      if(你的实力Map>实力Map){
19.          你的目标Map.each(setTrue)
20.      }
21.
22.  }
23.
24.  print("you win! ")
```

所以你需要不断磨练自己的素质，提升自己的能力，积累自己的知识，不断投资自己。自己强大后外界的帮助也会随之而来，你离成功就不远了。

架构师速成 3-开发者境界 - for5million 的专栏 - 博客频道 - CSDN.NET

修炼的境界自下而上分为：

筑基、开光、融合、心动、金丹、元婴、出窍、分神、合体、洞虚、大乘、渡劫

其实开发者也可以按照修炼的境界进行划分：

入门、对象、模式、框架、架构

1.入门 初学者就是为了把功能实现，不考虑其他，此时根本不考虑可读或者可修改性。
2.对象 以面向对象方式进行编码，把代码分开写到不同的对象中，能够进行跨对象的交互。
3.模式 关键点可以使用设计模式进行设计
4.框架 某一语言内部进行高度封装，使常用的功能开发步骤极度简化，提升开发效率，并极大降低对开发人员的要求。例如使用 spring mvc 进行封装，对错误、事务、日志等进行统一处理，或者更进一步对前后端交互进行封装。
5.架构 针对不同场景，进行跨语言、跨系统、跨容器进行设计及高度封装，使系统高可用，并能支持高流量，高并发。

http://www.100j.top/ top100 架构师

架构师速成 4-幼儿园 - for5million 的专栏 - 博客频道 - CSDN.NET

- 阶段： 幼儿园
- 完成任务耗时：1-3个月
- 升级标准
    - 熟练使用word、excel、ppt软件
    - 能写出简单的计算器，接受用户输入的+-x/运算
    - 掌握基本的沟通技巧

下面详细讲一下怎么做，其实很多人都比较讨厌那种讲空洞理论的书，me too。既然我们讲技能型的知识，就讲一下如何去实践。

word、excel、ppt 就不需要详细讲了，但是这 3 个工具会对你未来的发展有很的影响，尤其是 ppt 能力，写一个打动人心的 ppt，是你赢得赏识，快速晋升的法宝。下载一个金山 wps，练习一下 word 的排版，excel 的多行计算，制作图表，做几个简单的 ppt。幼儿园期间仅要求熟练的使用，不要求精通各种高级技能，后面会逐步要求提升。

下面几个是给你预习的材料，有余力的同学可以预习一下：

- 《别告诉我你懂PPT》思维导图读书笔记
- 《说服力：工作型PPT该这样做》思维导图读书笔记
- 《说服力：工作型PPT应该这么做》PPT制作七步法
- 《写给大家看的PPT设计书》思维导图读书笔记
- 《PPT，要你好看》读书笔记思维导图
- 《乔布斯的魔力演讲》思维导图读书笔记
- Robin Williams《写给大家看的设计书》思维导图读书笔记
- 《PPT演示之道》思维导图

掌握一门编程语言，后端语言更好，例如：java，scala，php，python 等等，这个是有诀窍的，一般人我不告诉他。以 java 为例，

1. 先上网搜索相关的入门书籍推荐，会找到比较多的人推荐，整理成列表<thinking in java>,<head first java>,<java 核心技术>，<java从入门到精通>等等。
2. 根据推荐度进行筛选，如果这时候咨询一下有经验的前辈，会得到比较中肯的答复。我推荐<head first java>。
3. 下载编辑器，例如eclipse，这个一定要下载，别听某人说某牛人用记事本编写了niubility的程序。那不是牛人，那是傻x。我的感觉就是，砍柴的时候有砍刀不用，非得用手，智商着急。
4. 用2天时间，先把书看一遍，建立一个宏观概念。java到底可以干什么，有什么东西要学，此时不要纠结很多看不懂的东西。
5. 开始每章突破，每章看完之后，自己敲所有的代码。切忌去网上下载了执行，而且尽量每个代码敲5遍（这个不信就算了）。此时如果有特别不明白的就问一

下前辈，如果没有可以去看视频。当然你看见有些后面才会讲的内容就不要纠结了，能执行就行。尽量高速的完成每一章，纠结太久是错误的，每章1-2天。

6. 做一个计算器。

掌握一项技能，沟通。"谁不会沟通啊，这还用学"有人暗骂。我只想说"呵呵"。说话大家都会，但是沟通不只是说话，而是把话说的让别人明白。好好学吧，你会发现这太有用了。怎么学呢？

1. 同java学习，我就是喜欢你这样能够举一反三的学生。
2. 但是这个要找人练习，直至做到无意识的使用沟通技巧，即变为被动技能。
3. 顺便推荐一些相关的资料
    1. http://www.write.org.cn/category/psychology/communication-skills

补充：前面学到的东西，后面阶段有可能会提到，有可能不会提到，但是默认你会自己安排时间不断的去练习。

架构师速成 5-小学 - for5million 的专栏 - 博客频道 - CSDN.NET

很高兴你很快的进入了小学，小学的东西会让你更加的耀眼。

- 阶段：小学
- 学时：2-3个月
- 升学标准
    - 能自己制定目标及计划，get thing done。
    - 可以轻松制作一个让你身旁人惊叹的ppt
    - 能做一个简单的网站（或者客户端软件），数据能保存到数据库。

实践经验干货来了。

先说 ppt 吧，这个上一期已经讲了，如果你 ppt 做到出神入化，基本不需要做架构这么苦逼的事情了。因为你很容易成为老板的心腹，军师，走上人生正道。作为一个苦逼的小学程序员，很羡慕吧。那就再努力学一下，不用多久,你就会升职加薪,当上总经理,出任 CEO,迎娶白富美,走上人生巅峰。想想还有点小激动呢,嘿嘿！

- 《别告诉我你懂PPT》思维导图读书笔记
- 《说服力：工作型PPT该这样做》思维导图读书笔记
- 《说服力：工作型PPT应该这么做》PPT制作七步法
- 《写给大家看的PPT设计书》思维导图读书笔记

- 《PPT，要你好看》读书笔记思维导图
- 《乔布斯的魔力演讲》思维导图读书笔记
- Robin Williams《写给大家看的设计书》思维导图读书笔记
- 《PPT演示之道》思维导图

再说 gtd，这个是你人生的利器，掌握了他，基本就会比同龄人花更少的时间，赚更多的钱，升更快的职，玩更多的

–

–

–

–

-，Dota。想歪了的，先站一旁反省一下。

用比较通俗的说法说一下，人生就像一次旅行，说走就走。有人坐飞机（x 二代），有人坐豪车（还是 x 二代），在这看我文章的人，肯定不是什么二代，所以只能靠自己的双脚。小 A 和小 C 都想靠自己的双脚来一次说走就走的旅行，小 A 真的说走就走了，啥都不带就走了。小 C 不着急，先找来地图，确定要去北京（为啥是北京啊！），然后查了一下大约多少公里，每天走多少公里，需要带什么东西。准备完毕，小 C 也出发了。那问题来了，挖掘机哪家强？谁先到北京？

有人这时候就要开始抬杠了，争论了，辩论了，我只想说别扯蛋。有闲功夫扯蛋的，都不适合做架构师。

gtd 就是目标+规划+执行。怎么掌握呢？

1. <小强升职记>看10遍。跟你身旁最亲的人讲2遍。
2. 推荐一个工具，绝对没有收钱的推荐。doit.im，免费的，每天都用，坚持1个月。
3. 找你的前辈帮你提出目标的建议。
4. 坚持执行你的计划，坚持！

怎么做一个网站或客户端呢？

我就以网站为例讲解一下，因为网站更难，客户端更简单一些。

1. 快速掌握语言
   html,css,javascript的基础知识，这些因为已经有了之前的基础，花费时间应该不多，具体方式参考java学习。不要求做到精通，仅要求掌握。
2. 快速掌握mysql的安装，使用，掌握sql语句。具体方式参考java学习。

3. 掌握一个web容器，如tomcat，jetty，学会如何启动，关闭，部署。
4. 综合起来进行web开发
    1. 找一本做web网站的书或视频（针对你会的语言找），速度过一遍，
    2. 下载demo跑起来。
    3. 修改前端，看看改了之后的效果。
    4. 改后端代码，打印一些日志。
    5. 再看配置文件（xml,properties,ini)之类的，搞明白每个部分的作用。
    6. 找一个从前到后例子，自己实现一遍。至少实现5个类似的功能。
    7. 自己从头到尾创建项目进行开发，直到跑起来为止。

简单吧，请抓紧时间完成你的学业。

www.100j.top

架构师速成 6-初中 - for5million 的专栏 - 博客频道 - CSDN.NET

- 阶段：中学
- 学时：6-12个月
- 升学标准
    - 学会如何思考，读每本书都能整理思维导图，能使用思维导图思考。
    - 学会知识整理，使用知识管理工具整理自己的知识。
    - 掌握设计模式，可以设计一套开发框架，使用此框架开发一个产品，而且新人也可以快速使用此框架进行开发。
    - 掌握软件设计理念，有一套自己的设计、开发思路，并使用此思路完整的完成一个产品设计。
    - 精通linux

中学自然要学习的东西更多，不知道你能不能坚持下来，如果不能坚持，请放弃这个职业。

思维导图又一个神器，好好掌握，后面单独一个篇幅讲解。
知识管理工具，wiz(为知)，evernote等，后面单独一个篇幅讲解。
linux 作为一个牛逼开发人员，不得不面对。
小学你已经学会如何从0搭建一个简单网站，那么你已经可以找到一份工作。找到工作后，你肯定会被安排做更大的网站。

这时候你应该沉浸在学新知识的快乐中，抓紧一切时间学习你身边可以学到的东西。同时你需要思考，如果从零开始做你目前的这个工作，应该怎么做？你会想到下面这些：

- 需要人员管理
- 需要了解需求

- 需要制定计划
- 需要进行设计
- 需要实现
- 需要测试
- 需要发布
- 需要收集反馈

或者更多的事情。
中学其实就希望你能学会整个软件的开发过程，形成自己的一套设计、开发思路。另外有思路还是不够的，希望你能沉下心把做的事情简化掉，对已有的轮子进行封装，让你从0开始做一个网站时易如反掌

架构师速成 7-高中 - for5million 的专栏 - 博客频道 - CSDN.NET

- 阶段：高中
- 学时：6-12个月
- 升学标准
    - 掌握速读技能，能迅速吸收海量知识，取其精华，弃其糟粕。
    - 学习你涉及知识的底层标准及协议，能够讲解标准和协议产生的原因。
    - 搭建持续交付平台，能够持续自动化部署，devops。
    - 能够带团队顺利完成一个任务的开发，团队有自己的规范、流程、风格。
    - 进行性能测试。能够发现系统瓶颈，并进行调优。

经过初中的锻炼你已经有了做框架的能力，设计思路，而且掌握了思维导图和知识管理 2 大利器。你现在的技术实力已经足够可以尝试带领一帮兄弟进行独立的开发，那就好好争取这样的机会。

当然不要忘了前面已经学习的超级能力，ppt 及时间管理。如果你一直在磨练，那现在你应该也可以很得领导赏识，成为领导的御用 ppt 达人。而且你也已经有了自己的短期，中期目标，有了朝着目标前进的计划，既然有了足够的规划能力。那你争取带队的机会应该比较 easy。

上面后 3 项应该是你成功一个 leader 才能实施的，不过你可以充分的学习前 2 项，第 3 项也涉及到很多知识，可以先行探索。等你有机会就可以马上实施。

www.100j.top

架构师速成-目录 - for5million 的专栏 - 博客频道 - CSDN.NET

天地会总舵，陈近南给了韦小宝一本武功秘笈，韦小宝说：“嗯？这么大一本我看要练个把月啊！”
陈近南说：“这本只不过是绝世武功的目录，那边才是绝世武功的秘笈！”

这就是架构速成的秘笈目录

架构师速成 4.1-幼儿园要学会如何学习（转载自 36 氪） - for5million 的专栏 - 博客频道 - CSDN.NET

在这个资讯爆炸、新东西层出不穷的时代，学习是件终生的事业。书到用时方狠少，很多时候我们总是后悔当初没学，然后抱怨时间不够多，最后开始反思—时间对每个人来说都是公平，所以我们的问题应该是如何才能更快地学会东西？也许 Sean Kim 的这篇学习终极指南可以参考。

如何缩短学习曲线这个课题已经被研究了几十年。其中讨论到的一些加速学习进程的核心原则这份指南都会讨论到。利用好这些原则，无论你学的是语言、乐器等等，都可以学得更快。

> 在当今我们生活的这个时代，如果你想过上一段非凡的人生，就得掌握快速学习这项本领。 —Anthony Robbins

不做重复工

我们学东西有个倾向是试图自己掌握，但是有个学会的人帮助可以节省你很多的时间和精力。

想想自己以前是怎么学语言或技能的。是不是一开始走了很多弯路？有了你的帮助，后面的人再学是不是就可以避免很多你犯过的错误？

要想更快掌握东西，第一步就得去请教这个领域做得最好的人，然后按照他开辟好的路走下去就行了（等你到一定水平了再走自己的路）。

正如 Tony Robbins 所言：

> 许多伟大的领袖已经证明，掌握任何技能、策略或目标的最快方式就是照着前面的人开辟的路走下去。如果你能找到已经拿到你想要的结果的人并且采取跟他们一样的行动，你也可以获得同样的结果。

> 你的年龄、性别、背景如何都没有关系，照着做让你可以快速跟踪并以短得多的时间实现你的梦想。

在今天这个时代，你几乎从通过书本、博客、培训视频、咨询或网上获得任何解决方案。

毕加索曾说过：

> 好的艺术家抄袭 伟大的艺术家剽窃。

（这话乔布斯也剽窃过。:)）

技能解构

破解学习曲线的下一步是将要掌握的技能解构为基本要素。把这些要素分解，然后找出最重要的首先进行练习。

这个做法源自帕累托的 2/8 原则：即用 20%的努力获得 80%的结果。

这个原则几乎到处都可以体现：

- 商业（80%的销售来自20%的客户）
- 员工效能（80%的成果出自20%的员工）
- 快乐（80%的快乐来自20%的关系）
- 旅游体验（旅游80%可用20%的精彩时刻概括）

下面这张图是 2/8 原则的另一种表现形式（横轴为努力，纵轴为效果）：

2/8 原则的要点是只有很少的东西会对我们的生活（包括学习）产生很大影响。

那么我们的目标就是把这能产生 80%结果的 20%的学习材料独立出来。

实际上一些快速学习专家早已采取这种意识形态。

Josh Kaufman 在 TED 演讲中提出，掌握一项技能并不需要 1 万小时的练习。关键是前面的 20 小时要学会最重要的子技能获得最大的效应：

> 动作与认知技能获取领域的若干研究表明，新技能前面几个小时的练习对表现产生了最大的影响。

> 一般模式是这样的：一开始时你很害怕，但你学会技能最重要的部分之后改进就会非常快。

相关学习：帕金森定律。

举例：

- 学习一门乐器必须知道少数最常见的和弦，这样80%的曲子都能弹。

- 学习新外语应该聚焦在最常见的1500到2000个字，这样80%的文字都能懂。

一心不能二用

多任务是我们在连续通告和移动应用时代培养出来的一种罪恶快感。每 10 分钟就要查查邮件、刷刷微博、看看微信或者跟经过的同事聊聊天等都是表现。

但是多任务处理会成为加快学习的最大障碍。

可以想想自己的电脑。

浏览器打开了 20 个以上标签页时，计算机就会开始变慢，然后执行后续动作就会放缓。

横轴：集中注意力的时间，纵轴：完成需要的总时长

研究表明，人如果分心后平均需要 **25** 分钟才能让心思回到手头的工作上。

更值得注意的是，加州大学的 Irvine 发现，自己的同事往往只干了 **11** 分钟就开始分心。

长期专注也一样。我们很多人都无法腾出 6 到 12 个月以上的时间去学习一项技能，因为总会有新项目、新想法、新爱好冒出来。

当我们决定把注意力转移到新的事情上面时，往往很难再对之前技能保持同样的激情和专注。

在解构出能产生最大结果的子技能之后，就得把精力集中在改进这些子技能上，在掌握这些子技能之前不要学其他任何东西。

重复练习

这是我们大多数人最纠结的部分。

是的！更快掌握任何东西的关键是练习。

这需要频繁持续地反复练习同一种技能，直到不假思索就能下意识地做到。

全世界表现最好的人都理解这个学得更快成为最佳的"秘密"，但是却很少会讨论它的重要性，因为这听起来并不性感。

专家级表现主要是专家级练习的结果，而不是因为与生俱来的的天才。

佛罗里达州立大学的 K. Anders. Ericsson 指出：

> 大家认为，专家表现跟一般表现之所以有质的不同，一定是因为专家有与生俱来的天分。这种看法让科学家忽视了按照普通心理学定律原则去检视专家的表现。

寻求即时反馈

1960 年当时还还默默无名的甲壳虫乐队去德国汉堡的场子演出。

报酬低、音效差、听众不欣赏就是当时他们的感受。如何避免呢？

不停地练习然后马上听取反馈迫使他们变得越来越好。

Macolm Gladwell 在《异类》中总结认为，这是甲壳虫登上音乐殿堂巅峰的关键不同。

哥几个并不只是呆在车库里面埋头练习，他们同时还努力站在现场观众面前，无论是喷口水还是建设性意见，均能获得第一手的反馈。

随着他们水平的提高，观众要求他们演出的时间也加长了。到了 1962 年，他们每晚的演出时间提高到了 8 小时，而且 1 周 7 天不间断。到了 1964 年他们在国际舞台崭露头角时，甲壳虫乐队已经完成了 1200 场音乐会。

相比较而言，今天的大部分乐队整个职业生涯的演出都达不到这个数字。

坚持

不幸的是，大多数人在到达赛斯·高汀所谓的"低谷（The Dip）"期间或之前就已经放弃努力。

高汀说，尽管知道适时学会放弃很重要，但是很多有可能取得成功的人正是没有选好放弃的时机。

> 你没能成为全世界最好有 5 个原因：

- 你时间用完（然后放弃）了。
- 你钱用光（然后放弃）了。
- 你害怕（然后放弃）了。
- 你不认真对待（然后放弃）了。
- 你没兴趣（然后放弃）了。

心理学家也研究过所谓的转变周期。

这是指人经历变故（如悲伤事件）或新奇事件（如学新东西）时的心理变化周期过程。

如图所示，我们一开始学新东西时都会经历过精神愉快。这也是我们看社交媒体的通知会上瘾的原因，因为每次都会释放多巴胺。

不过，随着蜜月期渐逝，我们就会进入“低落”期，进展开始放缓或者减少。这段时间也是最多人放弃的。

把这个周期进行可视化很重要，因为这会让你明白这是黎明前的黑暗。如图所示，熬得过这段时间的人很快就经历了新的突飞猛进。

总结：

- 以专家为榜样，不必自己重新探索
- 解构技能，找出实现80%效果的那20%
- 不要一心二用
- 练习练习再练习！然后获得即时反馈
- 坚持，不要在低谷期放弃

架构师速成 4.2-幼儿园要学会如何高效学习 - for5million 的专栏 - 博客频道 - CSDN.NET

《如何高效学习》，这本书的作者是 scotthyoung，最知名是的 1 年内自学完成 4 年麻省理工学院计算机科学的 33 门课程，同时也写了一个学习方法的 Blog,他使用费曼技巧来加强理解和学习。
费曼技巧有很多种理解，最简单的是：

1. 拿张白纸；
2. 在白纸顶部写上你想理解的某想法或某过程；
3. 用你自己的话解释它，就像你在教给别人这个想法。

最要紧的是，对一个想法分而化之，虽然可能重复解释某些已经弄懂的知识点。但你最终会到达一个临界点，无法再解释清楚。那里正是你需要填补的知识缺口。为了填补这个缺口，你可以查课本、问老师、或到互联网搜寻答案。通常来说，一旦你精准地定义了你的不解或误解，找到确切的答案则相对而言更轻松。

另一种稍微复杂一点的说法:

第一步 - 选择一个你想要理解的概念，然后拿出一张白纸, 把这个概念写在白纸的最上边.

第二步 - 设想一种场景，你正要向别人传授这个概念，在白纸上写下你对这个概念的解释, 就好像你正在教导一位新接触这个概念的学生一样. 当你这样做的时候, 你会更清楚地意识到关于这个概念你理解了多少, 以及是否还存在理解不清的地方.

第三步 - 如果你感觉卡壳了, 就回顾一下学习资料，无论何时你感觉卡壳了, 都要回到原始的学习资料并重新学习让你感到卡壳的那部分, 直到你领会得足够顺畅, 顺畅到可以在纸上解释这个部分为止.

第四步 - 为了让你的讲解通俗易懂，简化语言表达，最终的目的, 是用你自己的语言, 而不是学习资料中的语言来解释概念. 如果你的解释很冗长或者令人迷惑, 那就说明你对概念的理解可能并没有你自己想象得那么顺畅 -- 你要努力简化语言表达, 或者与已有的知识建立一种类比关系, 以便更好地理解它。

http://jingyan.baidu.com/article/dca1fa6f59896bf1a440528d.html 费曼学习法的详细讲解

我看了这本书的中文版，发现还有一个很重要的概念，就是通过联想不断的建立你掌握知识的关联，这个跟思维导图的逻辑有些想通。举个完整的例子，程序相关的吧：

缓存 其实是一个攻防战，用户是进攻方，架构师防守方，第 1 层阵地就是用户的浏览器端，第 2 层是 cdn，第 3 层是 nginx 等 web 容器静态缓存，等等 这是讲解。

当然缓存还会跟其他的知识进行关联 http 协议、db 除了缓存还可以进行分布式等等。你发现整个大领域的知识都是相关的，你不停的关联，你的知识体系就会越牢固。

如果这个学习法，我早些学到，哇哈哈！

架构师速成 4.3-幼儿园要学会查找资料 - for5million 的专栏 - 博客频道 - CSDN.NET

有很多人整天用电脑，但是碰到问题就不知所措，好像从来没有用过互联网一样。

互联网给了我们最多的知识，但是也给我了最大的诱惑，也给了我们最多的垃圾。如果迅速找到自己想要的知识，取其精华，弃其糟粕是一个非常有用的学问。还是举例子：

比如我要学习一门新的语言 scala，

- 列出对自己帮助比较大的几种资料或途径
    - 书籍
    - 视频
    - 别人学习的心得和经验
    - 相关论坛和答疑群、博客，有疑问可以咨询
- 光撒网，搜集所有相关资源
    - 书籍

      从百度文库或者百度网盘，另外还有一个资源网站 http://www.lefashn.com/other/resource.htm
    - 从豆瓣搜索相关书籍和别人评价
    - 视频 从优酷或百度网盘，或it相关的视频网站
    - 心得或经验， 就需要baidu或google 看有没有相关人的博客或论坛
    - 论坛或答疑群、博客 同上
- 去粗取精
    - 书籍

      前期肯定找入门和基础的，口碑特别好的，如何排除就看别人心得、豆瓣评价、销量
        - 快学scala
        - scala编程
        - scala程序设计
        - effective scala
        - twitter的入门教程
    - 视频 也粗略的看一下目录，找一套就可以
    - 论坛和博客都找最近有更新，比较活跃的
- 速读
    - 迅速的过一下所有的书籍，明确知识概况，确定重点难点。
    - 制定学习计划
- 学编并行
    - while(!ok){
    - 先看书
    - 敲代码
    - 如果有视频就看视频，可以没有
    - }

架构师速成 5.1-小学 gtd 进阶 - for5million 的专栏 - 博客频道 - CSDN.NET

人生没有理想,那和咸鱼有什么区别。

有了理想如何去实现，这就是 gtd 需要解决的问题。简单说一下 gtd 怎么做？

1. 确定你的目标，如果不能确定长期目标，至少需要一个2年到3年的目标。
    1. 目标必须是可以衡量的，不能模棱两可。
    2. 目标必须符合你的价值观，如果确定是否符合价值观，假设这件事给你10万，让你不要继续做，你说滚。
2. 针对你的目标，确定需要做的事情，脑子里想到的所有事情都放到收集箱，放空自己。
    1. 必须把所有的事情放进来，让大脑做到空无一物。
    2. 放到收集箱不行，必须确定有解决方案，否则大脑潜意识还是会纠结。
3. 收集箱的事情，进行分类整理，分成马上执行、项目、日程、下一步计划、将来也许
    1. 马上执行的事情，马上解决掉
    2. 项目必须进一步细化，细化成日程或者下一步计划
    3. 日程设定定时执行
4. 每天早上把脑子里的事情放到收集箱，同上整理一遍，然后过一下所有的下一步计划和日程，今天需要做的放到今日执行。另外过一遍所有的目标，看看有没有新的想法需要实施。
5. 将今天需要做的事情分清优先级，按照优先级逐个完成。
    1. 优先级，请按照四象限方法进行安排，一定要注意，不要被紧急不重要的事情占用时间，尽管我这样说了，但是还是有很多人整天被紧急不重要的事情占得满满的。
    2. 一旦确定优先级，就一定要按照优先级执行，不要被其他人打断。
    3. 如果有人要打断你，一定不要被打断，尽量完成手头的工作。如果非要打断不可，那一定要保存好断点，让你回来马上能够接上。
6. 晚上过一遍今天的工作，反省一下。
    1. 确定一下自己的计划是否完成
    2. 是否被人打断
    3. 今天有哪些做的不够好的地方需要改进
    4. 今天有哪些做的好的地方，需要坚持

有人生来就是当臭咸鱼的，但你不是，那就 get things done！

架构师速成 5.2-如何掌握综合性技能 - for5million 的专栏 - 博客频道 - CSDN.NET

买菜、洗菜、切菜很多人都会，但是把几种菜混在一起做成一道可口的佳肴就不是一般人能掌握的。

今天就讲解一下如何学会做一道好菜。

举个例子，做网站，其实跟做菜一样。

1. 先看菜谱，看需要哪些配料，做网站，当然需要html,css,js,java,mysql这些菜，当然还需要设计模式、面向对象、框架这些调料，eclipse这样的刀，另外还需要tomcat这样的锅。
2. 磨刀 基本工具要首先用熟，快捷键都记住了吧。
3. 洗菜
   主菜要多花些功夫，配菜只需要一点。html，javal算作前后端主菜，要反复洗脸。css，js，mysql算作配菜，稍作处理即可。
4. 放料 菜好了，料也不能少，配料要少而精，什么火候放都是学问啊。
5. 开炒
   准备工作都ok了，放入锅中，一起翻炒。翻炒时要不断的研究，仔细的观察，弄懂主次搭配，调料的作用，前后贯通。
6. 好菜出锅，如果不行就多来几次，一盘好菜就ok了。

学习其他综合性技能也是类似的思路，先根据菜谱，分别准备，有主有次，从前到后分析流程原理，熟能生巧。

架构师速成 6.2-知识什么时候才是你的 - for5million 的专栏 - 博客频道 - CSDN.NET

很多人上了很多年学，读了很多年书，他们其中的很多人压根就没有用上这些知识，我就是其中的一员。当时我还是化学课代表，化学牛人，高三的化学卷，别人做 1 个半小时，我只需要半小时，而且顶多会错 2 个选择。然而并没有什么卵用，我发现上了大学之后，化学、物理统统交回去了。所以学过!=你的知识。

另外我还看过很多书，然而并没有什么卵用，都忘了。所以看过!=你的知识。

参加工作以来，我用过的语言也有不下 10 种，然而并没有什么卵用，现在很多都忘了。所以用过!=你的知识。

那什么才算是你的知识呢？就像吃饭一样，你吃了很多，但是最终留下的才是你的，其他的都变成了大便。

你总结的思想，你掌握的技能，你可以讲授的东西，才是你的知识，才是你的价值。

前面的费曼学习法就是很好的总结自己知识的方法，思维导图是进行总结归纳的好工具，另外你要有自己的知识管理工具，类似"为知"之类。

知识要挑肥拣瘦，不要全盘接收。打个比方，你去吃自助餐，海鲜，肉类，菜类，甜点，饮料，水果应有尽有。有人就上来喝了 2 杯啤酒，吃了一个西瓜，后来只能干瞪眼看别人吃，自己喜欢吃的吃不下了。有人喜欢吃肉，来了就吃肉，吃的很尽兴，有人喜欢吃海鲜，就主攻海鲜，吃的也很 happy。

- 要有自己的目标，知道自己需要什么知识，专门去整理和学习这些知识。

还有人猛吃便宜的，但是很占肚子的，比如来了就啃面包，那还不如在家自己啃面包，何必来吃自助。

- 对于字典型知识，只需要知道，什么时候会用到他，我该如何去查就ok了，而不需要记住。不要因为字典型知识的学习占用过多的时间。

吃完了，要总结归纳，下次来怎么吃才爽，分享给后来人。

- 比如你爱吃肉，随着你多次吃自助，总结出规律，先吃牛肉，再吃五花，再吃鸡翅比较过瘾。总结下来，归纳下来，顺便给同样是吃货的朋友分享，碰撞。画一个吃自助攻略思维导图，或许更好。

不断的跟其他的饭店进行比较和串联，吃自助优势在哪，吃大酒店怎么样，一周 7 天该怎么吃。

- 把你的知识跟其他知识串联起来，说不定就可以出一本《吃遍中国》

一个喜欢的店，要多去才能显示你的真爱。

- 要不断的回顾和反思你总结的知识，归纳整理分类。

我都觉得我像个吃货了，减肥去了，100 个俯卧撑！

架构师速成 6.3-设计开发思路 - for5million 的专栏 - 博客频道 - CSDN.NET

面向对象，是一个伟大的设计思想，应该是软件开发史上的一次革命。

当然理解面向对象也很难，有好多人用着面向对象的语言，写着面向过程的逻辑，而且一写就是好多年。但是有高手，用 c 照样可以写出很牛的面向对象的程序。面向对象其实是一种思考问题的方式，重点如下：

1. 面向对象是用来反映显示世界的，而不是强行创造世界。
    - 这句话，说起来简单，但是做起来很难。现实世界中你绝对不会把狗腿，按在一个人身上，但是写程序的时候，你常常会创造出一个狗腿人。
    - 有人还会创造一些一些稀奇古怪的万能类，或者融合了n种物种的怪物。或者只有一条腿的狗。
    - 一定要记住，只有反映显示的才是长久的。
2. 是我的就是我的，不是我的就不是，不增不减。
    - 有时候有些属性，或者方法，不知道该如何放就随便放在一个类里，这就大错特错。
3. 真正的对象是有血有肉的，而不是只有一堆属性，或者只有一堆方法的怪物。
4. 按照现实世界的关系安排对象之间的交互。

怎样锻炼你的面向对象思考能力，很简单，每天把和你产生交互的人和物，写成代码，让他们和你的交互变成方法的调用。例如今天你坐车去了超市，买了一堆东西。抽象出 car，person，market，goods，cash 等等，用代码实现他。

设计模式的确是很好的，它们是前人给我们总结了的一些秘诀，这是国人最喜欢的了，对吧，所以记住什么时候使用他们。

有了上面的知识，就可以思考如何有一套自己的设计开发思路了。做任何事，都有一套最适合自己的方法，你要试着摸索出来，并固化到你的血液里。比如现在给你一个项目，做一套简单的进销存系统，需要你设计出来，并进行编码，你会怎么办？

想一想？有很多种方式去进行设计，但是你总得有你的一套，所以你自己总结吧，我不会把我的秘诀告诉你的。

架构师速成 6.4-开发框架 - for5million 的专栏 - 博客频道 - CSDN.NET

经历了很多公司，看过好多代码。传统行业公司普遍都有自己一套统一的开发框架，封装的非常傻瓜化，门槛极低，便于不同的项目快速开发上线。比如有一个银行业框架封装到开发人员只需要在界面上拖拽就完成大部分工作，然后在拖拽好的模块里面添加一些业务代码就 ok 了。用友，华为也是类似。当然传统行业缺点是，一套框架用 n 年，老掉牙了也在用，有时候跟不上时代。

小的互联网公司有时候就不太注重，基本就以快速上线为主，草草完成功能就可以了。阿里巴巴在开发框架封装方面做的也不是很好，跟发展历史有关，也跟商业驱动有关，这个就不多谈了。

开发框架一说，有做 java 就认为 spring 就 ok 了，或者 spring 我简单封装一下就 ok 了。大错特错，做框架的目的是把与业务无关的所有细节隐藏掉，让开发人员以最快最简单的方式完成业务。拿做一个网站来说，抛开架构，仅就单机开发一个 java 网站，需要做的事情就很多。

我觉得起码需要规范或封装的部分：

1. 后端
    - 后端编码规范
    - 分层规范 controller service dao。这个非常重要
    - 基础工具类 xxUtil
    - Ioc aop spring可以实现
    - mvc spring可以实现
    - 数据库 数据源 事务 这个可以通过spring实现
    - 日志记录 通过spring aop记录耗时日志，参数日志等
    - 模板封装 jsp或vm等
    - 数据校验
2. 前端
    - 前端js/css编码规范
    - 前端统一字体、样式、控件规范
    - 前端模块化开发 例如:sea.js
    - form 校验
    - ajax
    - 无js化，参见angularjs jui(国内前端框架)
3. 前后端结合
    - 分页
    - 异常处理及展示
    - 页面跳转规范或ajax load规范
4. 业务
    - 单点登录

- 权限判定
- 评论
- 等等

总之有很多，一定需要封装到让一个刚毕业的学生在 2-3 周就可以熟练开发。而且框架代码由专门的框架开发人员进行升级维护及培训，保持框架的活力。

封装框架可以增强对设计模式及面向对象，aop 的理解，只有当你成功的完成了一个框架的封装，你才可以继续往架构的方向前进。

当然对此有开发人员颇有争议，我见过很多程序员，当他从一个公司离开后，发现脱离了公司的框架自己什么也不会。这个说明 2 个问题：

1. 他们公司框架做的好，让他了解的细节足够少。
2. 他没有合理的利用时间，去把框架掌握到手。

跟阿里的同事也有讨论，结果大家都比较认同：

1. 一个团队必须有自己的开发框架，虽然再去为阿里全集团做一套统一的框架很难，但自己团队内部必须有一套框架，可以帮助团队快速开发。其实支付宝是有一整套开发框架的，这是因为支付宝沿袭自金融行业。
2. 另外要组织全团队的框架培训，让每个人都对框架有足够的熟悉，甚至提出改进意见。
3. 通过框架节省下的时间，可以研究更先进的技术，反哺框架本身。
4.

架构师速成 7.1-速读、速记 - for5million 的专栏 - 博客频道 - CSDN.NET

速读速记一直是很多人梦寐以求的技能，想象一下，别人看一本书 2 天，你看一本半天，而且记得比别人清楚，这是一件多么开心的事情。当然有人不相信有这样的技能的存在，我只能说呵呵。如果你看过记牌或者任意图像的记忆分辨，你会吓哭的。

我觉得我好像可以速读，但是速记练过一段时间，没有坚持下来，所以速记算是没有的。不过当时练习记牌，一副牌洗乱了，我能看 2 遍记下来。后来觉得对于记录数字比较有用，而且需要练习图像记忆的思维，坚持不下来就失败了。

那么说一下速读吧，这个好像是我小时候偷看小说炼成的技能。当时家里人不让看武侠小说，而且也不会给钱买小说。我的表哥有时候会去租书看，但是一般他看完就需要还，我就要抓紧看，而且要偷偷看。结果无意中就炼成速读技能，我觉得重点如下：

1. 明白自己看书的目标，要得到什么
    - 比如武侠小说，就是为了进入武侠世界，享受江湖
    - 比如技术书籍，比如java入门，那就需要掌握编码技能。
2. 根据目标，先大体看一遍目录，确定重点
    - 这个适用于技术书籍
    - 武侠小说看了目录也没啥用，不需要确定重点
3. 开始快速泛读一遍
    - 不要发出声音
    - 每页设定固定时间，必须翻页，这个电纸书有优势了，自动翻页，越练越快
    - 不要用手指每个字
4. 重点部分精读一遍
    - 武侠小说，除非特别精彩，否则看一遍就行
    - 找到重点部分，而不是所有的都精读
5. 总结xmind
    - 做到能自己讲一遍给别人

速记，大家如果有兴趣可以学习一下超右脑快速记忆，这个我就不展开了，因为我只会记牌。

另外大家可以搜索一下 战隼的学习探索

http://www.read.org.cn/html/category/memory

架构师速成 7.3-devops 为什么很重要 - for5million 的专栏 - 博客频道 - CSDN.NET

---

evops 是一个很高大上的名字，其实说的简单点就是开发和运维本身就是一个团队的，要干就一起把事情干好。谁出了问题，网站都不行。作为一个架构师，必须要 devops，而且要知道如何推行 devops。

首先要自动化，举个阿里的例子，阿里通过 aone 系统来实现半自动化部署：

- 开发人员开发代码先自测通过后，提交代码到git。
- 在aone中一键部署到日常环境。部署是自动化扫描依赖冲突，系统安全等问题。
- 测试接到部署成功的通知，进行测试，如果测试通过，则审批通过，可以线上发布。

- 线上运维人员一键部署到线上，部署可以分配部署，进行A/B测试。如果出现问题可以一键回滚。

当然这里面牵扯到了很多角色，其实如果是一个公司，这个团队应该一体的，不分彼此。团队中每个角色都是程序员，每隔一段时间需要轮换岗位。

其实不只是部署需要自动化，测试、扩容、监控、分析等等都需要自动化。当然这需要根据你公司的实际情况进行实践，最先需要的其实也是部署自动化，如何实现呢：

- 使用开源产品 git,hudson,ansible或者puppet
- 制定相应的规范，自动化部署是有规范的，每个人都必须遵守，否则故障就会马上出现
- 进行定制化开发，使整个系统形成一个整体，
    - 上面的开源产品并不是一个整体，需要通过自己编码使其成为一套系统。
    - 涉及web系统开发，底层脚本编写，打包规范制定
- 不断发现重复劳动，使其自动化
- 要有对应的监控系统，及时发现异常

有了这一个自动化系统之后，你可以考虑一下后面的产品安装、测试、扩容、监控、分析系统的自动化。

另外持续改进也是 devops 的核心，要有发现改进的眼睛。只要有事情重复做 3 次以上，就考虑是否需要自动化。

如果你不进行 devops，随着业务的增多，系统的增多，最终还是需要进行从头到尾的改造。不如一开始建设的时候就预先设计考虑好，其实这些东西如果你懂的，开始的时候就做好，并不浪费太多时间，反而能节省大量的时间，何乐而不为。

架构师速成 7.2-为什么要学习协议、规范 - for5million 的专栏 - 博客频道 - CSDN.NET

我们在开发过程中会碰到很多协议，标准规范类的东西，比如 http 协议，比如 javaEE 标准等等。有人觉得这些很枯燥，根本不需要去关注，但是恰恰相反，这个很重要。

- 协议和标准规范，是一个时代人类智慧的结晶，汇聚了很多牛人和专家的智慧。你需要从中吸取他们设计的理念，对程序的设计也会有很大的帮助。
- 理解协议对于你的故障排查很有帮助，比如一个http请求出错，如果了解协议很容易判断，是浏览器端还是服务器端问题，可以很快定位问题。

- 理解协议可以自己开发对应的实现，帮助你快速建立用户群，比如你可以开发开放的http协议服务，鉴权遵循oauth2标准。这样无需提供额外的说明，大部分用户都可以很快接入。
- 底层协议和规范是最基础的，你理解这些之后，有新技术出现，你会发现似曾相识。
- 可以举一反三，实现自己的开放协议，互联网时代，
    - 1等公民是建立规范和协议的人
    - 2等公民是提供服务的人
    - 3等公民是开发软件的人
    - 4等公民是卖硬件的人

顺便提供一个 web 网站快速定位故障的秘诀，如果出现了问题，没有很明显的提示帮助你快速定位，可以按照下面步骤快速确定问题：

1. 确定是前端还是后端的问题
    1. 使用chrome浏览器等直接可以显示http请求和返回结果的浏览器，如果没有那就用fiddler这个抓包工具
    2. 查看请求发出信息，确定你希望发出的数据都在请求中发出，如果跟预期不一样，那就需要排查前端。
    3. 查看respone结果的数据，看是否有返回或者返回的数据，是否有错误码。一般情况下，请求按照预期发出，基本都是后端问题了。
2. 定位前端问题，根据请求基本就可以确定，无非就是：
    1. form属性值错误
    2. url错误
    3. 请求方式错误等等
3. 定位后端问题，直接debug就可以了
    1. 传入参数解析是否正确
    2. 处理逻辑是否正确
    3. 请求转向是否符合预期

这个我一般不告诉别人。

架构师速成 6.5-也谈设计模式 - for5million 的专栏 - 博客频道 - CSDN.NET

回头来回顾之前写的文章，发现初中阶段的内容缺了一块很重要的内容就是设计模式。设计模式是程序员的另一个 G 点，无论吹牛，还是面试，还是设计都会秀一下设计模式，这也奠定了设计模式不可动摇的地位。

我主要讲一下设计模式如何学习，哪些虚的我就不讲了。

1. 找2本书就可以了，大话设计模式和head first设计模式
2. 速读2遍，对设计模式有总体的概念
3. 整理思维导图，先把印象最深的5个列下来，用你的语言通俗易懂的给别人讲清楚，另外需要重点侧重于什么场景下使用。
    - 举个例子，代理模式，怎么跟人讲呢，http://yangguangfu.iteye.com/blog/815787 看看这篇文章，很黄很暴力
    - 当然你还要侧重一下怎样用，在需要用比较通用和复杂的对象指针代替简单的指针的时候，使用
      Proxy模式。下面是一些可以使用Proxy模式常见情况：
    - 远程代理（Remote Proxy）为一个位于不同的地址空间的对象提供一个本地的代理对象。这个不同的地址空间可以是在同一台主机中，也可是在另一台主机中，远程代理又叫做大使(Ambassador)
    - 虚拟代理（Virtual
      Proxy）根据需要创建开销很大的对象。如果需要创建一个资源消耗较大的对象，先创建一个消耗相对较小的对象来表示，真实对象只在需要时才会被真正创建。
    - 保护代理（Protection
      Proxy）控制对原始对象的访问。保护代理用于对象应该有不同的访问权限的时候。 智能指引（Smart
      Reference）取代了简单的指针，它在访问对象时执行一些附加操作。
    - Copy-on-
      Write代理：它是虚拟代理的一种，把复制（克隆）操作延迟到只有在客户端真正需要时才执行。一般来说，对象的深克隆是一个开销较大的操作，Copy-on-
      Write代理可以让这个操作延迟，只有对象被用到的时候才被克隆。
4. 直到你讲的一个不懂开发的人也很清楚的时候，开始整理下5个。
5. 整理完所有的之后，理清所有的模式之间的联系和区别，例如
    - 适配器Adapter
      为它所适配的对象提供了一个不同的接口。相反，代理提供了与它的实体相同的接口。然而，用于访问保护的代理可能会拒绝执行实体会执行的操作，因此，它的接口实际上可能只是实体接口的一个子集。
    - 装饰器模式Decorator：尽管Decorator的实现部分与代理相似，但Decorator的目的不一样。Decorator为对象添加一个或多个功能，而代理则控制对对象的访问。
6. 整理到1个思维导图上，做到只要看到这份思维导图，你就能讲出所有。
7. 试着重新画这副思维导图，多画几次，直到你很容易的就能画出。
8. 尝试用代码描述你日常见到的所有可以使用设计模式的地方，每天做1个例子。

ok 不出半个月你已经是设计模式高手了。请让我膜拜一下奥！

架构师速成 7.4-架构师为什么要带团队 - for5million 的专栏 - 博客频道 - CSDN.NET

有人说架构师明明只需要做架构，干嘛要扯出来带团队，带团队不是项目经理或者 CTO 之类的管理人员干的事情吗？

其实这个是一个误区，架构师其实是一个全栈的特殊人物，应该项目开发的所有的环节和角色都有深入了解，尤其是带过团队对你的帮助会更大。那种只做架构，而且仅会做架构的架构师，是大公司畸形的产物，在我看来，不太接地气。大公司人员体系庞大，分工明确而且细致，技术只是负责技术就好了，管理自然有专门的管理人员来做。我简单列举一下架构师带团队的优势：

1. 架构设计时会从整个项目的角度考虑
    - 开发人员使用更方便
    - 测试人员测试更容易
    - 前后端开发分离
    - 运维人员如何自动化运维
    - 项目管理人员如何把控，架构如何符合项目进度
    - 如何实现项目1到n的复制
2. 跟各个角色人员讲解、沟通、培训更流畅
3. 程序员往往希望被比自己技术牛的人领导
4. 可以为程序员进行定制化培训，符合项目要求
5. 架构更接地气，而不是高高在上
6. 创业更有优势，CTO就是你

当然如果在大公司就算了，不要便宜都被你占了，考虑一下别人的感受。当然如果不是特别大的公司，一定争取带团队的机会，否则有很多事情很难实现。

- 框架、架构理念推广
- 定制化开发工具及开发流程
- 人员培养
- devops
- 实践先进的技术理念

所以尽量争取，你必须先是 team leader，然后才能成为架构师。

架构师速成 6.6-知识的收集整理学习 - for5million 的专栏 - 博客频道 - CSDN.NET

知识如何学习前面已经讲了 2 节，这节主要讲知识的整理和沉淀。

其实我之前也一直没有好好的思考过这个问题，今天在整理自己的 wiz 知识库的时候突发灵感，所以有了这一节。

其实知识获取的过程分为搜索->收集->整理->精化->应用->分享，前一部分跟时间管理的收集也很相近吧。知识获取的思路适用于有目的的知识收集和日常的备忘性的知识收集。当然你随机收集一些资料记录下来其实效果并不是很理想，重要的是你要有目的的学习才能最大的发挥你的心智以及潜意识。当你主动要学习一项知识时，你的潜意识会主动关注一切相关数据和知识，举个例子，当你要学架构师相关知识时，当你在随便浏览网页时，如果出现架构师这几个字时，你会一眼就发现，而且会特别关注，这就是潜意识。当然今天不是讲潜意识，而且你的主动意识。

我说一下我是怎么进行管理的，其实我发现需要 2 套工具来进行知识管理，1 是 wiz 类知识收集工具，2 个云盘用来存放大的文件，如 pdf，视频等等。但是 2 者的目录结构是完全一致的，这样便于管理和分享。

先抛砖一下我的目录结构

├─01 收集箱
│ ├─团队分享
│ └─股票
├─02 有效资料
├─03 我要学习
│ ├─01 马上学习
│ └─02 即将学习
├─04 我的能力
│ ├─01 学习能力
│ │ ├─01 阅读技巧
│ │ ├─02 记忆力
│ │ └─03 学习方法
│ ├─02 思考能力
│ ├─03 沟通能力
│ └─04 管理能力
│ ├─01 时间管理
│ ├─02 知识管理
│ └─03 财务管理
├─05 我的知识
│ ├─01 家庭生活

| ├─02 健康养生
| └─03 工作必备

我说一下，如果我要学一门新的知识我会怎么办，比如要学习 scala 吧，我会先搜集入门资料：

1. 去搜相关的论坛，豆瓣，收集资料，有很好的经验、心得就用wiz抓下来，放到收集箱
2. 确定要看的书，确定好之后下载电子版，电子版放到云盘里面，如果没有电子版就买纸质的。
3. 确定相关的好的项目，比如twitter的util库，akka等，在看完书之后实践用。也下载

整理资料：

1. 在我要学习－
   马上学习目录里建立scala入门，scala进阶的文件夹，把相关pdf，代码放到里面
2. 在wiz的相同目录，将抓取得文章，放到scala入门，scala进阶目录。

开始学习：

1. 先快速浏览一下所有的入门文章，在我的知识－工作必备-scala
   建立目录，将吸收的知识转化为自己的语言整理到里面。当然看懂的资料先放一下，等后面再回头看。
2. 速读相关书籍，整理xmind读书心得
3. 进行练习，整理练习心得，所有的都沉淀下来，变成自己的。
4. 反复进行

整理自己的知识&分享：

1. 学一段时间后，回顾自己整理的心得，知识，做进一步总结概况，并能给别人讲解，放到网站上或者论坛上。
2. 把已经完全吸收的文章或者书籍，进行删除，仍有价值或者可以备查的资料，放到我的知识－工作必备-scala-归档目录进行保存。

大概就是这样了，肚子饿了，赶紧洗洗睡了。

架构师速成 8.1-谈做技术人员的态度 - for5million 的专栏 - 博客频道 - CSDN.NET

谦卑，永远保持谦卑。

热情，永远保持热情。

同行相轻在中国好像是一个惯例，互相看不起，互相贬低，也充斥了软件行业的每个角落。想成为一个架构师请先保持你的内心的谦卑，永远不要嘲笑或贬低任何一个人。因为当你嘲笑或者贬低一个人时，世界就给你关了一扇门。其实群体的意志才是最强大的，当你有一个团队，一般情况下团队的力量总是会比一个人要强大。所以要时刻想着如何激发整个团队的热情，而不是轻视你的队友，当你谦卑时，你就会拥有整个团队的力量。同行相轻这些事情在大公司的同学身上尤为明显，大公司的同学会不自然的体现一种优越感，可能本人无法察觉，尤其是在面对其他小公司的同学时。但是我见过真正的牛人，他们反而是最没有架子，最谦卑、最低调的人。这也说明为什么牛人能做到那样的地步，先做人再做事，古人诚不欺我。

热情，是前进的动力，是团队的火种。保持对新技术的热情，保持对知识的热情，保持对同伴的热情，用你的热情感染周围的一切，世界终将因你而变。

怎么一股浓浓的鸡汤味！这段时间晚上没有饭吃的原因吧，已经 2 周多没有吃晚饭，减了 6 斤了，请祝贺我。

架构师速成 8.2-架构师要懂产品 - for5million 的专栏 - 博客频道 - CSDN.NET

产品和架构两个截然不同的职业，好像风马牛不相及，其实不是这样的。产品的思想需要经过技术的手来成为现实，在成为现实之前，需要技术理解、评估、碰撞、优化、把控、验证等等。当然架构师就承担了这一系列技术的责任，而且在一个产品的实现过程中，技术架构并不是很重要的，前期可以没有架构，简单快速验证，只有在用户多了之后，架构才有真正的用处。在初创公司，很多架构师都等不到用户多了的那一天，来实现自己的架构梦。所以产品这一关架构一定要把好，只有你把好了，后面才有机会让你去架构。

当然架构师的懂产品，是懂产品的生命周期，产品发展的客观规律，产品的评估及验证。其实架构师承担的是一个美食家的角色，美食家并不需要去做菜，但是要知道如何评价菜的味道，懂得菜是如何做出来的。你不需要去养猪，只管吃肉就行了，听上去很美好，非常美好！

1. 产品提出初步思路，这时候你就需要参与进来，用客观的角度去评价思路的正确性，而不是觉得这个做产品的人跟我很熟，我觉得产品思路就不错。这个非

常重要，目标错了，200%的努力去实现，只会更糟。如何用客观的眼光去评价呢，我有一个诀窍，就是代入法，假设这个产品是你的竞争对手做的，你如何干掉他。

2. 产品需求调研，你要亲自参加，使用精益创业的手段来进行理性的验证。

后面不写了，再写可以专门写一本了，反正就是你要懂产品，不是让你去做原型，而是让你去品这个产品！

架构师速成 4.8-幼儿园书单资料推荐 - for5million 的专栏 - 博客频道 - CSDN.NET

- java学习
    - head first java
- 高效学习
    - 如何高效学习
    - 学习之道
    - 学习要像加勒比海盗
    - 如何高效阅读
- 沟通
    - 沟通的艺术
    - 沟通:用故事产生共鸣
    - 演讲之禅
    - 60天完美口才打造计划
- 学习资料站
    - http://www.read.org.cn/
    - http://book.douban.com/

架构师速成 5.5-小学书单资料推荐 - for5million 的专栏 - 博客频道 - CSDN.NET

- ppt进阶
    - 写给大家看的PPT设计书
    - 说服力 让你的PPT会说话
    - PPT,要你好看
    - 别告诉我你懂PPT
- 时间管理
    - 小强升职记
    - 把时间当作朋友

        - 高效能人士的七个习惯
- 技术类
    - head first java
    - head first sql
    - head first html css
    - thinking in java
    - effective java

架构师速成 7.5-性能优化为什么写的这么晚？ - for5million 的专栏 - 博客频道 - CSDN.NET

性能优化是程序员的 G 点，一碰就会高潮！（小朋友不懂的不不要懂了）

但是我为啥这么晚才抛出这个命题，其实有人早就急不可待吧。我这么晚写是有这么晚写的理由的，其实性能优化，在做一个小的网站，根本没有什么卵用。一个流量小的网站，框架做好，架构做好，表结构设计好，根本没有太大的必要去优化，因为机器都闲着没有什么卵用，有时间不如把产品做好，吸引更多的人气。

其实我在到阿里之前，做过几个网站，而且也提了很高的要求，比如 用户一次请求，服务器端必须在 100ms 内返回结果。也做了一些性能优化，但是在寥寥无几的用户面前，这纯粹是浪费时间。但是到了阿里就不同了，阿里很注重性能优化，注重稳定，注重底层的参数设定。为啥？流量大了，你优化 1%都会节省大批的机器，这都 money。说个笑话，之前和同事一起优化了一个 mr，算下来每天能节省 70 万，要是这个钱发给我，那就 happy 了。

过早优化是万恶之源，永远不要过早的去优化，去过度设计。当然不是说基本的代码规范不要了，有最佳实践当然按照最佳实践进行编码。过早优化是指的去做代码结构和一些主要逻辑的优化，这些没有必要过早去实施，真正发现瓶颈再去优化。

而且优化时不要想当然的去优化，一切要以测试为准。我经历过很多次代码优化，有时候很多代码改动，并不能带来性能的提升，反而增加了代码复杂度。印象最深的一次是优化了很多代码，都没有很大的效果，结果调整了一下 jvm 的参数，性能提升 20%，所以一切过早的优化都是扯淡。还没有上线，没有测试的依据就进行优化，80%都是徒劳。

架构师速成 4.6-软技能和硬技能 - for5million 的专栏 - 博客频道 - CSDN.NET

在投职和面试的过程中，雇主通常会查看求职者的两种技能：硬技能与软技能。
硬技能就是能够通过培训或可以容易量化的技能。硬技能包括下面这些：

- 熟练的外语能力
- 文凭或证书
- 打字速度
- 计算机程序编写能力

而软技能，则是较难量化的主观性技能，例如：

- 团队合作
- 沟通
- 时间管理

而且绝多数的日常软性技能都是可以跨领域迁移，硬技能有时候随着工作的变迁，就失去了价值。我也极为推崇软技能的练习和培养。首先一个人应该最先具备的软技能如下：

- 学习能力
- 思考能力
- 沟通能力
- 管理能力

学习能力是第一应该掌握的，你学习能力比别人强，那就凡事快人一步了。学习能力再细分为：

- 阅读技巧，就是如何又快又好的读一本书，这个非常重要，尤其是目前各种书籍泛滥的情况下，速读就是一大利器。
- 记忆力　　记忆力是可以锻炼的，记得有一期最强大脑，有一位患老年痴呆的老人，通过锻炼不但恢复而且能记下一整本书。大家可以去搜搜。
- 学习方法　推荐前面提到的学习方法

思考能力，这个算第 2 吧，因为你学习速度上去了，再学习这个更快一些：

- 思维导图
- 创造性思维
- 批评性思维
- 冥想
- 自省
- 心智管理

沟通能力，这个也非常重要，人是群体性动物，如果缺少沟通能力，寸步难行。

- 语言

- 演讲
- 说服力

管理能力，这是自我管理与管理他人的能力的集合，你成功的必备要素:

- 时间管理：最大限度的利用时间，不拖延不浪费，努力提升团队效率
- 知识管理：能在最短的时间内建立一个新知识的整体框架，并能确保在团队内的知识协作和共享
- 财务管理：你不理财，财不理你。
- 团队管理：三人为众，众志成城。如果你有一只铁军，没有什么做不成的。
- 人脉管理：人都是靠别人的帮助才取得成功的，人脉是你发展的最大助力。

有时间先把你的软技能练好吧，你会发现没有什么能难得倒你。

架构师速成 6.18-初中书单资料推荐 - for5million 的专栏 - 博客频道 - CSDN.NET

思维导图

- 你的第一本思维导图操作书
- 画出好成绩-通过思维导图提升分数
- 思维导图系列

知识管理

- 你的知识需要管理

面向对象

- 写给大家看的面向对象编程书
- 面向对象分析与设计
- 深入浅出面向对象分析与设计

软件设计

- UML精粹
- UML基础、案例与应用
- Head First软件开发
- 领域驱动设计

写好代码

- 重构:改善既有代码的设计
- 代码大全
- 代码整洁之道

设计模式：

- 大话设计模式
- head first 设计模式

架构师速成 7.6-高中书单资料推荐 - for5million 的专栏 - 博客频道 - CSDN.NET

速读速记：

- 如何高效阅读
- 10倍速影像阅读法
- 超右脑快速记忆法

项目管理：

- 敏捷开发的艺术
- 敏捷软件开发
- 硝烟中的Scrum 和XP
- 精益开发实践
- 走出软件作坊
- 人件
- 人月神话
- 黑客与画家
- 死亡之旅

企业架构：

- 企业应用架构模式

devOps：

- 持续集成:软件质量改进和风险降低之道

性能：

- 深入理解Java虚拟机

- Java性能优化权威指南

架构师速成 5.2-价值观和目标 - for5million 的专栏 - 博客频道 - CSDN.NET

价值观和人生目标是 gtd 的最上层要素，如果你整个人生的价值观和目标都不清晰，后面的路会因为失去方向而走的比较艰辛。但好多人都不太清楚价值观和人生目标如何确定，这 2 个东西太抽象了。首先要先说一下价值观是什么？

心理学家发现我们会受到不同的事物所鼓舞和推动，这些动力往往归纳为：需要、兴趣、价值、信念及喜好。对部分人来说，这些动力根深蒂固，而一般认为这是由于我们成长中所接受的教育，以及性格上的基因分别所致。一般价值观分为：

1. 理性价值观
   它是以知识和真理为中心的价值观。具有理性价值的人把追求真理看的高于一切。
2. 美的价值观
   它是以外形协调和匀称为中心的价值观，他们把美和协调看的比什么都重要。
3. 政治性价值观
   它是以权力地位为中心的价值观，这一类型的人把权力和地位看的最有价值。
4. 社会性价值观
   它是以群体和他人为中心的价值观，把为群体、他人服务认为是最有价值的。
5. 经济性价值观
   它以有效和实惠为中心的价值观。认为世界上的一切，实惠的就是最有价值的。
6. 宗教性价值观，它以信仰为中心的价值观。认为信仰是人生最有价值的。

可以使一下李开复老师的报纸头条检测法。
所谓"报纸测试法"，就是在事后想一想：明天，如果在一份你的亲朋好友都会阅读的报纸上，你做的事被刊登为头条新闻，你会不会因此而感到羞愧？会不会无法面对自己的良心？如果不会，你做的事才对得起你自己的价值观。

李开复本人讲述了一个"报纸头条测试法"的真实使用例子。那是李开复在苹果公司工作时遇到了公司裁员，当时李开复必须要从两个员工中裁掉一位。第一位员工毕业于卡内基·梅隆大学，是李开复的师兄。他十多年前写的论文非常出色，但加入公司后很是孤僻、固执，而且工作不努力，没有太多业绩可言。他知道面临危机后就请他和李开复的共同的老师来提出希望我顾念同窗之谊，放他一马。另一位是刚加入公司两个月的新员工，还没有时间表现，但他应该是一位有潜力的员工。

李开复内心里的"公正"和"负责"的价值观告诉自己应该裁掉师兄，但是李开复的"怜悯心"和"知恩图报"的观念却告诉李开复应该留下师兄，裁掉那位新员工。

于是，李开复为自己做了"报纸测试"，想象在明天的报纸上，自己希望看到下面哪一个头条消息：

（1）徇私的李开复，裁掉了无辜的员工；

（2）冷酷的李开复，裁掉了同窗的师兄。

虽然李开复极不愿意看到这两个"头条消息"中的任何一条，但相比之下，前者给李开复的打击更大，因为它违背了李开复最基本的诚信原则。如果违背了诚信原则，那么李开复既没有颜面见到公司的领导，也没有资格再作职业经理人了。于是，李开复裁掉了师兄，然后李开复告诉他，今后如果有任何需要自己的地方，自己都会尽力帮忙。

如果用"报纸测试法"得到令自己羞愧的结果，就有必要深刻反省，下定决心将来再也不做类似的事。

ok，你可以用这种方法确定你的价值观，价值观是你确定人生目标及做事评判的唯一标准，每当你失去方向，犹豫不决时，想想你的价值观，你就会找到方向。

架构师速成 6.7-设计开发思路-uml - for5million 的专栏 - 博客频道 - CSDN.NET

---

uml 是什么东西？统一建模语言，一门语言，是用来进行软件设计的一门语言。

其实一门语言的诞生并不伟大，让大多数人都使用才足够伟大。uml 就是一门伟大的语言，因为目前软件设计的唯一语言就是它。

UML 其实还是比较简单的，就那么几个图形，那么几种模式，但是因为他是唯一的语言，所以有设计能力的人都能很容易看懂你说的什么，这就是他的伟大之处。

我说一下在软件设计中最常用的几个，以及我的心得：

1. 用例图，在了解用户需求时非常有效，他仅用来描述系统需要提供的功能，本身没有顺序，不要用来描述流程。注意使用扩展和包含。那个小人即可以是使用者也可以是其他系统。
2. 类图，这是面向对象设计的真谛，不要和ER图混为一谈，类图是用来描述类与类之间的交互关系，本身可以没有任何属性。当然也可以有很多属性，但是不要用设计数据库的思路来设计类图。类图只是用来反映现实，在设计类图时，可以认为数据会存储在DB中，也可能存储在XML中，也可以存储在文件中，不要去考虑存储。
3. 对象图，用的不太多
4. 序列图，描述对象之间的交互顺序，着重体现对象间消息传递的时间顺序，这个比较有用，但是不是很难。

5. 状态图，状态机就是它了，当你被复杂的状态搞晕的时候，用它来画清楚，实现就用状态模式，perfect。
6. 活动图，表示两个或多个对象之间在处理某个活动时的过程控制流程，这个也很重要，但是不难。

其他我用的就不多了，学习这门语言真的很重要，请重视。掌握他之后，学习设计模式会更加得心应手！

架构师速成 6.8-设计开发思路-领域驱动 - for5million 的专栏 - 博客频道 - CSDN.NET

领域驱动设计简称 DDD,很好的名字，先来普及一下相关的名词缩写：

测试驱动设计 TDD，行为驱动设计 BDD，面向对象设计 OOD，面向过程设计 OPD。

设计思路和方法是一项专门的技能，区别于设计模式，编程语言。UML 是设计的工具，设计方法是设计的灵魂，而且设计方法并没有好坏之分。关键是你需要掌握各种设计方法，在做项目时信手拈来，才是真正的高手。为什么要讲领域驱动设计，因为在做大型系统时，领域驱动设计会让你事半功倍，得心应手。

网上找到一篇总结的文章，写到比我写的好，那就直接转帖吧
http://www.cnblogs.com/netfocus/p/4492486.html，原帖地址。

1. 领域驱动设计（DDD）是一种基于模型驱动的软件设计方式。它以领域为核心，分析领域中的问题，通过建立一个领域模型来有效的解决领域中的核心的复杂问题。Eric
   Ivans为领域驱动设计提出了大量的最佳实践和经验技巧。只有对领域的不断深入认识，才能得到一个解决领域核心问题的领域模型。如果一个应用的复杂性不是在技术方面的，而是在领域本身，即领域内的业务很复杂，那这种应用，使用领域驱动设计的价值就越大。
2. 领域驱动开发也是一种敏捷开发过程（极限编程，XP），强调迭代开发。在迭代过程中，强调开发人员与领域专家需要保持密切的合作关系。极限编程假设我们能通过不断快速重构完善设计。所以，对开发人员的要求非常高。
3. 领域驱动设计提出了一套核心构造块（Building
   Blocks，如聚合、实体、值对象、领域服务、领域工厂、仓储、领域事件，等），这些构造块是对面向对象领域建模的一些核心最佳实践的浓缩。这些构造块可以使得我们的设计更加标准、有序。
4. 统一语言（Ubiquitous
   Language），是领域驱动设计中一个非常重要的概念。任何一个领域驱动设计的项目，都需要一种通用语言，一套通用的词汇。因为没有通用的语言，就没

有一致的概念，沟通就会遇到障碍，最后的领域模型和软件也就无法满足领域内的真实业务需求。通用语言是领域专家和开发人员在对领域问题的沟通、需求的讨论、开发计划的制定、领域模型的设计，以及开发人员之间对领域模型的具体编码落地实现，等一系列过程中，所有人员使用的一种通用语言。话句话说，就是无论是沟通时所用的词汇、还是领域模型中的概念、还是代码中出现的类名与方法，只要是相同的意思，那就应该使用相同的词汇。可以看出，这种通用语言不是一下子就可以形成，而是在一个各方人员讨论的过程中，不断发现、明确，与精炼出来的。

5. 领域模型是领域驱动设计的核心。统一语言中的所有关键词汇，在领域模型上应该都能找到。各方人员沟通时，都应该以领域模型为基础。通过讨论的不断深入，大家对领域的认识也会不断深入，领域模型也会不断得到完善，统一语言的词汇也会不断丰富和精准。需要特别强调的是，开发人员应该尽量保证代码实现和领域模型相绑定，时刻保持代码与模型的一致。如果不绑定，那代码就会慢慢和模型相脱节，就会出现像我们以前那样的设计文档和代码相脱节一样的问题，甚至模型还会起到误导作用。通过这样一种思路，我们确保语言、模型、代码三者紧密绑定，确保最后实现出来的软件可以准确无误的实现业务需求，并且还能让我们的软件可以快速的和业务同时演进。而不像传统的开发方式那样，分析、设计、实现三个阶段完全脱节，最后出来的软件没有很好的满足业务需求，也不能在未来很快的跟业务需求一起演进。所以，领域模型同时承载了分析的结果和设计的结果，这里的分析是指对领域内业务需求的分析，设计是指对模型的设计以及软件的设计。所以，我们的领域模型，不能只考虑业务需求，还要同时考虑软件设计的原则，是一种综合考虑的、平衡的设计结果。
6. 领域模型可以复用，因为特定的领域模型解决的都是某个特定的问题域；比如淘宝网有个商品中心，有个商品模型，核心概念有商品分类、商品；商品模型负责解决电子商务领域中的商品目录（Product Catalog）子域。后来阿里又出了个天猫，也会有商品中心，但是这两个商品中心基本是一样的问题域。所以，我们可以复用之前淘宝实现的商品中心领域模型，并复用之前淘宝商品中心的解决方案，来解决天猫的商品维护和展示。当然，这个只是我个人的认识，一个例子。具体阿里是否是一个商品中心同时解决淘宝和天猫的业务，没具体调研过。
7. Bounded Context，属于一种软件构件，作用是用来对领域模型进行划分。Bounded Context有两层含义：
    - Bounded，即有边界的，表示领域模型有边界；这个边界定义了模型的适用范围，以便让负责该模型的团队知道什么该在模型中实现，什么不该；

- Context，即领域模型的产生是在某个上下文中产生的；上下文是一个和环境相关的概念。比如一次头脑风暴会议大家达成了一个模型，那这次会议的讨论就是该模型的上下文；比如某本书中谈到了某个东西，那这个东西的上下文就是那本书，那个东西要有意义的前提离不开那本书这个上下文；所以，上下文是模型有意义的前提；

8. 领域建模的方法有很多种，我分享一下自己的一种基于场景为核心的分析方法。大概的思路是：
   - 通过与领域专家和业务需求人员沟通，找出领域中的关键业务场景；
   - 针对每个业务场景分析出有哪些场景参与者，哪些参与者以对象（聚合）的形式参与，哪些参与者以服务的形式参与；
   - 分析每个场景参与者对象的基本状态特征；
   - 分析每个场景参与者对象分别扮演什么角色参与场景，整个场景的完整交互过程是怎样的，对象在参与场景的过程中执行了哪些交互行为；
   - 分析如何记录和跟踪这一次交互行为，分析这次交互行为会产生哪些额外的信息；
   - 上面，只是简单列了一下条目，具体的描述，请参看我的另一篇文章，有详细的叙述。
9. 关于领域（Domain）、领域模型（Domain Model）、边界上下文（Bounded Context）的关系
   - 领域就是问题域，问题空间；
   - 领域模型是一种模型，表达了领域中哪些业务需求以及业务规则必须被满足；
   - 每一个领域中的问题，都会有一个对应的领域模型去解决；
   - Bounded Context的作用是用来对领域模型进行划分；
   - 划分领域就是对问题空间的划分，通俗的理解，就是将大问题拆分为小问题；
   - 划分Bounded Context就是将一个大的领域模型划分为多个小的领域模型；
   - 可以把Bounded Context看成是一种解决方案空间，所以，Bounded Context也可以理解为是对解决方案空间的划分；
   - 理论上，一个Domain可能会对应多个Bounded Context；同样，一个Bounded Context可能也会对应多个Domain；所以他们之间没有绝对的关系。主要是他们划分的依据不同，一个是针对领域（问题空间），一个是针对领域模型（解决方案空间）；理想情况，一个Domain最好对应一个Bounded Context；
10. 关于Domain、Sub Domain、Core Domain、Generic Domain，以及Shared Kernal的理解：

    - 一个领域（Domain）会拆分为多个子领域（Sub Domain）；
    - 子领域中最核心（最重要）的那个叫Core Domain；我们应该讲团队的核心资源用在核心子域上，因为它是产品成败的关键；
    - 除了Core Domain外，其他的是支撑子域（Supporting Subdomain）；
    - 有些支撑子域比较特殊，因为它解决的是一类通用问题，比如账号和权限；这类子域我们叫做通用子域（Generic Subdomain）；通常，通用子域对应的Bounded Context，会跨域多个子域；
    - 多个子领域有时会有相交的部分，我们称作共享内核（Shared Kernel）；体现到代码上，就是同一份代码，在两个领域模型中复用；
    - 一般只有Domain比较大的时候，我们才会划分出Sub Domain；
11. 为什么一个大的领域模型需要划分？因为，通常一个大的领域模型需要多个团队合作完成。如果多个团队基于一个共同的领域模型工作，由于每个团队的关注点不同，且一些看似叫法一样的概念，对于不同的团队，其背后的意思完全不同。所以，这样的概念含义模糊会给团队以及成员之间的合作带来很大的困扰。所以，我们需要通过一种手段（Bounded Context），将领域模型划分为不同的部分，确保同一个Bounded Context内的领域模型所表达的概念含义明确。然后，同一个Bounded Context下面，相关人员都使用一种统一的语言，以此来保证团队成员之间沟通能畅通无阻；

当然，如果有时间还是看看《领域驱动设计》这本书吧。

架构师速成 6.9-如何写好代码 - for5million 的专栏 - 博客频道 - CSDN.NET

读过很多代码，有些人写了 5 年以上的代码，但是代码烂到直接让人无从读起。有人写的代码让人读的赏心悦目，为什么人与人之间的差距这么大呢？

看来如何写好代码还是一件很值得一说的事情。

关键思想：

1. 人写代码是用来给别人读的，而不是给机器执行的。写代码就应该像写小说一样，让人读的轻松愉快。
    - 软件的生命周期中，维护周期占1年以上，所以不看避免要有人来改动你的代码
    - 你如果更换工作岗位，你的代码需要别人接手
    - 如果你自己的代码，3天后自己都看不懂

    - 牛人的代码总是让你爱不释手
2. 写代码需要注意层次的一致性，不要芝麻和西瓜混在一起。
    - 凳子，桌子，椅子，兮子，床，你觉得那个不对
    - 再举一个做菜的例子：
        1. 洗菜
        2. 切菜
        3. 倒油爆炒
        4. 油温慢慢的从10度上升到60C，油分子之间间隙越来越大，油沸腾起来。油分子和菜分子之间产生了反应。
        5. 出锅。
    - 每个层级负责自己下面的7个子节点。
3. 不要有过多的注释，你注释说明你怕别人看不懂。
    - 注释越多说明我越认真，但是有一天代码变了，注释把人害了。
    - 只需要写生成doc的注释
    - 方法名，参数和类名本身就是注释
4. 一个团队必须有一致的编码风格，如果有人不能统一，特立独行，那么out。
    - 我要自由，我是不羁的风，那立刻给我刮走
    - 我牛，所以我不需要遵守，sorry，真的牛吗？
    - 我之前的公司没有编码规范，那是他们傻。
5. 要有最佳实践，这个可以显著提高代码质量
    - 比如effective java
    - 比如 “abc”.equals(xxx)

有时间读一下重构，代码大全，你会爱上编码，编写诗一般的代码。

架构师速成 6.11-开发框架-后端封装思路 - for5million 的专栏 - 博客频道 - CSDN.NET

要做一个好的开发框架并不是直接找一个现成的 Spring 就 ok 了，这个对于一般新手来说学习难度还是有点高。另外 Spring 还是一个通用性的框架，我们需要针对业务的定制化封装。我以 java 为例讲解一下后端需要做什么：

1. 确定层数结构
    1. parent
        1. common
        2. test
        3. dao
        4. service-common
        5. service

6. web-common
7. web

2. parent统一引入的类库及版本，通过maven方式进行类库管理，在parent中定义所有引入的类库及版本，不需要其他人私自引入类库。java类库的膨胀及冲突控制的确是一项很头疼的事情，一不小心最终的项目就达到几百兆，每次编译发布就需要很长时间。我在这里强烈推荐一下一个国内开源框架nutz，无引入其他任何jar包，提供了spring和hibernate的最常用功能，只有1M。我们之前使用Spring，后来换成nutz，而且严格限制引入的jar包后，web最终打完包，才10M左右。
3. common工具类确定及封装，不要引入过多的包，只引入最基础而且是最必须的工具类。比较极端的情况下，自己写。
    1. StringUtils，DateUtils等，这个引入common-lang
    2. 语法糖，比如 map,list等的泛型构造，isEmpty(Object) length(Object）等
    3. JodaTime，谁用谁知道
    4. guava，缓存，限制流量
    5. log，java的日志是一大堆，建议slf4j+logback,其他的都桥接到这上面。
    6. Exception，定义异常接口 ISystemException,IBusinessException，IAuthException，及几个实现类，这个会单独起一个异常章节进行讲解。
    7. Ioc，这个一般你使用的框架已经提供了。
    8. Aop，这个一般你使用的框架已经提供了。
4. test
   单元测试，集成测试的辅助类封装，让写单元测试跟玩一样，否则没有人愿意写的。
    1. IOC容器，Spring上下文及测试类封装
    2. 通用的Mock类
5. dao
   对于db或者hbase，mongodb等一系列存储的封装，当然如果有必要还可以拆分。我针对db先说一下，其他的先不说：
    1. 全局唯一主键生成算法
    2. 主备库，多数据源
    3. 统一的IdEntity，IAuthorEntity（带修改人，修改时间记录的），所有数据库实体都继承自他2个
    4. 针对统一类的泛型增删改查封装
    5. 扩展自定义的标注，比如@Column @MyId等等用于实现自己特殊的逻辑
    6. 数据级别的权限控制

7. 数据库事务，这个一般你使用的框架已经提供了。

6. service-common，封装业务逻辑所需要的工具类或基础类，如果没有多少，可以和service合并。
7. service真正的业务逻辑，根据不同的业务及规模可以进一步拆分service-a,service-b
8. web-common,封装于界面及mvc相关，filter等等的逻辑
    1. mvc，url-mapping　这个一般框架已经提供，比如spring，nutz都会有。
    2. 异常处理，后面会单独写
    3. 单点登录filter，其他统一的filter
    4. 访问权限控制
    5. 不同后缀的处理，json，html，json及jsonp的封装
    6. 动静分离，静态文件使用nginx进行加载，设置缓存时间等等
    7. 界面统一的变量封装，比如根路径，user
    8. 用户获取

做完这些，后端的基本封装才算稍微有一个样子。

架构师速成 6.12-开发框架-前端封装 - for5million 的专栏 - 博客频道 - CSDN.NET

做一个网站不只有写后端代码，为了更好的用户体验以及更优雅的代码实现，我们也需要对前端进行封装。但是在谈封装之前还是要先提一下统一的规范，前端统一的规范尤为重要，这是给用户一致性体验最核心的关键点。我举一个反面的例子，大家可能一下就明白前端规范的重要性了。之前做过一个网站，流程如下：

1. 设计给图片
2. 前端照图片进行静态页面生成
3. 后端再加入代码，最终成为一个网站

这个流程没有什么问题，但是设计没有统一的规范，他给出的每一页文字大小，表单，分页几乎都不太一样，都有些许的区别。前端做页面的同学，也严格要求自己，每一页都做的很符合原图。后端同学发现不能重用，那也只好每页单独做分页，做表单，做 js 校验，严重的是光用户头像就 7 个 size，我们到最后都要哭了。最后做出的网站给人感觉不太像一个网站，因为每页有些相似，但是又有细微不同。而且 css，js 超多，几乎每页都对应一个单独的 css 和 js。维护起来很费劲，比如加一些字段，还得设计再修改原图，前端再照着切，否则有些地方就不美观。设计，前端，后端每个人都疲于奔命，结果还是工期大大延后。

所有前端规范非常重要：

1. 除首页外，统一定义列表页，详情页，表单维护页的模板，全站使用一套。
2. 模板要抽取公用部分，统一头，统一尾，统一头像（大，中，小），统一翻页
3. 字体大小统一（大，中，小），网站所有的输入框，标题，错误提示都规范等等
4. 统一颜色，不能超过3个配色
5. 页面其他元素也要统一，例如 提示，弹框等等
6. 业务相关的元素也要统一设计，举个例子：比如当当的图书，淘宝的商品
7. 其他需要统一的部分

后来统一之后，设计只需要设计好那几个关键模板就可以了，有了新的业务元素设计一下，非常轻松。前端统一样式之后，css 就只有一个也是无比轻松，而且也不需要切页面了。后端同学也轻松了，拷贝一个模板，改改里面信息项就 ok 了。这个要向 bootstrap 学习。

另外前端需要的文件也很多，css,js,html,img 等等，文件夹规范也是很重要的。必要时请考虑动静分离，使用 nginx 加载静态资源，你会觉得爽多了。

说完规范，再说一下其他的关键点：

1. 前端模块化开发
   例如:SeaJS，RequireJS，如果要进行前端规模化开发，必须要考虑。如果只有一个前端，也尽量使用模块化开发方式。好处自己google。
2. 采用统一的底层类库，JQuery目前是使用率最高的了。不要张三说我用A，李四说我用B，结果互不妥协。必须只能选一个，就像后端你不能选了spring，又选nutz。
3. UI类库封装，弹框，消息，tooltip，翻页，form校验，tab页，日期，图表等等与界面相关的
4. 功能类库封装：ajax，异常处理，前端计算，模板加载等等
5. 无js化，参见angularjs
   jui(国内前端框架)，毕竟写js对初学者还是有些难，如果不需要写js，而只是加上一些属性，就可以实现js效果，初学者还是很喜欢的。

网站开发一前一后是必备的，必须进行完美的封装，让初学者也能很快上手。

架构师速成 6.13-开发框架-前后结合 - for5million 的专栏 - 博客频道 - CSDN.NET

前面说完了前端和后端的封装，本节再单独讲解一下哪些地方需要前后结合。

AJAX 的前后端结合：

1. 定义统一的返回消息体，{isError:true/false,message:'需要返回的消息',data:{返回的数据},postAction:{后续的动作}}
2. 根据消息体的返回，进行不同类别的展示，例如：错误提示，加载页面片段，数据刷新，页面跳转，或者自定义后续处理
3. 服务端能自动判断是JSONP还是json方式的请求，进行相应的处理

翻页的前后端结合：

1. 翻页首先分为页面跳转、ajax替换，ajax滚动追加
2. 前后端需要统一翻页的对象，Page对象，包含page,pageSize,totalCount等，另外返回的数据，是放到page对象内，还是外部一个单独的对象。这也是一个需要考虑的地方。
3. 后端支持将page对象传入，根据page对象返回结果，如果没有page对象，则使用默认的page，另外totalCount是否需要也是可选的
4. 前端根据返回的page对象进行页面渲染。
5. 尽量做到，程序员感知不到自己需要对翻页做任何的处理

表单验证的前后端结合：

1. 表单验证需要前后端都进行，有些同学说有前端校验就足够了，黑客同学也是这么想的。
2. 表单验证要保证前后端的一致性。如何保证？前端和后端验证需要无代码侵入式的验证，不需要写代码。
3. 如何防止重复提交。

业务相关的前后端结合：

1. 比如用户头像及用户基本信息
2. 比如评论
3. 其他业务需要结合的

错误的前后端结合：这个后面单独一节给讲解。

总之网站前后端要形成一个整体，需要开发人员了解的越少越好，这样才可以凸显框架的价值。

架构师速成 6.14-开发框架-异常处理 - for5million 的专栏 - 博客频道 - CSDN.NET

异常为什么需要单独一节来讲解呢？因为异常是软件设计里一项需要架构者自己构思处理的一个特殊部分。一般的框架没有提供统一的处理方式，而且会被很多同学忘记处理，或者是比较粗糙处理掉。想一想有很多网站出错时，用户会看到一大段的英文异常，甚至执行的 sql 都包含在里面，你会不会觉得很不专业呢？

首先异常应该分类，确定异常分类至关重要，这是后期进行不同处理的基础：

1. 业务异常，是用户在执行业务请求时，出现的业务出错或者不合法的信息，该异常的出现是合理的，应该让用户看到异常提示的信息，而且不需要记录异常日志。
2. 权限异常，是用户在执行某些需要特殊权限的业务，或者未登录用户操作需要登录相关的业务时，出现的异常，该异常出现是合理的，不需要记录异常日志。但是处理方式是不同的。
3. 系统异常，是出现一些不可预知的情况比如db宕机，此类异常不应该出现，出现时需要记录日志，而且此类异常需要及时报警，另外用户不应该看到具体的错误信息，此时应该提示比较优雅的错误信息，比如：网站可能正在维护中，请稍候重试，如果重试失败，请点击提交异常给相关人员，我们会及时处理。

这 3 类异常是所有异常的基础类，当然系统比较庞大时，针对这 3 类异常还需要进行细分，以进行更精细化的处理。当然我们先不再展开，先就这 3 类异常如何处理展开讨论。

再说一下用户请求的方式：

1. 页面跳转
2. ajax请求+ajax跨域请求

针对每种请求方式都会有如下 4 种情况：

1. 请求正常，正常返回
2. 请求出现业务异常，展示给用户
3. 请求出现权限异常，要求用户登录或申请相关权限
4. 请求出现系统异常，系统报警，给用户提示，尝试恢复

那如何进行异常的统一处理呢？

1. 首先确认普通请求和ajax请求及ajax跨域请求，建议使用后缀进行区分json方式是ajax请求，如果是json方式而且包含callback参数的是跨域请求。
2. 做一个统一的异常拦截器，先区分这3种方式，然后使用不同的handler来进行处理。
3. 每个handler中，分别针对3中不同的异常，进行不同的处理逻辑。

1. 页面跳转+业务异常，统一处理方式是跳转到统一的业务提示界面，提示用户对应的错误信息，然后在页面上可以选择返回上一页。当然也可以让用户在controller的方法上标示自定义的错误处理方式@Fail("我希望跳转到的页面")，此时会将错误信息放入上下文，在跳转到的页面进行展示。
2. 页面跳转+权限异常，调到登录页或权限申请页
3. 页面跳转+系统异常，跳转到统一的错误页面，触发报警，但是不要在页面展示错误的详细信息，用户可以选择返回重试，或者 提交错误报告。
4. ajax+业务异常，这个最简单了，直接提示错误信息就ok了。
5. ajax+权限异常，根据不同类别进行页面跳转。
6. ajax+系统异常，触发报警，但是不要在页面展示错误的详细信息，用户可以选择重试，或者提交错误报告
7. 跨域只是需要增加callback，其他同ajax。

4. 其中ajax，需要做统一的封装，针对请求的不同错误进行处理，另外发送ajax请求，如果页面刷新终止时，注意捕获ajax异常，不要提示出来了。
5. 当然针对权限异常，我们还会再开专门的一节进行讲解。

经过这样的封装之后，开发人员只需要在需要的时候抛出异常就可以了。一个菜鸟程序员都可以轻松的抛出自己的异常，而无需关心异常背后到底发生了什么。

另外如果系统复杂度增加时，可以增加异常的子类，进行不同的异常处理。另外系统异常也可以执行自动的补偿机制，比如 db 宕机，尝试服务重启之类，呵呵，当然后期可以通过架构的方式尽量避免此类问题。

架构师速成 6.15-开发框架-单点登录 - for5million 的专栏 - 博客频道 - CSDN.NET

单点登录应该归为架构的部分了，但是一般网站在开始的时候最好有单点登录的思想，防止后期再做大量的修改。而且单点登录对于开发人员来说并没有增加太多额外的工作量，所以提前讲一下对大家都是好的。

先说一下单点登录的机制（摘自百度百科，给我广告费）：

当用户第一次访问应用系统 1 的时候，因为还没有登录，会被引导到认证系统中进行登录；根据用户提供的登录信息，认证系统进行身份校验，如果通过校验，应该返回给用户一个认证的凭据－－ticket；用户再访问别的应用的时候，就会将这个 ticket 带上，作为自己认证的凭据，应用系统接受到请求之后会把 ticket 送到认证系统进行校验，检查 ticket 的合法性。如果通过校验，用户就可以在不用再次登录的情况下访问应用系统 2 和应用系统 3 了。

另外单点登录的好处是不需要每开发一个系统都需要做一套用户管理的功能了。你只需要开发一套用户管理系统，提供 sso 的 sdk，甚至可以再扩展一下提供 oauth2 的 api，这样就可以实现内部的单点登录及外部的用户认证。

具体实现机制我就不再巴拉巴拉的拷贝粘贴了，请自行 google sso 单点登录，oauth2。

我再说一下我们血的教训，最早做网站时根本没有使用单点登录，因为就一个系统，而且也没有想过这个问题。后来系统就渐渐多了起来，每个里面都有一套用户注册、登录、管理。但是这几个系统最终是服务同一批用户的，问题就来了，用户不想每个系统都注册、登录一次啊，而且每个系统的用户都没有关系。领导对此大发雷霆，只得每个系统进行改造，数据合并最头大，有些用户注册的用户名还不一样，只得多个账号都保留。当然改造完之后，神清气爽，为后来和新浪教育及其他教育公司对接提供了有力的支持。

架构师速成 8.3-架构师必须要了解的规则 - for5million 的专栏 - 博客频道 - CSDN.NET

作为一个架构师，有些规则是必须要掌握的，这就想软件的公理，如果你学物理不知道牛顿定律，那就不要学了。在软件行业也有类似的东西，我称之为软件定律。例如：

ACID,CAP,BASE

ACID

传统数据库系统中，事务具有 ACID 4 个属性

(1)原子性（Atomicity）：事务是一个原子操作单元，其对数据的修改，要么全都执行，要么全都不执行。

(2)一致性（Consistent）：在事务开始和完成时，数据都必须保持一致状态。这意味着所有相关的数据规则都必须应用于事务的修改，以保持数据的完整性；事务结束时，所有的内部数据结构（如 B 树索引或双向链表）也都必须是正确的。

(3)隔离性（Isolation）：数据库系统提供一定的隔离机制，保证事务在不受外部并发操作影响的“独立”环境执行。这意味着事务处理过程中的中间状态对外部是不可见的，反之亦然。

(4)持久性（Durable）：事务完成之后，它对于数据的修改是永久性的，即使出现系统故障也能够保持。

可以说，数据库系统是伴随着金融业的需求而快速发展起来。对于金融业，可用性和性能都不是最重要的，而一致性是最重要的，用户可以容忍系统故障而停止服务，但绝不能容忍帐户上的钱无故减少,而强一致性的事务是这一切的根本保证。

CAP

在 2000 的 PODC（Principles of Distributed Computing）会议上，Brewer 提出了著名的 CAP 理论。CAP 指的是：Consistency、Availability 和 Partition Tolerance。

（1）Consistency（一致性）：一致性是说数据的原子性，这种原子性在经典的数据库中是通过事务来保证的，当事务完成时，无论其是成功还是回滚，数据都会处于一致的状态。在分布式环境中，一致性是说多个节点的数据是否一致。

（2）Availability（可用性）：可用性是说服务能一直保证是可用的状态，当用户发出一个请求，服务能在有限时间内返回结果。

（3）Partition Tolerance（分区容错性）：Partition 是指网络的分区。可以这样理解，一般来说，关键的数据和服务都会位于不同的 IDC。

CAP 理论告诉我们，一个分布式系统不可能同时满足一致性，可用性和分区容错性这三个需求，三个要素中最多只能同时满足两点。三者不可兼顾，此所谓鱼与熊掌不可兼得也！而对于分布式数据系统而言，分区容错性是基本要求，否则就不称其为分布式系统了。因此架构设计师不要把精力浪费在设计如何能同时满足三者的完美分布式系统上，而是应该进行权衡取舍。这也意味着分布式系统的设计过程，也就是根据业务特点在 C（一致性）和 A（可用性）之间寻求平衡的过程，要求架构师真正理解系统需求，把握业务特点。

后来：CAP 理论的作者终于给了我长久以来想要的答案：CAP 理论并非严格的三选二，大多数情况下，A 和 C 是可以兼得的，因为大多数情况下，P 都不存在。
P 只有在结点之间通信延迟大于可接受的范围时才出现（结点之间开始近似隔离，状态开始不一致），即 P 一旦出现，我们选择继续提供服务那么状态就肯定不一致，也就等于放弃了 C；我们选择不提供服务，那么就等于放弃了 A。通俗一点，P 并不是目标，也不是手段，它是伴随着"多结点，网络，数据，共享"的要求而必然出现的，出现的原因是因为网络的不可靠性及结点通信延迟（延迟的原因可能是由于硬件，网络，或者压力太大而无法及时响应)。

弄清楚了 CAP 的 P，也就弄清楚了 CAP 理论的实质，戴在头顶的紧箍咒便永久摘掉了。

CAP 理论并不是要求我们悲观地放弃 A 和 C 任何一方，相反，它可以乐观地指导我们将 C 和 A 最大化；ACID 和 BASE 分别处于 CAP 理论的两个极端，ACID 强调强一致性，BASE 强调高可用性，两者把重点都放在 A 和 C 上，淡化了 P 也可变的事实；通过对 P

的出现检测，发现 P 之后的限制和约束，P 结束之后的补偿和恢复，通过采用千差万别的策略，我们可以避免 P 带来的 C 和 A 的严重损失，实现 A 和 C 的最大化来提高整个系统的正确性和可用性。ACID 和 BASE 并非水火不容，我们可以在同一个系统中，既使用 ACID，又使用 BASE。

BASE

BASE 来自于互联网的电子商务领域的实践，它是基于 CAP 理论逐步演化而来，核心思想是即便不能达到强一致性(Strong consistency)，但可以根据应用特点采用适当的方式来达到最终一致性(Eventual consistency)的效果。BASE 是 Basically Available、Soft state、Eventually consistent 三个词组的简写，是对 CAP 中 C & A 的延伸。
BASE 的含义：

（1）Basically Available：基本可用；

（2）Soft-state：软状态/柔性事务，即状态可以有一段时间的不同步；

（3）Eventual consistency：最终一致性；

BASE 是反 ACID 的，它完全不同于 ACID 模型，牺牲强一致性，获得基本可用性和柔性可靠性并要求达到最终一致性。

后面我会不断充实这本软件定律。
作者：arrowcat
出处：http://www.cnblogs.com/hustcat/

本文为 Gleasy 原创文章，转载请指明引自 Gleasy 团队博客

架构师速成 8.3-可用性 - for5million 的专栏 - 博客频道 - CSDN.NET

作为一个软件系统可用性是第一位的，如果一个系统不可用，你其他的地方做的再怎么好，然并卵。

一般什么情况下软件会不可用：

我方发生故障，导致系统不可用，当然会出现单机的不可用及 n 多机器群的全部不可用。

1. 程序故障 功能错误、程序退出
2. 系统故障 CPU超负荷、内存超负荷、网络超负荷
3. 物理故障 机器死机 断电 断网
4. 不可恢复故障 地震、海啸等等

客户方也会发生相同故障，导致系统不可用，当然会出现个别用户的不可用及区域性用户均不可用。

对于我方发生的问题，我们必须通过架构的方式进行解决，对于客户方发生的问题，我们尽量找方法解决，先解决区域性问题，再解决个别用户问题。解决方案必须要考虑到成本及战略来进行取舍，比如创业初期，根本没有大量资金，要解决不可恢复故障基本不太可能。

我们先试图从架构的方式来解决我方发生的故障，这种解决方案类似于设计模式，故称之为架构模式。

针对单机的不可用，有一个专业术语叫做单点故障，最好的方式就是部署多机器，通过多机器负载均衡，来规避单点故障。

1. 分布式
2. 负载均衡

针对多机的不可用，我们需要分类看如何解决：

1. 程序故障
   功能错误、程序退出，这种错误有同学说，可以加单元测试、功能测试，让测试来发现问题。是的，但是那是开发流程，我们先不讨论那个，我们从架构的角度讨论，主要的解决方案如下：
   - 分批自动化发布
   - 灰度发布
   - 异常监控
2. 系统故障 CPU超负荷、内存超负荷、网络超负荷
   - 流量控制
   - 功能降级
   - 动态扩容
   - 异常监控
3. 物理故障 机器死机 断电 断网
   - 异地多活
   - 异地热备or冷备
   - 异地数据同步

4. 不可恢复故障 地震、海啸等等
    - 同上

后面我会针对每个专题跟大家仔细讲解。

架构师速成 8.3-可用性之分布式 - for5million 的专栏 - 博客频道 - CSDN.NET

分布式算是软件界发展的一个里程碑，它开辟一个新的软件时代，其他的溢美之词我就不再乱说了。

分布式按照我的观点，应该分为有状态和无状态 2 种：

1. 有状态
    - 分布式数据库
    - 分布式存储
2. 无状态
    - 分布式计算
    - 分布式web系统

当然分布式尽量做成无状态的分布式，但是存储最终因为最终存储的是有状态的数据，所以不得不变的有状态。当然 web 系统也可以是有状态的，但是最好做成无状态的，因为无状态可扩展性更强，而有状态必须维护和确定状态和机器的对应关系，无谓增加了很多复杂性。例如：

无状态的集群，我随意增加或减少一台机器，对原有机器无任何影响。有状态的，我就需要知道新增或减少的机器，应该分配哪些数据哪些数据。

我们先从简单的无状态分布式系统说起，分布式计算我就不再展开了，针对 web 系统如何架构为无状态讲解一下其中的关键：

1. web系统有状态的原因是什么？如果用户都不需要登录，系统本身就是无状态的，所有人的看到都是一致的。但是用户登录之后，你要一直显示这个登录用户的相关数据，导致了系统变为有状态。
2. 如何解决这种有状态呢？如果这种状态可以作为数据每次通信时传递给web系统，那web系统就可以变为无状态。这也是常见的有状态变为无状态系统的解决方案，例如web系统将有状态的数据存储在db。解决方案很简单，就是cookie记录用户id。有人说用户登录之后有好多数据，我cookie存不下，而且有安全问题，那就加一个无状态的cache集群，存储所有的用户数据。cookie的id来了之后就去cache拿一下所需的数据，针对这个有个专业的名词叫做 share nothing architecture，sna架构。

3. 有了无状态的web系统，如果请求量增加，很easy，加机器就可以了。另外有些同学说加的机器怎样感知到，如果有机器宕机，又是如何感知到的，这个是负载均衡干的事情，后面会单独再讲。

无状态的讲完了，那再讲讲有状态的分布式，这个可能需要的篇幅比较多了，让我再单开一章吧。

刚才查了一下，果真有人已经提出了有状态和无状态，呵呵，看来还是有人更专业。

架构师速成-如何高效编程 - for5million 的专栏 - 博客频道 - CSDN.NET

今天看见有个 csdn 的征文大赛，谈谈如何高效编程，正好之前有些感触，先沉淀下来。

引子

赵云大喝一声，挺枪骤马杀入重围，左冲右突，如入无人之境。那枪浑身上下，若舞梨花；遍体纷纷，如飘瑞雪。

赵云是所有历史人物中我最喜欢的一个，如果放到现代，他走了 it 的道路，一定可以成为一个编程高手。为什么？

其实古时打仗也是一门技术活，需要有勇有谋，跟 it 没有什么太大的区别。打仗要修身、修技、修器才能左冲右突，如入无人之境，同样做 it 也要修身、修技、修器，才能高效编程，如入无人之境。

高效编程的修炼

何谓修身、修技、修器？

- 修身，古今都是修炼自己的体魄和思维，使自己体魄强健，思维敏捷
- 修技，古代修炼自己的武艺和兵法，如今修炼自己的设计方法、模式及设计技能
- 修器，子龙左手青釭剑，右手亮银枪，跨下白龙马，一身白盔白甲，帅到爆炸！如今我们左手xmind，右手eclipse，uml在脚下，也是无比潇（ku）洒（bi）。

待修到山花烂漫时，需求从中过，片叶不粘身。

修身

修身是一个很大的话题，要讲个几天几夜了，所以我们就先略过。当然修身之后，可以从思维的高度上确定大的方向，但是大家的看了题目也不太会关心如何修身，so pass！后面省略十万字，我可真够高产的（稿费不要省略啊，我不怕麻烦）。

修技

古时修技都是有秘笈，而且必须有独门功法历尽千辛万苦才能修炼成功。如今好了，有很多书和视频，还有我这种让人醍（提）醐（壶）灌顶的好教程，所以你只需要少吃一点苦就可以练成了，我算算啊-------你大概历尽 999 辛 9999 苦就可以了，看少了 1 点吧。

- 不明真相群众：揍他，揍他小舅子
- 作者：亚美蝶，揍我不行，揍我小舅子可以。
- 作者：哎呀，轻点
- 作者：呀，轻点
- 作者：轻点
- 作者：点
- 不明真相群众：KO
- 作者：请让我讲完，55555

修技大概分为：

- 面向对象、面向过程设计方法（心法）
- uml设计语言（秘笈）
- 设计模式、重构、xx最佳实践（先辈实战经验）
- 单元测试（对打练习）

等修炼到 10 成功力，才可以继续修炼终极秘笈架构模式，否则会走火入魔、oom。

- 不明真相群众：等一下，我们在讨论高效编程，你跟我们讲什么面向对象，你疯了吧！揍他，揍他小舅子
- 此处同上，省略100字。
- 作者：请让我讲完，55555

其实看到题目我首先想到的也是编程工具的使用，以及快捷键的使用。后来我发现那只占我浪费时间的 5%-10%，真正浪费我们时间的是：

- 老板说：哎呀，我们方向错了，我们重新搞一个方向吧！（产品、码农、测试心中下起了大雪，好冷，好累，我要屎了，砍死他.....看在钱的份上，先饶你一条狗命）
- 产品说：哎呀，需求不是这个样子的，我们应该改成这样.....（码农心中一万只草泥马跑过）
- 测试说：哎呀，这么多bug，快来改（测试心中鄙视的看着你，菜鸟，切！）
- 码农说：哎呀，这样地方设计不合理，我们需要重构。哎呀，这样地方if else好多，我晕了。哎呀，这地方出异常了。哎呀，.....（哎你妈个头啊!你有完没完啊!）
-
- 不明真相群众：好像很有道理的样子，先饶你一条狗命
- 作者：谢谢大侠

其实我们看看这些问题怎么来解决：

- 老板的问题，我先不解释了，老板最英明！（老板在后面看着呢）。《精益创业》《精益创业实战》，好像有点修身的味道。
- 产品的问题，有些错误的地方，你在设计时会感觉不对，那就问题所在。但是有些就是方向性的问题，这可以归结到第一个问题。（产品说：这个锅甩的好）《head first 面向对象分析与设计》《领域驱动设计》
- 测试的问题，单元测试起码保证最基础的代码的正确性，上面集成测试，功能测试，系统测试才有可能正确。《单元测试之道》《测试驱动开发》
- 码农的问题，先辈的经验先学会，你就不会有那么多需要哎呀的地方。《重构》《Head first 设计模式》《代码大全》，《Effective java》《代码整洁之道》

秘笈已经放在后面了，请大家笑纳！

- 不明真相群众：好像还不错，看在秘笈的份上先饶你一条狗命
- 作者：谢谢大侠
- 不明真相群众：等等，终极秘笈没有给
- 作者：sorry，各位大哥，《企业架构模式》，《大型网站技术架构》

修器

有小说讲到高手已经不需要好的武器，一片叶子，一根枯枝，都可以分分钟秒掉一片人。还有人谣传，最牛逼的开发人员只使用文本编辑器，巴拉巴拉，一个操作系统开发出来了。

我只问一个问题，你用不用电器，你开不开车？

如果用，那就继续，如果不用，那就闪开，骗子，你怎们看到我的文章的。

人活着的目的就是为了让更多人的获得更多的自由，时间自由，财务自由，人身自由。开发工具的目的是为了，让你更快的完成一些重复的繁琐的事情，让你有时间去享用你的自由。所以尽情的使用工具，工欲善其事，必先利其器。

- 不明真相群众：好伟大
- 作者：我只是说说而已，不要打我。

要高效，必须把最常用的工具修炼到极致：

1. 操作系统，尽量选择苹果、Ubuntu，window次之，当你用命令行爽呆的时候就明白了。当然如果用windows，尽量使用快捷键操作。
2. 开发工具，eclipse 快捷键，统一的格式，重构功能，findbug，checkstyle
3. 自动化编译发布，使用hudson进行定时自动化打包发布
4. 其他工具，xmind 快捷键
5. 时间管理，番茄工作法 控制时间
6. 好的框架，定义好一个好的框架，可以节省大量开发时间。具体内容详见我前面讲到的框架。
7. 代码生成器，输入uml设计的类图，自动生成相关的类，对于需要持久化的对象，可以实现从前到后的增删改查的代码。

实战

产品获得了一个用户需求给到我，我通常是这样做的：

1. 分析需求的合理性，并不是所有需求都是合理的，需要积极发现其中问题。发现的问题跟产品讨论，如果所有问题都解决掉，才进行下一步。如果后面的开发建立在一个错误的需求上，所有开发都是然并卵。
2. 进行分析设计，至少形成uml的用例图和类图。对于关键复杂逻辑，确定是否可以通过设计模式解决。复杂流程需要画出流程图，如果状态变化较多，还需要画出状态图。在uml设计时，仍然会发现需求的问题，此时仍需要跟产品确定，直至所有问题解决掉，才进行一步。
3. 编码实现，根据类图自动生成相关代码，省去了自己一个个再敲一遍的时间，前后端增删改查等重复逻辑都已经有了，只需要写具体的业务就ok了。
4. 编写单元测试，对于自定义代码，必须编写单元测试。这样后期修改代码或者进行集成测试时，才会比较顺畅。
5. 进行findbug，checkstyle自动检查，系统自动化发布，如果出现问题邮件通知。
6. 自测，自己实现功能后，先按照需求自测一般，保证正确性。
7. 提交测试，发现bug及时修改，基本做到日结。

8. 自动化发布上线。

当然中间会尽量节省各种时间：

1. 快捷键操作
2. 番茄工作法，设置不被打扰时间，全神贯注的投入编码
3. 3次以上重复的事情，就考虑实现自动化，比如 发布，代码检查，代码生成
4. 代码规范严格遵守，最优实践严格遵守

团队高效

其实编码不是一个人的事情，团队协作是必不可少的。团队尽量采用如下管理方式：

1. 需求、设计集体参与，不需要再每人讲解
2. 设计后进行分工，如果有新人就采取新老结合的结对编程，对复杂的代码也采取结对编程。
3. 推行结果导向，从产品，需求，设计，开发等，不要白白浪费时间做无结果的事情
4. 代码规范严格遵守，最优实践严格遵守
5. 规范流程，不断磨合各个环节人员，达到丝般顺滑
6. 人员互备，不会出现单点

祝福

上面是我做软件以来的一些感悟，当然一千个人眼中有一千零一个哈姆雷特，每个都会有最适合自己的方法，希望大家都早日修成自己的高效编程之道。

8 月 8 日是一个好日子！ 2015 年 8 月 8 日于阿里巴巴西溪园区

架构师速成 8.3-可用性之分库分表 - for5million 的专栏 - 博客频道 - CSDN.NET

有状态分布式，涉及的知识就比较多了，不过我们可以拿几个现实的例子由浅入深的来理解。

数据库的分库分表

1. 假设你是一个开发负责人，开始使用单机的数据库，突然一天数据库硬盘挂掉了。你没有做备份，然后就没有然后了。
2. 进入第2个公司，你意识到备份的重要性，每天定时备份到另一台机器，突然有一天，数据库硬盘挂掉了。你心想幸好我有备份，然后巴拉巴拉的恢复起来，用了2个小时。老板说不错，但是—-
   我们因为宕机造成大量用户流失，信誉下降，然后就又没有然后了。上面说的就是单点的问题。
3. 进入第3个公司，你觉得单点很可怕，所以主备做起来，数据自动同步到备库，做到随时准备切换。突然有一天，主数据库硬盘挂掉了，你从容的修改数据库连接指向备库，重启系统恢复了，只用了5分钟。此时掌声一片，你沉浸在无比的欢乐中，老板说不错，但是—
   就在这5分钟我们丢了一个上亿的单子。我擦，你不是故意的吧！（其实这有可能是真实的片段，我们创业时，就30分钟断网，结果正好在举行一个大型的营销策划，不说了，我擦一会眼泪），然后就又没有然后了。其实当你用上主备时，说明数据库已经有状态了，必须要区分谁是主，谁是备。
4. 进入第4个公司，你不但做了主备，还做了高可用，通过HA实现了瞬时切换。突然有一天，主数据库硬盘挂掉了，你从容的端起了你的屌丝杯，世界清静了。老板说不错，小子我看好你。从此你走向人生巅峰，出任CTO，迎娶白富美。但是没过多久问题来了，随着用户不断的增加，你的数据库摇摇欲坠，不时就抽疯。老板说搞定他，不然我就搞定你。
5. 咋办，分库分表啊！如何分，这就涉及到更多的规则了，比如按照用户id是最常见的做法。此时你不但需要管主备而且还需要在程序中确定如何路由，结果集合并，如果再有机器增加，还要涉及数据迁移，另外还要防止出现重复id的脏数据，需要全局唯一主键，等等。亚美蝶！知道有状态的痛苦了吧。这也是为什么有些同学转投nosql的存储的很大原因，nosql替你屏蔽了这些规则，他在内部实现了路由、分库、合并等等。
6. 提到这里不得不提一下淘宝的牛逼产品—
   drds（沈公子是不是应该给些广告费啊）。
   - 分布式SQL引擎
     - 将数据按照条件分散到多个数据节点(分库分表)，对于数据操作sql进行分布式优化，获得最佳执行效率
   - 自主运维
     - DRDS的用户运维平台提供DRDS接入、分布式DDL、拆分信息维护、平滑扩缩容、分布式DML、监控等常用功能，让运维工作变得更简单
   - 小表复制

        - 对于配置表，常量表等不经常变化的表进行多节点对等同步，加速该类表与其他拆分表做关联查询的速度
    - 分布式全局唯一id
        - 提供全局唯一数字id服务，帮助您在分布式环境下，继续保持类似唯一键、主键等数据的全局(所有节点)唯一性

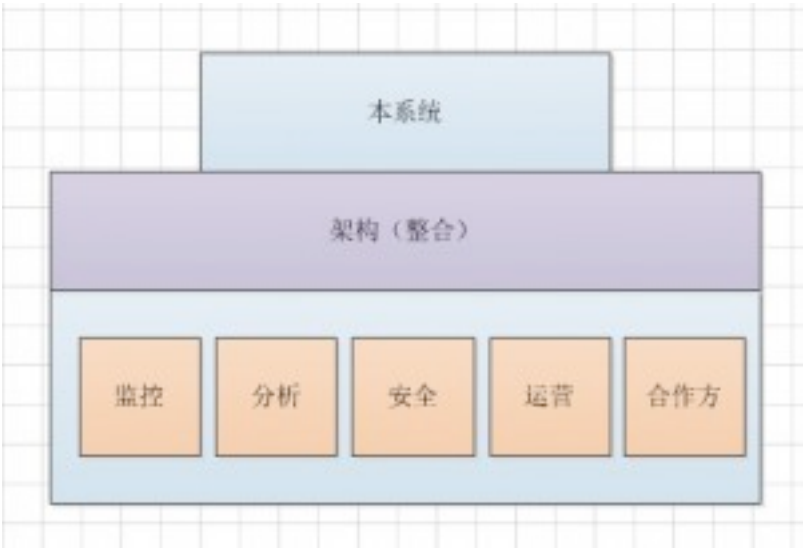
1. 看到了吧，这就是有状态带来的痛苦。为了把有状态变为无状态有时候你需要做大量的工作。

有关分库分表的关键点和难点，我新一章统一讲解。

架构师速成-有关架构的思考 - for5million 的专栏 - 博客频道 - CSDN.NET

架构是什么？架构的目标是什么？如果解决这 2 个问题，可能我能更好的梳理杂乱的架构理论。经过 2 天的思考，总算有了一点眉目。我们从一个产品的本质来说，追本朔源，自上而下：

大概就是这样的，当然架构不止需要解决这些问题，本产品只是其中一个部分，要支撑一个 web 产品还需要依赖很多的外部公共系统，对这些系统整合也算作架构的范畴。



架构，又名软件架构，是有关软件整体结构与组件的抽象描述，用于指导大型软件系统各个方面的设计，这是官方的定义。

在"软件构架简介"中，David Garlan 和 Mary Shaw 认为软件构架是有关如下问题的设计层次："在计算的算法和数据结构之外，设计并确定系统整体结构成为了新的问题。结构问题包括总体组织结构和全局控制结构；通信、同步和数据访问的协议；设计元素的功能分配；物理分布；设计元素的组成；定标与性能；备选设计的选择。"
但构架不仅是结构。IEEE Working Group on Architecture 把其定义为"系统在其环境中的最高层概念"。构架还包括"符合"系统完整性、经济约束条件、审美需求和样式。它并不仅注重对内部的考虑，而且还在系统的用户环境和开发环境中对系统进行整体考虑，即同时注重对外部的考虑。

大家自己想想自己心目中的架构定义吧，我好像触摸到他，但是还没有完全的感悟，这几天我需要再深入的思考一下。

架构师速成 8.4-分库分表的关键点 - for5million 的专栏 - 博客频道 - CSDN.NET

我们还是由浅入深（这个词我喜欢，你呢？）的讨论一下，分库分表的关键点（本故事纯属虚构，仅为搞笑）：

1. 当你的系统很小的时候，只有一个数据库，每个表的主键都是自增的，你都不去关心主键变成了多少，反正db保证自增，小日子过的很是惬意。但惬意的日子总是短暂的，你因为DB宕机被老板fire 3次（见上一个故事）。
2. 进入第4个公司的时候，你发奥涂墙，将集群改成主备HA,结果顺利出任CTO，迎娶白富美，走向了人生巅峰。当然这中间也出过一些小插曲，比如：张三注册时，刚点击完注册，DB主机宕机了。张三发现刚注册的账号不能登录了，张三很生气。你说这这不算啥，让他重新注册一下账号吧。但是李四刚付款买了

网站上一款价值9.98的玉镶金的超级玉佩，刚过1秒，订单就没有了。李四不愿意了，他还等着这个9.98的玉佩3天内升值500%大赚一笔呢。你灵机一动，我们写一个数据订正的程序吧，对于宕机丢失的数据进行订正，另外加送李四同学一块999的超级金树叶。问题都摆平了，你果真是维护世界和平的正义使者！

3. 由于只要9.98的超级玉佩口碑传播太好了，有无数人等着购买，甚至有人肯出更高的价购买，但是我们是有操守的，只卖9.98,9.98你买不了吃亏，买不了上当，现在购买还可以....此处省略1000字广告。于是网站的注册用户暴涨，数据库时不时卡死。
    - 老板发飙了："怎么回事,我造福全人类的大业，要毁在你手里，你马上解决，要是解决不了，你就是人民公敌，社会败类！我会让无数大爷大妈一人一口唾沫吐死你！"。
    - 你：看来我们要分库存储了，一台数据库，完全抵挡不住大爷大妈们的热情啊，马上给点经费吧。
    - 老板对你一阵痛骂："这得多少金镶玉啊，要不用金镶玉付款吧"。
    - 你：老板英明。
4. 又增加2台机器之后，突然发现你是的世界完全崩塌了。
    1. 哪些数据需要分库呢？
    2. 原来数据怎么分到这2台机器上呢？
    3. 我查询的时候怎么知道查哪一套集群？
    4. 自增主键太坑了，自增完都重复了，怎么办？
    5. 原来的关联查询（我无数牛叉的sql），分库之后怎么办？
    6. 我的事务一致性怎么办？
    7. 原来的count，sum,group怎么办？
    8. 要是需要再分我怎么办？但是你被fire3次之后，早已练成神功之————-

       死猪不怕开水烫，既然天降大任于我，我就全力去搞，顺便鄙视一下这个老板。
5. let's google，我靠，有专门的资料，http://blog.csdn.net/column/details/sharding.html，还有专门的书籍《MySQL性能调优与架构设计》，照猫画虎，大功告成。
    - 不明真相的群众:这样也太坑了，直接把最关键的略过了。退货，退货，揍他，揍他
    - 作者：我只是想告诉大家怎么思考，为什么会出现这些问题
    - 不明真相的群众：狡辩，揍他
    - 作者：饶命啊，大侠，我是觉得写起太费劲了。而且我也向大家展示了面向对象威力，这个分库分表就是一个完整的对象，我只需要引用他就可以了
    - 不明真相的群众：好像比较有道理，但是就是因为你数衍，揍他

6. 搞好之后，流量大增，老板心里乐开了花，对你大加称赞，并奖励了你一块金镶玉。
    - 老板：小子，我看好你！这块金镶玉就送给你，不日就升值1000倍，你就是大富翁了。
    - 你：呵呵，谢谢老板，老板英明
    - 老板：看你嘴甜，再送你10块
    - 你：呵呵呵，证据到手
    - 你：警察局吗，有人在诈骗
    - 老板：亚美蝶……，我是在造福全人类
    - 你：对不起，我是卧底！

架构师速成-架构体系 - for5million 的专栏 - 博客频道 - CSDN.NET

经过这段时间的反思和整理，终于对架构有了一个较为明确的理解。架构是产品从无到有以及慢慢壮大过程中所需要的全部技术体系总称，架构过程：

1. 配置、编码、测试、运维、监控分析、安全、运营等一系列技术体系的选型、取舍
2. 技术选型基础上进行规划、设计、实现、迭代、制定相关规范
3. 相关技术及规范运用到产品开发的整个过程中，并在产品迭代过程中对架构进行迭代优化

架构不止包含技术的框架，比如有人用了 spring 就觉得我已经是架构师了，其实架构并不是这么简单。我们以做一个新浪微博类似产品为例，现实应该是这样的：

1. 产品初期，经典的LAMP快速开发实现第一个版本，功能也无比简单就是加好友，发消息，开发人员也只有一个小的队伍。此时的架构就体现为纯的技术选型及实现，包含了
    - 配置：代码通过git管理，暂时无其他
    - 编码：技术选型为LAMP，基于LAMP的开发框架封装，并在开发团队内制定开发规范
    - 测试：技术人员手工测试
    - 运维：手工发布，主备2台，mysql也做主备
    - 监控：暂不需要
    - 安全：发帖过滤、屏蔽
    - 运营：后台删帖
2. 随着用户的暴增以及功能的增加，产品需要迭代，架构也需要迭代。产品功能增强，性能需要优化，开发人员的增加，此时架构就发生了一个较大的变革：

    - 配置：代码通过git管理，要分为多个模块，不同团队开发不同模块
    - 编码：技术选型增加缓存、消息队列、搜索引擎等，框架封装更加复杂，抽象出基础层和服务层。分团队进行代码的维护，制定不同模块之间通信标准。推拉模型也提炼出来。
    - 测试：单元测试+自动化测试
    - 运维：批量自动化发布、回滚、支持灰度发布
    - 监控：增加流量监控，机器监控
    - 安全：发帖过滤、屏蔽，防范其他攻击
    - 运营：后台删帖、大V管理等等
3. 用户和流量再次增加后，产品再次发生变革，架构也再次变革。
    - 配置：代码通过git管理，服务化，每个服务单独一个模块，不同团队开发不同模块
    - 编码：技术选型需要考虑异地容灾，层次继续抽取，服务粒度细化，增加开放api，改进推送架构。
    - 测试：单元测试+自动化测试
    - 运维：批量自动化发布、回滚、支持灰度发布，异地数据同步
    - 监控：增加流量监控，机器监控，增加缓存等监控
    - 安全：发帖过滤、屏蔽，防范其他攻击，开放api权限管理oauth2，防止恶意调用
    - 运营：后台删帖、大V管理等等

从上面的例子可以看出，架构是跟随产品进行迭代的，而且随着产品的越来越复杂，不可避免的需要不断拆分，多团队合作，运维自动化。架构也变成团队的工作，而不是一个架构师就搞定一切了。

阿里发展到现在架构也有些类似，有很多不同团队负责底层设施（中间件）的开发迭代及其架构的革新。业务也有不同的团队负责开发不同的业务模块，这个都使用统一的架构体系。技术保障部门维护统一的自动化运维工具，安全部门维护安全工具。不同部门都有自己的架构师，负责本部分的架构，不过比较遗憾的是没有一个总架构师的角度去推动总体架构演进及各个部门架构的优化。

架构师速成-架构的目标 - for5million 的专栏 - 博客频道 - CSDN.NET

架构的目标为了实现以下特性：

- 正确性
    - 系统首先需要正确，运行稳定
- 可用性

        - 软件系统对于用户的商业经营和管理来说极为重要，因此软件系统必须非常可靠，一般99.99%是一个比较基本的要求。
- 快速开发
    - 互联网目前是一个快鱼吃慢鱼的时代，已经不是大鱼吃小鱼了。因为小鱼在一夜之间就长大了，把大鱼吃掉了。诺基亚就是明证，facebook就是明证。
- 良好体验
    - 良好的体验对用户的吸引力是巨大的，某退公司往往是抄一个产品，把用户体验做好，然后原产品就没有然后了。
- 伸缩性
    - 用户激增的时候，网站可以伸缩来支持用户的增长或流量高峰。
- 安全性
    - 安全也是一个商业公司的命脉，攻击、泄密、破解，前一段闹的沸沸扬扬的各种用户信息泄露，足以说明安全的重要性。
- 扩展性
    - 网站在增加新模块或者新的技术时，能比较容易的扩展。
- 高性能
    - 性能其实也是用户体验的一部分，尤其是用户量不断增多，性能是节省成本的重要手段。
- 可定制化
    - 同样的一套软件，可以根据客户群的不同和市场需求的变化进行调整。入门版、企业版、旗舰版针对不同用户，不同人群是绝对必须的。
- 可维护性
    - 一个产品，一个网站上线之后，80%时间需要不断的更新及增加新的功能，可维护性是一个架构的基本需求。

为了实现以下特性，需要依赖本产品系统的架构模式以及流程规范，另外也需要支撑的系统支持及流程规范。 架构不只是要关注本系统的技术实现，同时也需要关注支撑系统，支撑系统也是架构不可缺的一部分。

架构师速成-架构目标之正确性 - for5million 的专栏 - 博客频道 - CSDN.NET

本系统架构模式：

- 统一异常
    - 统一异常处理是保证程序正确性的第一步，这是第一个架构模式。具体如何实现，详见前面的文章。
- 日志

        - 日志也是保证程序正确的一大手段，虽然是在错误出现后，日志才会记录。但是日志是快速确认问题，并分析出隐藏问题的重要手段。
        - 关键点
            - 日志文件按照级别进行区分，将错误和普通调试日志分开
            - 日志文件滚动方式，可以按天及按大小滚动，定时清理
            - 日志级别可以实时调整设置
            - 性能

支撑系统：

- 测试系统
    - 自动化单元测试，保证基础模块的正确性
    - 自动化功能测试，保证每次代码更新的正确性
- 监控系统
    - 监控异常日志，及时报警
- 运维系统
    - 自动化发布，减少人为操作造成错误
    - 分批次发布，可以在线上小数据量测试，保证功能正确后，再全部升级
- 对账系统
    - 对应金钱或库存相关的系统，需要有实时对账系统，校验每个订单是否符合预期，及时止损

架构师速成-架构目标之可用性 - for5million 的专栏 - 博客频道 - CSDN.NET

---

1. 服务器等，从而共同完成工作任务。各种负载均衡的软硬件有很多，我们可以单独讲解一下。
2. 配置中心，原来单一节点的配置，被类似zookeeper的多节点配置中心取代。
3. 流量控制，流量控制是保证大流量下系统可用性的重要手段，当系统流量不足以支撑所有流量时，只保留合理的流量处理。其他流量直接丢弃，否则系统会被压垮，造成雪崩。
4. 功能降级，另外大流量情况下，有些无关紧要的功能可以暂时降级，后期通过数据补全的方式进行修正，将核心的资源用于最关键的业务。比如双11时，为保证购买可以暂时不考虑推荐，这样省掉推荐资源，供给购买环节。
5. cdn，其实也算是分布式的一种解决方案，但是更专业，所以单独讲解。智能路由、资源加速等等是满足可用性的利器。

为满足可用性，需要的支撑系统：

1. 运维系统
    1. 运维系统需要提供分布式的自动扩容
    2. 热备、冷备
    3. 各种分布式系统的运维工具
    4. 异地多活
    5. 分批自动化发布
    6. 灰度发布
2. 监控系统
    1. 分布式服务的健康状态监控
    2. 流量监控
    3. 故障监控、异常发现

架构师速成-架构目标之伸缩性\安全性 - for5million 的专栏 - 博客频道 - CSDN.NET

为满足伸缩性，所需的架构模式包含：

1. 分布式，这个前面有单独的章节进行了讲解，分布式是互联网时代的主旋律。
2. 负载均衡，前面已经有讲解。
3. 服务拆分，按照业务进行系统服务的拆分并单独部署。

为满足伸缩性，需要的支撑系统：

1. 运维系统：
    1. 自动扩容，缩容
2. 监控系统
    1. 监控流量，确定何时伸缩

为满足安全性，所需的架构模式包含：

1. 数据加密，密码的加密存储、关键数据的加密传输，才有https。
2. 数据校验，数据传输时，同时传递明文和密文校验数据未被篡改。前端及后端进行数据的有效性校验，比如数字、日期等。

为满足安全性，需要的支撑系统：

1. 安全系统

1. 屏蔽字过滤，帮助过滤发言的屏蔽字，并能及时新增屏蔽字
2. 垃圾过滤，过滤广告机及其他
3. 安全扫描，自动扫描发现网站安全漏洞
4. 攻击行为识别，自动定义或识别用户的攻击行为，及时预警
5. 防攻击，通过防火墙防止各种攻击行为