27 软件开发的基本步骤，无套路不欢

更新时间：2020-08-17 09:43:23

> 劳动是一切知识的源泉。——陶铸

前言

你好，我是彤哥。

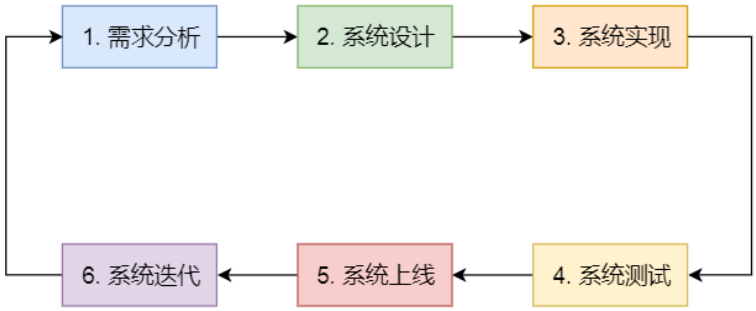
在前面的章节中，我们一起学习了 Netty 的发展历程，熟悉了它的核心组件，并从源码级别对它做了完整的剖析，通过这些学习，我们发现，Netty 确实做的很不错，不管是性能、效率，还是易用性，甚至是可学习性，它都做得很完美，将简单化做到了极致。

这里我们提到了易用性，前面也见识过了 Netty 的基本使用案例，确实非常简单，那么，在互联网级应用中，Netty 是否还是这么简单易用呢？

从本节开始，我将以游戏为背景介绍 Netty 是如何在互联网应用中落地的，但是，在正式介绍实战项目之前，我想先介绍一下软件开发的基本步骤，或者说是套路。

软件开发的基本步骤

在学校有学过软件设计或者系统设计的同学，对于软件开发的基本步骤都比较了解，不过，我想根据实际的情况对学校教过的知识再做一些补充，使这个体系更加完整健壮。



上面这张图涵盖了从 产品 -> 研发 -> 测试 -> 运维 的整个产品开发周期的全图谱，这也是目前互联网公司软件开发的基本步骤，作为系统架构师，需要对每一个环节进行把关，才能做出健壮的产品。

需求分析

对于需求分析，我将它分成三个部分：

1. 需求收集
2. 详细分析
3. 可行性分析

首先，在当前的互联网环境下，基本上能想到的产品都有人做过了，而且，很多公司已经帮我们培养好了用户习惯，所以，需求收集，简单来说，就是看看竞品是怎么做的，直接抄是最方便的，这也是很多研发同学吐槽产品经理的地方，但是，事实就是如此。比如，老板让你做一个类似阿里云的东西，难道你真的傻傻地在脑海中构造一幅蓝图？

当然，收集的需求不仅仅包括业务侧需求，还应该包括研发侧需求，在很多公司，第一个版本急于上线，会落下很多研发债务，这部分债务在后期也是需要通过研发经理转换为研发侧需求的，比如性能测试、安全整改等。

对于收集到的需求，下一步就是进行需求的详细分析，哪些需求是有意义的，哪些需求是锦上添花的，哪些需求是可有可无的，这一块光靠产品经理可能就无法想全面了，这时候就需要架构师出马，对于产品收集的需求进行详细的分析，并给出合理的建议。

经过第二步，可能砍掉了一些不合理的需求，但是，不想当将军的研发不是好研发，所以，最后一步，一定要站在老板的角度想问题，也就是可行性分析，研发关注的是实现需求和个人成长，而老板关心的永远只有一个字 —— 钱，所以，可行性分析最重要的点就是能不能带来收益，能带来多少收益，投入产出比是多少，只有当产出大于投入时，这才是一条好的需求，才是值得做的。

系统设计

经过了需求分析，拿到了完整的需求，架构师就要开始做系统设计了，通常来说，架构师会从下面几个方面来做系统设计：

1. 技术选型；
2. 领域模型设计；

3. 接口设计；
4. 部署架构设计；

首先，技术选型，对于大部分互联网公司，做的产品都是基于 RESTful 的应用，所以，一般都会选择 springboot 或者 springcloud 作为他们的后端架构，但是，对于游戏或者证券类的应用，因为需要实时刷新，此时，再使用 spring 家族的技术就很难做到了，对于这部分应用，如果后端是 Java 的话，我是非常建议使用 Netty 的。

对于 Netty 应用，我想在技术选型这里再扩展几个点：

1. 网络协议选型；
2. 数据协议设计；
3. 编解码设计；

网络协议选型，是基于 Http，还是基于 TCP，还是基于 WebSocket，亦或者是 UDP？如果有自己的 APP 端，使用 Socket 没问题，如果后面又要做小程序呢，又要做 Web 应用呢？所以，通常的做法，我建议使用 WebSocket，这样不管后面增加多少种前端，后端的网络协议基本不需要修改。

数据协议设计，有没有比较通用的数据协议设计呢？从大的方面来说，有，现在比较流行的做法基本上都是分成请求头和请求体，从小的方面来说，又没有，因为具体的字段还是要根据业务场景来定义，比如，对于游戏场景，为了保证消息的时序性，我们一般会添加类似 requestId 或者 sequenceId 的字段。

编解码设计，这一块就是我们前面介绍过的内容了，对于编解码，通常分为协议层的编解码（一次编解码）和 Java 对象的编解码（二次编解码，序列化），协议层的编解码，我们通常采用 长度 + 内容 的方式，对于 Java 对象的编解码，一般可以选择 JSON 或者 Protobuf，不过，对于游戏，我们通常还是会选择 Protobuf 的。

好了，对于技术选型这一块我们就先说这么多，选好了技术，下一步就是根据业务来做领域模型的设计了。

领域模型设计，通常也叫作数据库设计或者数据结构设计，简单点讲，就是把业务抽象成 Java 可以操作的对象，这些对象有可能会存储在数据库中，如果是基于领域模型设计的话，它们还会有一些行为（方法），不过，现在大部分的场景都是使用的 MVC 设计，行为逐渐被抽离到了 Service 层。

领域模型设计好了，就轮到接口设计了，通常来说，架构师是不会干接口设计这个活的，因为实在太多了，对于一个正常的系统来说，基本上都会有几百个接口，如果都依赖架构师来设计，就太延误工期了，所以，通常来说，架构师会分好模块，扔给下面的研发人员来设计，再收集上来，大概看一遍，没问题即可。

最后，就是部署架构设计了，这一块也是架构师需要提前想好的，是单机，还是多机，还是微服务，当然，这一块也会跟第一部分的技术选型有一定的关系。对于现在的大部分互联网应用来说，通常设计成服务无状态即可，这样如果一个节点无法支撑业务量，加一个节点就行了，非常简单，但是，对于游戏应用来说，情况就不太一样了，游戏为了追求性能，很多数据是存储在内存中的，所以，它是有状态的，这样又该如何扩展呢？这个问题，我们后面详细聊。

系统实现

系统设计完，就可以着手实现了，其实，在接口设计完就可以实现了，这一步没啥好说的，就是 coding、coding、再 coding，但是，对于我们本次的游戏实战项目呢，我想分成下面几个点来实现：

1. 协议实现，双端打通；
2. 领域模型实现；

3. 业务逻辑实现；
4. 客户端 Mock 实现；

服务端、客户端实现，是指实现服务端、客户端的框架实现，使得他们可以正常的通信，能够收到互相请求的内容等。

业务逻辑实现，把主要逻辑都实现了，在上面框架的基础上实现。

客户端 Mock 实现，因为本次课程我们主要的关注点还是服务端，所以，需要做一个 Mock 客户端来与服务端进行通信，当然，这个客户端很简单，使用命令行即可。

系统测试

系统测试，分成很多种，比如单元测试、冒烟测试、接口测试、功能测试、性能测试、白盒测试、黑盒测试、研发自测等，一般来说，研发需要编写单元测试用例、冒烟测试用例等，专门的测试人员要做接口测试、性能测试、冒烟测试等，在国内，大部分企业对测试这块的要求都不是很高，特别是小型互联网企业，可能更大一部分是依赖于专门的测试人员，我认为这是不完全正确的，作为一名合格的研发人员，一定要保证自己手中出去的东西是经过自测可用的。

在本次的实战项目中，我们会通过 Mock 客户端的方式进行自测。

系统上线

经过完整的系统测试，终于可以上线了，直接扔给运维吧？那你就错了，上线也是每一个后端成员特别是研发经理应该关注的事情，生产环境 JVM 参数如何配置？CPU 多少？堆内存多少？堆外内存多少？预估业务量多少？应急预案是什么？这些都是跟上线息息相关的问题，只有做到了深入理解，才能安心睡觉。在系统上线完成后，产品还要基于生产做验收等。

不过，在本次的实战项目中，我们也不需要上线，所以，也不会涉及过多关于系统上线的问题。

系统迭代

系统上线了，终于可以歇息了，那你就错了，产品还有源源不断的需求过来，急手急脚地上线，会产生大量的研发债务，这些债务又会转化成研发需求，同时，可能还有生产问题，甚至于，老板要找你看报表，这些都会转化成新一轮的需求，进入下一次迭代。

不过，对于本次的实战项目，我们并不关心产品需求，我们关心的只有研发债务以及老板的报表，所以，我在实战项目之后还安排了实战调优等着你哟 ^^

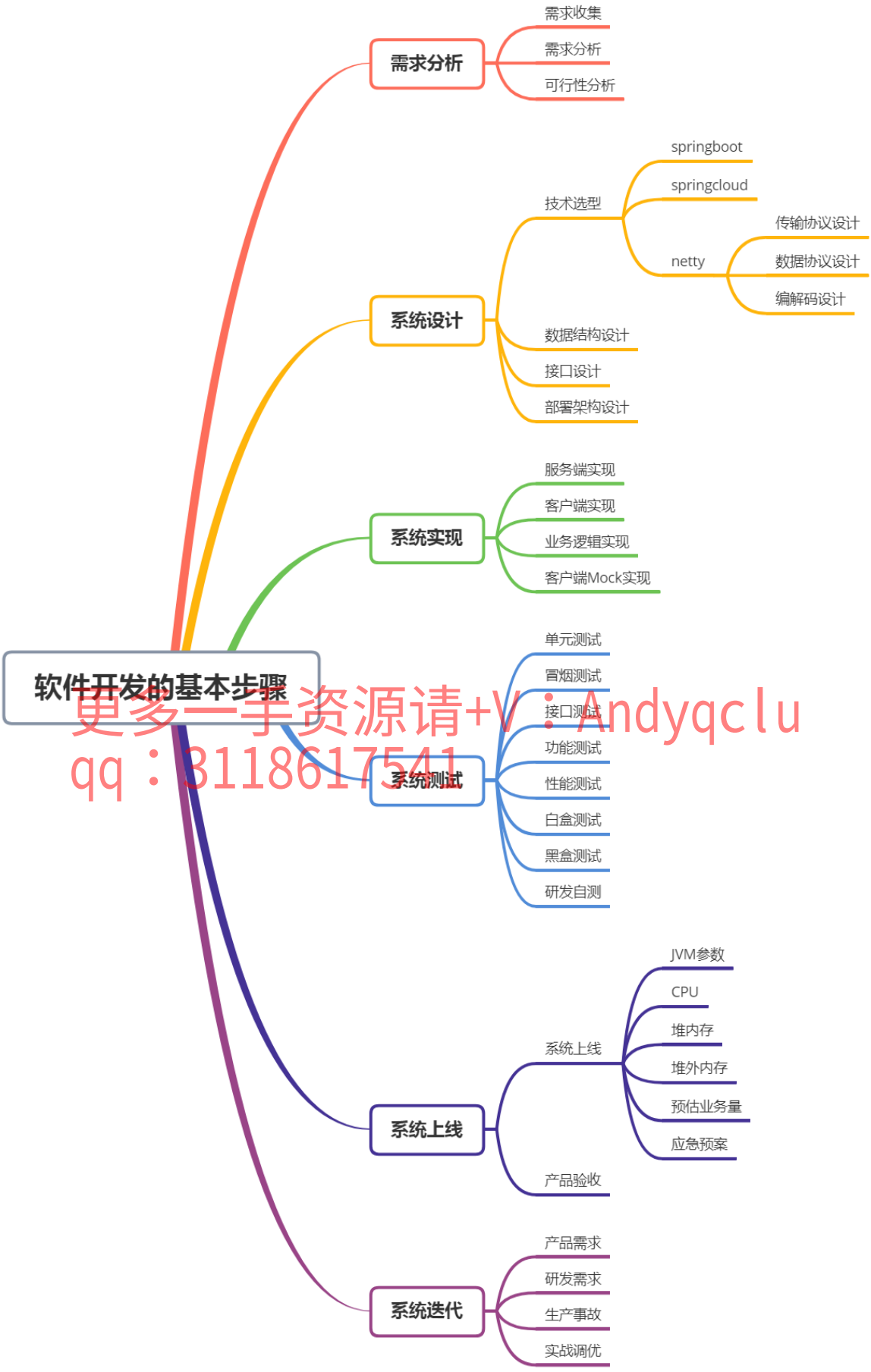
后记

本节，我们从软件开发的基本步骤出发，介绍了整个软件开发的生命周期，同时，对于每一个点，也结合了我多年的工作经验，对它们做了一些扩展。

在本次的实战项目中，除了个别步骤不会执行外，我们基本上会按照这个步骤，来实现我们的实战项目。

下一节，我们就从需求分析阶段入手，详细介绍本次实战项目的具体内容。

思维导图



}