去年此时，刚刚在Coursera上学完*Functional Programming Principles in Scala*，开始看其后续课程*Principles of Reactive Programming*，之前只听说过前端开发的Responsive Design，还是第一次听说**Reactive Programming**，更不要说**Functional** Reactive Programming。这门课一开始就从Monad讲起，几位教授的北欧口音（主要是Erik）略有些难懂，听的实在迷糊，中途就放弃了。

但心里的种子已经种下，时隔一年，下决心要这个心结给了了。重看这门课，忽然发现，很多之前觉得迷糊的东西，经过一年的发酵，竟然变得清晰起来：透过现象看本质，这个看起来很玄的FRP，和学校里学过的时间序列信号处理，异曲同工嘛。

响应式宣言

关于响应式的官方解释，参见响应式宣言。简单来说，响应式开发是为了解决传统软件架构难以满足现代对于大型系统的需求而提出的。

	传统	现代
运行环境	几十台服务器	从数不清的移动设备到上千台多核服务器组成的云集群
响应时间	数秒钟	毫秒级
维护方式	数小时宕机维护	全年无宕机
数据量	GB级	PB级

需要强调的是，正如在需求变化缓慢的前提下，敏捷并不比瀑布更高效一样；在系统规模不大的情况下，响应式开发带来的益处并不明显。

响应式宣言里提到，响应式系统有四个设计原则：

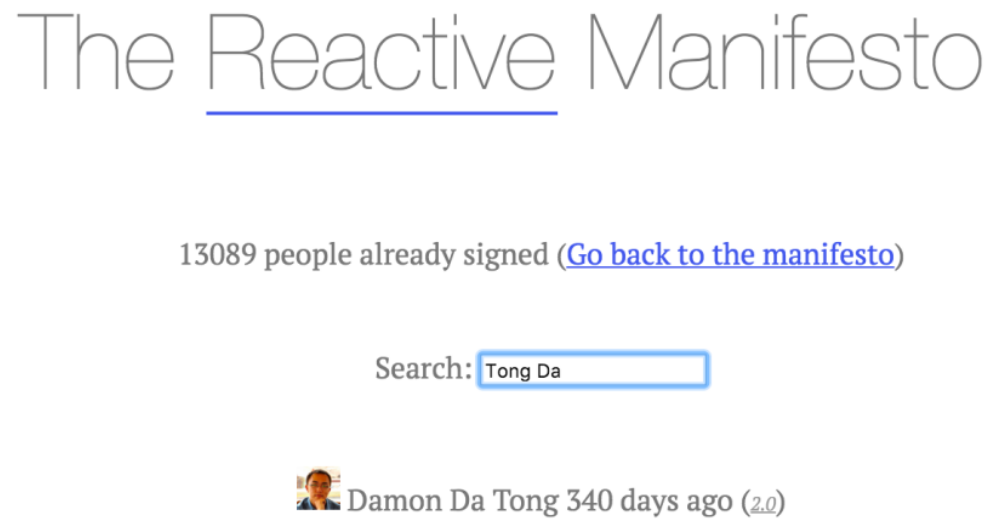
1. 灵敏响应（Responsive）：系统应尽可能做到及时响应。灵敏响应不仅仅是对可用性和实用性而言，更是对响应速度和效率提出了更高的要求。类似实时操作系统，响应式系统要能够在一定的上限响应时间内对用户做出响应，提供一致的服务质量（QoS）；
2. 韧性适应（Resilient）：系统能够在发生故障时保持响应能力。除了对于系统高可用（Hight-Availability）的要求外，韧性适应性要求系统还要能够通过其他内部或外部代理模块实现自动修复，同时故障要被隔离在模块内部，对系统其他部分，以及系统的客户端都不能造成影响；
3. 弹性伸缩（Elastic）：系统能够在不同负载环境下保持响应能力。在系统输入负载变化的过程中，系统要能够增加或降低提供服务的资源分配。这意味着系统设计不能包含竞争点（contention point）或者中心瓶颈，以提供通过分片或复制组件来分发系统负载的能

力。响应式系统还要能够通过实时性能监控，提前进行预测性的系统伸缩。

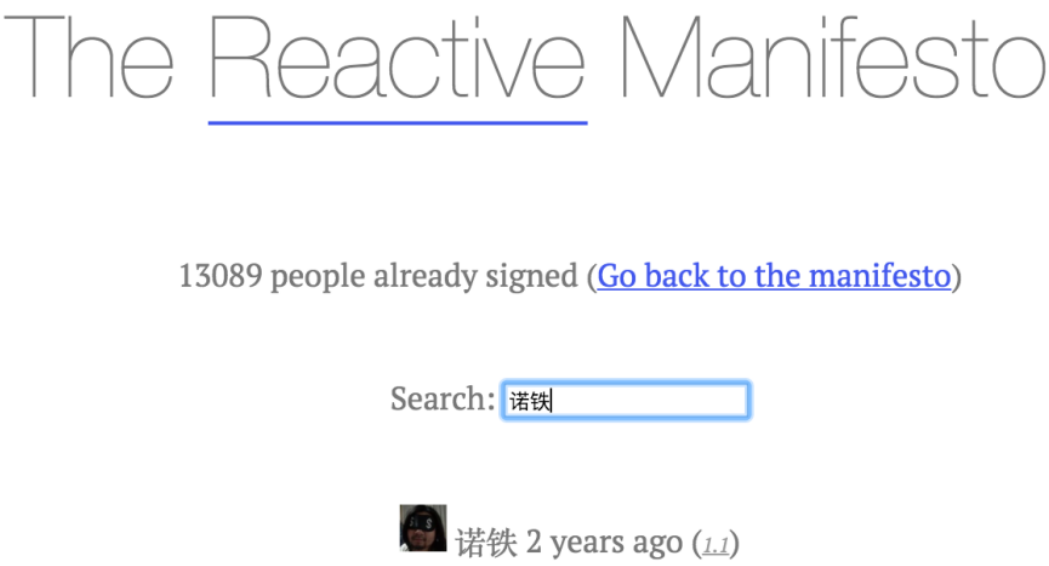
4. 消息驱动（Message Driven）：响应式系统依靠异步消息传递实现模块间的通信，以保证松耦合，隔离，地址透明性等。引入消息传递机制能够让系统通过监控消息队列状态实现负载管理，弹性，以及分流控制。地址透明的消息能够让系统更容易处理故障。非阻塞通信能够让接收方减少系统开销。

虽然这些原则是针对大型系统提出来的，但大型系统都是由小的系统组成，所以应该在系统设计之初，就贯彻这些原则。

顺便说一句，一年前虽然中断了课程，但还是在宣言上署了名字。



无意中发现，大魔头甚至在这个宣言刚提出没多久就署名了。



非官方解释

第一次看到FRP的官方解释，我是边读边点头，读完一个劲摇头。点头，是因为觉得说的好有道理；摇头，是因为觉得好像什么都没说。经过一年的冥想编程^1，我找到了一种对于工科学生（也许）比较容易理解的方式，来解释什么是FRP。文章开头提到，FRP其实就是时间序列信号处理在软件开发中的另一种形式。为什么呢？这要从数字电路说起。数字电路是大学工科专业的技术基础课，从最简单的与非门开始，设计3-7译码器，半加器，全加器等等，这些叫做组合逻辑电路。

> *组合逻辑电路：任一时刻的稳态输出，仅仅与该时刻的输入变量的取值有关，而与该时刻以前的输入变量取值无关。*

再看另一段摘自维基百科对于函数式编程的介绍：

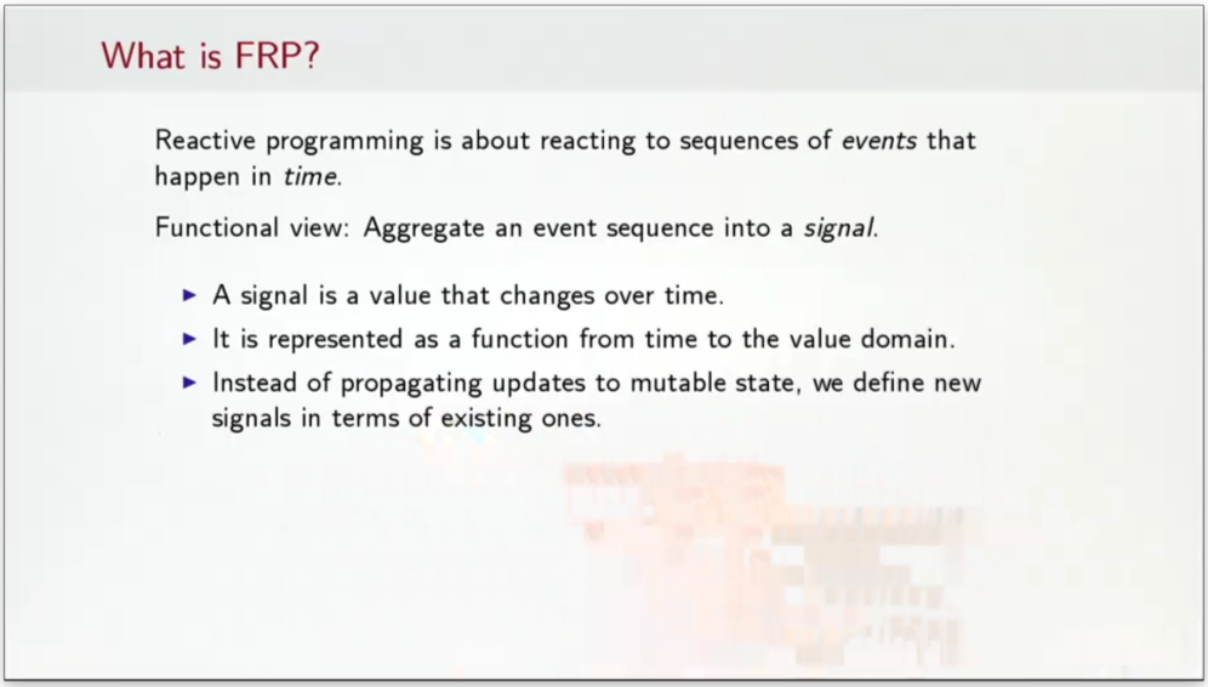
> *In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time.*

两者看起来是不是有点像？组合逻辑电路的最小组成单位是逻辑门，通过各种逻辑门的组合，可以实现各种各样的复杂功能；而函数式编程将函数作为最小单位，通过函数组合来实现复杂业务。数字电路的组合，通过将

器件之间的管脚相连，让一个器件的输出作为另一个器件的输入；函数式编程的组合，则是通过将一个函数的输出，作为另一个函数的输入来实现。这两者何止是相似，简直就是相似！

顺便说一句，函数式编程中作为最小单位的函数，如果要像逻辑门一样可以自由组合，就不能只是一般的函数，而是Monad。这也是为什么*Principles of Reactive Programming*这门课一开始就要将Monad的原因。至于什么是Monad，我们留在下一篇文章细讲。

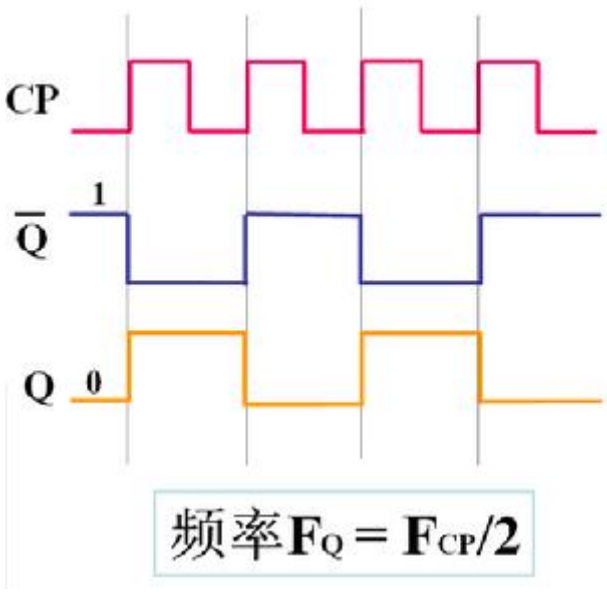
既然函数式编程（FP）可以映射到组合逻辑电路，剩下就是处理FRP中的R了。看看Scala的作者，Martin Odersky在*Principles of Reactive Programming*课程中定义的FRP：



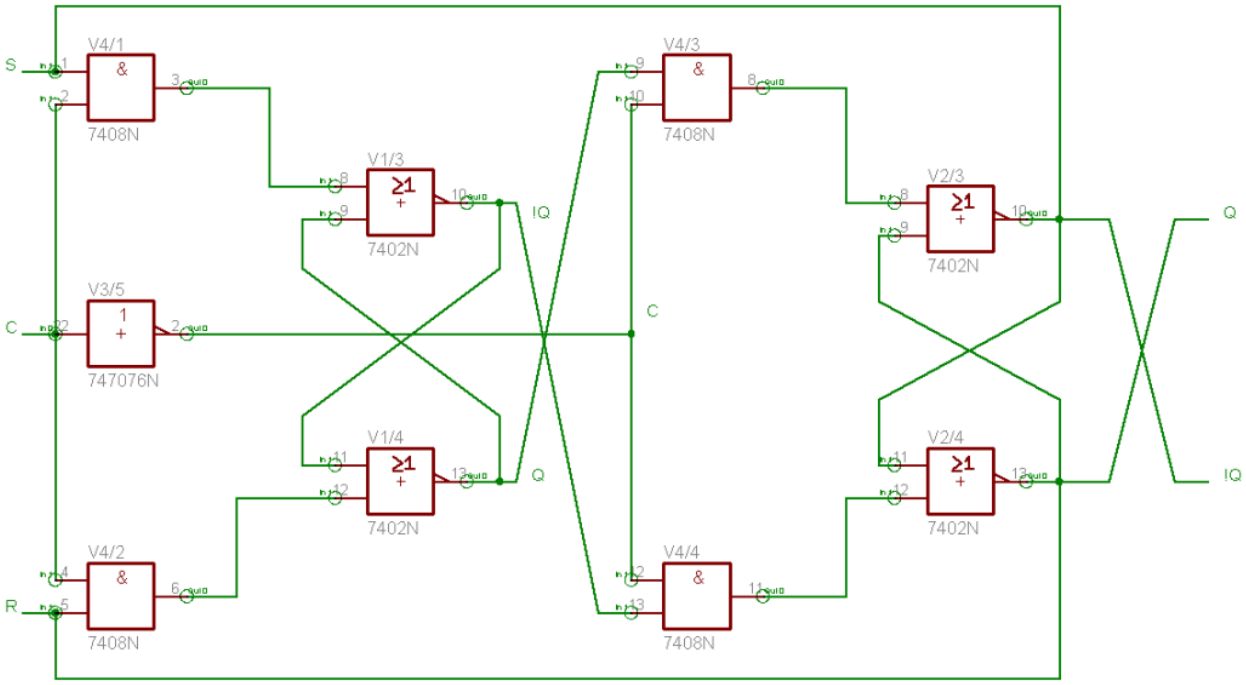
将一系列的时间看做一个**随时间变化的信号**，可以用一个**从时间域到值域的映射函数**来表示；新的信号是通过**对原始信号做变换**得到，而不是直接修改信号的值。哈，作为在哈工大5系待了6年，可不是白学的。这不就是数字信号处理在做的事情么！而且还是连基带调制都没做的信号处理。事实上，对没有调制过的信号处理，还没上升到数字信号处理这个级别——毕竟这是大三才学的课。用大二学的数字电路，足以应付。数字电路的下半部分，都在讲时序逻辑电路。

> *时序逻辑电路：电路任何时刻的稳态输出不仅取决于当前的输入，还与前一时刻输入形成的状态有关。*

举个最简单的2-分频器的例子，即输入一个频率为*f*的信号，输出为*f*/2的信号。先看时序图：



基于主从触发器实现的原理图[^2]：



[^2]: “Frequencydivider”作者Pascal Rehfeldt – 自己的作品。来自维基共享资源 – https://commons.wikimedia.org/wiki/File:Frequencydivider.png#/media/File:Frequencydivider.png根据CC BY-SA 3.0授权

简单来说，时序电路比数字电路多了状态；FRP比FP，嗯，也是多了状态。在时序电路之后，数字电路讲到了有限状态机。

> *有限状态机（英语：finite-state machine，缩写：FSM）又称有限状态自动机，简称状态机，是表示有限个状态以及在这些状态之间的转移和动作等行为的数学模型。*

如果把状态机状态转移的驱动信号当做消息，这不就是Akka的Actor模型么？所以，如果再有人问FRP是什么，我会回答他：

> *FRP就是时序信号处理在软件领域的另一个名字。*

脑洞大开

既然已经把信号处理和FRP联系在了一起，我们不妨将脑洞开得再大点。

脑洞1：数据谱分析，数据滤波器

上面说了，目前的FRP不过是发展到了信号处理领域的初级阶段，也就是大二课程那一部分。信号处理领域的高级技术在未来也可能会引入到软件开发中，比如频域变换，数字滤波，信号谱分析，小波分析等等。射频技术暂时还用不到，但至少基带处理的技术拿过来还是很有价值的。其实，这个趋势在近期已经有些体现，大数据处理中的很多算法，都来自信号领域的经典算法。正如范伟在小品中说的：别跟我整大二的，要整就整大三的！

脑洞2：数字电路与微服务

再看看到上面2-分频器的原理图，乍一看是不是有点像微服务的调用关系图？在数字电路设计中，一个元器件可以简单到只是一个非门，也可以复杂到一个数字FIR滤波器，这两者可以非常和谐的一起工作，其原因就在于一旦一个完整的逻辑电路设计完成，定义好其输入与输出端口，就可以将其封装成一个黑盒子（用过Matlab Simulink的应该都干过这个），再和其他元件组合使用时，和一个简单的非门没什么区别。如果两个元件不匹配，比如信号频率或者电压不匹配，只需要加上一个分频器，或者整流器就可以了。这种思想会不会在未来的微服务架构设计中有所体现呢？

脑洞3：再谈OO与FP

一谈到FP，就一定有人会问，OO（面向对象技术，Object Oriented）和FP的区别在哪？FP火了，是不是OO就要死了。如果把FP看作是信号处理

这个层面的技术，OO就是设计制造手机和电脑。信号处理技术虽然在实验室里解决了通信问题，但要想让相隔千里的普通人通过互相沟通，还是需要手机和电脑制造商。而且，即使用的是相同的信号处理技术，甚至相同的处理芯片，不同的手机，用起来还是不一样。那么，你说OO会不会死？

后续文章预告

本文是未来一系列关于FRP文章的开篇。未来计划如下：

1. Monad与Event Handling；
2. Obeservable与RxJava/RxScala；
3. Actors与Akka。

希望能通过这一系列文章，将FRP理顺，讲清楚。

发表评论

标签： FRP 响应式 响应式编程