响应式编程与响应式系统

在恒久的迷惑与过多期待的海洋中，登上一组简单响应式设计原则的小岛。

-- Jonas Bonér, Viktor Klang

本文导航

在恒久的迷惑与过多期待的海洋中，登上一组简单响应式设计原则的小岛。

下载 Konrad Malawski 的免费电子书《为什么选择响应式？企业应用中的基本原则》[1]，深入了解更多响应式技术的知识与好处。

自从 2013 年一起合作写了《响应式宣言》[2] 之后，我们看着响应式从一种几乎无人知晓的软件构建技术——当时只有少数几个公司的边缘项目使用了这一技术——最后成为中间件领域（middleware field）大佬们全平台战略中的一部分。本文旨在定义和澄清响应式各个方面的概念，方法是比较在响应式编程风格下和把响应式系统视作一个紧密整体的设计方法下编写代码的不同之处。

响应式是一组设计原则

响应式技术目前成功的标志之一是“响应式（reactive）”成为了一个热词，并且跟一些不同的事物与人联系在了一起——常常伴随着像“流（streaming）”、“轻量级（lightweight）”和“实时（real-time）”这样的词。

举个例子，当我们看到一支运动队时（像棒球队或者篮球队），我们一般会把他们看成一个个单独个体的组合，但是当他们之间碰撞不出火花，无法像一个团队一样高效地协作时，他们就会输给一个“更差劲”的队伍。从这篇文章的角度来看，响应式是一组设计原则，一种关于系统架构与设计的思考方式，一种关于在一个分布式环境下，当实现技术 implementation techniques、工具和设计模式都只是一个更大系统的一部分时如何设计的思考方式。

……拼凑在一起——尽管单独来看，这些软件都很优秀——和响应式系统之间的不同。在一个响应式系统中，正是不同组件 parts ……它使得不同组件能够独立地运作，同时又一致协作从而达到最终想要的结果。

……风格 style，它允许许多独立的应用结合在一起成为一个单元，共同响应它们所处的环境，同时保留着对单元内其它应用的“感……扩大/缩小规模 scale up/down，负载平衡，甚至能够主动地执行这些步骤。

……编程）写一个软件是可能的；然而，那也不过是拼图中的一块罢了。虽然在上面的提到的各个方面似乎都足以称其为“响……不足以让一个系统成为响应式的。

当人们在软件开发与设计的语境下谈论“响应式”时，他们的意思通常是以下三者之一：

- 响应式系统（架构与设计）
- 响应式编程（基于声明的事件的）
- 函数响应式编程（FRP）

我们将调查这些做法与技术的意思，特别是前两个。更明确地说，我们会在使用它们的时候讨论它们，例如它们是怎么联系在一起的，从它们身上又能到什么样的好处——特别是在为多核、云或移动架构搭建系统的情境下。

让我们先来说一说函数响应式编程吧，以及我们在本文后面不再讨论它的原因。

函数响应式编程（FRP）

函数响应式编程 Functional reactive programming，通常被称作 FRP，是最常被误解的。FRP 在二十年前就被 Conal Elliott 精确地定义过了[3]了。但是最近这个术语却被错误地[脚注1]用来描述一些像 Elm、Bacon.js 的技术以及其它技术中的响应式插件（RxJava、Rx.NET、 RxJS）。许多的库 libraries 声称他们支持 FRP，事实上他们说的并非响应式编程，因此我们不会再进一步讨论它们。

响应式编程

响应式编程 Reactive programming，不要把它跟函数响应式编程混淆了，它是异步编程下的一个子集，也是一种范式，在这种范式下，由新信息的有效性 availability 推动逻辑的前进，而不是让一条执行线程 a thread-of-execution 去推动控制流 control flow。

它能够把问题分解为多个独立的步骤，这些独立的步骤可以以异步且非阻塞 non-blocking 的方式被执行，最后再组合在一起产生一条工作流 workflow ——它的输入和输出可能是非绑定的 unbounded。

“异步地” Asynchronous [4] 被牛津词典定义为“不在同一时刻存在或发生”，在我们的语境下，它意味着一条消息或者一个事件可发生在任何时刻，也有可能是在未来。这在响应式编程中是非常重要的一项技术，因为响应式编程允许[非阻塞式 non-blocking]的执行方式——执行线程在竞争一块共享资源时不会因为阻塞 blocking 而陷入等待（为了防止执行线程在当前的工作完成之前执行任何其它操作），而是在共享资源被占用的期间转而去做其它工作。阿姆达尔定律 Amdahl's Law [脚注2] 告诉我们，竞争是可伸缩性 scalability 最大的敌人，所以一个响应式系统应当在极少数的情况下才不得不做阻塞工作。

响应式编程一般是事件驱动 event-driven，相比之下，响应式系统则是消息驱动 message-driven 的——事件驱动与消息驱动之间的差别会在文章后面阐明。

响应式编程库的应用程序接口（API）一般是以下二者之一：

- 基于回调的 Callback-based —匿名的间接作用 side-effecting 回调函数被绑定在事件源 event sources 上，当事件被放入数据流 dataflow chain

中时，回调函数被调用。

- 声明式的（*Declarative*）——通过函数的组合，通常是使用一些固定的函数，像 map、 filter、 fold等等。
- [illegible]带有基于流的操作符（*stream-based operators*），像 windowing、 counts、 triggers。
- [illegible]g [5] 有关是很合理的，因为它强调的是数据流而不是控制流。
- [illegible]呈抽象概念：
  - [illegible]具有读共享/写独占（*many-read/single-write*）的语义，即使变量尚不可用也能够添加异步的值转换操作。
  - 支持异步，非阻塞式，支持多个源与目的的反压转换管道（*back-pressured transformation pipelines*）。
- 数据流变量[8]——依赖于输入、过程（*procedures*）或者其它单元的单赋值变量（*single assignment variables*）（存储单元），它能够自动更新值的改变。其中一个应用例子是表格软件——一个单元的值的改变会像涟漪一样荡开，影响到所有依赖于它的函数，顺流而下地使它们产生新的值。

在 JVM 中，支持响应式编程的流行库有 Akka Streams、Ratpack、Reactor、RxJava 和 Vert.x 等等。这些库实现了响应式编程的规范，成为 JVM 上响应式编程库之间的互通标准（*standard for interoperability*），并且根据它自身的叙述是"……一个为如何处理非阻塞式反压异步流提供标准的倡议"。

响应式编程的基本好处是：提高多核和多 CPU 硬件的计算资源利用率；根据阿姆达尔定律以及引申的 Günther 的通用可伸缩性定律（*Günther's Universal Scalability Law*）脚注3，通过减少序列化点（*serialization points*）来提高性能。

另一个好处是开发者生产效率，传统的编程范式都尽力想提供一个简单直接的可持续的方法来处理异步非阻塞式计算和 I/O。在响应式编程中，因活动(active)组件之间通常不需要明确的协作，从而也就解决了其中大部分的挑战。

响应式编程真正的发光点在于组件的创建跟工作流的组合。为了在异步执行上取得最大的优势，把反压（*back-pressure*）[9] 加进来是很重要，这样能避免过度使用，或者确切地说，避免无限度的消耗资源。

尽管如此，响应式编程在搭建现代软件上仍然非常有用，为了在更高层次上理解（*reason about*）一个系统，那么必须要使用到另一个工具：响应式架构（*reactive architecture*）——设计响应式系统的方法。此外，要记住编程范式有很多，而响应式编程仅仅只是其中一个，所以如同其它工具一样，响应式编程并不是万金油，它不意图适用于任何情况。

事件驱动 vs. 消息驱动

如上面提到的，响应式编程——专注于短时间的数据流链条上的计算——因此倾向于事件驱动，而响应式系统——关注于通过分布式系统的通信和协作所得到的弹性和韧性——则是消息驱动的 [10] 脚注4（或者称之为消息式（*messaging*）的）。

一个拥有长期存活的可寻址（*long-lived addressable*）组件的消息驱动系统跟一个事件驱动的数据流驱动模型的不同在于，消息具有固定的导向，而事件则没有。消息会有明确的（一个）去向，而事件则只是一段等着被观察（*observe*）的信息。另外，消息式（*messaging*）更适用于异步，因为消息的发送与接收和发送者和接收者是分离的。

响应式宣言中的术语表定义了两者之间概念上的不同 [11]：

> *一条消息就是一则被送往一个明确目的地的数据。一个事件则是达到某个给定状态的组件发出的一个信号。在一个消息驱动系统中，可寻址到的接收者等待消息的到来然后响应它，否则保持休眠状态。在一个事件驱动系统中，通知的监听者被绑定到消息源上，这样当消息被发出时它就会被调用。这意味着一个事件驱动系统专注于可寻址的事件源而消息驱动系统专注于可寻址的接收者。*

分布式系统需要通过消息在网络上传输进行交流，以实现其沟通基础，与之相反，事件的发出则是本地的。在底层通过发送包裹着事件的消息来搭建跨网络的事件驱动系统的做法很常见。这样能够维持在分布式环境下事件驱动编程模型的相对简易性，并且在某些特殊的和合理的范围内的使用案例上工作得很好。

然而，这是有利有弊的：在编程模型的抽象性和简易性上得一分，在控制上就减一分。消息强迫我们去拥抱分布式系统的真实性和一致性——像局部错误（*partial failures*），错误侦测（*failure detection*），丢弃/复制/重排序（*dropped/duplicated/reordered*）消息，最后还有一致性，管理多个并发真实性等等——然后直面它们，去处理它们，而不是像过去无数次一样，藏在一个蹩脚的抽象面罩后——假装网络并不存在(例如EJB、 RPC [12]、 CORBA [13] 和 XA [14])。

这些在语义学和适用性上的不同在应用设计中有着深刻的含义，包括分布式系统的复杂性(complexity)中的弹性(resilience)、韧性(elasticity)、移动性(mobility)、位置透明性(location transparency)和管理(management)，这些在文章后面再进行介绍。

应式编程技术的，这样的系统中就即有事件也有消息——一个是用于沟通的强大工具（消息），而另一个则呈现现实（事

中定义的那样——是一组用于搭建现代系统——已充分准备好满足如今应用程序所面对的不断增长的需求的现代系统

，它可以被追溯到 70 和 80 年代 Jim Gray 和 Pat Helland 在串级系统(Tandem System)[15] 上和 Joe aomstrong 和 Robert Virding 在 Erlang 上做出的重大工作。然而，这些人在当时都超越了时代，只有到了最近 5 - 10 年，技术行业才被不得不反思当前企业系统最好的开发实践活动并且学习如何将来之不易的响应式原则应用到今天这个多核、云计算和物联网的世界中。

响应式系统的基石是消息传递(message-passing)，消息传递为两个组件之间创建一条暂时的边界，使得它们能够在时间上分离——实现并发性——和空间(space)——实现分布式与移动性(distribution mobility)。这种分离是两个组件完全隔离(isolation)[17] 以及实现弹性(resilience)和韧性(elasticity)基础的必需条件。

从程序到系统

这个世界的连通性正在变得越来越高。我们不再构建程序——为单个操作子来计算某些东西的端到端逻辑——而更多地在构建系统了。

系统从定义上来说是复杂的——每一部分都包含多个组件，每个组件的自身或其子组件也可以是一个系统——这意味着软件要正常工作已经越来越依赖于其它软件。

我们今天构建的系统会在多个计算机上操作，小型的或大型的，或少或多，相近的或远隔半个地球的。同时，由于人们的生活正变得越来越依赖于系统顺畅运行的有效性，用户的期望也变得越得越来越难以满足。

为了实现用户——和企业——能够依赖的系统，这些系统必须是灵敏的(responsive)，这样无论是某个东西提供了一个正确的响应，还是当需要一个响应时响应无法使用，都不会有影响。为了达到这一点，我们必须保证在错误(弹性)和欠载(韧性)下，系统仍然能够保持灵敏性。为了实现这一点，我们把系统设计为消息驱动的，我们称其为响应式系统。

响应式系统的弹性

弹性是与错误下的灵敏性(responsiveness)有关的，它是系统内在的功能特性，是需要被设计的东西，而不是能够被动的加入系统中的东西。弹性是大于容错性的——弹性无关于故障退化(graceful degradation)——虽然故障退化对于系统来说是很有用的一种特性——与弹性相关的是与从错误中完全恢复达到自愈的能力。这就需要组件的隔离以及组件对错误的包容，以免错误散播到其相邻组件中去——否则，通常会导致灾难性的连锁故障。

因此构建一个弹性的、自愈(self-healing)系统的关键是允许错误被：容纳、具体化为消息，发送给其他（担当监管者(supervisors)）的组件，从而在错误组件之外修复出一个安全环境。在这，消息驱动是其促成因素：远离高度耦合的、脆弱的深层嵌套的同步调用链，大家长期要么学会忍受其煎熬或直接忽略。解决的想法是将调用链中的错误管理分离，将客户端从处理服务端错误的责任中解放出来。

响应式系统的韧性

韧性(Elasticity)[18] 是关于欠载下的灵敏性(responsiveness)的——意味着一个系统的吞吐量在资源增加或减少时能够自动地相应增加或减少(scales up or down)（同样能够向内或外扩展(scales in or out)）以满足不同的需求。这是利用云计算承诺的特性所必需的因素：使系统利用资源更加有效，成本效益更佳，对环境友好以及实现按次付费。

系统必须能够在不重写甚至不重新设置的情况下，适应性地——即无需介入自动伸缩——响应状态及行为，沟通负载均衡，故障转移(failover)，以及升级。实现这些的就是位置透明性(location transparency)：使用同一个方法，同样的编程抽象，同样的语义，在所有向度中伸缩(scaling)系统的能力——从 CPU 核心到数据中心。

如同《响应式宣言》所述：

一个极大地简化问题的关键洞见在于意识到我们都在使用分布式计算。无论我们的操作系统是运行在一个单一结点上(拥有多个独立的 CPU,并通过 QPI 链接进行交流),还是在一个节点集群(cluster of nodes)(独立的机器,通过网络进行交流)上。拥抱这个事实意味着在垂直方向上多核的伸缩与在水平方面上集群的伸缩在…解耦 […],是通过异步消息传送以及运行时实例与其引用解耦从而实现的,这就是我们所说的位置透明性。

…样的方式与它交流。唯一能够在语义上等同实现的方式是消息传送。

…么其中一个最重要的点即是保证一个系统架构在开发和维护组件时,最小程度地减低生产效率,同时将操作…

…很重要,因为在一个系统的生命周期中——如果系统的设计不正确——系统的维护会变得越来越困难,理解、定位和解决问题所需要花费时间和精力会不断地上涨。

响应式系统是我们所知的最具 生产效率 的系统架构(在多核、云及移动架构的背景下):

- 错误的隔离为组件与组件之间裹上舱壁 [19] (LCTT 译注:当船遭到损坏进水时,舱壁能够防止水从损坏的船舱流入其他船舱),防止引发连锁错误,从而限制住错误的波及范围以及严重性。
- 监管者的层级制度提供了多个等级的防护,搭配以自我修复能力,避免了许多曾经在侦查(inverstigate)时引发的操作 代价(cost) ——大量的瞬时故障(transient failures)。
- 消息传送和位置透明性允许组件被卸载下线、代替或 重新布线(rerouted) 同时不影响终端用户的使用体验,并降低中断的代价、它们的相对紧迫性以及诊断和修正所需的资源。
- 复制减少了数据丢失的风险,减轻了数据 检索(retrieval) 和存储的有效性错误的影响。
- 弹性允许在使用率波动时保存资源,允许在负载很低时,最小化操作开销,并且允许在负载增加时,最小化 运行中断(outgae) 或 紧急投入(urgent investment) 伸缩性的风险。

因此,响应式系统使 生成系统(creation systems) 很好的应对错误、随时间变化的负载——同时还能保持低运营成本。

响应式编程与响应式系统的关联

响应式编程是一种管理 内部逻辑(internal logic) 和 数据流转换(dataflow transformation) 的好技术,在本地的组件中,做为一种优化代码清晰度、性能以及资源利用率的方法。响应式系统,是一组架构上的原则,旨在强调分布式信息交流并为我们提供一种处理分布式系统弹性与韧性的工具。

只使用响应式编程常遇到的一个问题,是一个事件驱动的基于回调的或声明式的程序中两个计算阶段的 高度耦合(tight coupling) ,使得 弹性 难以实现,因此时它的转换链通常存活时间短,并且它的各个阶段——回调函数或 组合子(combinator) ——是匿名的,也就是不可寻址的。

这意味着,它通常在内部处理成功与错误的状态而不会向外界发送相应的信号。这种寻址能力的缺失导致单个 阶段(stages) 很难恢复,因为它通常并不清楚异常应该,甚至不清楚异常可以,发送到何处去。

另一个与响应式系统方法的不同之处在于单纯的响应式编程允许 时间 上的 解耦(decoupling) ,但不允许 空间 上的(除非是如上面所述的,在底层通过网络传送消息来 分发(distribute) 数据流)。正如叙述的,在时间上的解耦使 并发性 成为可能,但是是空间上的解耦使 分布(distribution) 和 移动性(mobility) (使得不仅仅静态拓扑可用,还包括了动态拓扑)成为可能的——而这些正是 韧性 所必需的要素。

位置透明性的缺失使得很难以韧性方式对一个基于适应性响应式编程技术的程序进行向外扩展,因为这样就需要分附加工具,例如 消息总线(message bus) , 数据网格(data grid) 或者在顶层的 定制网络协议(bespoke network protocol) 。而这点正是响应式系统的消息驱动编程的闪光的地方,因为它是一个包含了其编程模型和所有伸缩向度语义的交流抽象概念,因此降低了复杂性与认知超载。

对于基于回调的编程,常会被提及的一个问题是写这样的程序或许相对来说会比较简单,但最终会引发一些真正的后果。

例如,对于基于匿名回调的系统,当你想理解它们,维护它们或最重要的是在 生产供应中断(production outages) 或错误行为发生时,你想知道到底发生了什么、发生在哪以及为什么发生,但此时它们只提供极少的内部信息。

为响应式系统设计的库与平台（例如 Akka [20] 项目和 Erlang [21] 平台）学到了这一点，它们依赖于那些更容易理解的长期存活的可寻址组件。当错误发生时，根据导致错误的消息可以找到唯一的组件。当可寻址的概念存在组件模型的核心中时，监控方案（monitoring solution）就有了一个 有意义的方式来呈现它收集的数据——利用传播的身份标识。

…现像可寻址能力和错误管理这些东西的范式，已经被证明在生产中是无价的，因它在设计中承认了现实并非一帆风顺，接…去尝试避免错误。

…的实现技术，可以用在响应式架构当中。但是记住这只能帮助管理一部分：异步且非阻塞执行下的数据流管理——通常只…，就需要开始认真地考虑像数据一致性（data consistency）、跨结点沟通（cross-node communication）、协调（coordination）、版本控制（versioning）、关注与责任（concerns and responsibilities）分离等等的东西——也即是：系统架构。

…把它作为构建响应式系统的工具来使用。构建一个响应式系统需要的不仅是在一个已存在的遗留下来的软件栈（software stack）上抽象特定的操作系统资源和少量的异步 API 和断路器（circuit breakers） [22]。此时应该拥抱你在创建一个包含多个服务的分布式系统这一事实——这意味着所有东西都要共同合作，提供一致性与灵敏的体验，而不仅仅是如预期工作，但同时还要在发生错误和不可预料的负载下正常工作。

总结

企业和中间件供应商在目睹了应用响应式所带来的企业利润增长后，同样开始拥抱响应式。在本文中，我们把响应式系统做为企业最终目标进行描述——假设了多核、云和移动架构的背景——而响应式编程则从中担任重要工具的角色。

响应式编程在内部逻辑及数据流转换的组件层次上为开发者提高了生产率——通过性能与资源的有效利用实现。而响应式系统在构建原生云（cloud native）和其它大型分布式系统的系统层次上为架构师及 DevOps 从业者提高了生产率——通过弹性与韧性。我们建议在响应式系统设计原则中结合响应式编程技术。

脚注

1. 参考 Conal Elliott，FRP 的发明者，见这个演示 [23]。
2. Amdahl 定律 [24] 揭示了系统理论上的加速会被一系列的子部件限制，这意味着系统在新的资源加入后会出现收益递减（diminishing returns）。
3. Neil Günter 的通用可伸缩性定律（Universal Scalability Law） [25] 是理解并发与分布式系统的竞争与协作的重要工具，它揭示了当新资源加入到系统中时，保持一致性的开销会导致不好的结果。
4. 消息可以是同步的（要求发送者和接受者同时存在），也可以是异步的（允许他们在时间上解耦）。其语义上的区别超出本文的讨论范围。

via: https://www.oreilly.com/ideas/reactive-programming-vs-reactive-systems

作者：Jonas Bonér [26]，Viktor Klang [27] 译者：XLCYun 校对：wxy

本文由 LCTT 组织编译，Linux中国 荣誉推出

LCTT 译者

XLCYun
共计翻译： **12** 篇
贡献时间：647 天

相关阅读

- [1]: 《为什么选择响应式？企业应用中的基本原则》 - http://www.oreilly.com/programming/free/why-reactive.csp?intcmp=il-webops-free-product-na_new_site_reactive_programming_vs_reactive_systems_text_cta
- [2]: 《响应式宣言》 - http://www.reactivemanifesto.org/
- [3]: 精确地定义过了 - http://conal.net/papers/icfp97/
- [4]: “异步地” - http://www.reactivemanifesto.org/glossary#Asynchronous
- [5]: 数据流编程 - https://en.wikipedia.org/wiki/Dataflow_programming
- [6]: Futures/Promises - https://en.wikipedia.org/wiki/Futures_and_promises

icle/p-frqbydrl-vc.html)
ww.voidcn.com/article/p-ghsjkcbs-bhy.html)
n/article/p-wzjdshsz-vr.html)
ttp://www.voidcn.com/article/p-bvtiydbu-bcq.html)
ttp://www.voidcn.com/article/p-yonbyvdv-baz.html)
n/article/p-zjjyybdf-zw.html)
n/article/p-qwdfrdoz-bdx.html)
n/article/p-sagpihfd-bmt.html)
- 9. 响应式编程总览 (http://www.voidcn.com/article/p-ephoyfmd-e.html)
- 10. ReactiveX--响应式编程 (http://www.voidcn.com/article/p-qdzaksnx-vb.html)
- 更多相关文章... (http://www.voidcn.com/relative/p-kupfukqd-bmt.html)

相关文章

- 1. 响应式编程 (http://www.voidcn.com/article/p-frqbydrl-vc.html)
- 2. 响应式编程 理解响应式编程 (http://www.voidcn.com/article/p-ghsjkcbs-bhy.html)
- 3. 响应式框架系统 (http://www.voidcn.com/article/p-wzjdshsz-vr.html)
- 4. 【译】反应式编程也称响应式编程 (http://www.voidcn.com/article/p-bvtiydbu-bcq.html)
- 5. Reactive programming - 响应式编程 (http://www.voidcn.com/article/p-yonbyvdv-baz.html)
- 6. iOS 响应式编程 (http://www.voidcn.com/article/p-zjjyybdf-zw.html)
- 7. 谈谈响应式编程 (http://www.voidcn.com/article/p-qwdfrdoz-bdx.html)
- 8. 理解响应式编程 (http://www.voidcn.com/article/p-sagpihfd-bmt.html)
- 9. 响应式编程总览 (http://www.voidcn.com/article/p-ephoyfmd-e.html)
- 10. ReactiveX--响应式编程 (http://www.voidcn.com/article/p-qdzaksnx-vb.html)

>>更多相关文章<< (http://www.voidcn.com/relative/p-kupfukqd-bmt.html)

相关标签/搜索

响应式编程 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E5%BC%8F%E7%BC%96%E7%A8%8B) 响应方式 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E6%96%B9%E5%BC%8F) rem响应式 (http://www.voidcn.com/tag/rem%E5%93%8D%E5%BA%94%E5%BC%8F) 响应式布局 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E5%BC%8F%E6%B5%81) 响应式 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E5%BC%8F) em响应式 (http://www.voidcn.com/tag/em%E5%93%8D%E5%BA%94%E5%BC%8F) 响应格式 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E6%A0%BC%E5%BC%8F) vue响应式 (http://www.voidcn.com/tag/vue%E5%93%8D%E5%BA%94%E5%BC%8F) html5 响应式 (http://www.voidcn.com/tag/html5+%E5%93%8D%E5%BA%94%E5%BC%8F) 响应式设计，技巧，响应式 (http://www.voidcn.com/tag/%E5%93%8D%E5%BA%94%E5%BC%8F%E8%AE%BE%E8%AE%A1%EF%BC%8C%E6%8A%80%E5%B7%A7%EF%BC%8C%E5%93%8D%E5%BA%94%E5 响应式编程 (http://www.voidcn.com/cata/2553793) 响应式 (http://www.voidcn.com/cata/2349369) 响应式 (http://www.voidcn.com/cata/2275967) 响应式 (http://www.voidcn.com/cata/2503817) 响应式 (http://www.voidcn.com/cata/3032835) 响应式 (http://www.voidcn.com/cata/1687057) 响应式编成 (http://www.voidcn.com/cata/5913209) 应急响应 (http://www.voidcn.com/cata/305338d4) 应急响应 (http://www.voidcn.com/cata/10710132d4) 响应式设计 (http://www.voidcn.com/cata/4412051) 响应式编程 (http://www.voidcn.com/search/ogmufj) ligerui响应式 (http://www.voidcn.com/search/bgryyr) 响应式自助建站系统 (http://www.voidcn.com/search/srpjtd) 响应式编程-消息模式 Actor 实现与 Scala、Akka 应用编程 (http://www.voidcn.com/search/ciannr) Capture响应方式和bubbing响应方式 (http://www.voidcn.com/search/bwroun) 响应式设计原理 (http://www.voidcn.com/search/xftpkh) 响应式布局 android (http://www.voidcn.com/search/rppqoa) bootstrap 响应式table例子 (http://www.voidcn.com/search/pbgxty) 响应式架构 akka pdf (http://www.voidcn.com/search/kduugp) 响应式架构 消息模式pdf (http://www.voidcn.com/search/flhnut)

0

分享到微博 分享到微信 分享到QQ

每日一句

每一个你不满意的现在，都有一个你没有努力的曾经。

作者信息

(http://www.voidcn.com/blog/xclwbg)

暂时无法查看

Linux中国 (http://www.voidcn.com/blog/xclwbg) 微信号：linux-cn

您的【关注和订阅】是作者不断前行的动力

近期文章

- 1. Neo4j 和图数据库起步 (http://www.voidcn.com/article/p-revaujnw-brn.html)
- 2. ESR：程序语言设计的要诣和真谛 (http://www.voidcn.com/article/p-ruumrofr-brn.html)
- 3. 你或许不知道的实用 GNOME Shell 快捷键 (http://www.voidcn.com/article/p-pfiddbyf-brn.html)
- 4. LinchPin：一个使用 Ansible 的简化的编配工具 (http://www.voidcn.com/article/p-trvemeek-brn.html)

- 5.【每日安全资讯】新安卓病毒Loapi爆发：绑架你的手机来挖矿 (http://www.voidcn.com/article/p-fyfqozjz-brn.html)

程序园

最新文章

本站(http://www.voidcn.com/)

- 1. 类型：如何使用未在第一个参数列表中显示的类型参数来改进Scala的类型推断？ (http://www.voidcn.com/article/p-hytlekke-bve.html)
- 2. jquery - 未捕获的SyntaxError：意外的令牌e (http://www.voidcn.com/article/p-naxupugn-bve.html)

欢迎关注本站公众号,获取更多程序园信息

p://www.voidcn.com/article/p-vsjcghzy-bve.html)
输出 (http://www.voidcn.com/article/p-skpbnyff-bve.html)
lin：解决方案文件夹中的项目不是“项目” (http://www.voidcn.com/article/p-fcjydaaa-bve.html)
http://www.voidcn.com/article/p-cuawtded-bve.html)
dcn.com/article/p-accldcba-bve.html)
e命令 (http://www.voidcn.com/article/p-mzplkbdk-bve.html)
导多？ (http://www.voidcn.com/article/p-eswtstkb-bve.html)
http://www.voidcn.com/article/p-xzngmbfk-bve.html)

近搜索 (http://www.voidcn.com/search) 最新文章 (http://www.voidcn.com/recent) 站长统计 (https://www.cnzz.com/stat/website.php?
dcc.com/) 程序問答 (http://hk.voidcc.com/)

相关文章

- 1. 响应式编程 (http://www.voidcn.com/article/p-frqbydrl-vc.html)
- 2. 响应式编程：理解响应式编程 (http://www.voidcn.com/article/p-ghsjkcbs-bhy.html)
- 3. 响应式栅格系统 (http://www.voidcn.com/article/p-wzjdshsz-vr.html)
- 4. 【流行】反应式编程也称响应式系统 (http://www.voidcn.com/article/p-bvtiydbu-bcq.html)
- 5. Reactive programming - 响应式编程 (http://www.voidcn.com/article/p-yonbyvdv-baz.html)
- 6. IOS-响应式编程 (http://www.voidcn.com/article/p-zjjyybdf-zw.html)
- 7. 谈谈响应式编程 (http://www.voidcn.com/article/p-qwdfrdoz-bdx.html)
- 8. 理解响应式编程 (http://www.voidcn.com/article/p-sagpihfd-bmt.html)
- 9. 响应式编程总览 (http://www.voidcn.com/article/p-ephoyfmd-e.html)
- 10. ReactiveX--响应式编程 (http://www.voidcn.com/article/p-qdzaksnx-vb.html)

>>更多相关文章<< (http://www.voidcn.com/relative/p-kupfukqd-bmt.html)