本文来讨论一下容器设计模式，来源于 Google 于 2016 年发布的论文 *Design patterns for conntainer-based distributed system*，作者是 *Brendan Burns* 和 *David Oppenheimer*，Kubernetes 项目创始人。从论文题目中我们可以看到几个关键词：

- container-base：应用都将是容器化的
- distributed system：分布式系统
- design pattern：设计模式

所以这篇论文的核心应该是探讨容器化的分布式系统的设计模式，我们下面来一起看一下。

1. 概述

论文观点，继上一波面向对象编程引领了软件开发革命（1980 年代和 1990 早期）之后，目前基于容器构建的微服务体系也在悄悄改变着分布式系统领域。这篇论文发表于 2016 年，目前来看，这个预言确实是没有错的。继 2018 年 Kubernetes 全面爆发之后，全民应用容器化的趋势已成定局。

论文总结了三大设计模式：

- 单容器模式（single-container management patterns）；
- 单节点，多容器模式（single-node，multi-container application patterns）；
- 多节点模式（multi-node application patterns）。

2. 单容器模式

容器类似 OOP 编程中的 Object，定义了一系列的 interface。不仅可以暴露应用相关的功能函数，还可以暴露一些方便应用管理的 hook 接口。

现状

目前容器暴露出来的接口非常有限：run，pause，stop。毫无疑问这些接口都是很有用的，但是我们其实可以暴露出来更丰富的接口给开发者和管理员来使用。另外鉴于基本每一种主流的系统都提供了通过 http server 来暴露必要的信息，这一块容器其实也可以加强。那么具体来说，针对容器，有哪些可以加强的呢?

upward 视角

向上视角，容器可以提供关于应用更加丰富的信息，包括监控指标（比如 QPS）、profiling 信息（比如线程相关信息、堆栈使用、锁竞争、网络统计信息等）、配置信息、log 信息等。通过这些信息，开发者可以做更多的事情，比如系统诊断和调优点。

举个具体的例子，容器管理系统，比如 Kubernetes、Aurora、Marathon 等都提供了通过 HTTP 协议暴露健康检测的功能。

downward 视角

向下视角，容器可以提供：

- lifecycle：生命周期管理，以及每个阶段的 hook 使用，比如 Kubernetes 提供的 postStart 和 preStart。
- priority：优先级管理，不同优先级应用的容器对应不同的优先级，我们应该优先保障高优先级的容器。
- replicate yourself：快速创建一组相同的应用容器以达到横向扩容的目的。

举个例子，考虑一下 Android Activity 模型，抽象出了一系列的 callback，比如 onCreate()、onStart()、onStop()，以及形式化的状态机来定义开发者如何去触发这些 callback。这种应用的生命周期管理，毫无疑问帮助了开发者降低心智负担。

3. 单节点、多容器模式

单节点、多容器模式考虑的是多个容器被调度到同一台机器的情况。在很多容器调度管理系统中，都支持这种共同调度的场景，比如 Kubernetes 中将多个容器组成一个 Pod，把 Pod 作为一个原子调度单位。下面讨论的前提是系统提供对类似 Kubernetes 系统中的 Pod 抽象的支持。

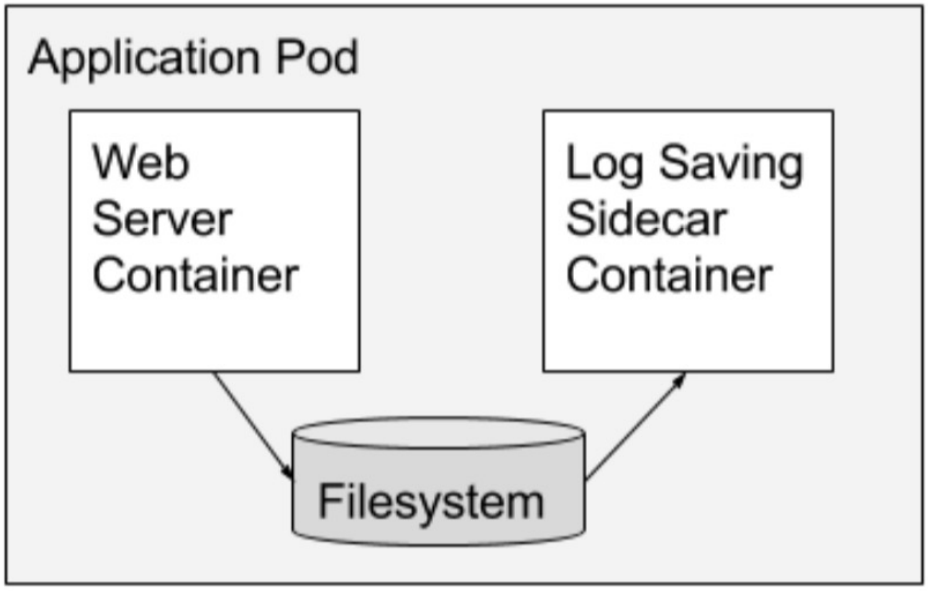
针对这种模式，论文提出了三种不同的方案，分别是：

- Sidecar 模式
- Ambassador 模式
- Adapter 模式

Sidecar

sidecar 是最常见的多容器调度的一种模式，简单来说这种模式下有一个主容器，然后其他容器都是针对主容器的扩展和增强，其他容器在这里的角色就类似 sidecar，也被称作 sidecar 容器。

举个例子，主容器是 web server 应用容器，sidecar 容器是一个日志收集容器用来将 web server 的日志收集并转存至特殊的文件系统中。



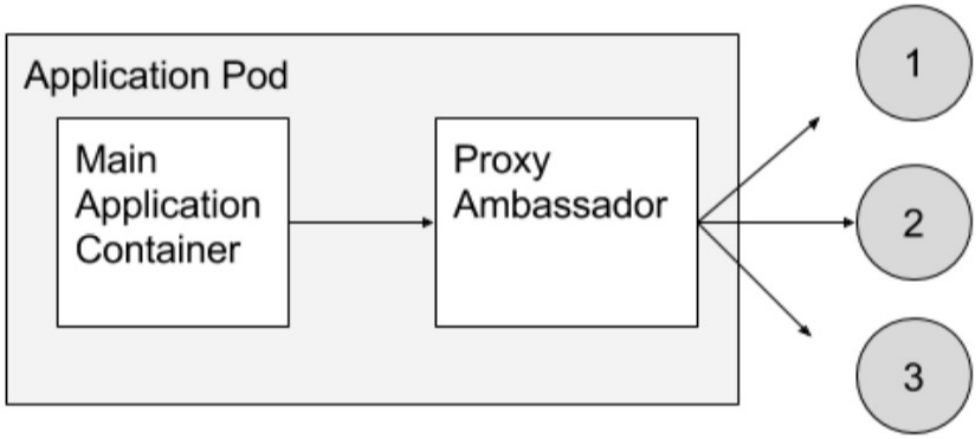
Ambassador

ambassador 这个单词中文一般翻译为大使，但是大使并不能准备表达其意思，我们看一下韦氏词典的准确翻译：

> a person who acts as a representative or promoter of a specified activity.

这样就很明了了，就是用来负责专门活动的人。ambassador 也是取自这个意思，ambassador 容器负责主容器通信的中转节点。

举个例子，有些 web 应用需要访问缓存，比如 memcached，这时 twemproxy 容器就可以作为一个 ambassador 容器和应用容器部署在同一个节点，web 应用像访问本地应用（localhost）一样访问 ambassador 容器，而 ambassador 容器在负责将请求路由到真正的 memcached 集群。



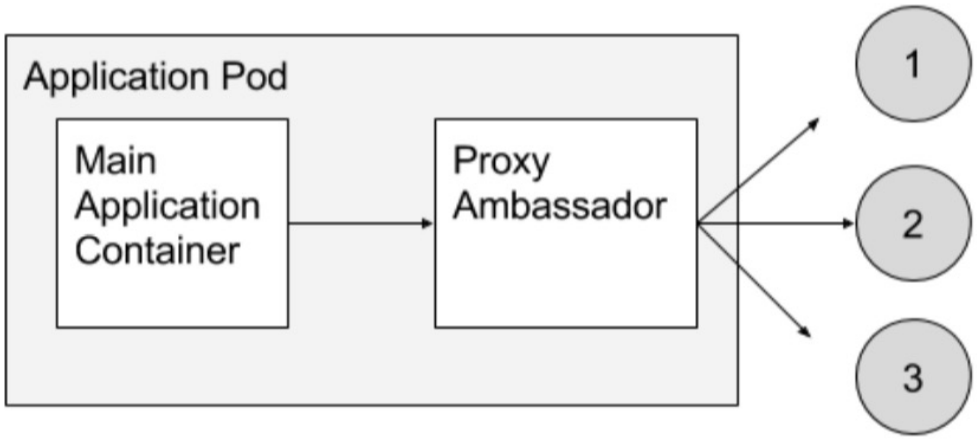
这种模式的好处在于精简了开发模式：

- 开发者只需要考虑自己的应用如何和本地的 memcached server 通过 localhost 访问
- 测试也非常简单，直接启动一个 standalone 的 memcached 实例即可
- 复用 twemproxy 的代码逻辑

Adapter

某些程度上，adapter 看上去和 ambassador 很像。区别在于 adapter 做的事情要更多，adapter 为应用容器提供了一个统一的视图。adapter 将不同容器的输出和交互都抽象成统一的结构。

一个简单的例子，通过 adapter 来实现同一个监控指标暴露。不同系统的监控指标会有不同的暴露方式，比如 jmx、statsd 等。对于由多种应用容器组成的微服务系统，如果每个应用透出监控指标的方式都不尽相同，那么外部的监控对接系统将会非常的麻烦。我们通过 adapter 将不用应用的指标做个标准化的封装然后再进行透出，这样会使得外部对接更加的方便。



4. 多节点模式

多节点模式是容器可能会部署到不同的节点上，当然这里讨论的前提也是系统提供对 Pod 抽象的支持。针对多节点部署模式，论文中讨论了三种不同的模式，分别是：

- Leader Election 模式
- Work Queue 模式
- Scatter/Gather 模式

Leader Election 模式

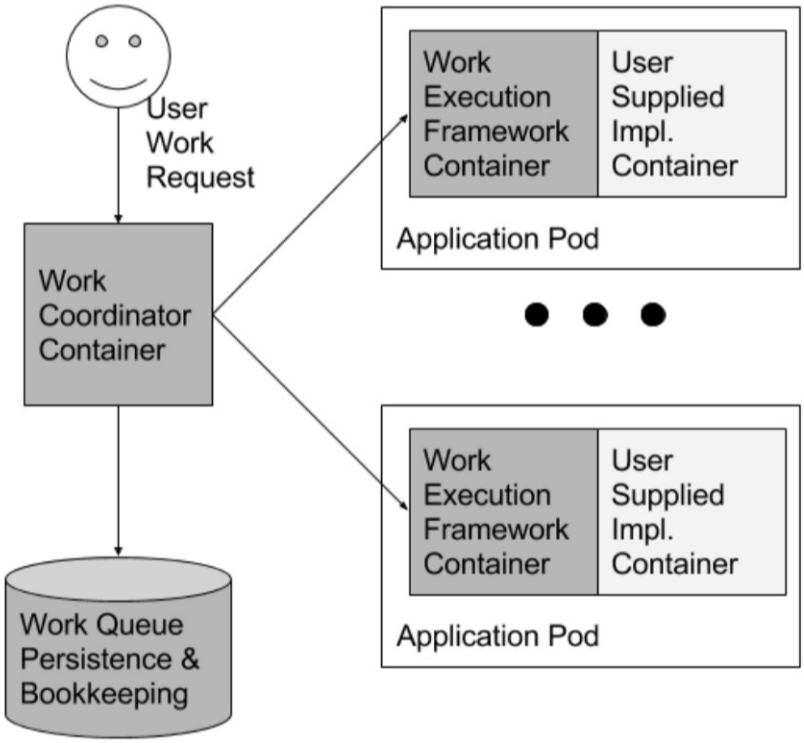
在分布式系统中一个非常常见的问题就是 Leader Election，中文一般叫做领导者选举。

尽管有很多 Leader Election 的算法库，但是大部分都是和特定的编程语言相关的。Leader Election 的一种容器解决方案是将选角算法封装到容器中，我们可以称之为 leader-election 容器，然后这些容器通过 HTTP 协议暴露特定的选举信息。当其他应用需要使用进行 Leader 选举时，直接和 leader-election 容器进行交互即可。leader-election 容器可以由有经验的专家开发，一旦开发完成，其他开发者要使用则可以直接通过网络进行交互，而不用考虑语言实现问题。这也是一种非常好的抽象和封装。

Work Queue 模式

Work Queue 就是工作队列，将 task 放入工作队列，然后进行统一处理。Work Queue 框架如同 Leader Election 一样，也是分布式系统讨论比较多的话题。类似 leader 选举，传统的工作队列框架同样和编程语言强相关。

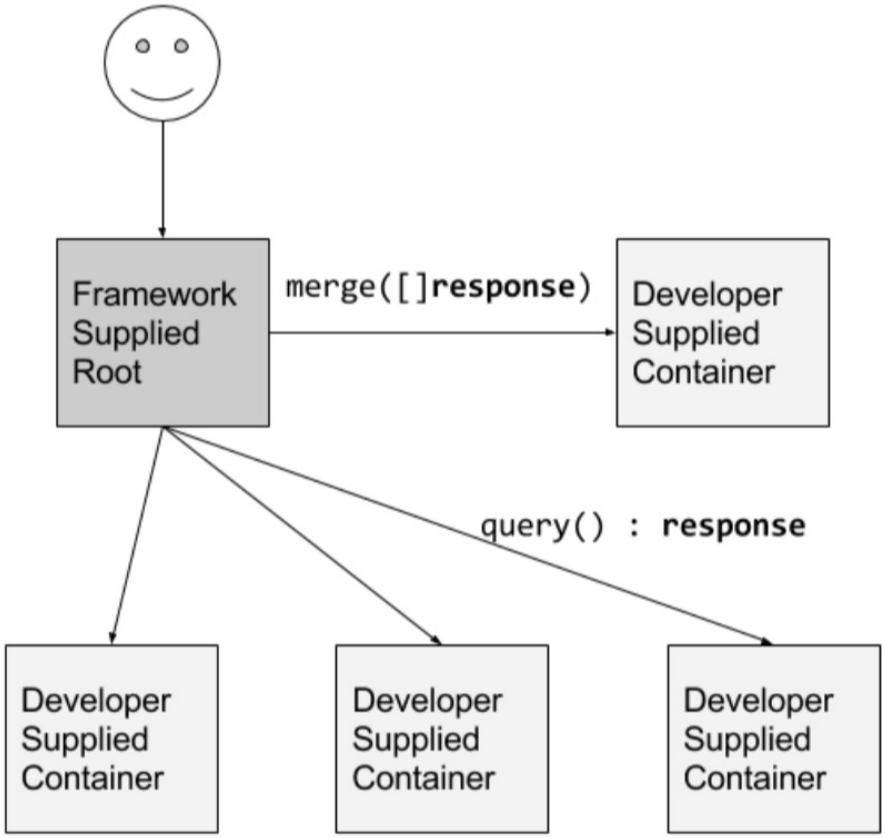
我们通过容器实现 Work Queue 模式同样可以作为一种好的抽象和封装。比如实现接口 run()、mount() 等，作为 Work Queue 的抽象。



Scatter/Gather 模式

scatter 是集中的意思，gather 是分散的意思，Scatter/Gather 模式也是一种形而上学的定义。具体来说，Scatter/Gather 模式讨论的场景是外部的 client 将请求发送给 root 或者 parent 节点，然后 root 将请求转发给后端多个 server 来做并行计算，最后将不同 server 的结果进行汇总。最常见的例子就是分布式查询引擎、搜索引擎等。

容器化的解决方案可以使用多个 leaf 容器和 1 个 merge 容器来实现通用的 scatter/gather 框架。



5. 总结

本文介绍了容器的多种设计模式，处处都能看到 Kubernetes 的影子，我们后面再介绍 Kubernetes。

}