极客时间 训练营

极客时间 Java 进阶训练营
第 19 课
分布式服务-深入分布式服务化

KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

极客时间 训练营

目录

1. 分布式服务治理

从 RPC 走向服务化->微服务架构

具体的分布式业务场景里，除了能够调用远程方法，我们还需要考虑什么？

1、多个相同服务如何管理？ ==> 集群/分组/版本 => 分布式与集群
2、服务的注册发现机制？ ==> 注册中心/注册/发现
3、如何负载均衡，路由等集群功能？ ==> 路由/负载均衡
4、熔断，限流等治理能力。 ==> 过滤/流控
5、心跳，重试等策略。
6、高可用、监控、性能等等。

还有没有其他要考虑的点？

RPC 与分布式服务化的区别

RPC：技术概念

》以 RPC 来讲，我们前面的自定义 RPC 功能已经差不多了。

》可以再考虑一下性能优化，使用 spring-boot 等封装易用性。

分布式服务化：服务是业务语义，偏向于业务与系统的集成

》以分布式服务化框架的角度来看，我们还差前面的这些非功能性需求能力。

》具体使用时，另外一个重点是如何设计分布式的业务服务。

注意：服务 != 接口，服务可以用接口或接口文档之类的语言描述。

JHWH == 耶和华 == 上帝

分布式服务化与 SOA/ESB 的区别

SOA/ESB：代理调用，直接增强

服务汇聚到 ESB：

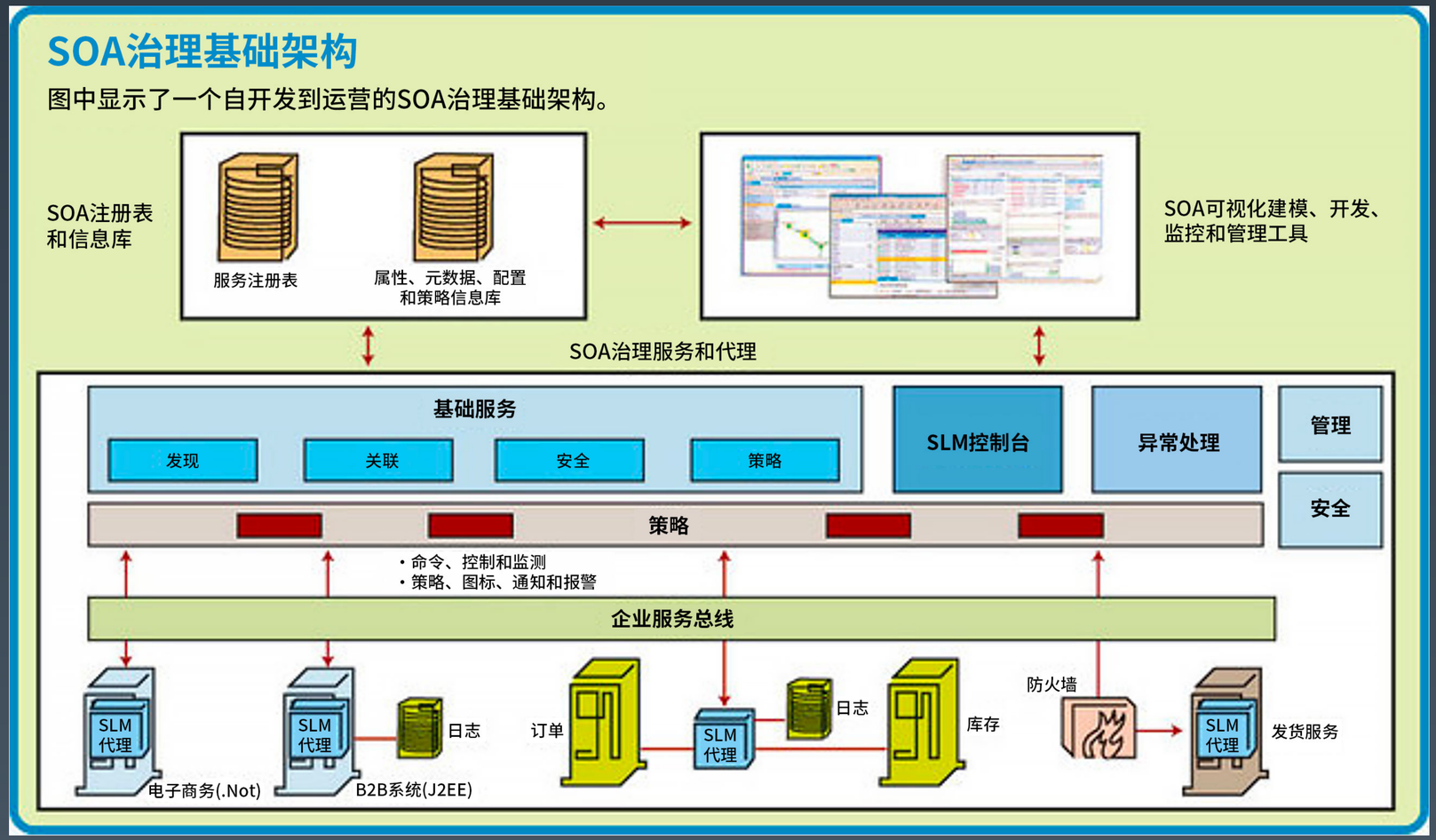
1、暴露和调用

2、增强和中介

3、统计和监控

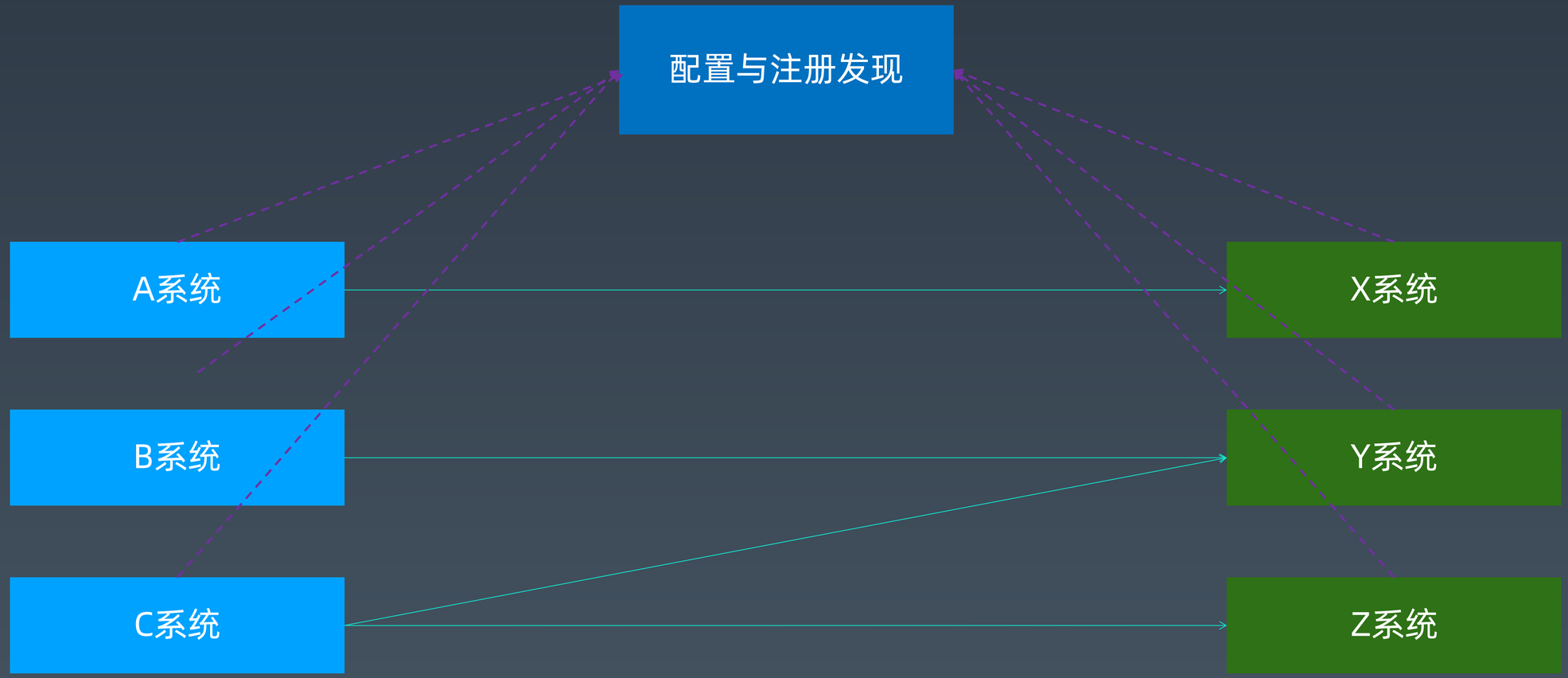
分布式服务化与 SOA/ESB 的区别

分布式服务化作为 SOA 的另一种选择，以不同方式把 ESB 的一些功能重做了一遍



分布式服务化与 SOA/ESB 的区别

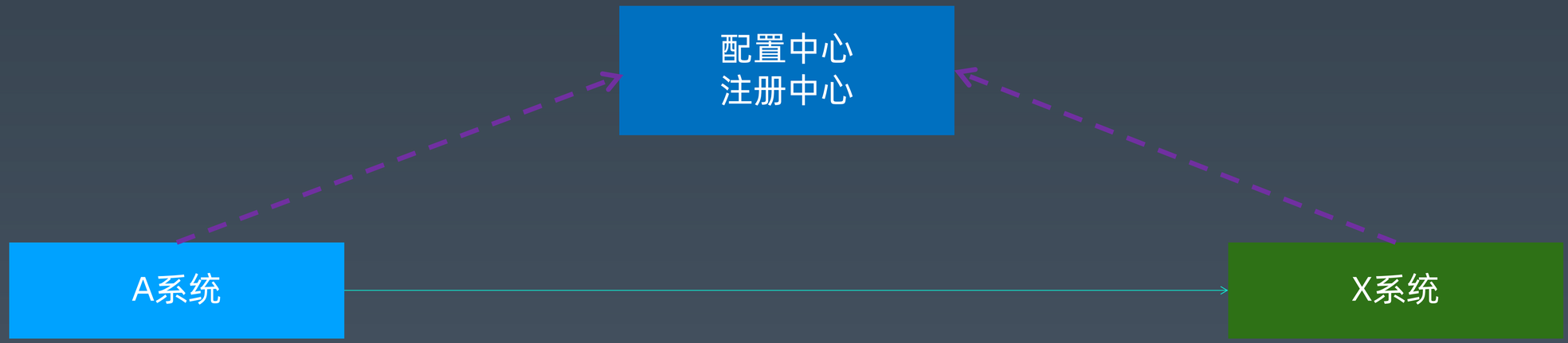
分布式服务化：直连调用，侧边增强



分布式服务化与 SOA/ESB 的区别

上面的配置/注册发现等就演化成了代替 ESB 容器的新组件：配置中心、注册中心等。

然后呢？原本的增强能力放到哪儿呢？

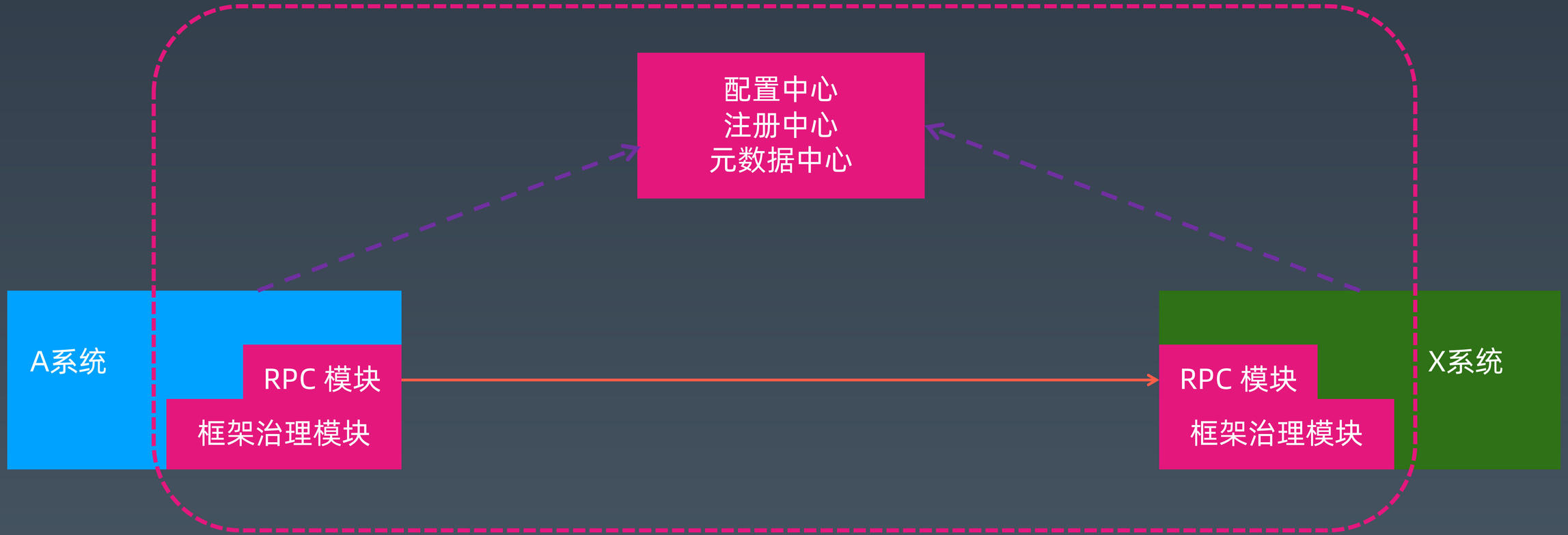


分布式服务化与 SOA/ESB 的区别

RPC 之上的增强能力根据特点：

1、有状态的部分，放到 xx 中心

2、无状态的部分，放到应用侧（具体来说是框架和配置部分，尽量不影响业务代码）



2. 配置/注册/元数据中心

配置、注册、元数据，有何异同？

配置中心（ConfigCenter）：管理系统需要的配置参数信息

注册中心（RegistryCenter）：管理系统的服务注册、提供发现和协调能力

元数据中心（MetadataCenter）：管理各个节点使用的元数据信息

相同点：都需要保存和读取数据/状态，变更通知

不同点：配置是全局非业务参数，注册中心是运行期临时状态，元数据是业务模型

为什么会需要配置中心？

想想看：

1、大规模集群下，如何管理配置信息，特别是批量更新问题。

2、大公司和金融行业，一般要求开发、测试、运维分离（物理隔离）。

3、运行期的一些开关控制，总不能不断重启？？

Zookeeper、etcd、Nacos、Apollo。。。

为什么会需要注册中心？

有什么办法，让消费者能动态知道生产者集群的状态变化？

1、hello.htm -> ok

2、DNS？VIP？

3、主动报告+心跳

这些信息很重要，后续的集群管控，分布式服务治理，都要靠这个全局状态。

为什么会需要元数据中心？

一般情况下，没有问题也不大。

有了更好。

元数据中心，定义了所有业务服务的模型。

如何实现 XX 中心？

最核心的两个要素：

1、需要有存取数据的能力，特别是临时数据的能力。

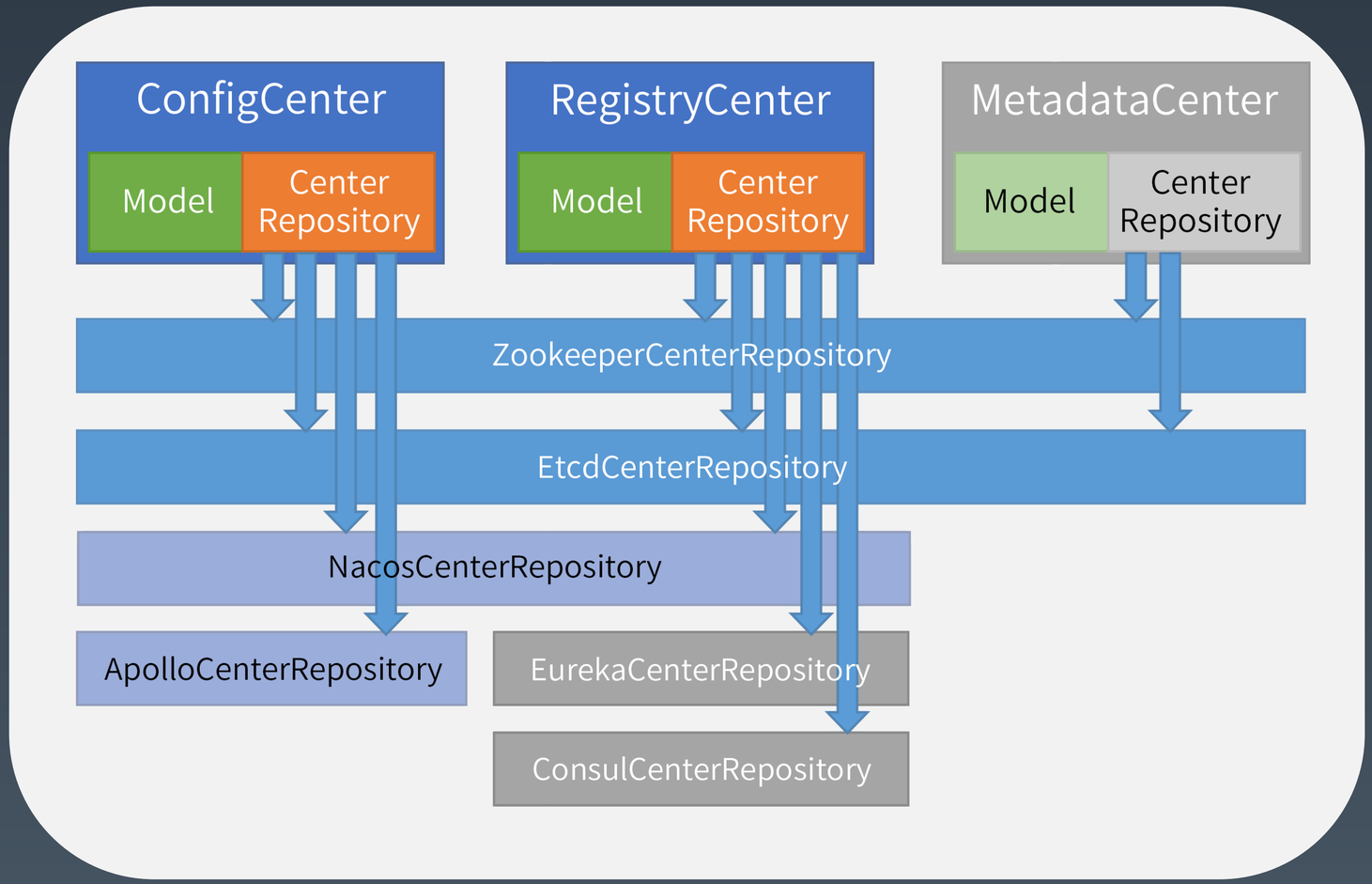
2、需要有数据变化的实时通知机制，全量或增量。

主流的基座，一般都可以使用 namespace 的概念，用来在顶层隔离不同环境。

zk 没有，但是我们一般用第一个根节点作为 namespace/group。

如何实现 XX 中心？

以某开源软件为例，设计的 XX 中心与基座



3. 服务的注册与发现

服务注册

服务提供者启动时，

- 将自己注册到注册中心（比如 zk 实现）的临时节点。

- 停止或者宕机时，临时节点消失。

服务注册

注册的数据格式

- 节点 key，代表当前服务（或者服务+版本）

- 多个子节点，每一个为一个提供者的描述信息

服务发现

服务消费者启动时，

- 从注册中心代表服务的主节点拿到多个代表提供者的临时节点列表，并本地缓存（why？？？）。

- 根据 router 和 loadbalance 算法从其中的某一个执行调用。

- 如果可用的提供者集合发生变化时，注册中心通知消费者刷新本地缓存的列表。

例如 zk 可以使用 curator 作为客户端操作。

4. 服务的集群与路由

服务集群

多个服务提供者都提供了同样的服务，这时应该如何处理？

>> 大家回忆一下，我们提到了多少种处理方式。

对于完全相同能力的多个服务，我们希望他们能一切协同工作，分摊处理流量。

- 路由

- 负载均衡

服务路由（Service Route）

跟网关的路由一样

1、比如基于 IP 段的过滤，

2、再比如服务都带上 tag，用 tag 匹配这次调用范围。

服务负载均衡（Service LoadBalance）

跟 Nginx 的负载均衡一样。

多个不同策略，原理不同，目的基本一致（尽量均匀）：

1、Random（带权重）== dubbo 默认的策略

2、RoundRobin（轮询）

3、LeastActive（快的多给）

4、ConsistentHashLoadBalance（同样参数请求到一个提供者）

5.服务的过滤与流控

服务过滤

所有的复杂处理，都可以抽象为管道+过滤器模式（Channel+Filter）

这个机制是一个超级 bug 的存在，

可以用来实现额外的增强处理（类似 AOP），也可以中断当前处理流程，返回特定数据。

对比考虑一下，我们 NIO 网关时的 filter，servlet 的 filter 等。

为什么需要服务流控（Flow Control）

稳定性工程：

1、我们逐渐意识到一个问题：系统会故障是正常现象，就像人会生病

2、那么在系统出现问题时，直接不服务，还是保持部分服务能力呢？

系统的容量有限。

保持部分服务能力是最佳选择，然后在问题解决后恢复正常状态。

响应式编程里，这就是所谓的回弹性（Resilient）。

需要流控的本质原因是，输入请求大于处理能力。

服务流控

流控有三个级别：

1、限流（内部线程数，外部调用数或数据量）

2、服务降级（去掉不必要的业务逻辑，只保留核心逻辑）

3、过载保护（系统短时间不提供新的业务处理服务，积压处理完后再恢复输入请求）

6.总结回顾与作业实践

第 19 课总结回顾

分布式服务治理

配置/注册/元数据中心

服务的注册与发现

服务的集群与路由

服务的过滤与流控

第 19 课作业实践

1、（选做）rpcfx1.1: 给自定义RPC实现简单的分组(group)和版本(version)。

2、（选做）rpcfx2.0: 给自定义RPC实现：

1）基于zookeeper的注册中心，消费者和生产者可以根据注册中心查找可用服务进行调用(直接选择列表里的最后一个)。

2）当有生产者启动或者下线时，通过zookeeper通知并更新各个消费者，使得各个消费者可以调用新生产者或者不调用下线生产者。

3、（挑战☆）在2.0的基础上继续增强rpcfx实现：

1）3.0: 实现基于zookeeper的配置中心，消费者和生产者可以根据配置中心配置参数（分组，版本，线程池大小等）。

2）3.1：实现基于zookeeper的元数据中心，将服务描述元数据保存到元数据中心。

3）3.2：实现基于etcd/nacos/apollo等基座的配置/注册/元数据中心。

第 19 课作业实践

4、（挑战☆☆）在3.2的基础上继续增强rpcfx实现：

1）4.0：实现基于tag的简单路由；

2）4.1：实现基于Random/RoundRobbin的负载均衡；

3）4.2：实现基于IP黑名单的简单流控；

4）4.3：完善RPC框架里的超时处理，增加重试参数；

5、（挑战☆☆☆）在4.3的基础上继续增强rpcfx实现：

1）5.0：实现利用HTTP头跨进程传递Context参数（隐式传参）；

2）5.1：实现消费端mock一个指定对象的功能（Mock功能）；

3）5.2：实现消费端可以通过一个泛化接口调用不同服务（泛化调用）；

4）5.3：实现基于Weight/ConsistentHash的负载均衡；

5）5.4：实现基于单位时间调用次数的流控，可以基于令牌桶等算法；

6、（挑战☆☆☆☆）6.0：压测，并分析调优5.4版本。