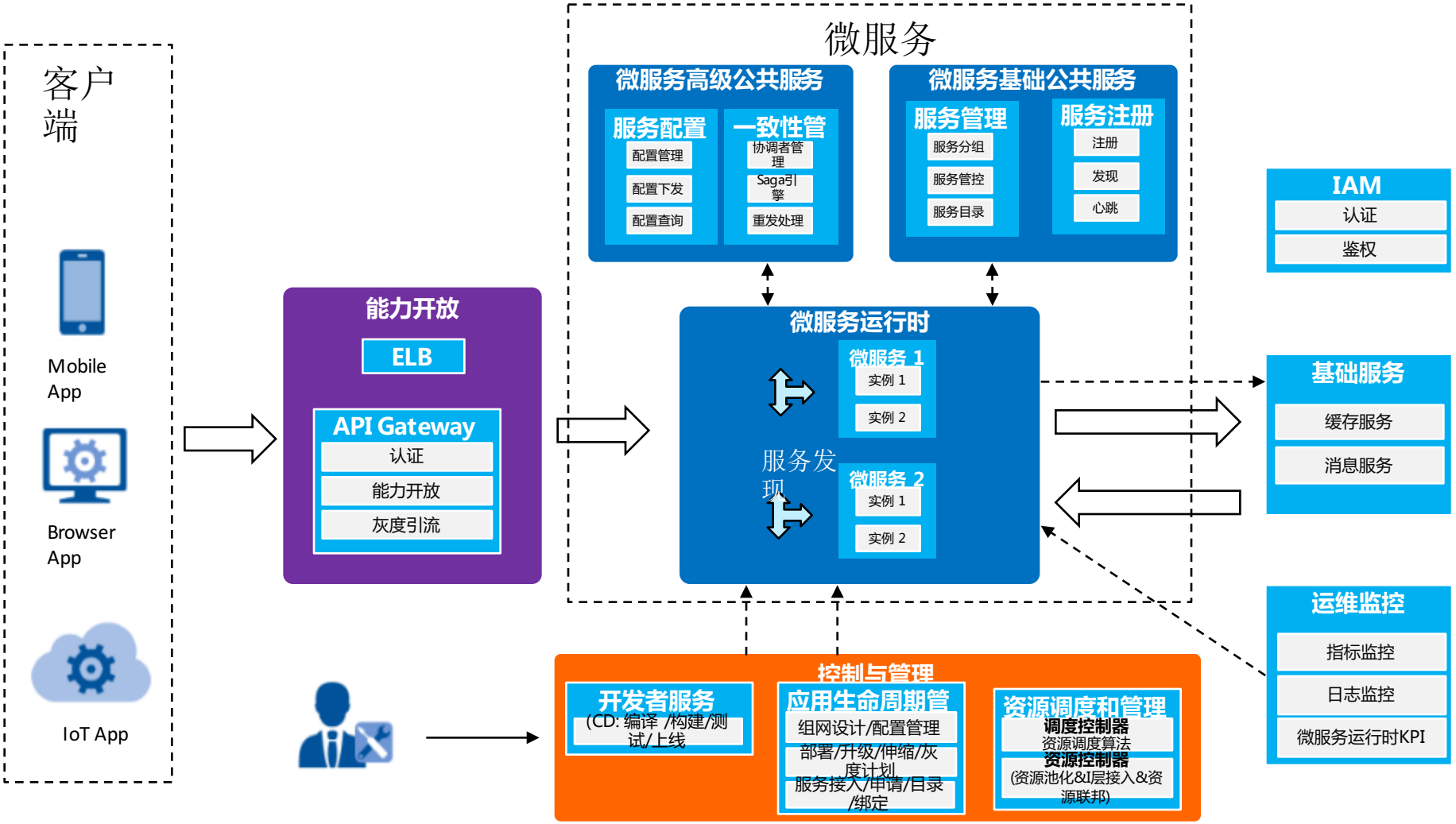
刘宝

华为高级软件工程师，当前负责微服务相关的设计和开发工作。

为什么开发微服务SDK？

微服务运行环境和基础设施



为什么开发微服务SDK？

Spring Cloud: http://projects.spring.io/spring-cloud/

RPC：支持Spring MVC/JAX-RS风格REST开发，提供多种J2EE运行容器支持，如tomcat。
微服务：服务注册与发现、负载均衡、配置管理
高级特性：分布式消息(stream)、全局锁、有状态服务(integration)

dubbo: http://dubbo.io/

RPC：支持RPC风格的开发，采用hession进行对象序列化，当前只提供JAVA语言实现。
微服务：服务注册与发现(zookeeper/redis)、负载均衡
高级特性：协议扩展

Tars : https://github.com/Tencent/Tars

RPC：支持RPC风格的开发，使用自定义的IDL语言定义接口。
微服务：内建在管理平台
高级特性：

gRPC : http://www.grpc.io/

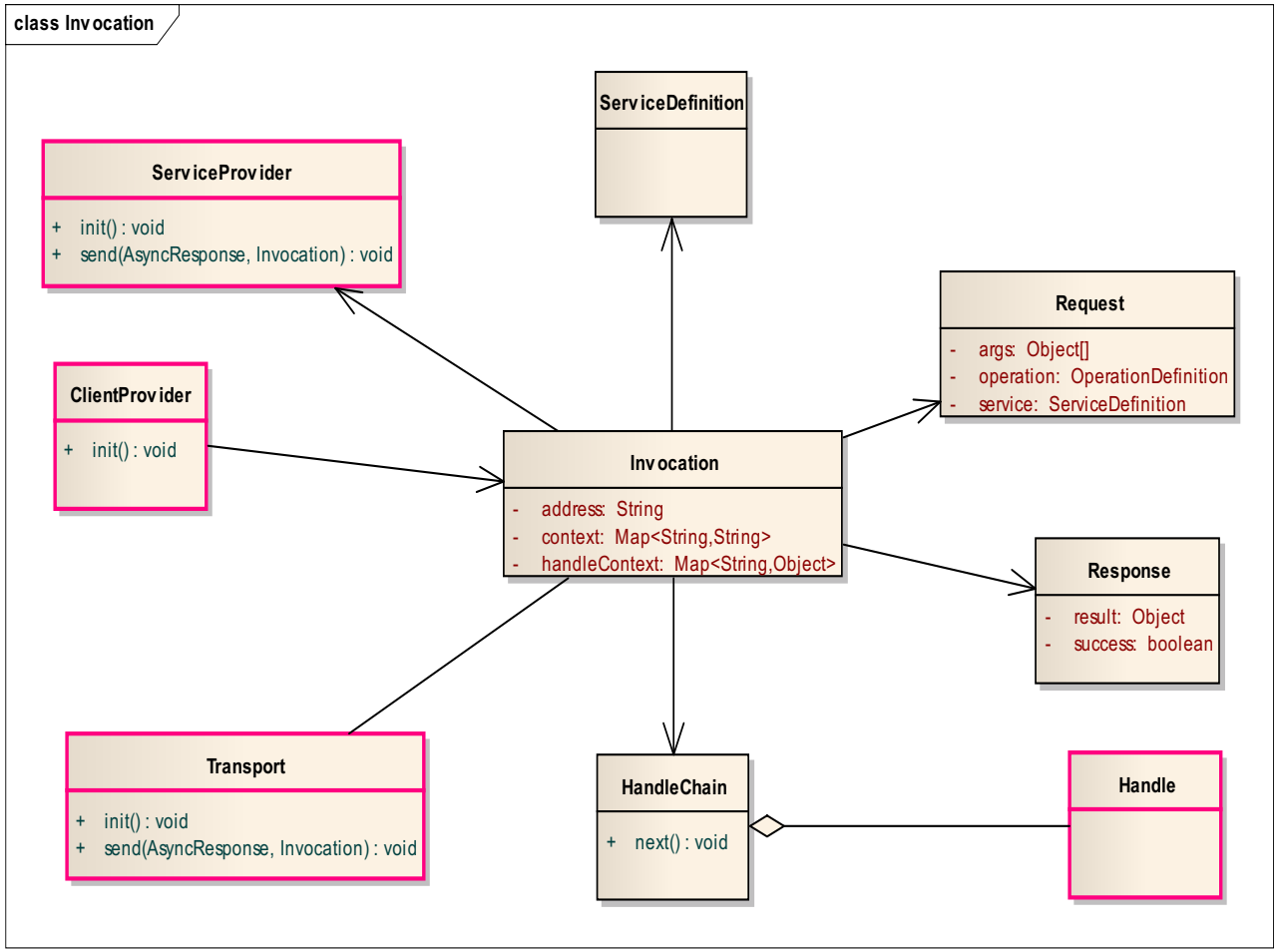
RPC：支持RPC风格的开发，使用Protocol Buffer进行对象序列化。
微服务：
高级特性：

微服务开发SDK

- 使用openapi标准定义服务契约，实现模块解耦和跨语言通信。为构建大型复杂分布式系统提供协议基础。目前支持JAVA 、go两种语言实现。
- 编程模型和通信模型分离，不同的编程模型可以灵活组合不同的通信模型。应用开发者在开发阶段只关注接口开发，部署阶段灵活切换通信方式；支持legacy系统的切换，legacy系统只需要修改服务发布的配置文件（或者annotation），而不需要修改代码。
- 灵活扩展和自由组合的调用链。业务可以自由开发认证、服务管控等功能。

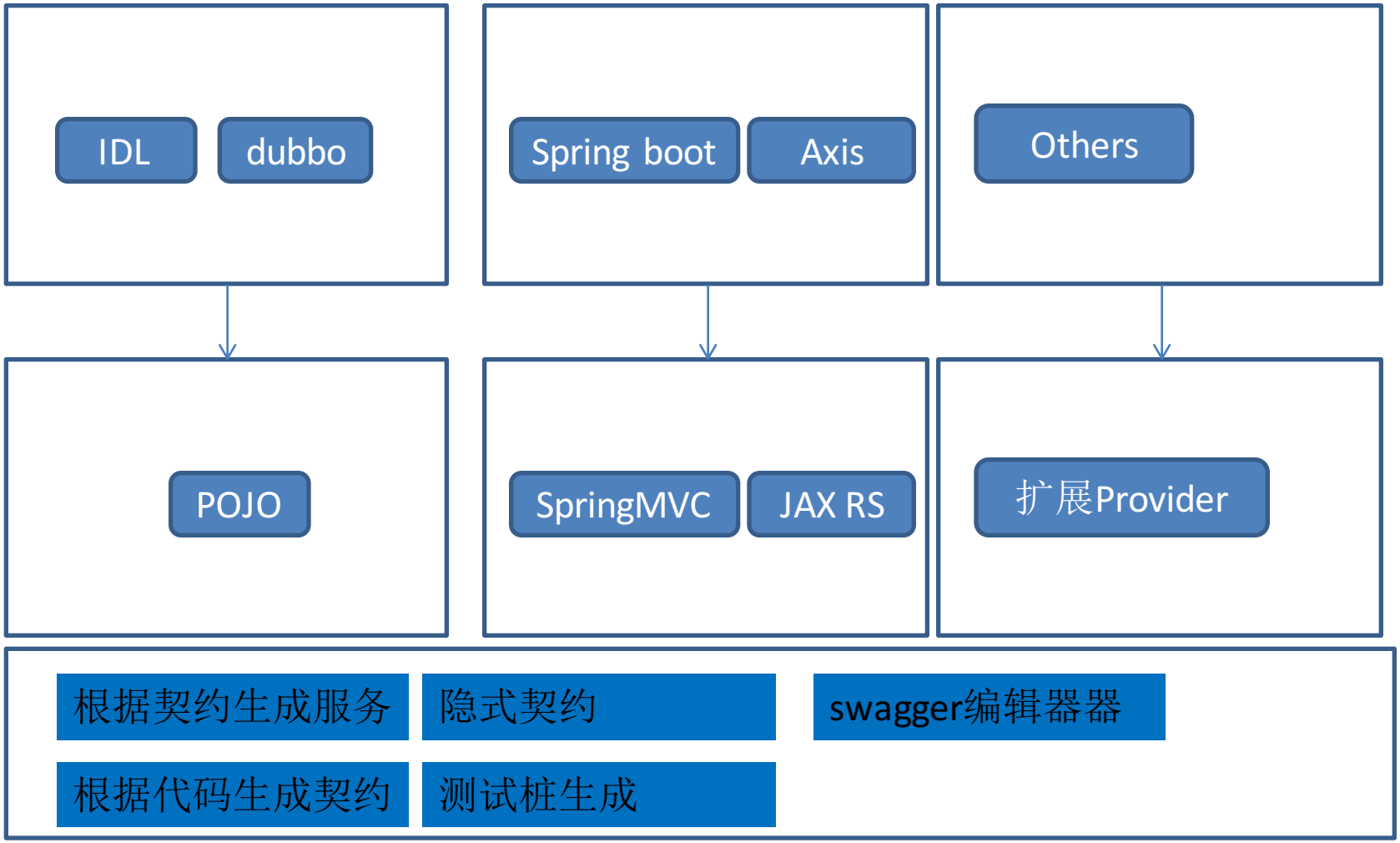
idea: 灵活可扩展的全开放设计

- 编程模型、运行模型、通信模型均可扩展
- 通过Invocation对象和契约实现组件独立
- 开发者参与建设



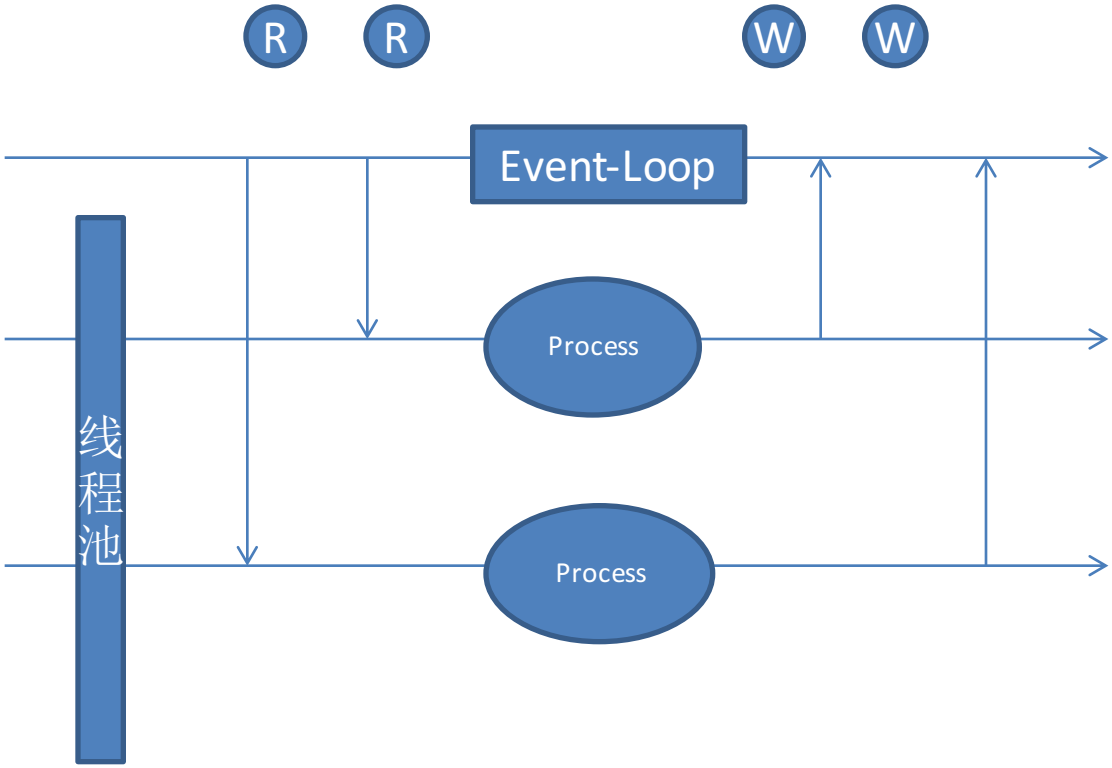
idea: 快速实现微服务改造和遗留系统迁移

- 提供多种开发风格
- 完善的开发工具链



idea: 满足高性能要求

- 采用纯异步实现，处理线程池可以按照服务隔离。
- 标准、开放、协议健壮性对于框架更加重要，选择http作为缺省的应用层协议。
- 优先考虑使用弹性扩容缩容解决性能问题。
- 开发框架的性能在于细节，而不仅仅是协议。



各主流微服务框架性能对比结果：

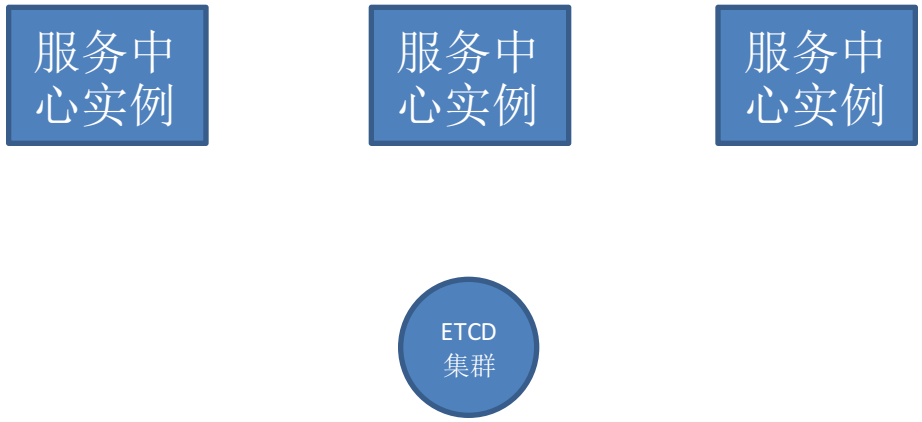
机型	协议	规格(CPU/内存/网卡)	EDAS	实例数	报文大小	调用线程数	TPS	时延(ms)	服
VM	RPC	8U16G1G	EDAS	1	1k	100	58431	1.73	2
物理机	RPC	24U64G1G	EDAS	1	1k	100	97586	1.01	3

机型	协议	规格(CPU/内存/网卡)	CSE SDK	实例数	报文大小	调用线程数	TPS	时延(ms)
VM	REST	8U16G1G	CSE SDK	1	1k	100	70669	1.414
物理机	REST	24U64G1G	CSE SDK	1	1k	100	107126	0.933

机型	协议	规格(CPU/内存/网卡)	Tars	实例数	报文大小	调用线程数	TPS	时延(ms)	服
VM	RPC?	8U16G1G	Tars	1	1k	100	75386	1.32	21

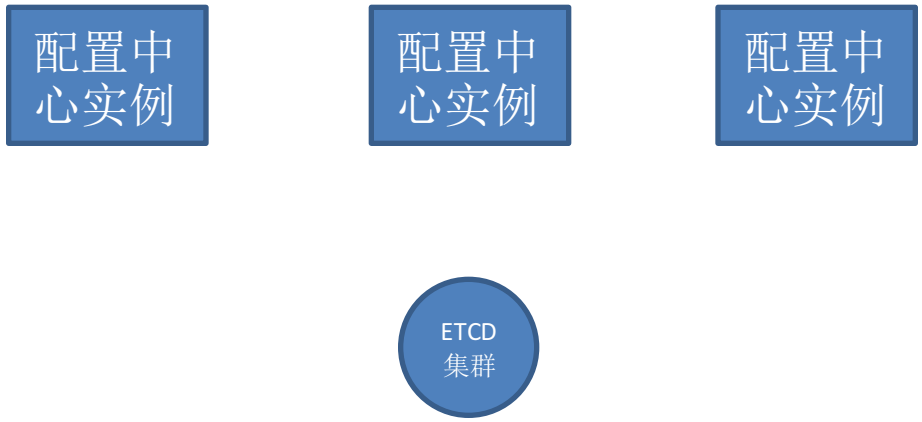
机型	协议	规格(CPU/内存/网卡)	SpringCloud	实例数	报文大小	调用线程数	TPS	时延(ms)
VM	REST	8U16G1G	SpringCloud	1	1k	100	22507	4.70
物理机	REST	24U64G1G	SpringCloud	1	1k	100	51382	2.84

服务中心



- 支持pull/push两种模式监控实例变化
- 实例动态扩容，海量的长连接或者短连接
- 支持灰度发布、服务分组等高级管理特性

配置中心



- 支持pull/push两种模式监控实例变化
- 实例动态扩容，海量的长连接或者短连接
- 配置分层，支持按照租户、应用、服务、实例等多维度进行配置管理

在Spring boot/cloud中使用微服务SDK

场景描述：使用Spring boot/cloud开发应用，并让服务运行于微服务SDK容器中，使用其高性能通信、服务治理、分布式事务管理等功能。

启用依赖

```
<dependency>
  <artifactId>spring-boot-starter-provider</artifactId>
</dependency>
```

配置处理链和协议

```
cse:
  rest:
    address: 0.0.0.0:7999
  highway:
   address: 0.0.0.0:7070
  handler:
    chain:
      Provider:
        default: tcc-server,bizkeeper-provider
```

在Spring boot/cloud中使用微服务SDK

场景描述：使用Spring boot/cloud开发应用，并使用服务中心。

启用依赖

```
<dependency>
<artifactId>spring-boot-starter-discovery</artifactId>
</dependency>
```

使用服务发现

```
@SpringBootApplication
@EnableDiscoveryClientpublic  class DiscoveryClient {
   public static void main(String[] args) throws Exception{
     SpringApplication.run(DiscoveryClient.class, args);
   }
}
```

在Spring boot/cloud中使用微服务SDK

场景描述：使用Spring boot/cloud开发应用，并使用配置中心。

启用依赖

```
<dependency>
<artifactId>spring-boot-starter-configuration</artifactId>
</dependency>
```

使用配置

```
DynamicPropertyFactory.getInstance()
    .getStringProperty("server.port",  null).get()
```

在Spring boot/cloud中使用微服务SDK

其他特性：

将微服务框架运行于Spring boot embeded tomat

启用依赖

```
<dependency>
<artifactId>spring-boot-starter-transport</artifactId>
</dependency>
```

在微服务框架中使用Spring boot/cloud

场景描述：使用第三方服务注册和发现。

扩展接口

接口名称：ServiceRegistryClient

```
/**
 *
 * 注册微服务静态信息
 * @param microservice
 * @return serviceId
 */
String registerMicroservice(Microservice microservice);

/**
 *
 * 根据微服务唯一标识查询微服务静态信息
 * @param microserviceId
 * @return
 */
Microservice getMicroservice(String microserviceId);
```

在微服务框架中使用Spring boot/cloud

场景描述：Spring Cloud Router and Filter: Zuul

扩展接口

接口名称：Handler

```
void handle(Invocation invocation, AsyncResponse asyncResp) throws Exception;
```

在微服务框架中使用Spring boot/cloud

场景描述：Spring Cloud Sleuth

扩展接口

接口名称：Handler

```
void handle(Invocation invocation, AsyncResponse asyncResp) throws Exception;
```

examples：小型系统的设计