云时代的号角

Dubbo-go新注册模型

邓明

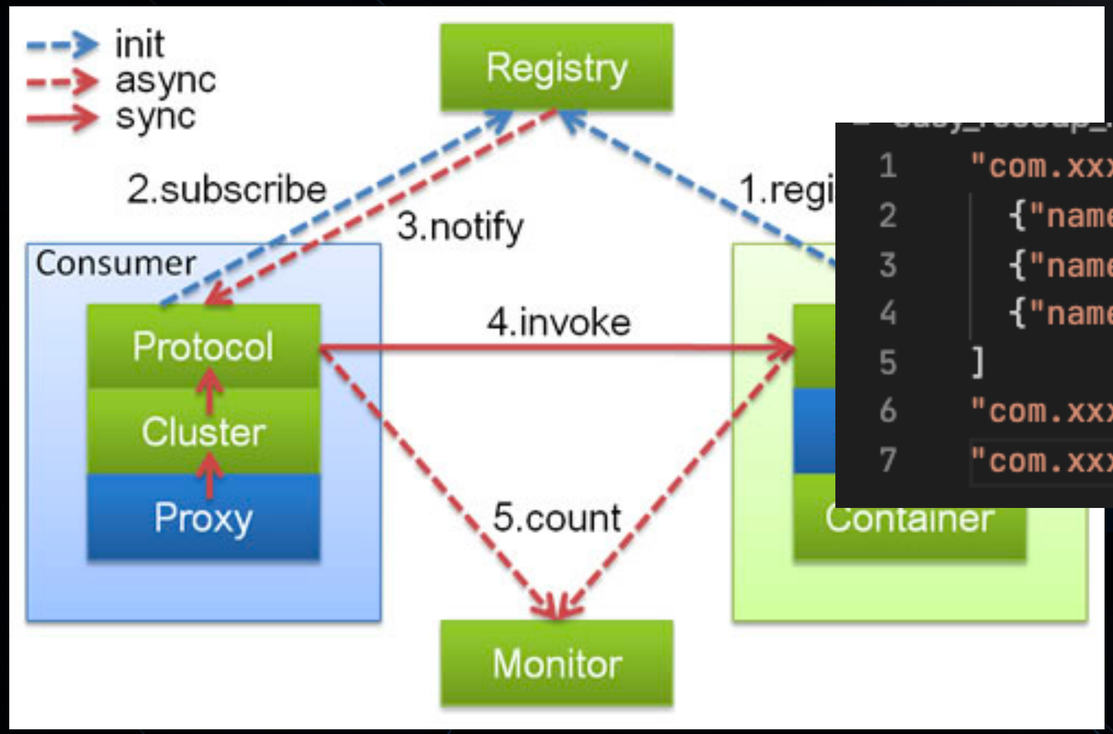
2020.07.18

Content

- 为什么引入新的模型
- Dubbo-go 新注册模型
- 设计与实现
- 总结

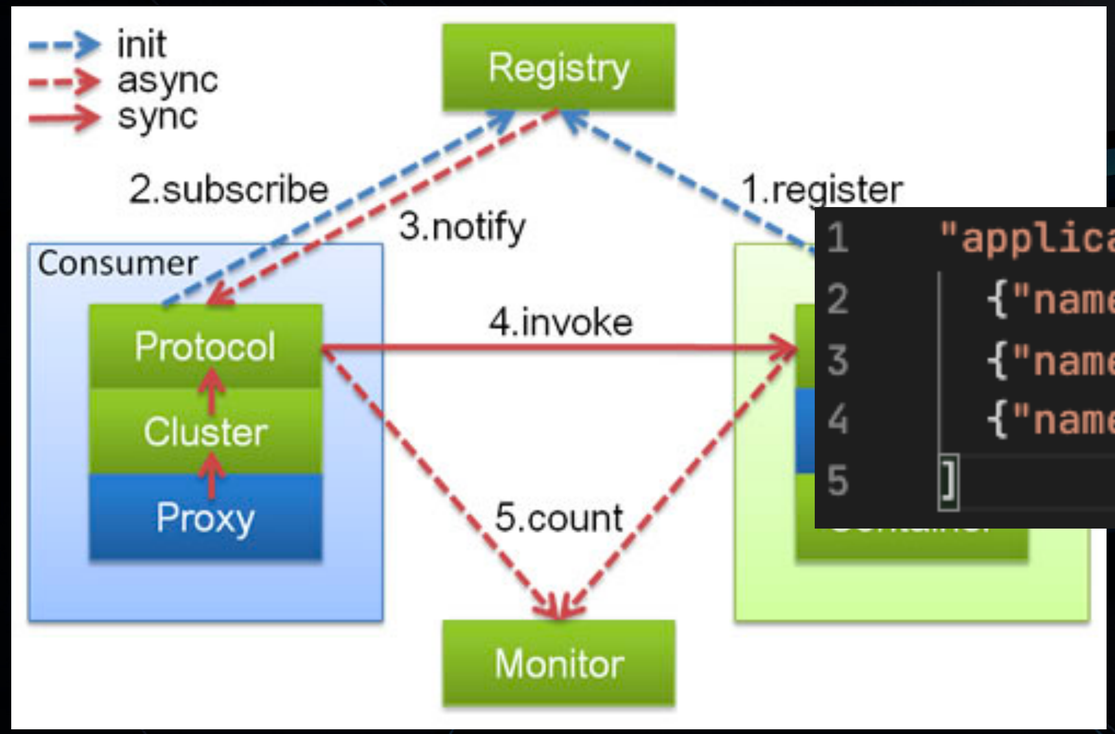
为什么引入新的注册模型

接口维度注册模型




```
"com.xxx.User": [
  {"name":"instance1", "ip":"127.0.0.1", "metadata":{"timeout":1000}},
  {"name":"instance2", "ip":"127.0.0.2", "metadata":{"timeout":2000}},
  {"name":"instance3", "ip":"127.0.0.3", "metadata":{"timeout":3000}},
]
"com.xxx.Product": [Instance list],
"com.xxx.Order": [Instance list]
```

应用维度注册模型

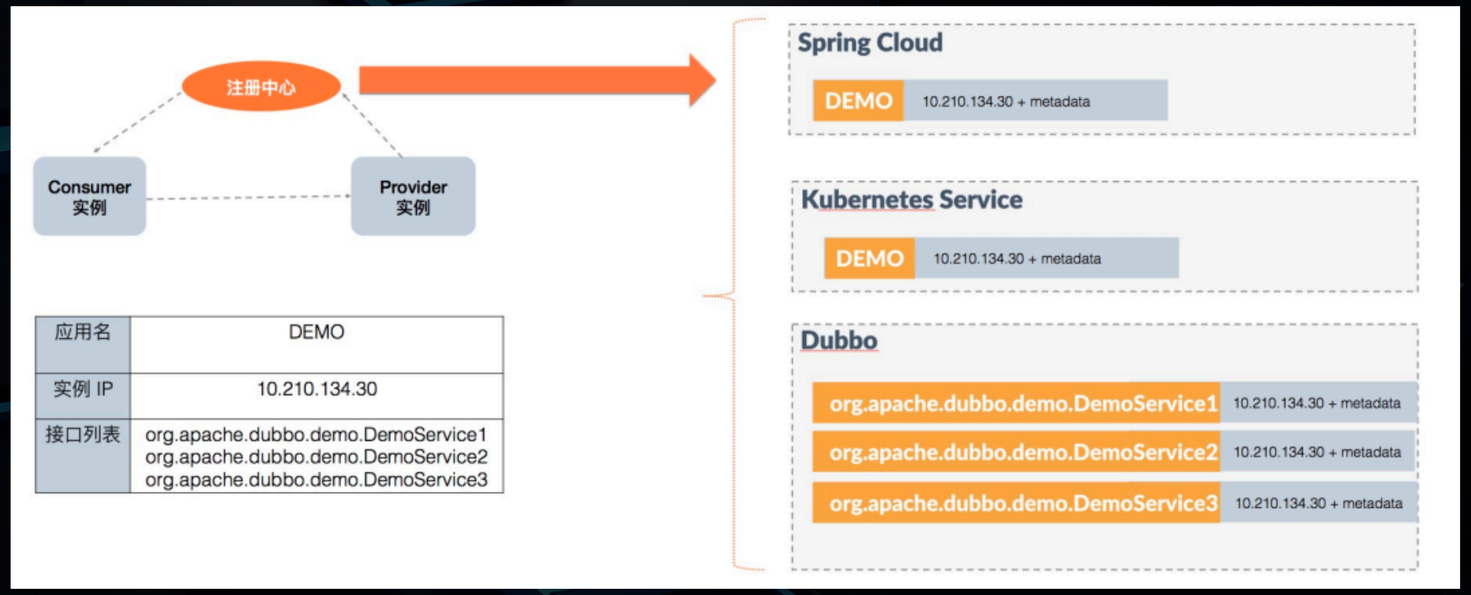



```
"application1": [
  {"name":"instance1", "ip":"127.0.0.1", "metadata":{}},
  {"name":"instance2", "ip":"127.0.0.2", "metadata":{}},
  {"name":"instanceN", "ip":"127.0.0.3", "metadata":{}}
]
```

Why?

与主流注册模型保持一致

支持大规模集群



Dubbo-go新注册模型

设计目标

完全兼容现有模型——用户无感知迁移

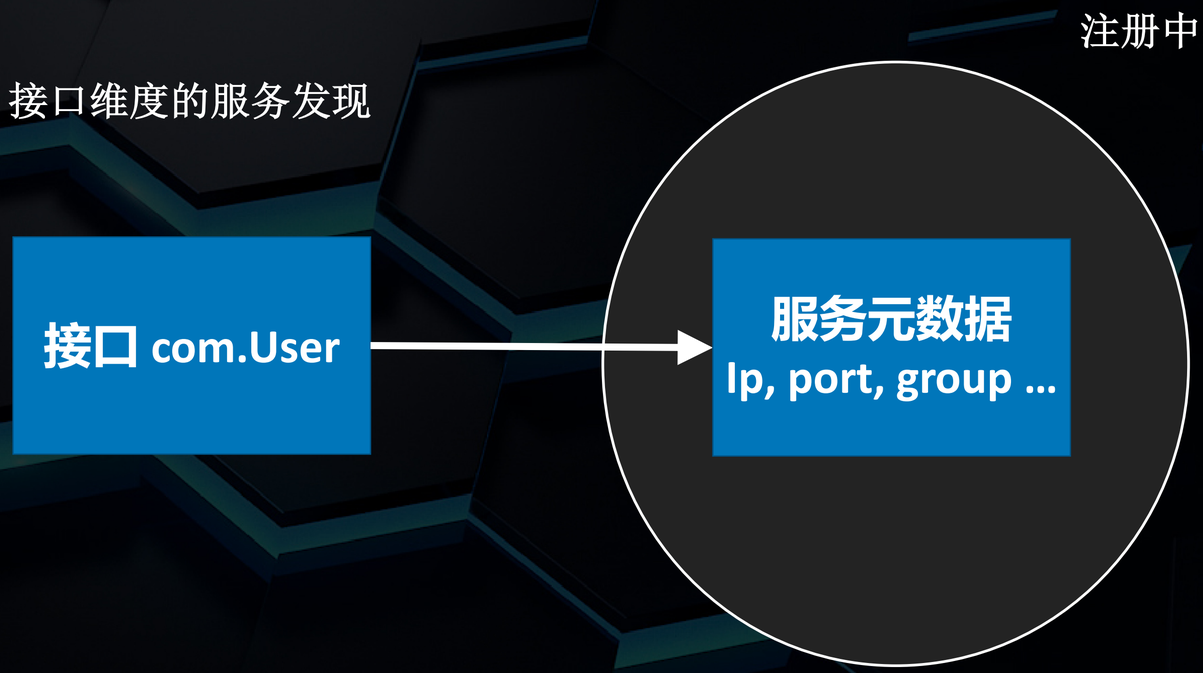
保留现有接口维度的服务配置——保留细粒度的控制服务的能力

```
references:
  "UserProvider":
    # 可以指定多个registry，使用逗号隔开;不指定默认向所有注册中心注册
    registry: "demoZk"
    protocol : "dubbo"
    interface : "com.ikurento.user.UserProvider"
    cluster: "failover"
    methods :
    - name: "GetUser"
      retries: 3
```

我只有一个接口，怎么获得发起调用所需的所有信息？

接口维度注册模型直接获取服务元数据

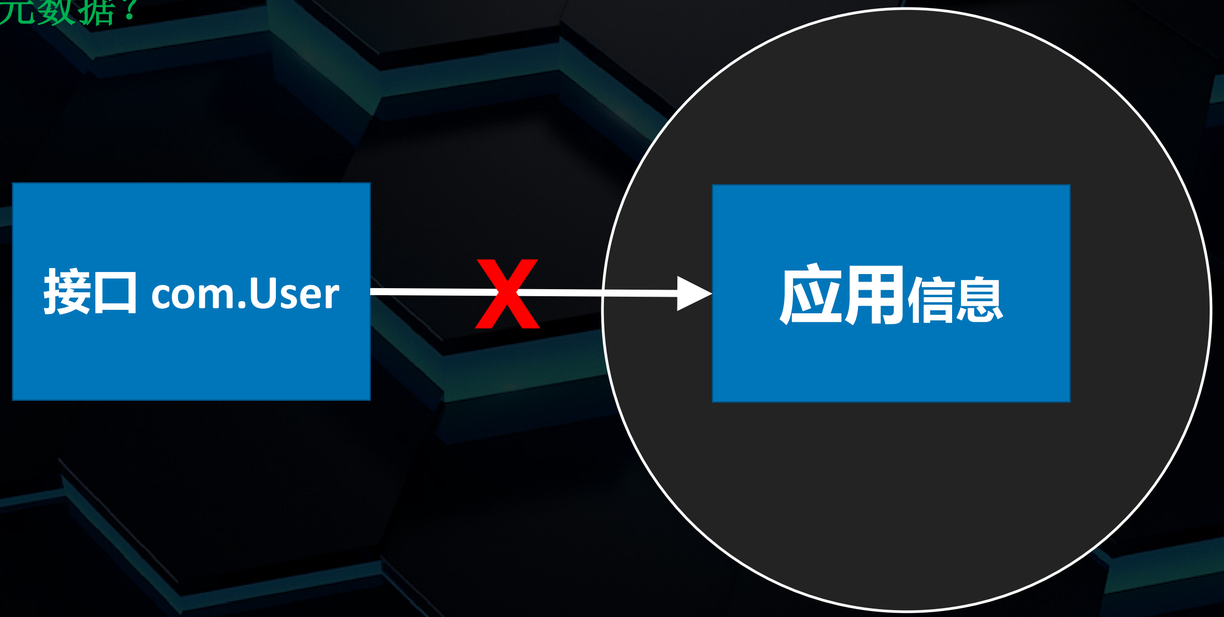
```
references:
  "UserProvider":
    # 可以指定多个registry，使用逗号隔开;不指定默认向所有注册中心注册
    registry: "demoZk"
    protocol : "dubbo"
    interface : "com.ikurento.user.UserProvider"
    cluster: "failover"
    methods :
    - name: "GetUser"
      retries: 3
```



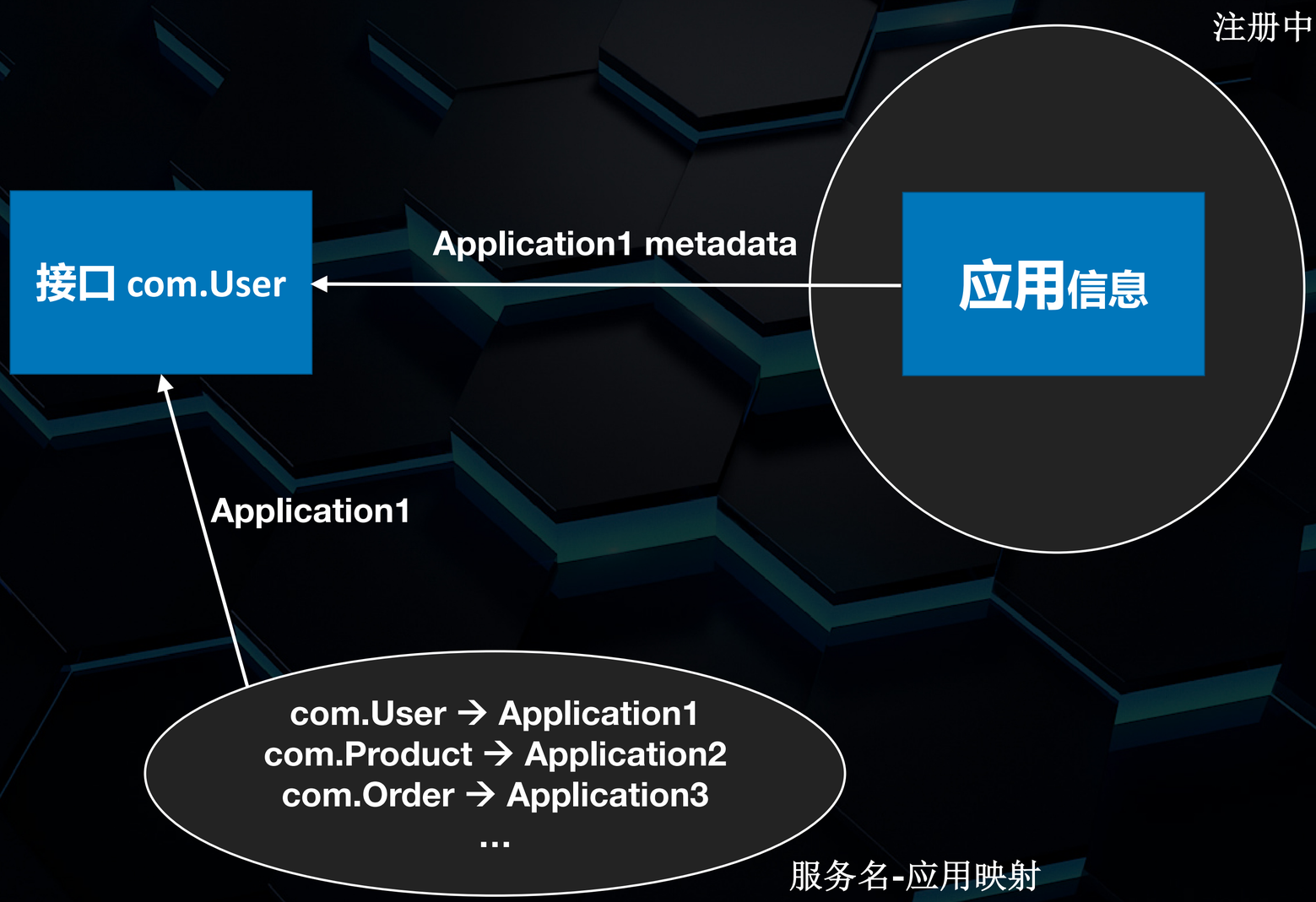
在应用维度模型下，怎么获得发起调用所需的所有信息？

这个接口对应的服务，是由哪个应用提供的？

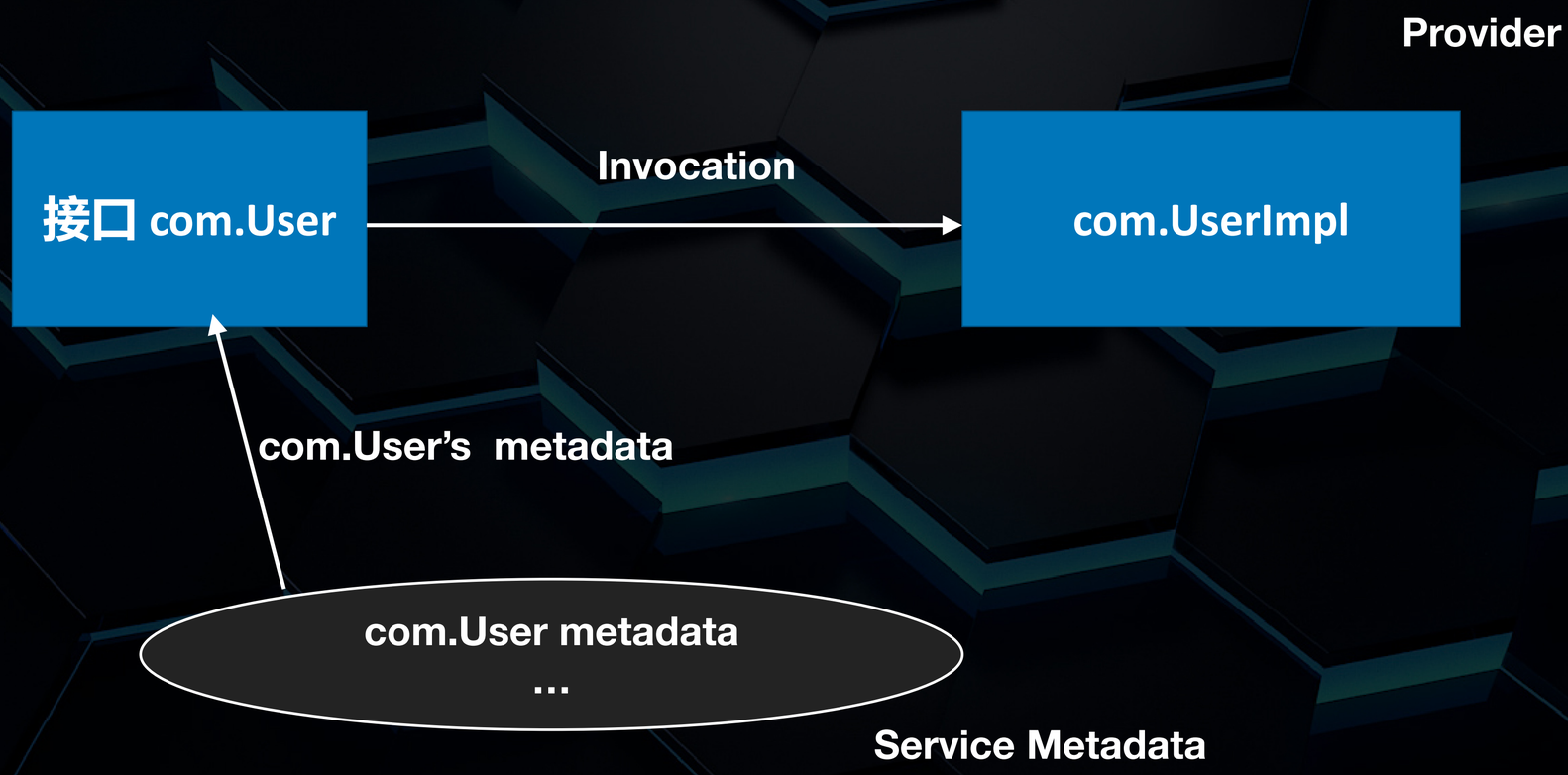
我拿到应用信息之后，怎么拿到服务元数据？

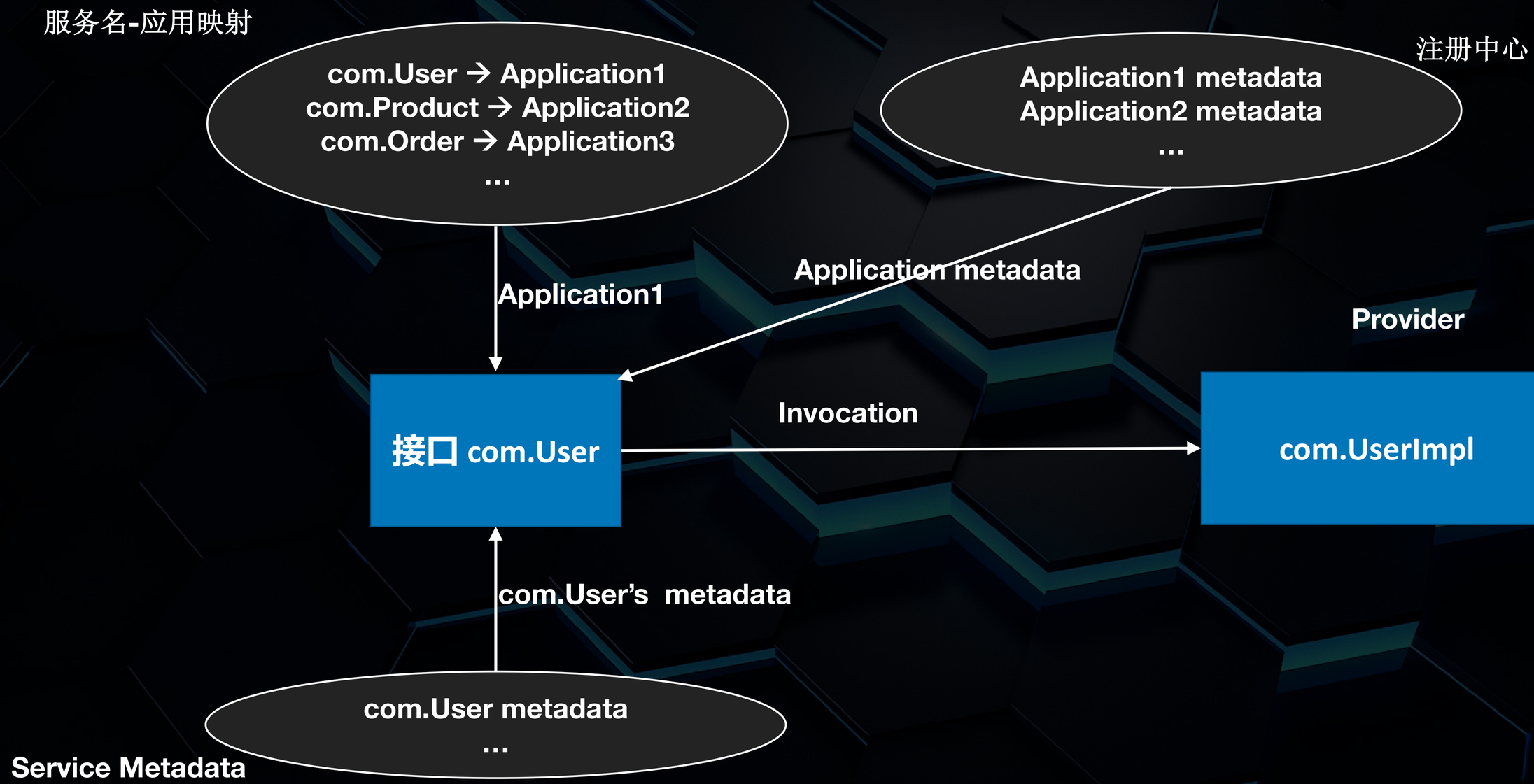


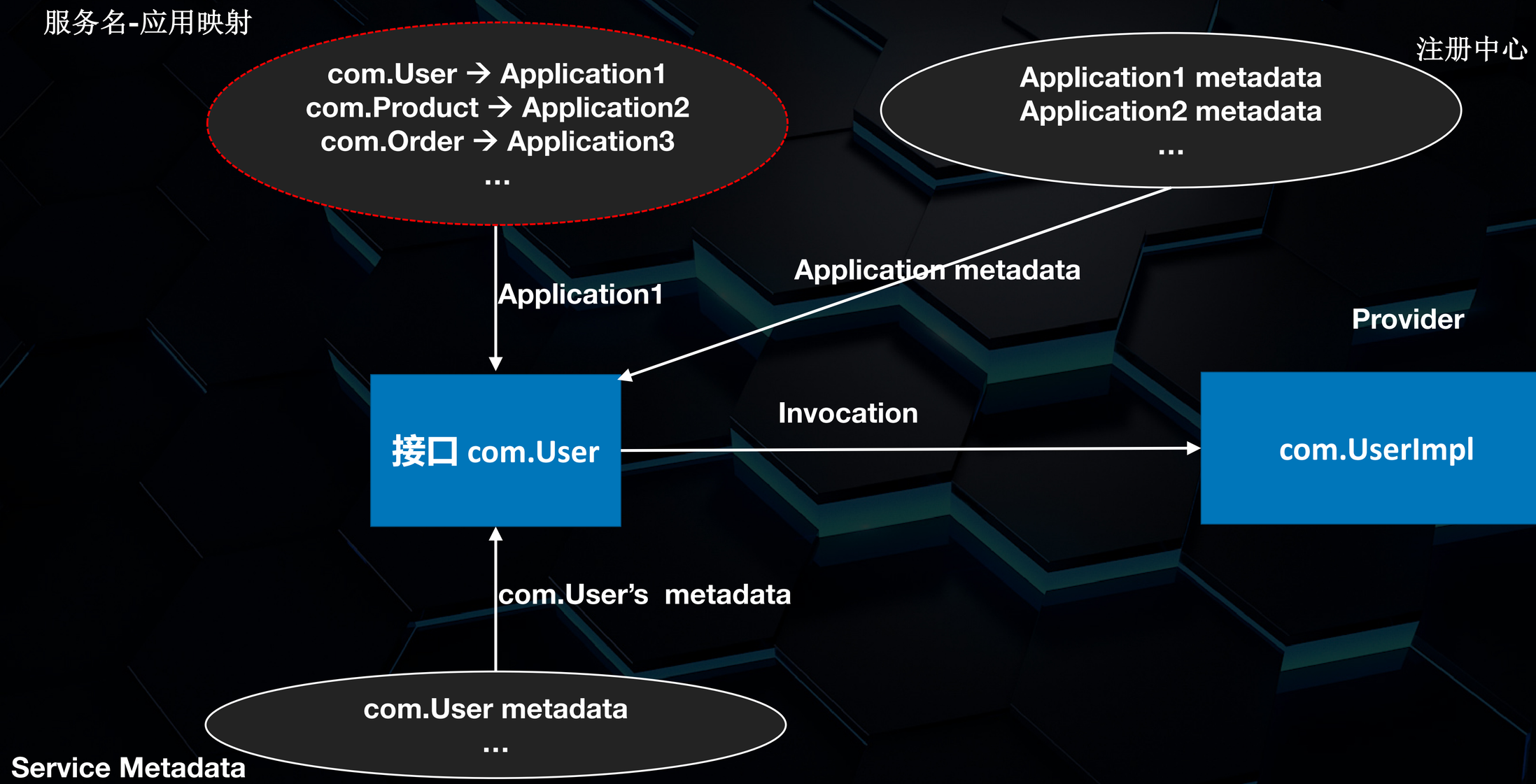
Step1 获得应用信息

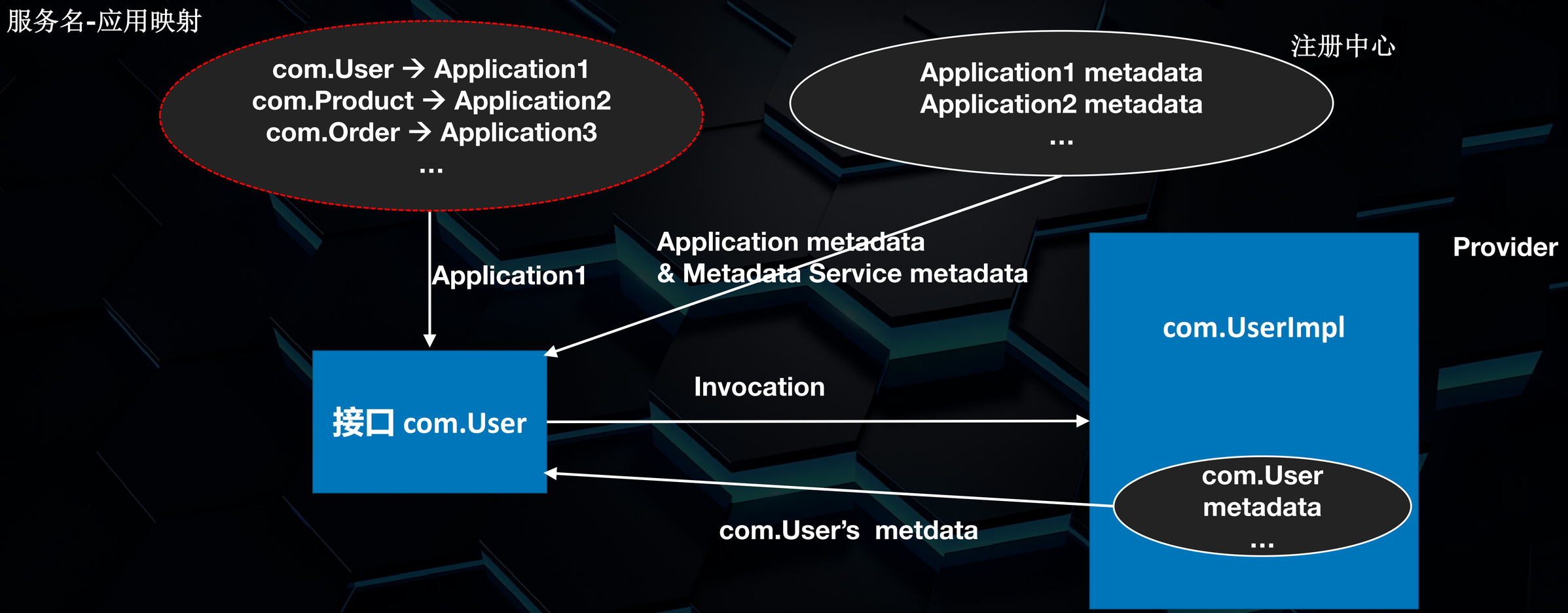


Step2 获得服务元数据



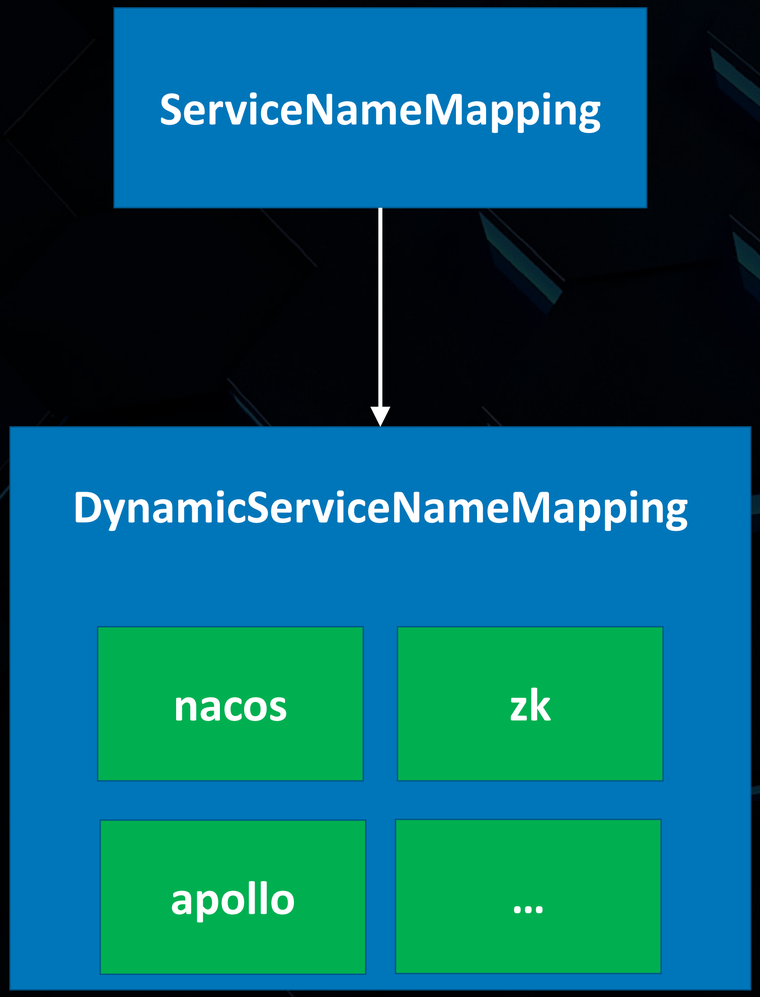

服务名-应用映射
com.User → Application1
com.Product → Application2
com.Order → Application3
...
注册中心
Application1 metadata
Application2 metadata
...
Application1
Application metadata
Provider
接口 com.User
Invocation
com.UserImpl
com.User's metadata
com.User metadata
...
Service Metadata


服务名-应用映射
com.User → Application1
com.Product → Application2
com.Order → Application3
...
注册中心
Application1 metadata
Application2 metadata
...
Application1
Application metadata
Provider
接口 com.User
Invocation
com.UserImpl
com.User's metadata
com.User metadata
...
Service Metadata


服务名-应用映射
com.User → Application1
com.Product → Application2
com.Order → Application3
...
注册中心
Application1 metadata
Application2 metadata
...
Application1
Application metadata
& Metadata Service metadata
Provider
com.UserImpl
接口 com.User
Invocation
com.User
metadata
...
com.User's metdata

接口与实现

ServiceNameMapping

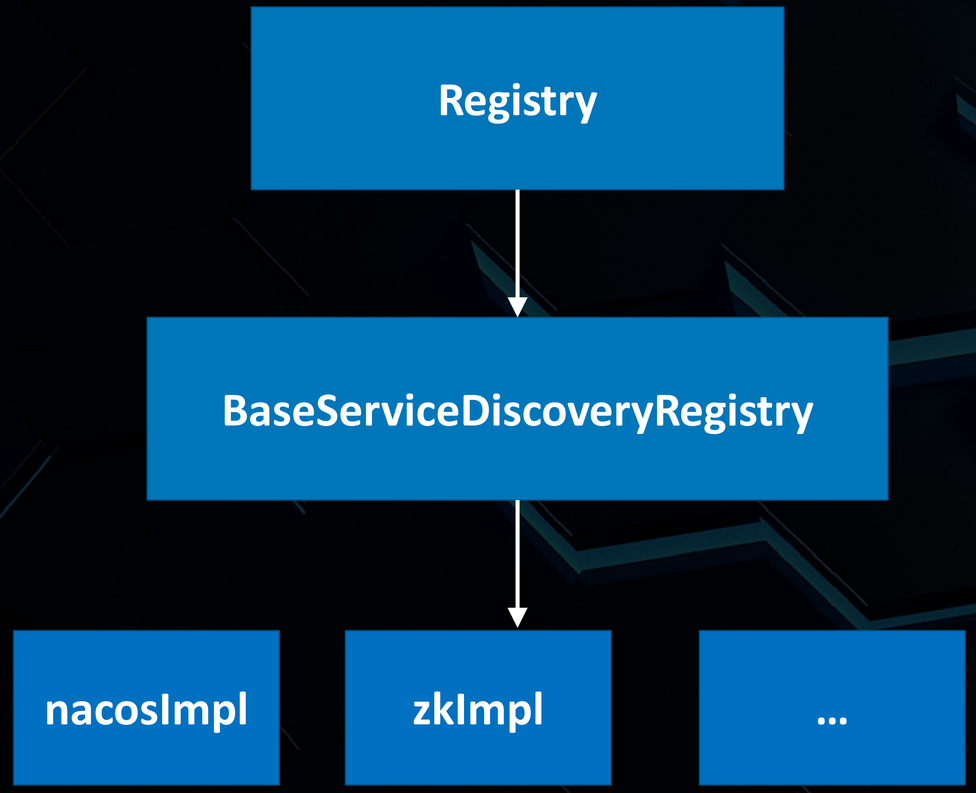


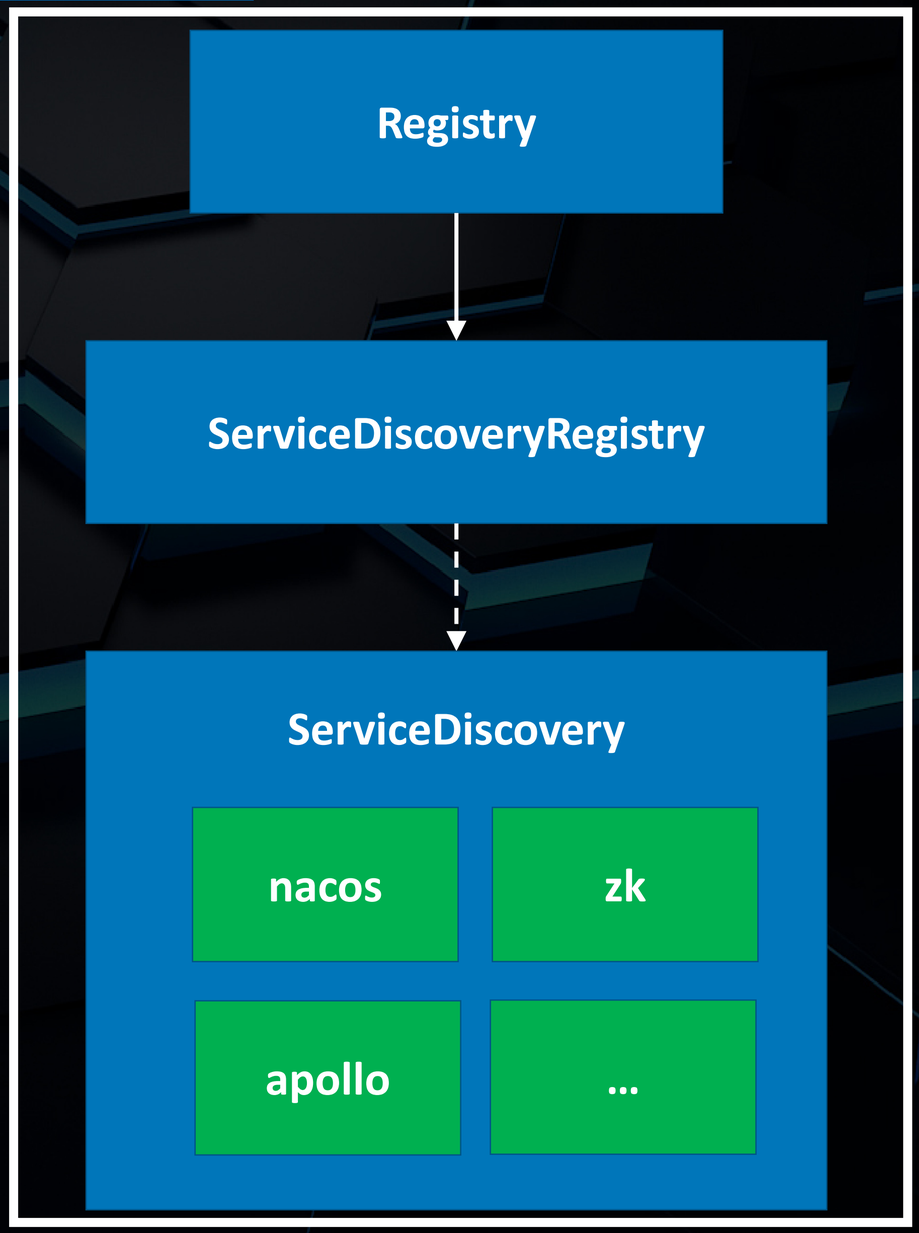

```yaml
references:
  "UserProvider":
    # 可以指定多个registry，使用逗号隔开;不指定默认向所有注册中心注册
    registry: "demoServiceDiscovery"
    protocol : "dubbo"
    provide_by: "user-info-server"
    interface : "com.ikurento.user.UserProvider"
    cluster: "failover"
    methods :
    - name: "GetUser"
      retries: 3
```

ServiceNameMapping

```go
// ServiceNameMapping try to build the mapping between application-level service and interface-level service.
type ServiceNameMapping interface {

    // Map will map the service to this application-level service
    Map(serviceInterface string, group string, version string, protocol string) error

    // Get will return the application-level services
    Get(serviceInterface string, group string, version string, protocol string) (*gxset.HashSet, error)
}
```

ServiceDiscoveryRegistry

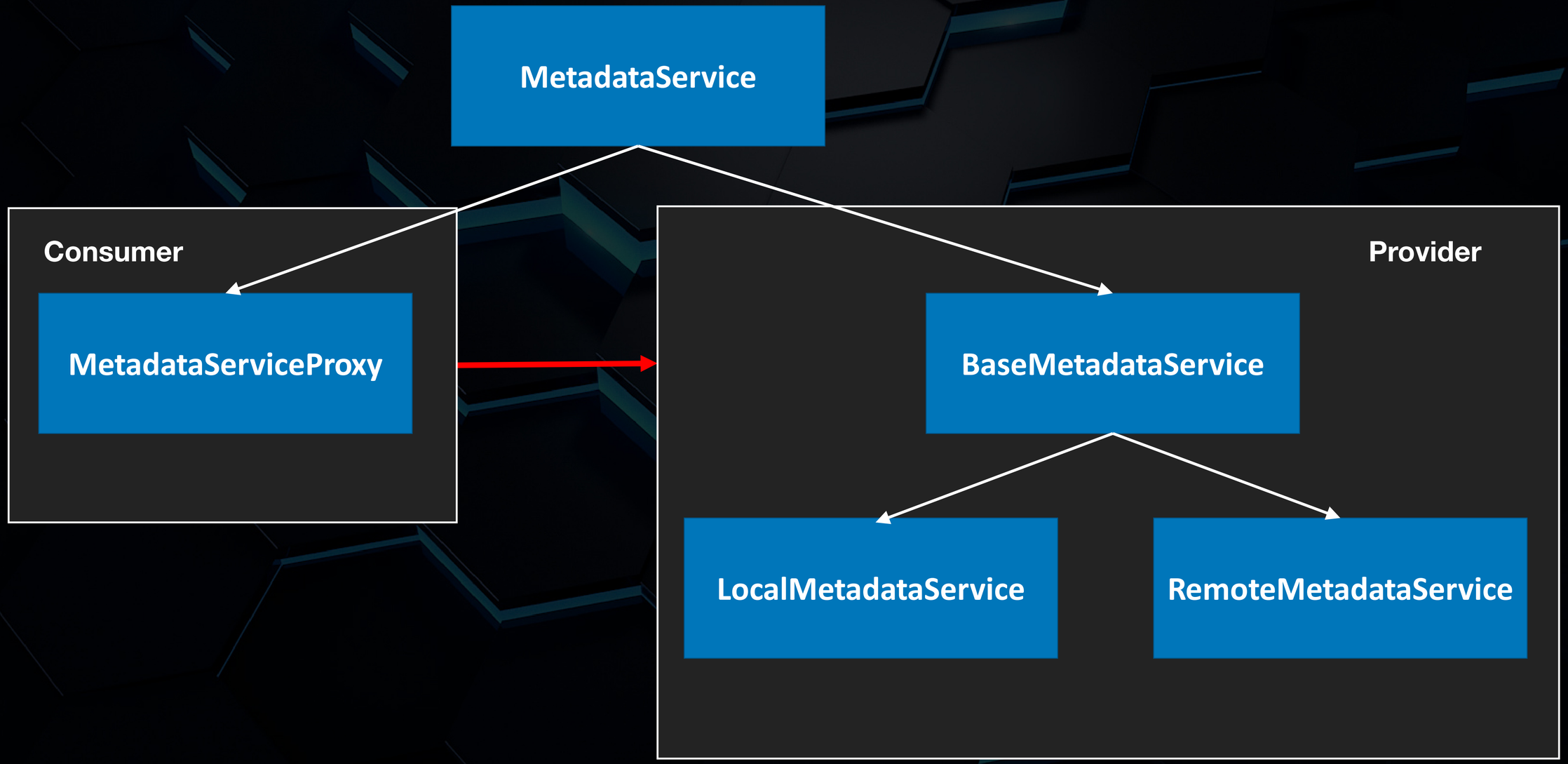




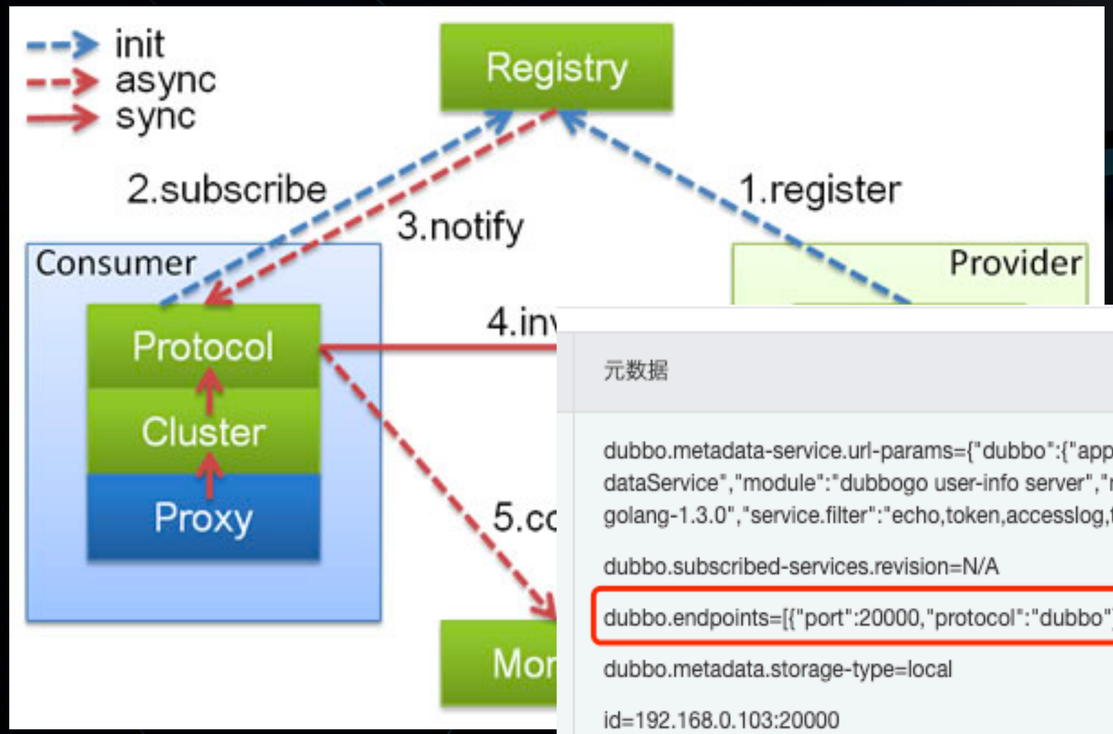
ServiceDiscovery

```
// Register will register an instance of ServiceInstance to registry
Register(instance ServiceInstance) error

// Update will update the data of the instance in registry
Update(instance ServiceInstance) error

// Unregister will unregister this instance from registry
Unregister(instance ServiceInstance) error

// ---------------- discovery ------------------
// GetDefaultPageSize will return the default page size
GetDefaultPageSize() int

// GetServices will return the all service names.
GetServices() *gxset.HashSet

// GetInstances will return all service instances with serviceName
GetInstances(serviceName string) []ServiceInstance
```

MetadataService



应用维度注册模型



元数据

dubbo.metadata-service.url-params={"dubbo":{"app.version":"0.0.1","bean.name":"MetadataService","environment":"dev","interface":"org.apache.dubbo.metadata.MetadataService","module":"dubbogo user-info server","name":"user-info-server","organization":"ikurento.com","owner":"ZX","port":"20000","registry.role":"3","release":"dubbo-golang-1.3.0","service.filter":"echo,token,accesslog,tps,generic_service,execute,pshutdown","side":"provider","version":"1.0.0"}}

dubbo.subscribed-services.revision=N/A

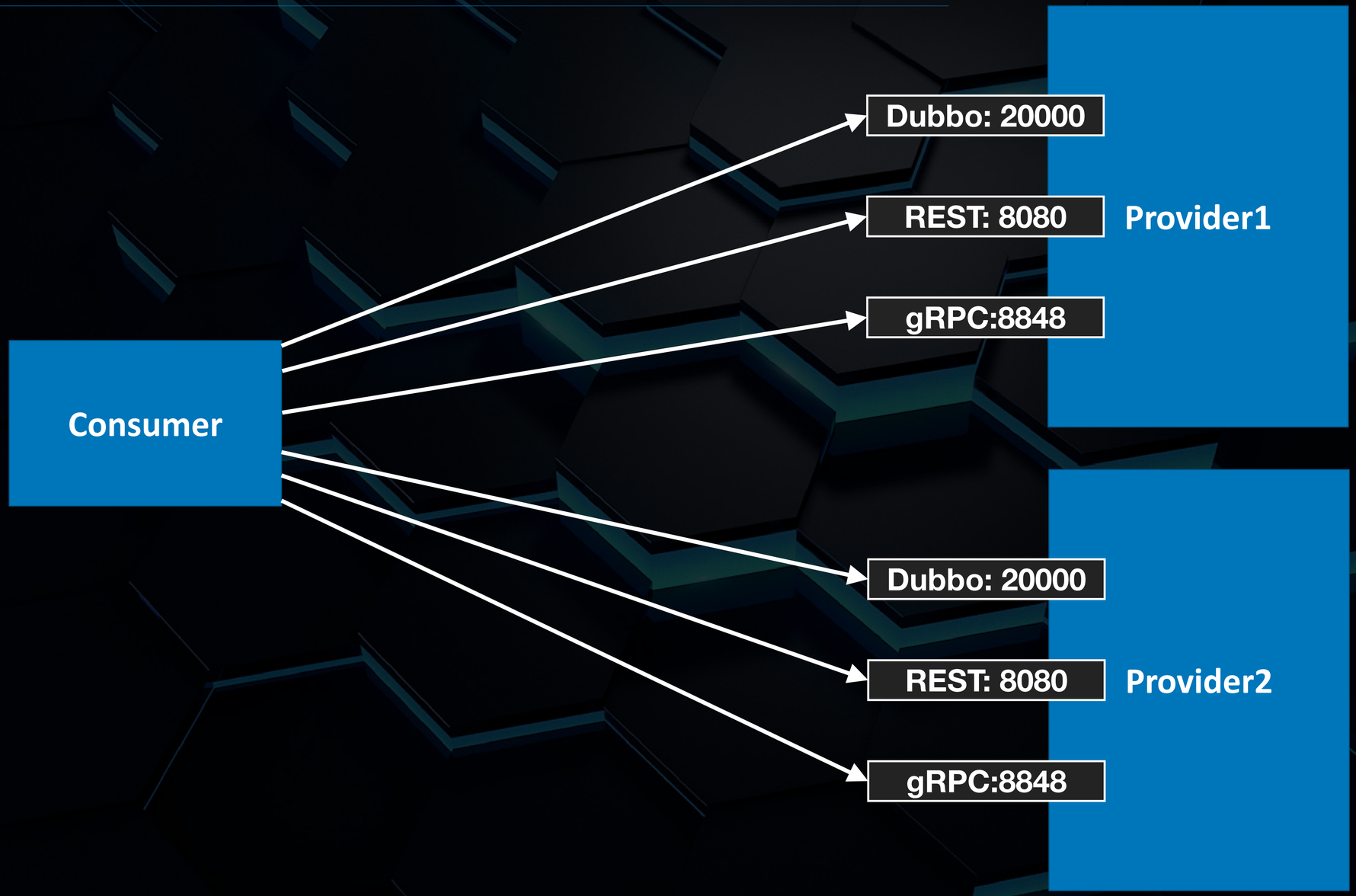
dubbo.endpoints=[{"port":20000,"protocol":"dubbo"}]

dubbo.metadata.storage-type=local

id=192.168.0.103:20000

dubbo.exported-services.revision=3092375190

Provider endpoints



元数据服务

元数据

dubbo.metadata-service.url-params={"dubbo":{"app.version":"0.0.1","bean.name":"MetadataService","environment":"dev","interface":"org.apache.dubbo.metadata.MetadataService","module":"dubbogo user-info server","name":"user-info-server","organization":"ikurento.com","owner":"ZX","port":"20000","registry.role":"3","release":"dubbo-golang-1.3.0","service.filter":"echo,token,accesslog,tps,generic_service,execute,pshutdown","side":"provider","version":"1.0.0"}}

dubbo.subscribed-services.revision=N/A

dubbo.endpoints=[{"port":20000,"protocol":"dubbo"}]

dubbo.metadata.storage-type=local

id=192.168.0.103:20000

dubbo.exported-services.revision=3092375190

元数据服务

```go
func (m *MetadataServiceProxy) GetExportedURLs(serviceInterface string, group string, version string, protocol string

    siV := reflect.ValueOf(serviceInterface)
    gV := reflect.ValueOf(group)
    vV := reflect.ValueOf(version)
    pV := reflect.ValueOf(protocol)

    const methodName = "getExportedURLs"

    inv := invocation.NewRPCInvocationWithOptions(invocation.WithMethodName(methodName),
        invocation.WithArguments([]interface{}{siV.Interface(), gV.Interface(), vV.Interface(), pV.Interface()}),
        invocation.WithReply(reflect.ValueOf(&[]interface{}{}).Interface()),
        invocation.WithAttachments(map[string]string{constant.ASYNC_KEY: "false"}),
        invocation.WithParameterValues([]reflect.Value{siV, gV, vV, pV}))

    res := m.invkr.Invoke(context.Background(), inv)
    if res.Error() != nil {
        logger.Errorf( fmt: "could not get the metadata service from remote provider: %v", res.Error())
        return []interface{}{}, nil
    }

    urlStrs := res.Result().(*[]interface{})

    ret := make([]interface{}, 0, len(*urlStrs))

    for _, s := range *urlStrs {
        ret = append(ret, s)
    }
    return ret, nil
}
```

总结

关键点

- 注册信息按照 应用-实例 的模式进行组织
- 在应用和接口之间建立了映射关系
- 建立了元数据同步机制

服　　务　　自　　省

Thank you !