Лекция 8: Проектирование распределённых приложений

Юрий Литвинов
y.litvinov@spbu.ru

19.05.2022

Распределённые системы

- Компоненты приложения находятся в компьютерной сети
- Взаимодействуют через обмен сообщениями
- Основное назначение — работа с общими ресурсами
- Особенности
  - Параллельная работа
  - Независимые отказы
  - Отсутствие единого времени

Частые заблуждения при проектировании распределённых систем

- Сеть надёжна
- Задержка (latency) равна нулю
- Пропускная способность бесконечна
- Сеть безопасна
- Топология сети неизменна
- Администрирование сети централизовано
- Передача данных “бесплатна”
- Сеть однородна

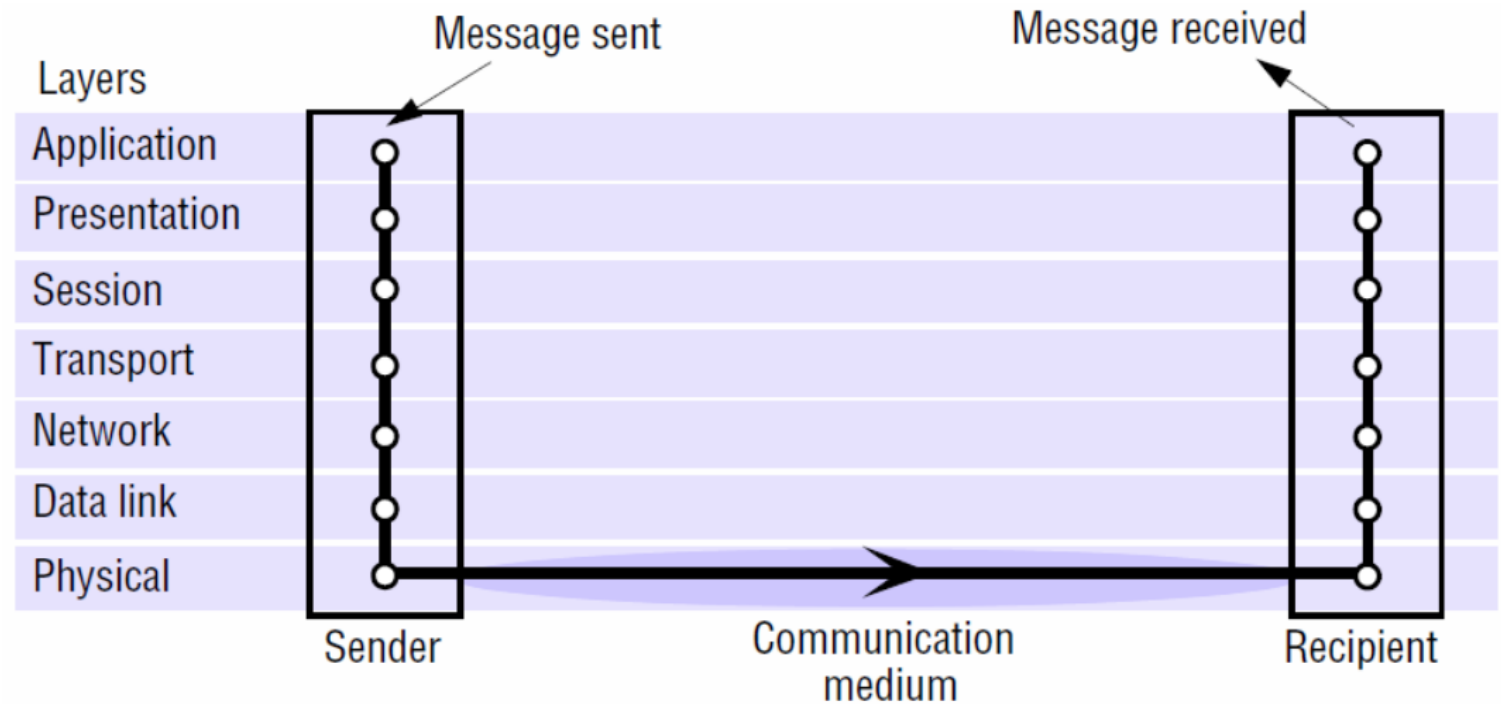
Виды взаимодействия

- Межпроцессное взаимодействие
- Удалённые вызовы
  - Протоколы вида “запрос-ответ”
  - Удалённые вызовы процедур (remote procedure calls, RPC)
  - Удалённые вызовы методов (remote method invocation, RMI)
- Неявное взаимодействие
  - Групповое взаимодействие
  - Модель “издатель-подписчик”
  - Очереди сообщений
  - Распределённая общая память

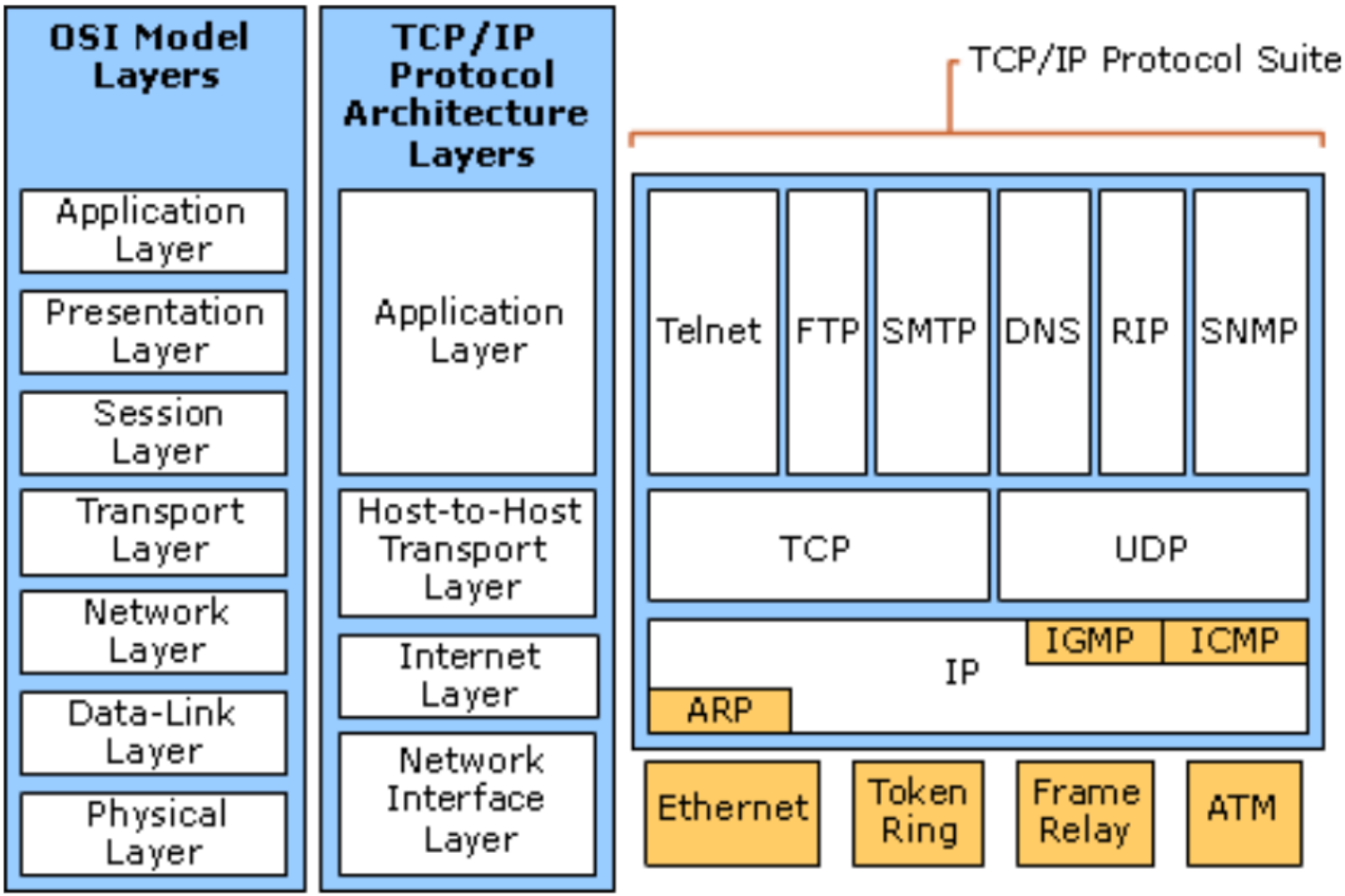
Типичные архитектурные стили

- Уровневая архитектура
  - ОС
  - Коммуникационная инфраструктура (Middleware)
  - Приложения и сервисы
- Клиент-сервер
  - Тонкий клиент
  - Бизнес-логика и данные — на сервере
- Трёхзвенная и N-уровневая архитектуры
  - Бизнес-логику и работу с данными часто разделяют

Модель OSI

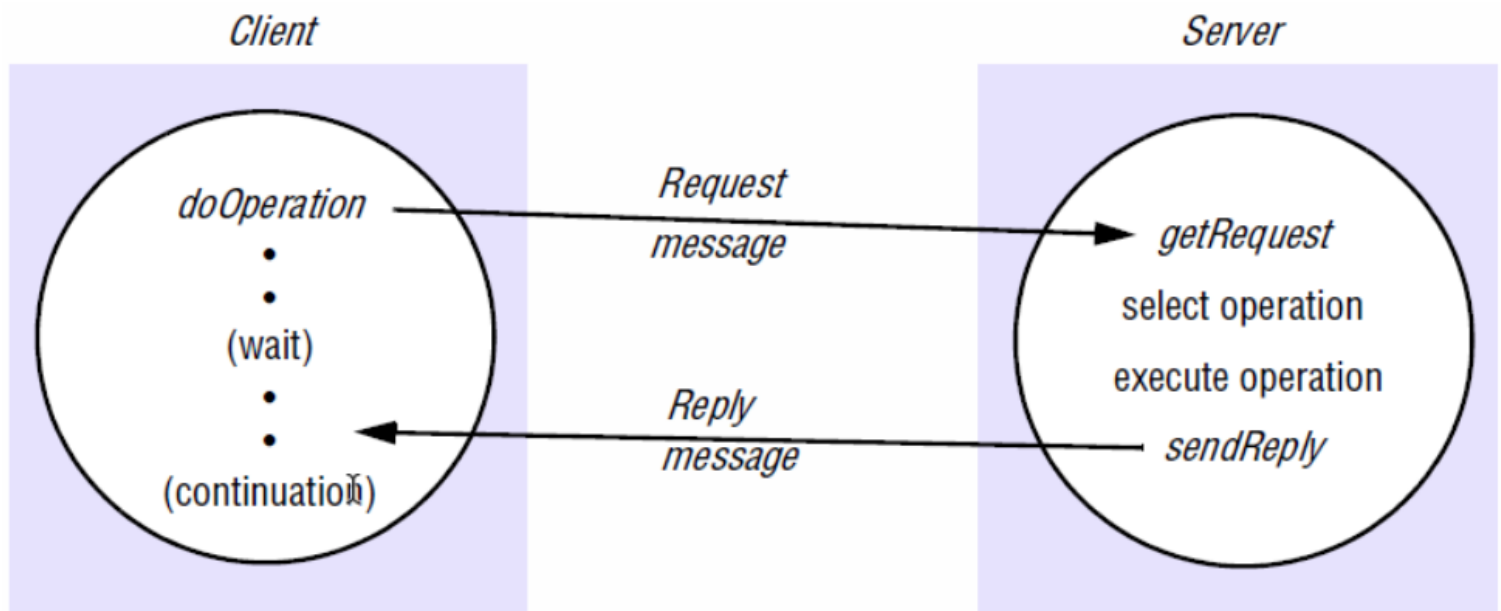

Message sent
Message received
Layers
Application
Presentation
Session
Transport
Network
Data link
Physical
Sender
Communication medium
Recipient

Стек протоколов TCP/IP



Протоколы “запрос-ответ”

- Запрос, действие, ответ
- Преимущественно синхронные вызовы



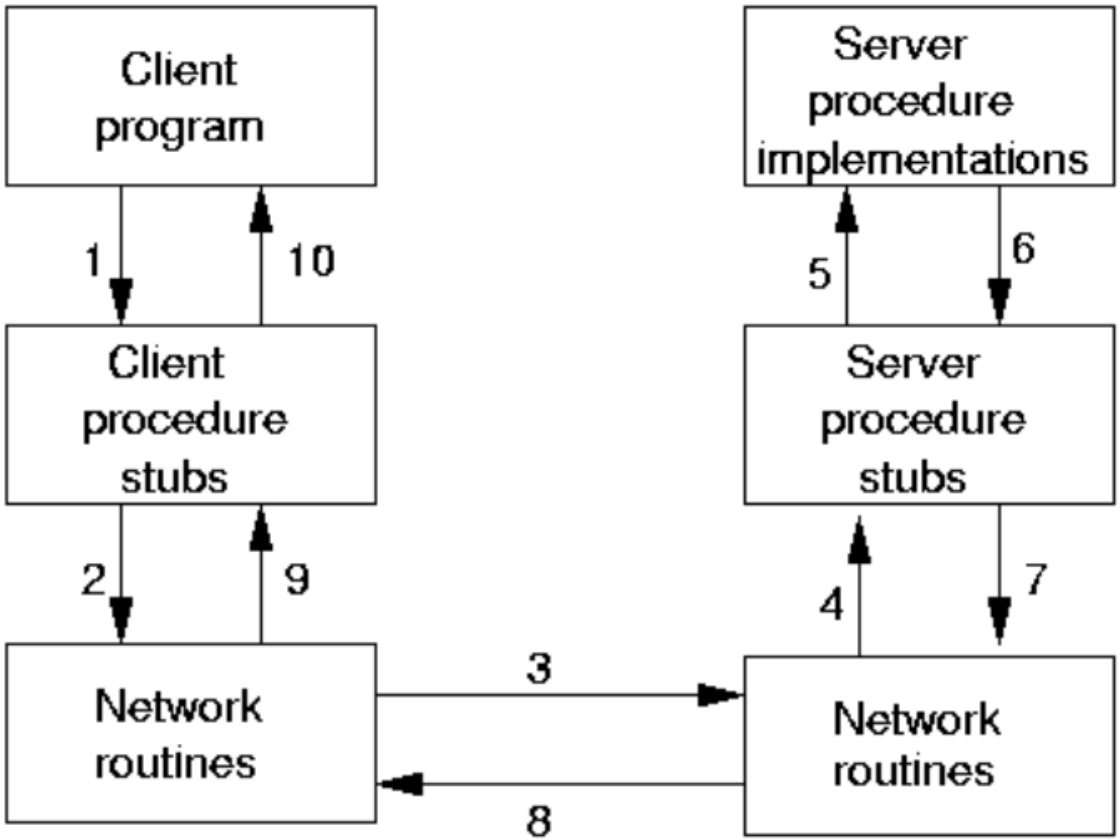
“Запрос-ответ” поверх UDP

+ Уведомления не нужны
+ Установление соединения — в два раза больше сообщений
+ Управление потоком не имеет смысла
- Потери пакетов
  - Таймаут + повторный запрос на уровне бизнес-логики
  - Защита от повторного выполнения операции (хранение “истории”)
  - Новый запрос как подтверждение получения прошлого
- Неопределённый порядок пакетов

“Запрос-ответ” поверх TCP

+ Использование потоков вместо набора пакетов
  - Удобная отправка больших объёмов данных
  - Один поток на всё взаимодействие
+ Интеграция с потоками ОО-языков
+ Надёжность доставки
  - Отсутствие необходимости проверок на уровне бизнес-логики
  - Уведомления в пакетах с ответом
  - Упрощение реализации
- Тяжеловесность коммуникации

RPC


Client program
Server procedure implementations
1
10
5
6
Client procedure stubs
Server procedure stubs
2
9
4
7
3
Network routines
Network routines
8

Прозрачность RPC-вызовов

- Изначальная цель — максимальная похожесть на обычные вызовы
  - Location and access transparency
- Удалённые вызовы более уязвимы к отказам
  - Нужно понимать разницу между отказом сети и отказом сервиса
    - Exponential backoff
  - Клиенты должны знать о задержках при передаче данных
    - Возможность прервать вызов
- Явная маркировка удалённых вызовов?
  - Прозрачность синтаксиса
  - Явное отличие в интерфейсах
    - Указание сематики вызова

Удалённые вызовы методов (RMI)

- Продолжение идей RPC
  - Программирование через интерфейсы
  - Работа поверх протоколов “запрос-ответ”
  - At-least-once или at-most-once семантика вызовов
  - Прозрачность синтаксиса вызовов
- Особенности ОО-программ
  - Наследование, полиморфизм
  - Передача параметров по ссылкам
  - Исключения
  - Распределённая сборка мусора

Protocol buffers

protobuf

- Механизм сериализации-десериализации данных
- Компактное бинарное представление
- Декларативное описание формата данных, генерация кода для языка программирования
  - Поддерживается Java, Python, Objective-C, C++, Go, JavaNano, Ruby, C#
- Бывает v2 и v3, с некоторыми синтаксическими отличиями
- Хитрый протокол передачи, https://developers.google.com/protocol-buffers/docs/encoding
  - До 10 раз компактнее XML

Пример

Файл .proto:

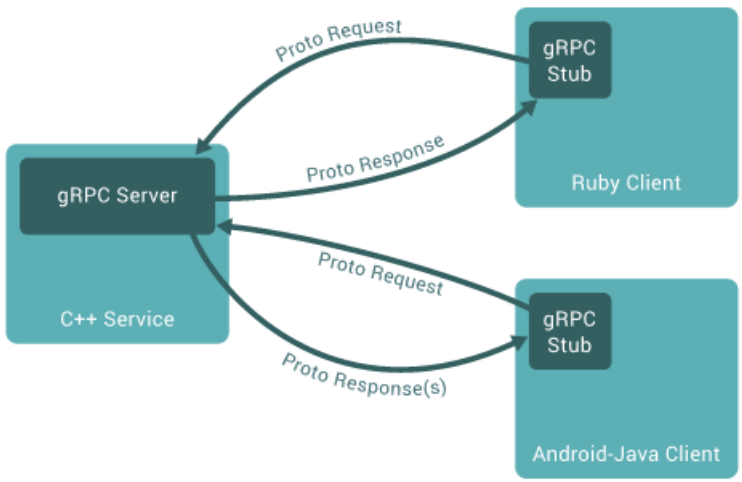
```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
}
```

Файл .java:

```
Person john = Person.newBuilder()
    .setId(1234)
    .setName("John Doe")
    .setEmail("jdoe@example.com")
    .build();
output = new FileOutputStream(args[0]);
john.writeTo(output);
```

gRPC

- средство для удалённого вызова (RPC)
- Работает поверх protobuf
- Разрабатывается Google
- Поддерживает C++, Java, Objective-C, Python, Ruby, Go, C#, Node.js



Технические подробности

- Сервисы описываются в том же .proto-файле, что и протокол protobuf-a
- В качестве типов параметров и результатов — message-и protobuf-a

```
service RouteGuide {
  rpc GetFeature(Point) returns (Feature) {}
  rpc ListFeatures(Rectangle) returns (stream Feature) {}
  rpc RecordRoute(stream Point) returns (RouteSummary) {}
  rpc RouteChat(stream RouteNote) returns (stream RouteNote) {}
}
```

- Сборка — плагином grpc к protoc

Реализация сервиса на Java

```
private static class RouteGuideService extends RouteGuideGrpc.RouteGuideImplBase {
  ...
  @Override
  public void getFeature(Point request, StreamObserver<Feature> responseObserver) {
   responseObserver.onNext(checkFeature(request));
   responseObserver.onCompleted();
  }

  @Override
  public void listFeatures(Rectangle request, StreamObserver<Feature> responseObserver) {
   for (Feature feature : features) {
     ...
     int lat = feature.getLocation().getLatitude();
     int lon = feature.getLocation().getLongitude();
     if (lon >= left && lon <= right && lat >= bottom && lat <= top) {
       responseObserver.onNext(feature);
     }
   }
   responseObserver.onCompleted();
  }
```

Реализация сервиса на Java (2)

```
@Override
public StreamObserver<RouteNote> routeChat(
      final StreamObserver<RouteNote> responseObserver) {
 return new StreamObserver<RouteNote>() {
  @Override
  public void onNext(RouteNote note) {
   List<RouteNote> notes = getOrCreateNotes(note.getLocation());
   for (RouteNote prevNote : notes.toArray(new RouteNote[0])) {
    responseObserver.onNext(prevNote);
   }
   notes.add(note);
  }
  @Override
  public void onError(Throwable t) {
   logger.log(Level.WARNING, "routeChat cancelled");
  }
  @Override
  public void onCompleted() {
   responseObserver.onCompleted();
  }
 };
}
```

Реализация клиента на Java (1)

```
public RouteGuideClient(String host, int port) {
  this(ManagedChannelBuilder.forAddress(host, port).usePlaintext(true));
}

public RouteGuideClient(ManagedChannelBuilder<?> channelBuilder) {
  channel = channelBuilder.build();
  blockingStub = RouteGuideGrpc.newBlockingStub(channel);
  asyncStub = RouteGuideGrpc.newStub(channel);
}
```

Реализация клиента на Java (2)

```
public void getFeature(int lat, int lon) {
  Point request = Point.newBuilder().setLatitude(lat).setLongitude(lon).build();
  Feature feature;
  try {
    feature = blockingStub.getFeature(request);
  } catch (StatusRuntimeException e) {
    warning("RPC failed: {0}", e.getStatus());
    return;
  }
  if (RouteGuideUtil.exists(feature)) {
    info("Found feature called \"{0}\" at {1}, {2}",
      feature.getName(),
      RouteGuideUtil.getLatitude(feature.getLocation()),
      RouteGuideUtil.getLongitude(feature.getLocation()));
  } else {
    info("Found no feature at {0}, {1}",
      RouteGuideUtil.getLatitude(feature.getLocation()),
      RouteGuideUtil.getLongitude(feature.getLocation()));
  }
}
```

Веб-сервисы

- Перенос специализации клиент-сервера в web
- Сложные приложения как интеграция веб-сервисов
- HTTP-запрос для выполнения команды
  - Асинхронное взаимодействие
  - Ответ-запрос
  - Событийные схемы
- XML или JSON как основной формат сообщений
  - SOAP/WSDL/UDDI
  - XML-RPC
  - REST

SOAP-ориентированные сервисы

- Simple Object Access Protocol
- Web Services Description Language
- Universal Discovery, Description and Integration

SOAP-сообщение

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
      <n:priority>1</n:priority>
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>
    </n:alertcontrol>
  </env:Header>
  <env:Body>
    <m:alert xmlns:m="http://example.org/alert">
      <m:msg>Get up at 6:30 AM</m:msg>
    </m:alert>
  </env:Body>
</env:Envelope>
```

WSDL-описание

```
<message name="getTermRequest">
   <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
   <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
   <operation name="getTerm">
      <input message="getTermRequest"/>
      <output message="getTermResponse"/>
   </operation>
</portType>
```

Достоинства SOAP-based сервисов

- Автоматический режим описания сервисов
- Автоматическая поддержка описаний SOAP-клиентом
- Автоматическая валидация сообщений
  - Валидность xml
  - Проверка по схеме
  - Проверка SOAP-сервером
- Работа через HTTP
  - Хоть через обычный GET

Пример: WCF

- Платформа для создания веб-сервисов
- Часть .NET Framework, начиная с 3.0
- Умеет WSDL, SOAP и т.д., очень конфигурируема
- Автоматическая генерация заглушек на стороне клиента
- ABCs of WCF:
  - Address
  - Binding
  - Contract

© http://www.c-sharpcorner.com

Пример, описание контракта

```
[ServiceContract(Namespace = "http://Microsoft.ServiceModel.Samples")]
public interface ICalculator
{
    [OperationContract]
    double Add(double n1, double n2);

    [OperationContract]
    double Subtract(double n1, double n2);

    [OperationContract]
    double Multiply(double n1, double n2);

    [OperationContract]
    double Divide(double n1, double n2);
}
```

Пример, реализация контракта

```
public class CalculatorService : ICalculator
{
    public double Add(double n1, double n2)
        => n1 + n2;

    public double Subtract(double n1, double n2)
        => n1 - n2

    public double Multiply(double n1, double n2)
        => n1 * n2;

    public double Divide(double n1, double n2)
        => n1 / n2;
}
```

Пример, self-hosted service

```
static void Main(string[] args)
{
    Uri baseAddress = new Uri("http://localhost:8000/ServiceModelSamples/Service");
    ServiceHost selfHost = new ServiceHost(typeof(CalculatorService), baseAddress);

    try {
        selfHost.AddServiceEndpoint(typeof(ICalculator), new WSHttpBinding(), "CalculatorService");

        ServiceMetadataBehavior smb = new ServiceMetadataBehavior();
        smb.HttpGetEnabled = true;
        selfHost.Description.Behaviors.Add(smb);

        selfHost.Open();
        Console.WriteLine("The service is ready. Press <ENTER> to terminate service.");
        Console.ReadLine();

        selfHost.Close();
    } catch (CommunicationException ce) {
        Console.WriteLine($"An exception occurred: {ce.Message}");
        selfHost.Abort();
    }
}
```

Пример, клиент

- Генерация заглушки:

  svcutil.exe /language:cs /out:generatedProxy.cs /config:app.config^
  http://localhost:8000/ServiceModelSamples/service

- Клиент:

```
static void Main(string[] args)
{
    CalculatorClient client = new CalculatorClient();

    double value1 = 100.00D;
    double value2 = 15.99D;
    double result = client.Add(value1, value2);
    Console.WriteLine($"Add({value1},{value2}) = {result}");

    client.Close();
}
```

Пример, конфигурация клиента

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <!-- specifies the version of WCF to use-->
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5,Profile=Client" />
  </startup>
  <system.serviceModel>
    <bindings>
      <!-- Uses wsHttpBinding-->
      <wsHttpBinding>
        <binding name="WSHttpBinding_ICalculator" />
      </wsHttpBinding>
    </bindings>
    <client>
      <!-- specifies the endpoint to use when calling the service -->
      <endpoint address="http://localhost:8000/ServiceModelSamples/Service/CalculatorService"
        binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_ICalculator"
        contract="ServiceReference1.ICalculator" name="WSHttpBinding_ICalculator">
        <identity>
          <userPrincipalName value="migree@redmond.corp.microsoft.com" />
        </identity>
      </endpoint>
    </client>
  </system.serviceModel>
</configuration>
```

Очереди сообщений

- Используются для гарантированной доставки сообщений
  - Даже если отправитель и получатель доступны в разное время
  - Локальное хранилище сообщений на каждом устройстве
- Реализуют модель “издатель-подписчик”, но могут работать и в режиме “точка-точка”
- Как правило, имеют развитые возможности маршрутизации, фильтрации и преобразования сообщений
  - Разветвители, агрегаторы, преобразователи порядка

RabbitMQ

- Сервер и клиенты системы надёжной передачи сообщений
  - Сообщение посылается на сервер и хранится там, пока его не заберут
  - Продвинутые возможности по маршрутизации сообщений
- Реализует протокол AMQP (Advanced Message Queuing Protocol), но может использовать и другие протоколы
- Сервер написан на Erlang, клиентские библиотеки доступны для практически чего угодно

Пример, отправитель

```
using System;
using RabbitMQ.Client;
using System.Text;

class Send
{
    public static void Main()
    {
        var factory = new ConnectionFactory() { HostName = "localhost" };
        using (var connection = factory.CreateConnection())
        {
            using (var channel = connection.CreateModel())
            {
                channel.QueueDeclare(queue: "hello", durable: false, exclusive: false,
                                     autoDelete: false, arguments: null);

                string message = "Hello World!";
                var body = Encoding.UTF8.GetBytes(message);

                channel.BasicPublish(exchange: "", routingKey: "hello",
                                     basicProperties: null, body: body);
            }
        }
    }
}
```

Пример, получатель

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Text;

class Receive
{
   public static void Main()
   {
      var factory = new ConnectionFactory() { HostName = "localhost" };
      using (var connection = factory.CreateConnection())
      using (var channel = connection.CreateModel())
      {
         channel.QueueDeclare(queue: "hello", durable: false, exclusive: false, autoDelete: false, arguments: null);

         var consumer = new EventingBasicConsumer(channel);
         consumer.Received += (model, ea) =>
         {
            var body = ea.Body;
            var message = Encoding.UTF8.GetString(body);
            Console.WriteLine(" [x] Received {0}", message);
         };
         channel.BasicConsume(queue: "hello", autoAck: true, consumer: consumer);
      }
   }
}
```

Representational State Transfer (REST)

- Модель клиент-сервер
- Отсутствие состояния
- Кэширование
- Единообразие интерфейса
- Слои

Интерфейс сервиса

- Коллекции
  - http://api.example.com/resources/
- Элементы
  - http://api.example.com/resources/item/17
- HTTP-методы
  - GET
  - PUT
  - POST
  - DELETE
- Передача параметров прямо в URL
  - http://api.example.com/resources?user=me&access_token=ASFQF

Пример, Google Drive REST API

- GET https://www.googleapis.com/drive/v2/files — список всех файлов
- GET https://www.googleapis.com/drive/v2/files/fileId — метаданные файла по его Id
- POST https://www.googleapis.com/upload/drive/v2/files — загрузить новый файл
- PUT https://www.googleapis.com/upload/drive/v2/files/fileId — обновить файл
- DELETE https://www.googleapis.com/drive/v2/files/fileId — удалить файл

Достоинства

- Надёжность
- Производительность
- Масштабируемость
- Прозрачность системы взаимодействия
- Простота интерфейсов
- Портативность компонентов
- Лёгкость внесения изменений

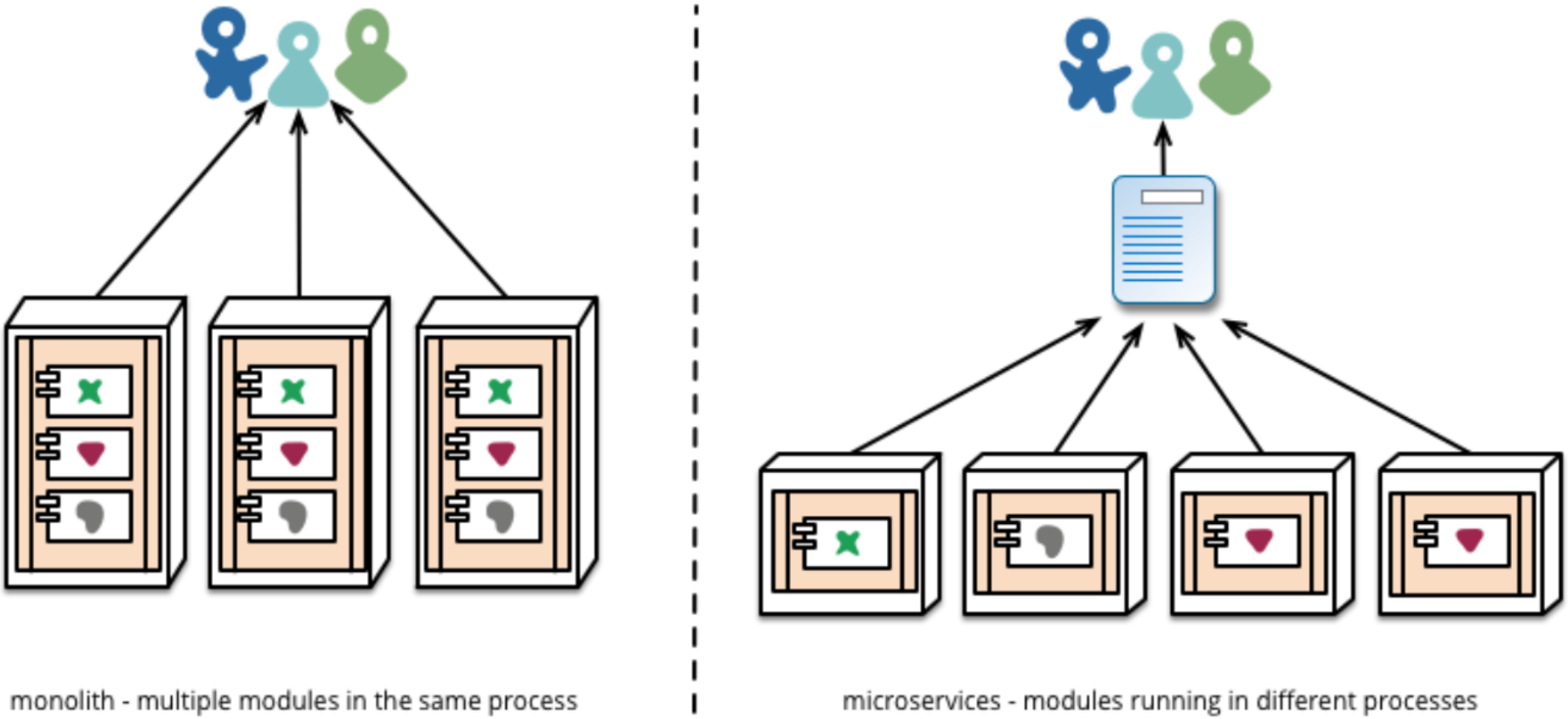
Микросервисы

- Набор небольших сервисов
  - Разные языки и технологии
- Каждый в собственном процессе
  - Независимое развёртывание
  - Децентрализованное управление
- Легковесные коммуникации

Монолитные приложения

- Большой и сложный MVC
- Единый процесс разработки и стек технологий
- Сложная архитектура
- Сложно масштабировать
- Сложно вносить изменения

Разбиение на сервисы



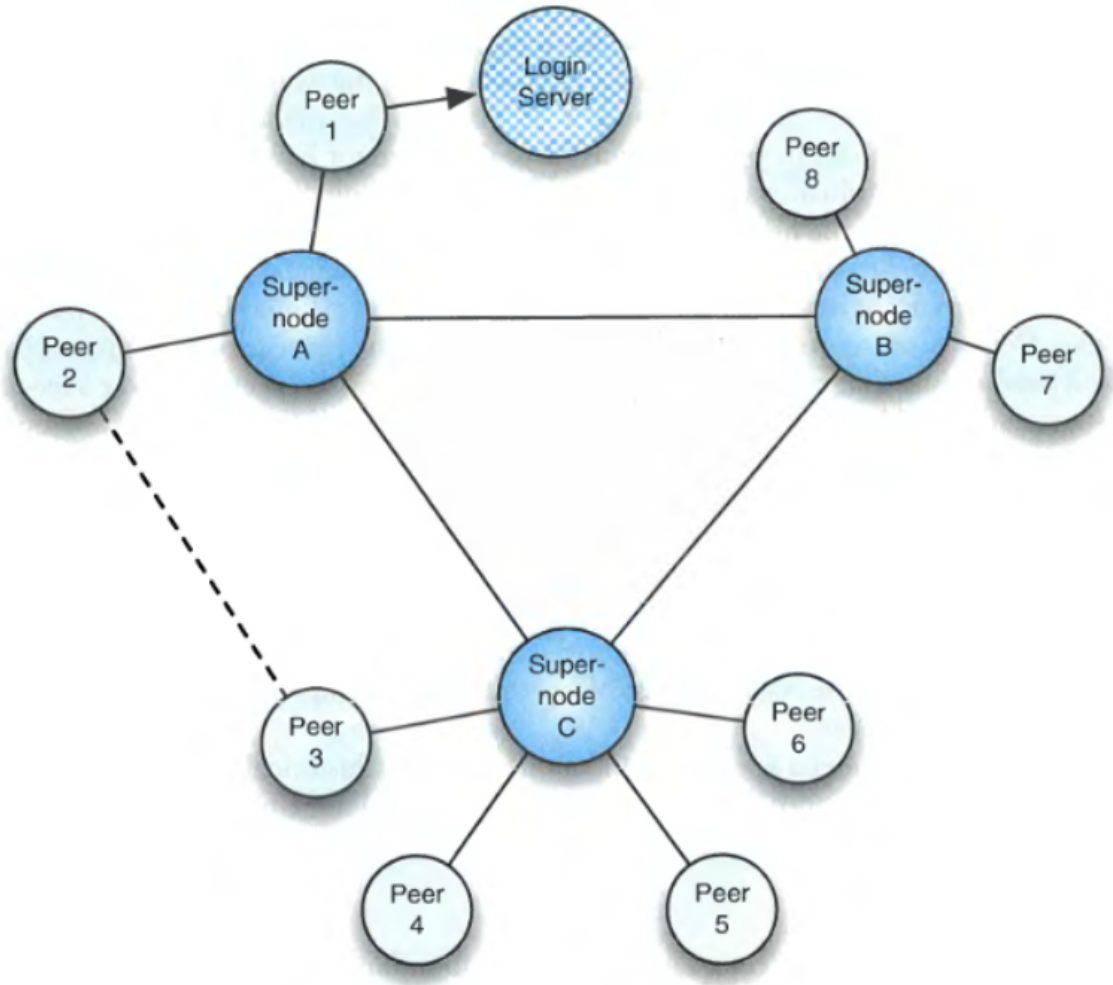
Основные проблемы

- Сложности выделения границ сервисов
- Перенос логики на связи между сервисами
  - Большой обмен данными
  - Нетривиальные зависимости
- Нетривиальная инфраструктура
- Нетривиальная переиспользуемость кода

Архитектура Peer-to-Peer

- Децентрализованный и самоорганизующийся сервис
- Динамическая балансировка нагрузки
  - Вычислительные ресурсы
  - Хранилища данных
- Динамическое изменение состава участников

Skype: Overlayed P2P



BitTorrent : Resource Trading P2P

- Обмен сегментами
- Поиск не входит в протокол
- Трекеры
- Метаданные
- Управление приоритетами
- Бестрекерная реализация