Лекция 9: Проектирование распределённых приложений

Часть первая: технические вопросы

Юрий Литвинов
y.litvinov@spbu.ru

12.05.2026

Распределённые системы

- Компоненты приложения находятся в компьютерной сети
- Взаимодействуют через обмен сообщениями
- Особенности
  - Параллельная работа
  - Независимые отказы

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

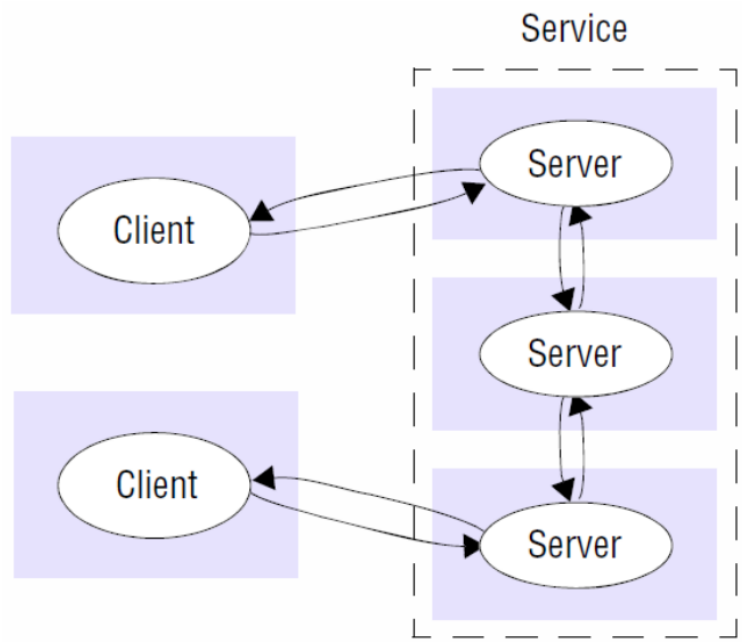
© https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

Виды взаимодействия

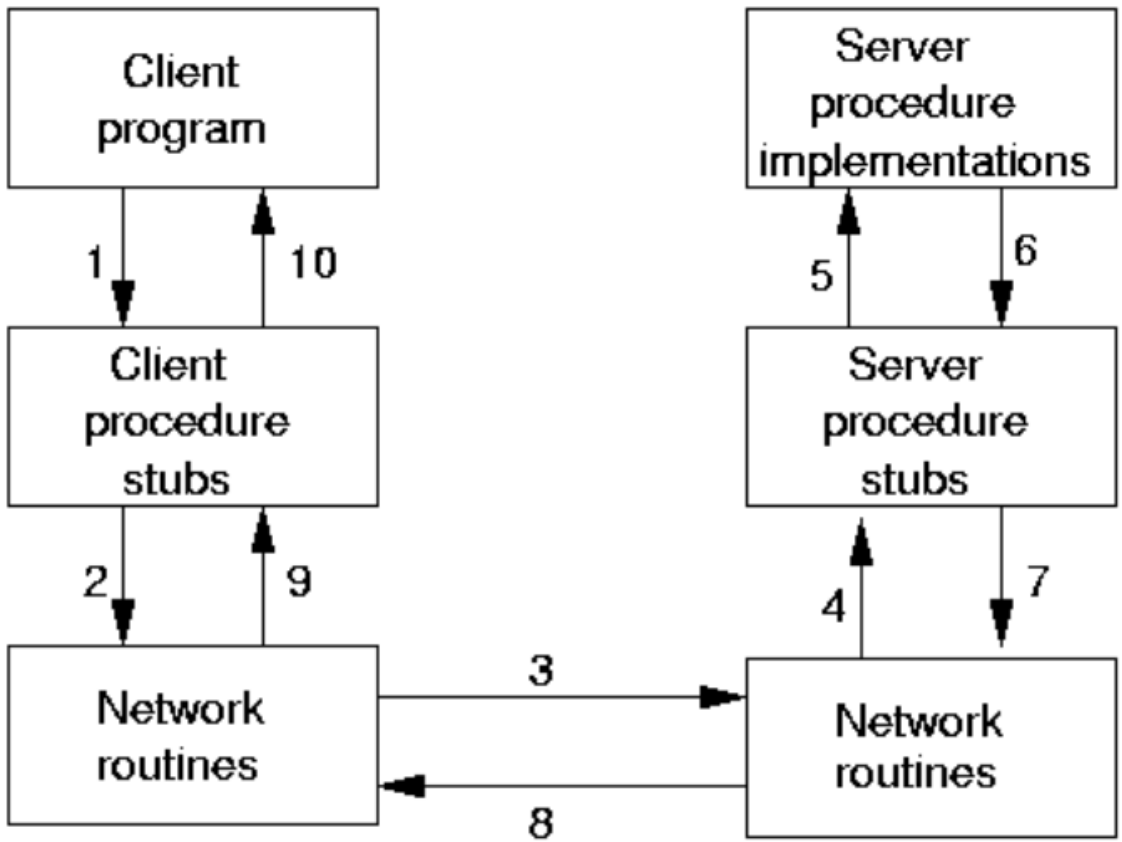
- Межпроцессное взаимодействие
- Удалённые вызовы
  - Протоколы вида “запрос-ответ”
  - Удалённые вызовы процедур (remote procedure calls, RPC)
  - Удалённые вызовы методов (remote method invocation, RMI)
- Неявное взаимодействие
  - Распределённая общая память
  - Очереди сообщений
  - Модель “издатель-подписчик”

Варианты размещения

- Разбиение сервисов по нескольким серверам
- Мобильный код
- Мобильный агент
- Кеширование



RPC

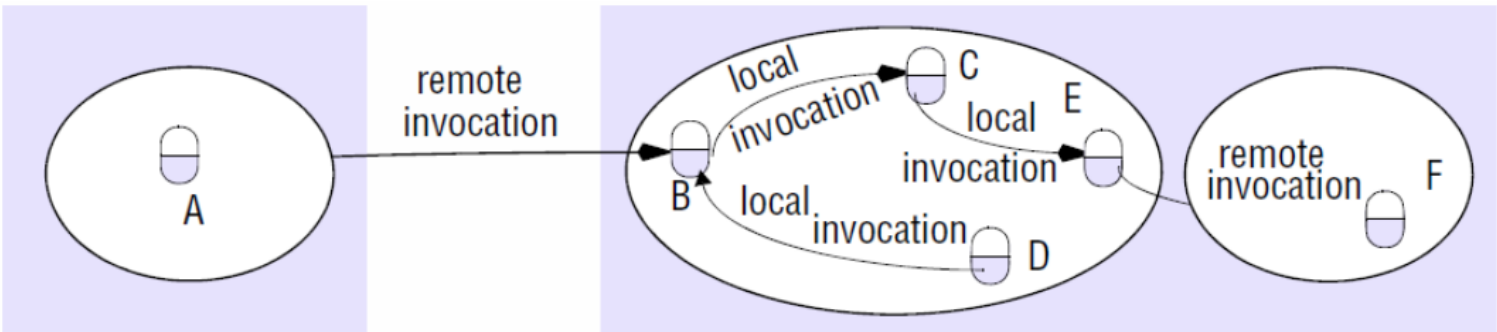

Client program
Server procedure implementations
1
10
5
6
Client procedure stubs
Server procedure stubs
2
9
4
7
3
Network routines
Network routines
8

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 семантики вызова

RMI

- Локальные и удалённые объекты
- Интерфейсы удалённых объектов
- Ссылки на удалённые объекты
  - Как параметры или результаты удалённых вызовов
- Умеют исключения



Protocol buffers

protobuf

- Механизм сериализации-десериализации данных
- Компактное бинарное представление
- Декларативное описание формата данных, генерация кода для языка программирования
  - Поддерживается Java, Python, Kotlin, Objective-C, C++, Go, Ruby, C#, Dart
- 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:

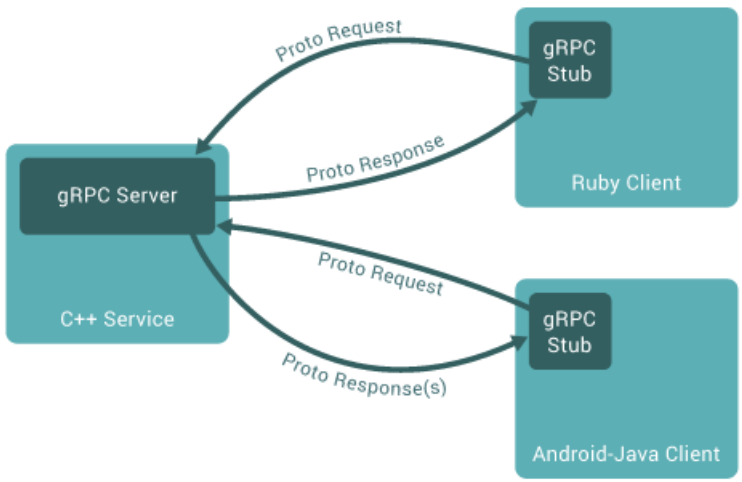
```
syntax = "proto3";

message Person {
    string name = 1;
    int32 id = 2;
    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
- Тоже от Google, поддерживает те же языки, что и protobuf
- Весьма популярен



Технические подробности

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

```
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) {}
}
```

© https://grpc.io/docs/languages/java/basics/

- 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

- Каждый веб-сервис — отдельная система, представляющая что-то вроде RPC/RMI интерфейса
- 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

Недостатки SOAP-based сервисов

- Огромный размер сообщений
- Сложность описаний на клиенте и сервере
- Один запрос — один ответ
  - Поддержка транзакций на уровне бизнес-логики
- Сложности миграции при изменении описания

Windows Communication Foundation

- Платформа для создания веб-сервисов
- Часть .NET Framework, начиная с 3.0
  - Сейчас замещается ASP.NET Web APIs
- 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

```
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:

```
var 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>
```

Representational State Transfer (REST)

- Самая популярная сейчас архитектура веб-сервисов
- Передача всего необходимого в запросе
  - Нельзя хранить состояние сессии
- Стандартизованный интерфейс, очень простые запросы
- Стандартные протоколы (в основном поверх HTTP)
- Обычно JSON как формат сериализации
- Кеширование

Интерфейс сервиса

- Коллекции
  - http://api.example.com/customers/
- Элементы
  - http://api.example.com/customers/17
- HTTP-методы (GET, POST, PUT, DELETE), стандартная семантика, стандартные коды ошибок
- Передача параметров прямо в URL
  - http://api.example.com/customers?user=me&access_token=ASFQF

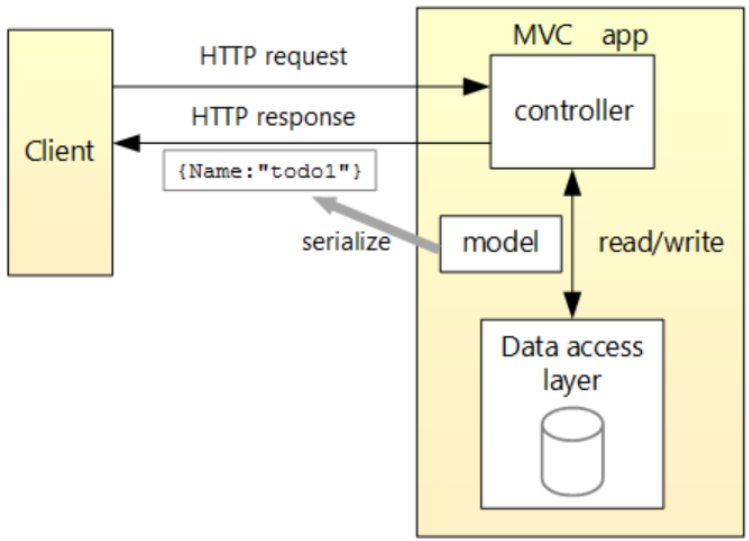
Пример, Google Drive REST API

- 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

ASP.NET

- Вообще — серверный фреймворк для разработки веб-приложений под .NET
- Появился в 2002 году (вместе с самим .NET), переписан под .NET Core в 2016
- Кроссплатформенный, open-source, лицензия MIT
- Имеет механизм быстрого создания REST-сервисов

ASP.NET Web APIs



© https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api

Пример

Полный код сервиса!

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

var summaries = new[] {
    "Freezing", "Bracing", "Chilly", "Cool", "Mild",
    "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", () => {
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateTime.Now.AddDays(index),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        )).ToArray();
    return forecast;
});

app.Run();

internal record WeatherForecast(DateTime Date,
        int TemperatureC, string? Summary) {
    public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
}
```

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;

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);
```

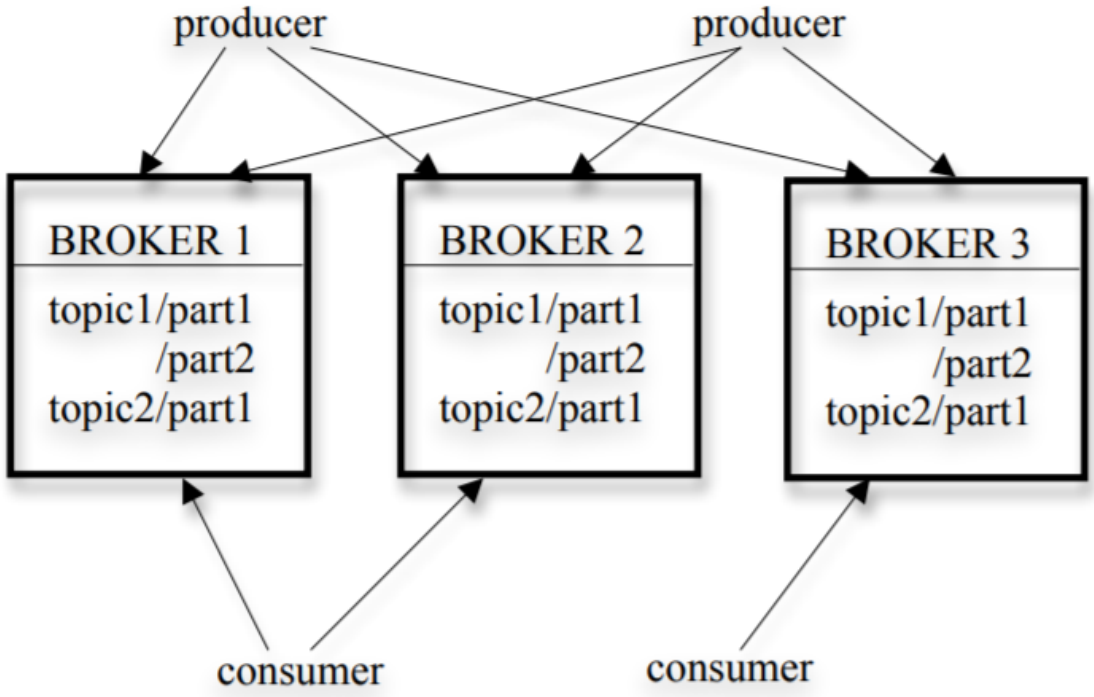
Пример, получатель

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

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);
```

Apache Kafka

- Несколько другой подход к очередям: лог событий
  - Сообщение посылается на сервер и хранится там вечно (ну, почти), получатель при обработке его не удаляет
  - Индекс текущего сообщения хранит сам получатель, может отмотать назад
  - Подход “Event Sourcing” — не храним состояние, храним набор событий, позволяющих его получить
  - Гораздо лучше с распределённостью
- Быстрее RabbitMQ, лучше масштабируется
- Хуже с маршрутизацией (по идее), немного сложнее в настройке

Apache Kafka, устройство



© J. Kreps et al., Kafka: a Distributed Messaging System for Log Processing, 2011

- Топики — каналы, куда можно писать
- Разделы — логические куски топиков
- Брокеры — отдельные сервера, балансируют нагрузку
- Сегменты — файлы на диске, куски разделов, хранящиеся у брокеров

Пример, отправитель

```
using Confluent.Kafka;

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

using var p = new ProducerBuilder<Null, string>(config).Build();
try
{
    var deliveryResult = await p.ProduceAsync(
        "test-topic", new Message<Null, string> { Value = "test" });
}
catch (ProduceException<Null, string> e)
{
    Console.WriteLine($"Delivery failed: {e.Error.Reason}");
}
```

Пример, получатель

```
using Confluent.Kafka;

var conf = new ConsumerConfig {
    GroupId = "test-consumer-group",
    BootstrapServers = "localhost:9092",
    AutoOffsetReset = AutoOffsetReset.Earliest
};

using var consumer = new ConsumerBuilder<Ignore, string>(conf).Build();
consumer.Subscribe("my-topic");

var cts = new CancellationTokenSource();
Console.CancelKeyPress += (_, e) => {
    e.Cancel = true;
    cts.Cancel();
};

try {
    while (true) {
        var consumeResult = consumer.Consume(cts.Token);
        Console.WriteLine($"Consumed message '{consumeResult.Message.Value}'.");
    }
}
catch (OperationCanceledException) {
    consumer.Close();
}
```