

# Лекция 13: Проектирование распределённых приложений

Часть первая: транспортные вопросы

Юрий Литвинов  
yurii.litvinov@gmail.com

07.05.2019г

# 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

# Архитектура распределённых систем

- ▶ Какие сущности взаимодействуют между собой в распределённой системе?
- ▶ Как они взаимодействуют?
- ▶ Какие (возможно изменяющиеся) роли и ответственности имеют эти сущности в рамках всей архитектуры?
- ▶ Как они размещаются на физическую инфраструктуру?

# Виды сущностей

- ▶ Узлы-процессы-потоки — сущности уровня ОС (или сами вычислительные узлы, если ОС не поддерживает даже процессы)
- ▶ Объекты — обычные объекты из ООП, с интерфейсами, описанными на IDL, вызывающие друг друга по сети
- ▶ Компоненты — более высокоуровневые сущности, как правило, предполагают middleware
- ▶ Веб-сервисы — ещё более высокоуровневые сущности, независимые приложения с чётко определённым способом их использовать

# 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

# Роли и обязанности

- ▶ Клиент-сервер
- ▶ Peer-to-peer

## Варианты размещения

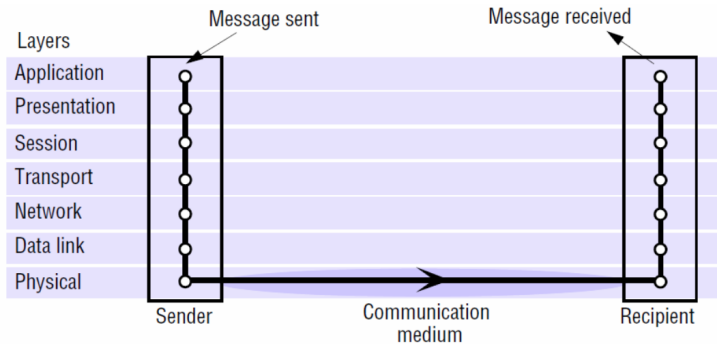
- ▶ Разбиение сервисов по нескольким серверам
- ▶ Кэширование
- ▶ Мобильный код
- ▶ Мобильный агент



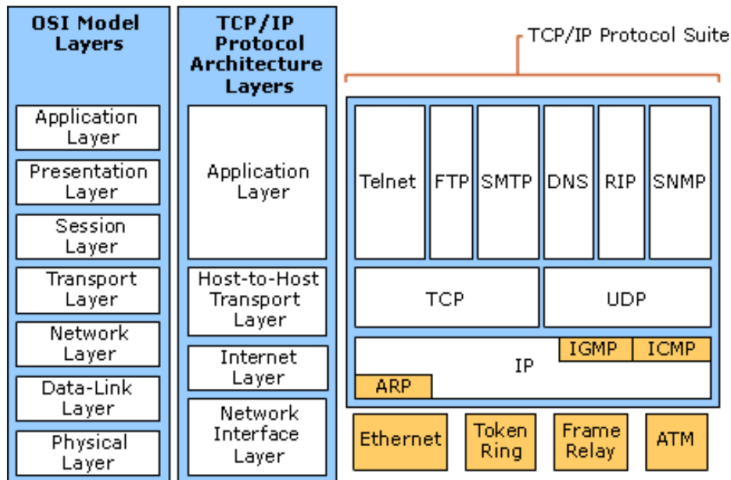
# Типичные архитектурные стили

- ▶ 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

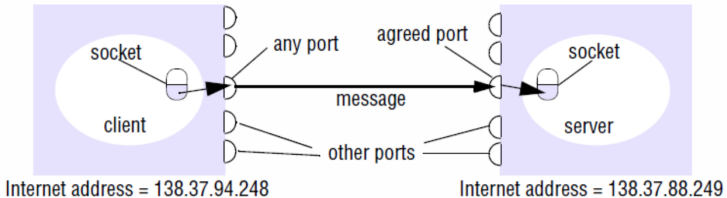
# Модель OSI



# Стек протоколов TCP/IP

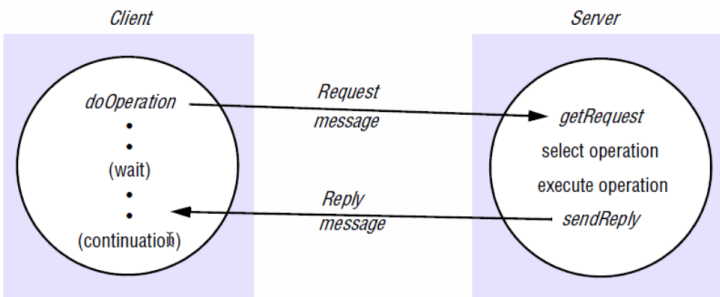


# Абстракция сокета



# 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

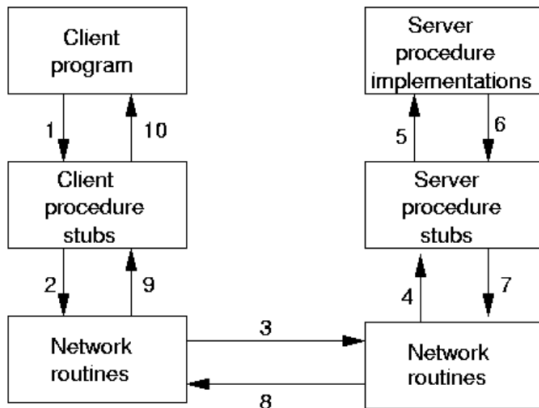
- + Использование потоков вместо набора пакетов
  - ▶ Удобная отправка больших объёмов данных
  - ▶ Один поток на всё взаимодействие
- + Интеграция с потоками ОО-языков
- + Надёжность доставки
  - ▶ Отсутствие необходимости проверок на уровне бизнес-логики
  - ▶ Уведомления в пакетах с ответом
  - ▶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

# HTTP

- ▶ Пример протокола “запрос-ответ”
- ▶ Реализован поверх TCP
- ▶ Соединение на всё время взаимодействия
- ▶ Маршалинг данных в ASCII
  - ▶ MIME
- ▶ HTTP 2.0
  - ▶ Бинарный протокол
  - ▶ Обязательное шифрование
  - ▶ Мультиплексирование запросов в одном TCP соединении
  - ▶ “Предсказывающая посылка данных”



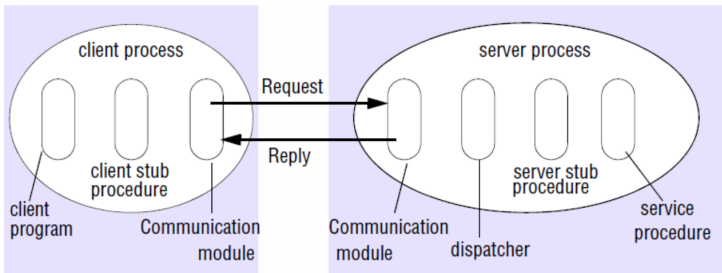
# RPC



# 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

# Структура RPC middleware

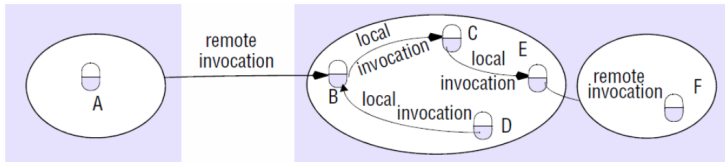


# Удалённые вызовы методов (RMI)

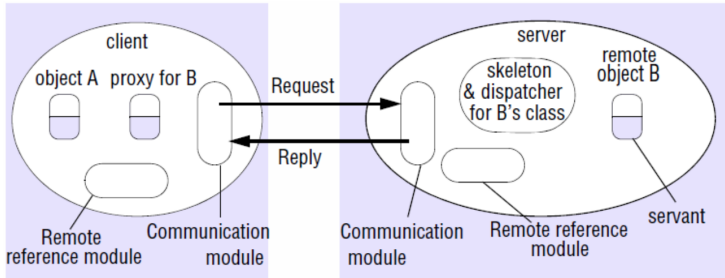
- ▶ Продолжение идей RPC
  - ▶ Программирование через интерфейсы
  - ▶ Работа поверх протоколов “запрос-ответ”
  - ▶ At-least-once или at-most-once семантика вызовов
  - ▶ Прозрачность синтаксиса вызовов
- ▶ Особенности ОО-программ
  - ▶ Наследование, полиморфизм
  - ▶ Передача параметров по ссылкам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

# Локальные и удалённые вызовы

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



# Структура RMI middleware



# 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;  
}
```

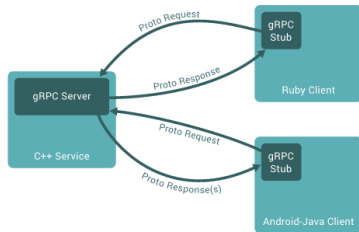
Файл .java:

```
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