Эффективное взаимодействие сервисов: язык Protobuf и протокол GRPC

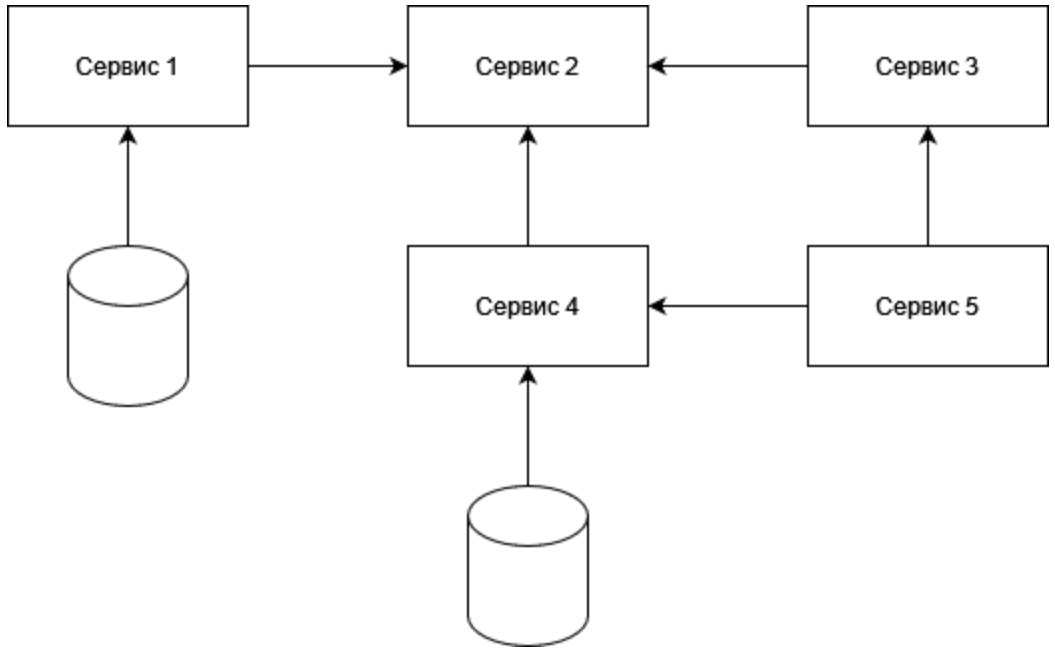
Шаниязов Ростислав

МИЭМ ВШЭ

Для кого доклад?

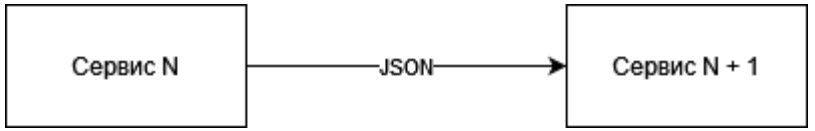
- Разработчики серверных приложений
- Что-то слышали о GRPC

Модель взаимодействия



Модель взаимодействия Сервис-Клиент

```
public class SimpleModel {
    private int id;
    private String client;
}
```

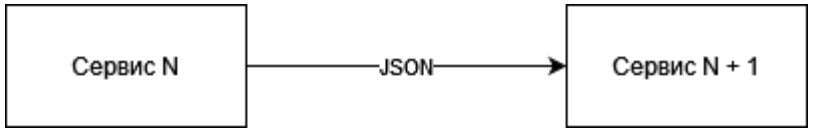



```
public class SimpleModel {
    private int id;
    private String client;
}
```

public class SimpleModel { private int id; private String client; }

Модель взаимодействия Сервис-Клиент (сломано)

```
public class SimpleModel {
    private int id;
    private String client;
}
```




```
public class SimpleModel {
    private int id;
    private Client client;
}
```

```
public class Client {
    private long id;
    private String name;
    private List<String> attributes;
}
```

Как это можно решить?

- Жёсткая фиксация API:
  - Нельзя изменять поля в модели
  - Жёсткое ревью на Merge Request
- Использовать инструменты языка Protobuf

Protobuf введение

- Язык описания API (как openapi/swagger).
- Имеет строгую типизацию.
- Позволяет генерировать контроллеры и клиенты на на различных языках (C++, Python, Go, Java)

1. Define proto

```
import "person_info.proto";

package persons;

message Person {
    PersonInfo info = 1; // characteristics of the person
    repeated Friend friends = 2; // friends of the person
}
```

2. Compile proto

```
_PERSON = _descriptor.Descriptor(
  name='Person',
  full_name='persons.Person',
  filename=None,
  file=DESCRIPTOR,
  containing_type=None,
  create_key=_descriptor._internal_create_key,
  fields=[
...
```

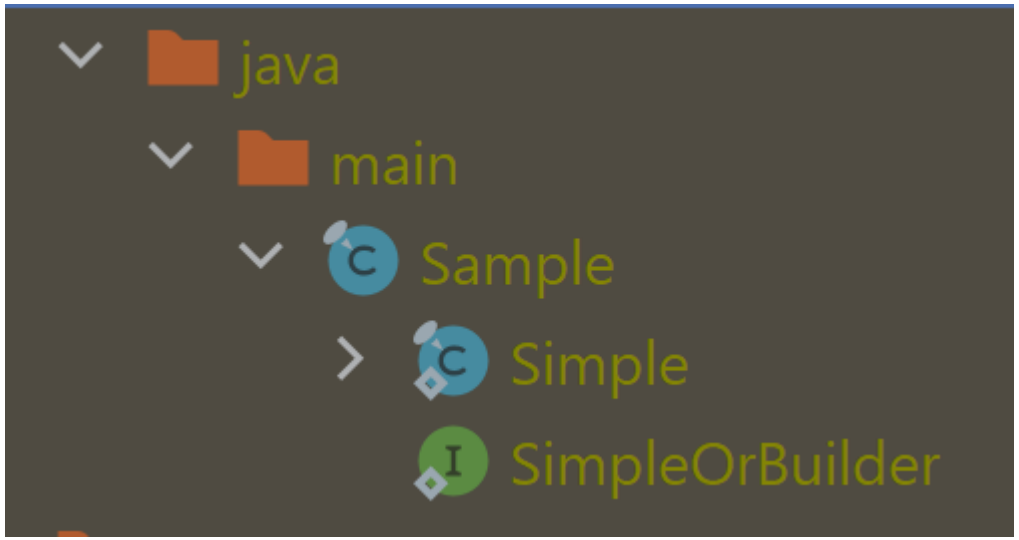
3. Serialize

```
info {
  age: 30
  height: 184
}
friends {
  friendship_duration: 365.1000061035156
  shared_hobbies: "books"
  shared_hobbies: "daydreaming"
  shared_hobbies: "unicorns"
  person {
    info {
      age: 40
      height: 165
    }
  }
}
```

Как работать с protobuf?

```
message Simple {
    string query = 1;
    int32 page_number = 2;
    int32 result_per_page = 3;
}
```

1. Определить protobuf



2. Скомпилируйте код

```
Simple sample = Simple.newBuilder()
    .setPageNumber(10)
    .setQuery("lol")
    .build();
```

3. Используйте

Проверка обратной совместимости (protolock)

- *protolock status*
- *protolock commit*

> protolock status
CONFLICT: "Simple" ID: "3" has been removed, but is not reserved [sample.proto]
CONFLICT: "Simple" field: "result_per_page" has been removed, but is not reserved [sample.proto]

Пример *protolock status*, когда удалили поле в модели

```
message Simple {
    string query = 1;
    int32 page_number = 2;
}
```

Как сохранять обратную совместимость? (reserved)

- Используется только для протокола GRPC

```
message Simple {
  reserved 3;
  reserved "result_per_page";
  string query = 1;
  int32 page_number = 2;
}
```

Как использовать линтер?

- Единый стиль для всего API

```
message Simple {
    string query = 1;
    int32 page_number = 2;
    int32 result_per_page = 3;
    string someString = 4;
}
```

> protolint lint .\sample.proto
[sample.proto:7:3] Field name "someString" must be underscore_separated_names like "some_string"

Пример линтера с ошибками

Protocol GRPC

- Надстройка над HTTP 2.0
- Использует длинные запросы
- Кроссплатформенный

https://medium.com/@onufrienkos/benchmarking-performance-grpc-vs-rest-on-node-js-bf766127f170

Плюсы GRPC

- Строгая типизация
- Производительность (?)
- Компиляция под все популярные и средне-популярныне языки
- Потоковая передача данных

Минусы GRPC

- Мало интеграций с существующими библиотеками
- Данные не в человоекочитаемом формате (bloomRpc)
- Длинные запросы
- Без service mesh вы не обойдётесь (Istio).

Спасибо за внимание