Протоколы обмена информацией

Протоколы сериализации

Протокол передачи данных — набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии программного обеспечения разнесённой в пространстве аппаратуры, соединённой тем или иным интерфейсом.

Все протоколы можно разделить на два типа:

Символьные

- **JSON** (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве языка программирования JavaScript. JSON-текст представляет собой пару ключ – значение. В качестве значения может выступать свтрока, число, true/false, массив, состоавной объект. `json { "name": "Alex", "login": "ronin", "work": "Innotech", "address": { "city": "Moscow", "street": "Molostovih" }, "hobbies": [ "MTB", "Ski Freeride" ] }`
- **XML** (eXtensible Markup Language) – разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программам и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете, но у них не получилось :(Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка. Расширение XML – это конкретная грамматика, созданная на базе XML и представленная словарем тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов. Все составляющие части документа обобщаются в пролог и корневой элемент. Корневой элемент — обязательная часть документа, составляющая всю его суть (пролог, вообще говоря, может отсутствовать). Может включать (а может не включать) вложенные в него элементы и символьные данные, а также комментарии. Вложенные в корневой элемент элементы, в свою очередь, могут включать вложенные в них элементы, символьные данные и комментарии, и так далее. Пролог может включать объявления, инструкции обработки, комментарии. Его следует начинать с объявления XML, хотя в определённой ситуации допускается отсутствие этого объявления. Элементы документа должны быть правильно вложены: любой элемент, начинающийся внутри другого элемента (то есть любой элемент документа, кроме корневого), должен заканчиваться внутри элемента, в котором он начался. Символьные данные могут встречаться внутри элементов как непосредственно так и в специальных секциях «CDATA». Объявления, инструкции обработки и элементы могут иметь связанные с ними атрибуты. Атрибуты используются для связывания с логической единицей текста пар имя-значение. `xml <?xml version="1.0" encoding="UTF-8"?> <user> <login>ronin</login> <name>Alex</name> <work>Innotech</work> <address> <city>Moscow</city> <street>Molostovih</street> </address> <hobbies> <element>MTB</element> <element>Ski Freeride</element> </hobbies> </user>`
- **YAML** (Yet Another Markup Language) Формат сериализации данных, концептуально близкий к языкам разметки, но ориентированный на удобство ввода-вывода структур данных языков программирования. YAML с версии 1.2 — это надмножество JSON. То есть, все, что правильно для JSON, годится и для YAML. Иногда это можно использовать для улучшения читаемости. Формат YAML:
  - отступы из пробелов (символы табуляции не допускаются) используются для обозначения структуры;
  - комментарии начинаются с символа # и могут начинаться в любом месте строки и продолжаются до конца строки;
  - списки обозначаются начальным дефисом с одним членом списка на строку, либо члены списка заключаются в квадратные скобки и разделяются запятой и пробелом;
  - массивы представлены двоеточием с пробелом в виде ключ: значение, по одной паре ключ-значение на строку, либо в виде пар, заключённых в фигурные скобки и разделенных запятой и пробелом;

- ключ в ассоциативном массиве может иметь в качестве префикса вопросительный знак (?), что позволяет указать сложный ключ, например представленный в виде списка;
- YAML позволяет задавать подстановки с помощью якорей & и алиасов (*).`yaml vms: region: Europe domains: ['ru', 'com'] _basic: &basic cpu: 2 ram: 2 disk: 10 profiles: small: {<<: *basic, cpu: 1} large: {<<: *basic, cpu: 4} default_attr1: &reference "Basic configuration" default_attr2: *reference`

Бинарные

JSON является широко распространённым и популярным форматом для обмена данными. Его изящность, простота обработки и относительно богатая система типов стали естественным выбором для многих разработчиков, которым необходимо быстро и просто сохранять или передавать данные между системами. В современных реализациях скорость конвертации из/в json очень высокая, тем не менее, json все равно содержит ряд избыточных данных, например повторяющиеся имена полей в массисе объектов. При передаче больших объемов данных (> 5Mb) это может быть проблемой, поэтому в таких ситуациях стоит рассматривать бинарные протоколы.

Бинарные протоколы делятся на протоколы со схемой данных и без. В протоколах со схемой данных, эта схема так же хранится на получателе и входящий объект валидируется и разбирается с ее помощью.

Протоколы без схемы данных:

- **BSON** (Binary JavaScript Object Notation) – объекты состоят из сортированных списков элементов. Каждый элемент состоит из имени поля, типа и значения. Имена полей — это строки. Типы включают: string, int, byte[], bool, null, DateTime, BsonObject, BsonObject[]. В сравнении с JSON, BSON спроектирован и для эффективного в отношении занимающего дискового пространства хранения данных, и скорости сканирования. Большие элементы в документе BSON имеют префикс с длиной документа для облегчения сканирования. BSON во многом аналогичен Protocol Buffers, но BSON является более свободным от схемы данных. Тем самым, бóльшая гибкость BSON уменьшает преимущества в производительности в случае, когда схема определена.
- **MessagePack** - бинарный формат сериализации, он позволяет передавать данные между различными языками, но в отличие от JSON объем передаваемых данных меньше на 15-20%.

Протоколы со схемой данных:

- **Protobuf** (Google Protocol Buffer) — язык описания сообщений и данных, предложенный Google, как компактная, быстрая и простая двоичная альтернатива текстовому формату XML. Протокол использует специальный .proto-файл, в котором на псевдокоде описан объект сериализации. После того как файл с нужной структурой данных создан, необходимо скомпилировать его компилятором для вашего языка программирования, чтобы сгенерировать класс доступа к этим данным. Этот класс будет содержать простейшие методы доступа ко всем полям типа get/set, а также методы для сериализации и десериализации вашей структуры данных в/из массива байтов.
- **Apache Avro** – протокол сериализации, основанный на схеме данных, умеет выполнять сериализацию в json или бинарный формат. Так же бинарный формат можно записывать в файл используя при этом компрессию данных. Формат сериализованного сообщения состоит из заголовка с форматом данных и массива самих данных, без дублирования схемы на каждом описании. Схема описывается в формате json, она состоит из:
  - name – имени типа данных;
  - fields – описания полей;
  - alias – альтернативные имена полей;
  - doc – блока содержащего описание элемента схемы.

При использовании бинарных форматов мы сталкиваемся с проблемами:

- Использование внутренних типов данных, исключительно присущих только двоичным форматам и изначально не включённых в стандарт JSON, делает непригодными для

широкого использования вышеуказанные спецификации, так как, в зависимости от реализации, каждый такой тип может интерпретироваться по разному.
- Одни форматы позволяют добиться более высокой производительности, а другие — более компактного представления за счёт более сложной, запутанной спецификации. Что, в свою очередь, замедляет или делает невозможных их распространение и внедрение. Простота JSON позволила создать реализации на различных языках программирования и сделала его удобным и понятным сразу для всех, кто использует ваши данные непосредственно.

Примеры

Serialization Protocols

Литература

1. google/gson
2. Некоторые приемы YAML
3. Protocol Buffers
4. Apache Avro Specification