Распределённые объектные технологии: **Сериализация данных**

Д. А. Усталов

УрФУ и ИММ УрО РАН

22 марта 2016 г.

Содержание

Концептуальный вопрос

- Данные можно передавать по сети при помощи сокетов.

Концептуальный вопрос

- Данные можно передавать по сети при помощи сокетов.
- Как передать структуру данных?

Концептуальный вопрос

- Данные можно передавать по сети при помощи сокетов.
- Как передать структуру данных?
  - Например, массив из двух целых чисел: `{19, 2000}`.

Концептуальный вопрос

- Данные можно передавать по сети при помощи сокетов.
- Как передать структуру данных?
  - Например, массив из двух целых чисел: `{19, 2000}`.
- Наивный подход: копировать данные из памяти в сокет.

Концептуальный вопрос

- Данные можно передавать по сети при помощи сокетов.
- Как передать структуру данных?
  - Например, массив из двух целых чисел: `{19, 2000}`.
- Наивный подход: копировать данные из памяти в сокет.
- Что может пойти не так?

Ответ на вопрос

Всё может пойти не так.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` = `sizeof(long)`, т. е. $4 = 4$.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` $=$ `sizeof(long)`, т. е. $4 = 4$.

  x86_64 `sizeof(int)` $\neq$ `sizeof(long)`, т. е. $4 \neq 8$.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` $=$ `sizeof(long)`, т. е. $4 = 4$.

  x86_64 `sizeof(int)` $\neq$ `sizeof(long)`, т. е. $4 \neq 8$.

- Принимающая сторона может использовать другой порядок байтов.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` $=$ `sizeof(long)`, т. е. $4 = 4$.

  x86_64 `sizeof(int)` $\neq$ `sizeof(long)`, т. е. $4 \neq 8$.

- Принимающая сторона может использовать другой порядок байтов.
- Компиляторы могут выравнивать структуры по-разному.

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` $=$ `sizeof(long)`, т. е. $4 = 4$.

  x86_64 `sizeof(int)` $\neq$ `sizeof(long)`, т. е. $4 \neq 8$.

- Принимающая сторона может использовать другой порядок байтов.
- Компиляторы могут выравнивать структуры по-разному.
- Что произойдёт, когда формат станет *сложным*?

Ответ на вопрос

Всё может пойти не так.

- Принимающая сторона может воспринимать целые числа по-разному.

  i686 `sizeof(int)` $=$ `sizeof(long)`, т. е. $4 = 4$.

  x86_64 `sizeof(int)` $\neq$ `sizeof(long)`, т. е. $4 \neq 8$.

- Принимающая сторона может использовать другой порядок байтов.
- Компиляторы могут выравнивать структуры по-разному.
- Что произойдёт, когда формат станет *сложным*?

http://www.joelonsoftware.com/items/2008/02/19.html

Сериализация данных

- **Сериализация данных** — процесс представления объекта в виде одномерного потока битов.

Сериализация данных

- **Сериализация данных** — процесс представления объекта в виде одномерного потока битов.
- **Десериализация данных** — обратный процесс восстановления состояния объекта из потока битов.

Достоинства и недостатки

- Упрощение организации взаимодействия.

Достоинства и недостатки

- Упрощение организации взаимодействия.
- Независимость от языка программирования.

Достоинства и недостатки

- Упрощение организации взаимодействия.
- Независимость от языка программирования.
  - Исключения: встроенные форматы в языках.

Достоинства и недостатки

- Упрощение организации взаимодействия.
- Независимость от языка программирования.
  - Исключения: встроенные форматы в языках.
- Удобство работы.

Достоинства и недостатки

- Упрощение организации взаимодействия.
- Независимость от языка программирования.
  - Исключения: встроенные форматы в языках.
- Удобство работы.
- Накладные расходы на представление данных.

Достоинства и недостатки

- Упрощение организации взаимодействия.
- Независимость от языка программирования.
  - Исключения: встроенные форматы в языках.
- Удобство работы.
- Накладные расходы на представление данных.
  - Зависят от выбранного формата.

Проблемы реального мира

В промышленных системах всё не так гладко.

Проблемы реального мира

В промышленных системах всё не так гладко.

- Передаваемые структуры данных могут быть *сложны*.

Проблемы реального мира

В промышленных системах всё не так гладко.

- Передаваемые структуры данных могут быть *сложны*.
- Требования к системе могут эволюционировать со временем.

Проблемы реального мира

В промышленных системах всё не так гладко.

- Передаваемые структуры данных могут быть *сложны*.
- Требования к системе могут эволюционировать со временем.
- Нужна обработка ошибок и интеграция в ПО.

Проблемы реального мира

В промышленных системах всё не так гладко.

- Передаваемые структуры данных могут быть *сложны*.
- Требования к системе могут эволюционировать со временем.
- Нужна обработка ошибок и интеграция в ПО.
- Поддержка собственного протокола затруднительна.

Подходы к решению проблем

- Унификация схемы данных.

Подходы к решению проблем

- Унификация схемы данных.
- Кодогенерация.

Подходы к решению проблем

- Унификация схемы данных.
- Кодогенерация.
- Удалённый вызов процедур.

Содержание

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:
  - двоичные: MessagePack, Protocol Buffers;

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:
  - двоичные: MessagePack, Protocol Buffers;
  - текстовые: JSON, XML, YAML.

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:
  - двоичные: MessagePack, Protocol Buffers;
  - текстовые: JSON, XML, YAML.
- **по наличию схемы**:

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:
  - двоичные: MessagePack, Protocol Buffers;
  - текстовые: JSON, XML, YAML.
- **по наличию схемы**:
  - бессхемовые: JSON, XML, YAML, MessagePack;

Виды сериализации

Форматы сериализации различаются:

- **по формату вывода**:
  - двоичные: MessagePack, Protocol Buffers;
  - текстовые: JSON, XML, YAML.
- **по наличию схемы**:
  - бессхемовые: JSON, XML, YAML, MessagePack;
  - с интерфейсом описания: Protocol Buffers, XML+XSD.

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`

Десериализация

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`

Десериализация

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`
- `'"a":1,"b":2'`

Десериализация

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`
- `'"a":1,"b":2'`

Десериализация

- `s = '{"a": 19, "b": 2000}'`

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`
- `'"a":1,"b":2'`

Десериализация

- `s = '{"a": 19, "b": 2000}'`
- `JSON.load(s)`

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`
- `'"a":1,"b":2'`

Десериализация

- `s = '{"a": 19, "b": 2000}'`
- `JSON.load(s)`
- `Object {a: 19, b: 2000}`

Принцип функционирования (без схемы)

Сериализация

- `foo = {a: 1, b: 2}`
- `JSON.stringify(foo)`
- `'"a":1,"b":2'`

Десериализация

- `s = '{"a": 19, "b": 2000}'`
- `JSON.load(s)`
- `Object {a: 19, b: 2000}`

Пример использования: AJAX.

Принцип функционирования (со схемой)

Сериализация со схемой работает сложнее.

Принцип функционирования (со схемой)

Сериализация со схемой работает сложнее.

- Интерфейс объекта описывается в специальном формате.

Принцип функционирования (со схемой)

Сериализация со схемой работает сложнее.

- Интерфейс объекта описывается в специальном формате.
- Генерируется код для целевого языка программирования.

Принцип функционирования (со схемой)

Сериализация со схемой работает сложнее.

- Интерфейс объекта описывается в специальном формате.
- Генерируется код для целевого языка программирования.
- Сериализация и десериализация выполняется при помощи этого кода.

Принцип функционирования (со схемой)

Сериализация со схемой работает сложнее.

- Интерфейс объекта описывается в специальном формате.
- Генерируется код для целевого языка программирования.
- Сериализация и десериализация выполняется при помощи этого кода.

Пример рассмотрим чуть позже.

Protocol Buffers

Protocol Buffers — технология Google для переносимой двоичной сериализации структурированных данных (2008 г.).

- https://developers.google.com/protocol-buffers/

Protocol Buffers

Protocol Buffers — технология Google для переносимой двоичной сериализации структурированных данных (2008 г.).

- https://developers.google.com/protocol-buffers/

Включает в себя:

- язык описания интерфейсов;

Protocol Buffers

Protocol Buffers — технология Google для переносимой двоичной сериализации структурированных данных (2008 г.).

- https://developers.google.com/protocol-buffers/

Включает в себя:

- язык описания интерфейсов;
- средства кодогенерации.

Apache Thrift

Apache Thrift — технология Facebook для разработки *сервисов* на различных языках программирования (2007 г.).

- https://thrift.apache.org/

Apache Thrift

Apache Thrift — технология Facebook для разработки *сервисов* на различных языках программирования (2007 г.).

- https://thrift.apache.org/

Отличия от Protocol Buffers:

- более развитая система типов;

Apache Thrift

Apache Thrift — технология Facebook для разработки *сервисов* на различных языках программирования (2007 г.).

- https://thrift.apache.org/

Отличия от Protocol Buffers:

- более развитая система типов;
- поддержка исключений и констант;

Apache Thrift

Apache Thrift — технология Facebook для разработки *сервисов* на различных языках программирования (2007 г.).

- https://thrift.apache.org/

Отличия от Protocol Buffers:

- более развитая система типов;
- поддержка исключений и констант;
- наличие протокола **удалённого вызова процедур**.

Содержание

Демонстрация работы Protocol Buffers

Protocol Buffers на примере «**Томита-парсера**».

- http://www.slideshare.net/Tatiana.lando/tomita
- https://tech.yandex.ru/tomita/.../about-docpage/
- https://github.com/yandex/tomita-parser/

Спорный, но работающий подход к использованию технологии.

Демонстрация работы Apache Thrift

Сетевой калькулятор на основе **Apache Thrift**.

- https://thrift.apache.org/tutorial/py

Содержание

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.
- Существует проблема масштабирования и обеспечения отказоустойчивости.

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.
- Существует проблема масштабирования и обеспечения отказоустойчивости.

Примеры использования:

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.
- Существует проблема масштабирования и обеспечения отказоустойчивости.

Примеры использования:

- Facebook и Quora: разработка сервисов на различных ЯП;

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.
- Существует проблема масштабирования и обеспечения отказоустойчивости.

Примеры использования:

- Facebook и Quora: разработка сервисов на различных ЯП;
- Evernote: основа программного интерфейса;

Обсуждение

- Protocol Buffers и Thrift — не единственные технологии: Avro, RMI, XML-RPC, JSON-RPC, ~~CORBA~~, и др.
- Существует проблема масштабирования и обеспечения отказоустойчивости.

Примеры использования:

- Facebook и Quora: разработка сервисов на различных ЯП;
- Evernote: основа программного интерфейса;
- Hadoop и HBase: интерфейс для внешних приложений.

Содержание

Домашнее задание

Разработать сервис разбора базовых метаданных Open Graph в коде Веб-страниц: `og:{title,type,image,url}`.

- **Входные данные:** строка с адресом страницы.
  Результат: структура с её метаданными.

Домашнее задание

Разработать сервис разбора базовых метаданных Open Graph в коде Веб-страниц: `og:{title,type,image,url}`.

- **Входные данные:** строка с адресом страницы.
  Результат: структура с её метаданными.
- Использовать любой формат сериализации со схемой.

Домашнее задание

Разработать сервис разбора базовых метаданных Open Graph в коде Веб-страниц: `og:{title,type,image,url}`.

- **Входные данные:** строка с адресом страницы.
  Результат: структура с её метаданными.
- Использовать любой формат сериализации со схемой.
- Сервер и клиент писать на разных языках.

Домашнее задание

Разработать сервис разбора базовых метаданных Open Graph в коде Веб-страниц: `og:{title,type,image,url}`.

- **Входные данные:** строка с адресом страницы.
  Результат: структура с её метаданными.
- Использовать любой формат сериализации со схемой.
- Сервер и клиент писать на разных языках.
- Предусмотреть: HTTP-перенаправление, отсутствие страницы, отсутствие метаданных, отсутствие сети.

Домашнее задание

Разработать сервис разбора базовых метаданных Open Graph в коде Веб-страниц: `og:{title,type,image,url}`.

- **Входные данные:** строка с адресом страницы.
  Результат: структура с её метаданными.
- Использовать любой формат сериализации со схемой.
- Сервер и клиент писать на разных языках.
- Предусмотреть: HTTP-перенаправление, отсутствие страницы, отсутствие метаданных, отсутствие сети.

https://developers.facebook.com/tools/debug/og/object/

Спасибо за внимание!

Вопросы?

Дмитрий Усталов

https://linkedin.com/in/ustalov

http://kvkt.urfuclub.ru/courses/dot/

https://telegram.me/doturfu

dmitry.ustalov@urfu.ru