Как отправить данные?

Данные в HTTP-запросе передаются в текстовом виде, если рассматривать HTTP/1.1.

В HTTP/2 все передается в бинарном. Однако это не означает, что мы можем передавать только строки. Чтобы передавать сложные структуры данных, надо просто выбрать формат для передачи.

Form-Data

Form-data — это формат передачи данных в виде ключ-значение. Вот пример, как это выглядит в HTTP-запросе:

```
POST / HTTP/2.0
Host: lms.metaclass.kts.studio
Content-Type: application/x-www-form-urlencoded
Content-Length: 13
say=Hi&to=KTS
```

Плюсы Form-Data

Почти все данные в формах на сайте можно представить в виде пары ключ-значения. Поэтому этот формат уже встроен в браузеры и в стандарт html.

То есть можно просто написать такой HTML-код...

```
<html>
  <form action="/add_user" method="post">
    <input type="email" name="email">
    <input type="submit" value="отправить email">
  </form>
</html>
```

... получить такую разметку...

... и при нажатии на клавишу «отправить email» данные с помощью метода POST уйдут на указанный в атрибуте action URL.

С помощью form-data можно отправить файлы. При отправке файлов в http-запрос включается так называемый boundary, в котором находится содержимое файлов, приведенное к унифицированному формату. Здесь можно подробнее прочитать про то, как работает отправка файлов через form-data.

В профессиональной разработке стоит использовать form-data только для передачи файлов. С остальными данными гораздо лучше справляются более комплексные форматы передачи.

XML

Не очень хочется говорить об этом формате, но надо...

XML — это формат для структурированного представления информации с помощью тегов, таких же по устройству, как HTML-теги. Вот пример XML-документа:

```
<course>
  <name>Начинающий backend-разработчик на Python</name>
  <chapters>
    <chapter>Алгоритмы</chapter>
    <chapter>Сетевое взаимодействие</chapter>
    <chapter>Базы данных</chapter>
    <chapter>Тестирование</chapter>
  </chapters>
</course>
```

В наше время XML в сфере web-программирования можно встретить чаще всего в legacy-продуктах. Во всех современных приложениях чаще всего используется JSON.

Минусы XML для разработки веб-приложений

Избыточность. Пример выше можно переписать на JSON так:

```
{
  "course": {
    "name":"Начинающий backend-разработчик на Python",
    "chapters":["Алгоритмы", "Сетевое взаимодействие", "Базы данных", "Тестирование"]
  }
}
```

Его размер 195 символов против 230 в XML. Эта разница растет практически экспоненциально со сложностью и вложенностью передаваемых объектов.

Необходимость согласования и документирования. Чтобы разработать API, который отдает XML, надо написать документацию с указаниями: какие существуют возможные теги, какие атрибуты могут использоваться для этих тегов, как можно эти теги вкладывать друг в друга и т.д. Это добавляет избыточную сложность в разработку — в случае JSON нет возможности указать какие-то специальные атрибуты тега: массив — это всегда массив.

Сложность парсинга. Как понять, что chapters — это массив? Надо найти закрывающий тег и посмотреть, что между тегами есть ряд одинаковых ключей.

Как понять это в JSON? Найти квадратную открывающуюся скобку!

Удаленность от языков программирования. Как выглядит массив:

1. в Python: [“1”, “2”]
2. JavaScript: [“1”, “2”]
3. JSON: [“1”, “2”]
4. XML: <array><value>1</value><value>2</value></array>

Надеюсь, пример достаточно показателен.

JSON

Самый любимый язык разметки веб-разработчиков. Он прекрасно интегрируется с существующими типами в языке, а для его парсинга есть готовые функции, встроенные в языки. Он в меру избыточен и эта избыточность является его плюсом: он человекочитаемый и очень гибкий. Пример разметки этого формата можно посмотреть выше.

Бинарные форматы передачи

Это форматы, которые оптимизированы под быстродействие. Например, формат ProtoBuf передает данные в бинарном виде, на каждое значение, в котором выделено жестко заданное количество бит. Также этот формат не передает ключи. Например, надо передать возраст пользователя и флаг, обозначающий, авторизован он или нет:

В **JSON** это выглядело бы так: {“age”:12,“authorized”:true} => 28 символов = **56 байт** в кодировке UTF-8.

А **ProtoBuf** формат выглядит примерно так: 12true => {uint32}{bool} => {32 бит}{1 бит}{15 пустых бит для 16-битного выравнивания} => 48 бит = **6 байт.**

Разница в размерах впечатляет.

Минусы бинарных форматов

Минусы перечеркивают плюсы от быстродействия.

Бинарные форматы не человекочитаемы — в текстовом виде они будут выглядеть как кракозябры.

Бинарные форматы должны быть жестко заданы как на клиенте, так и на сервере. Если клиент отправляет набор битов, то сервер должен уметь декодировать их по какой-то схеме.

Хорошее применение бинарных форматов — взаимодействие между микросервисами. В этом случае действительно важна скорость, а схема данных может быть обозначена в одном месте кода и синхронно обновляться как для отправляющего, так и для получающего.