Comparing Protobuf to AMF

Oleg Sivokon

<2018-06-27 Wed>

Outline

Some historical context

Metadata vs no metadata

Technical details of Protobuf

Reference

Some historical context

Battle for web dominance

- HTML rules the web. XSL and then SOAP are poorly accepted.
- PHP, unlike ASP or JSP streams page content.
- Macromedia, then Adobe Flash needs streaming for interactivity.
- AMF gives Flash a competitive edge over SOAP.
- AMFPHP first, then others implement AMF.
- Adobe and Mozilla lose standards battle to Google, MS and Apple.
- Protobuf is positioned to replace AMF.

Metadata vs no metadata

AMF: yes to metadata

- Easy debugging.
- No need for codegeneration.
- Some overhead in short messages.
- Potential for references, richer vocabulary.

Protobuf: no to metadata

- Invention of Descriptor (fixed point description of description).
- Extensive infrastrucutre. Pros: more tools. Cons: more work.
  - Special RPC description language.
  - Special language-independent language for binary data description.
- Multiple encoding layers: wire-type, value-type, application-type.

Technical details of Protobuf

Message description

- Formal grammar attempted, but not verified.
- Proliferation of special cases.
- Confusion about semicolons.
- Implementing parser is not enough.

Illustration I

```
syntax = "protobuf3";

import "another.proto";

enum Enum {
    MEMBER_1 = 1;  // Enumerations must not be zero
    MEMBER_2 = 3;
}
```

Illustration II

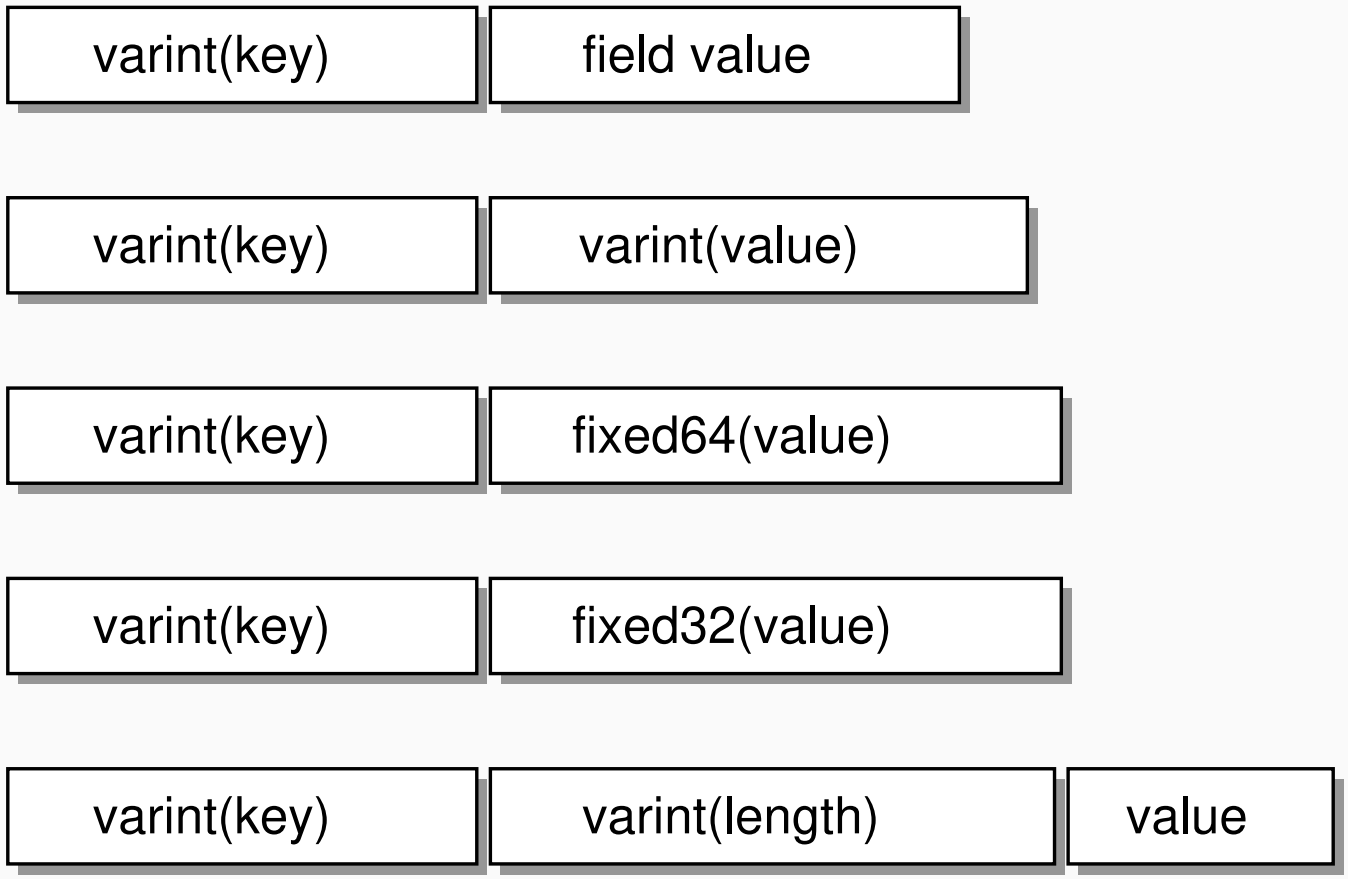
```
syntax = "protobuf3";
package some.package;

message Message {
    message NestedMessage {
        repeated int32 repeated_field = 1;
    }
    NestedMessage nested = 1;
    oneof unnecessary_name {
        int32 either = 2;  // Must be distinct
        string or = 3;
    }
    map<int32, NestedMessage> a_map = 4;
}
```

Binary protocol

- Wire-types: very limited.
- Value-types: no pointers.
- Application types: open to interpretation.

Illustration

Wire-level deals with key-value pairs



Value types

- `int` and `uint` types behave exactly the same way
- `repeated` types are, in fact, encoded differently, depending on the type being a `scalar` or not.
- *sintXX*(V) are encoded as *intXX*$(|V| \times 2 - (\frac{V}{|V|} - 1) \times \frac{1}{2})$.

varint	int32	int64	sint32	sint64
fixed64	fixed64	sfixed64	double	
fixed32	fixed32	sfixed32		
lenght-delimited	bytes	string	message	repeated

Application types

- Must be messages or enumerators.
- May contain extra fields, or may lack fields.
- Contain no identifying information.
- Repeated fields (and map fields) cannot be nested.

Reference

Links

- Cannasse: Ecmascript 4 and the web
- Cannasse: Ecmascript discharmony
- AMFPHP
- Wikipedia: AMF
- Protobuf language specification