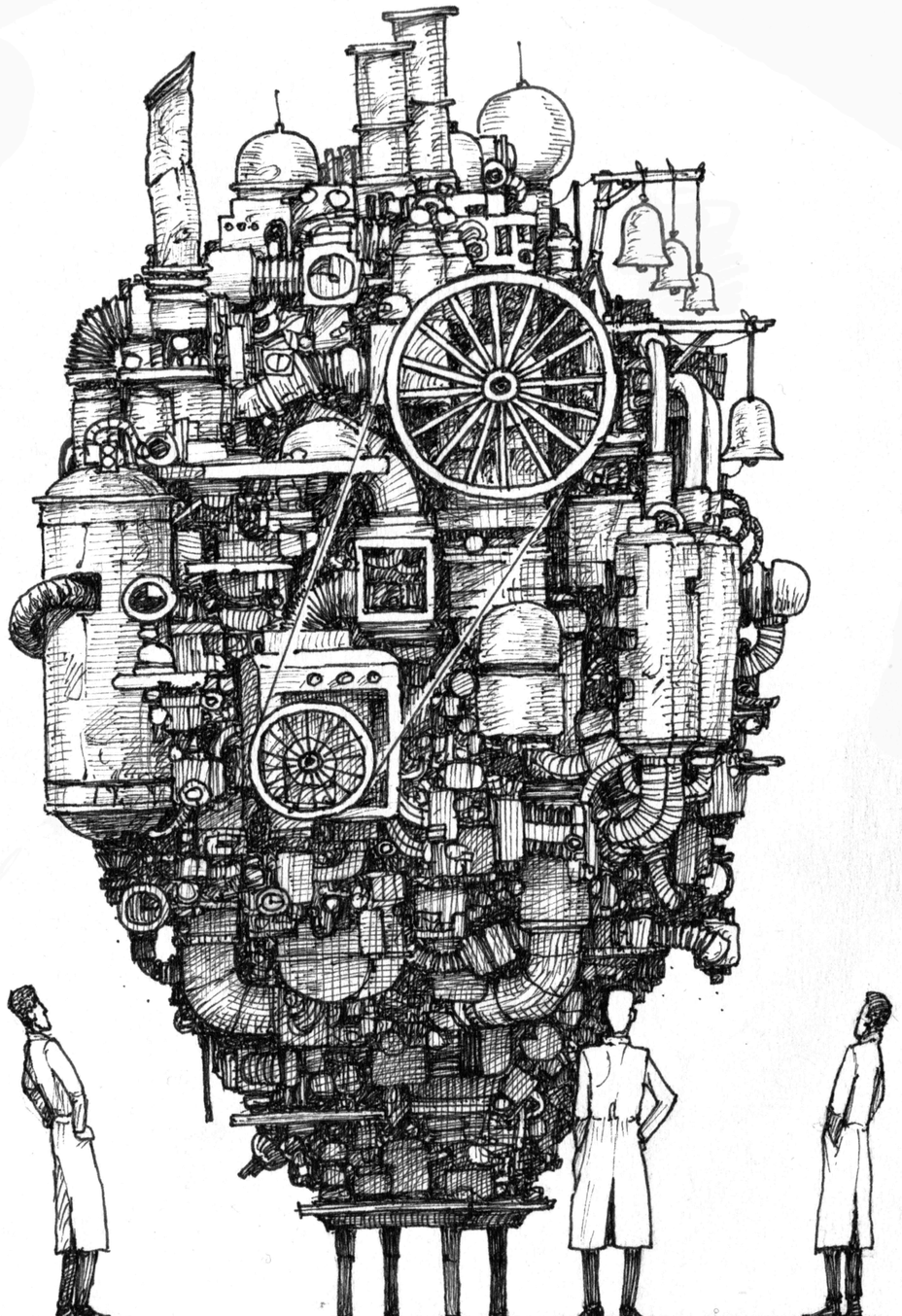
gRPC Demystified or hand-writing a gRPC handler in 7 minutes

October 2022

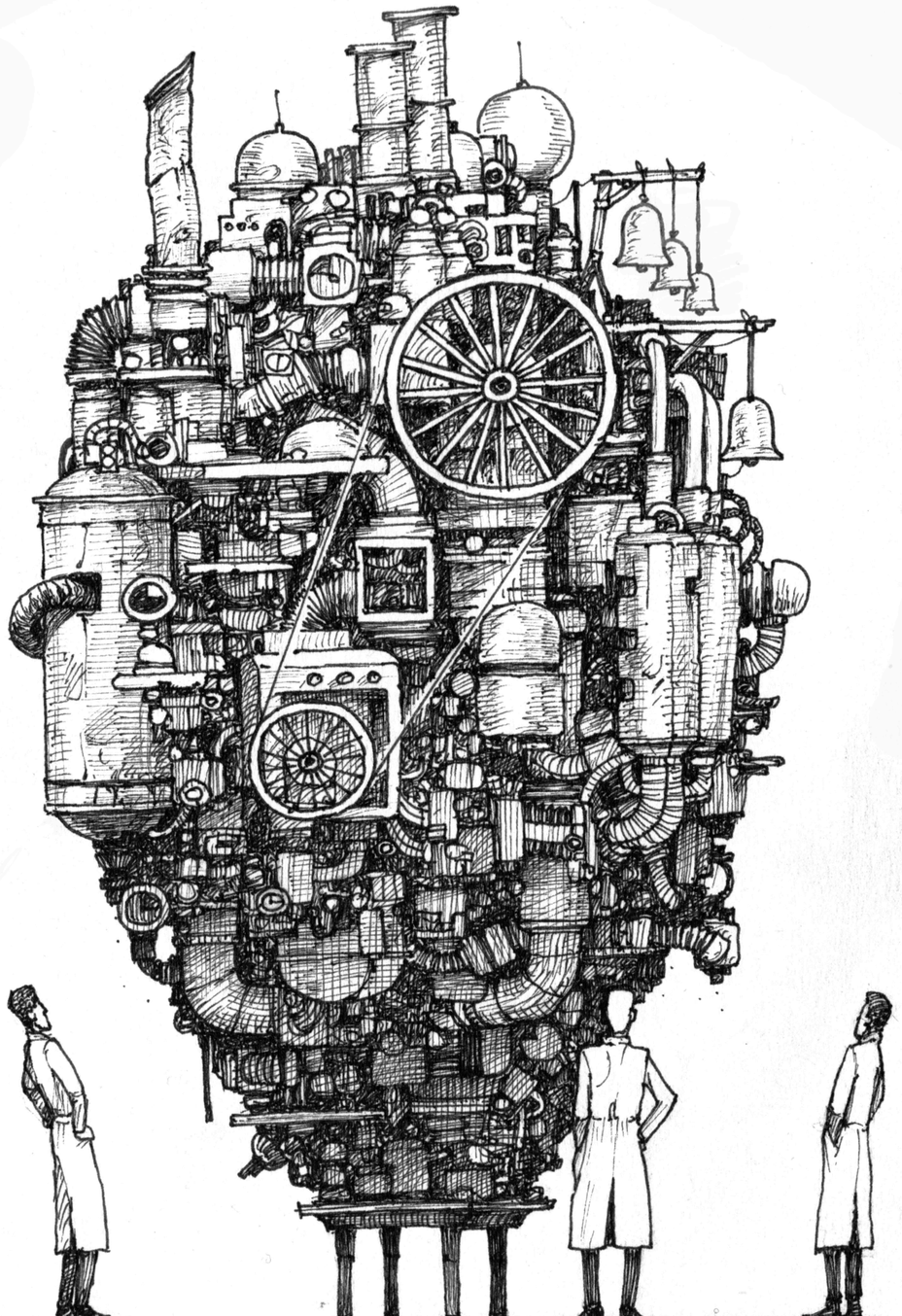
@akshayshah



gRPC Demystified or hand-writing a gRPC handler in 7 minutes

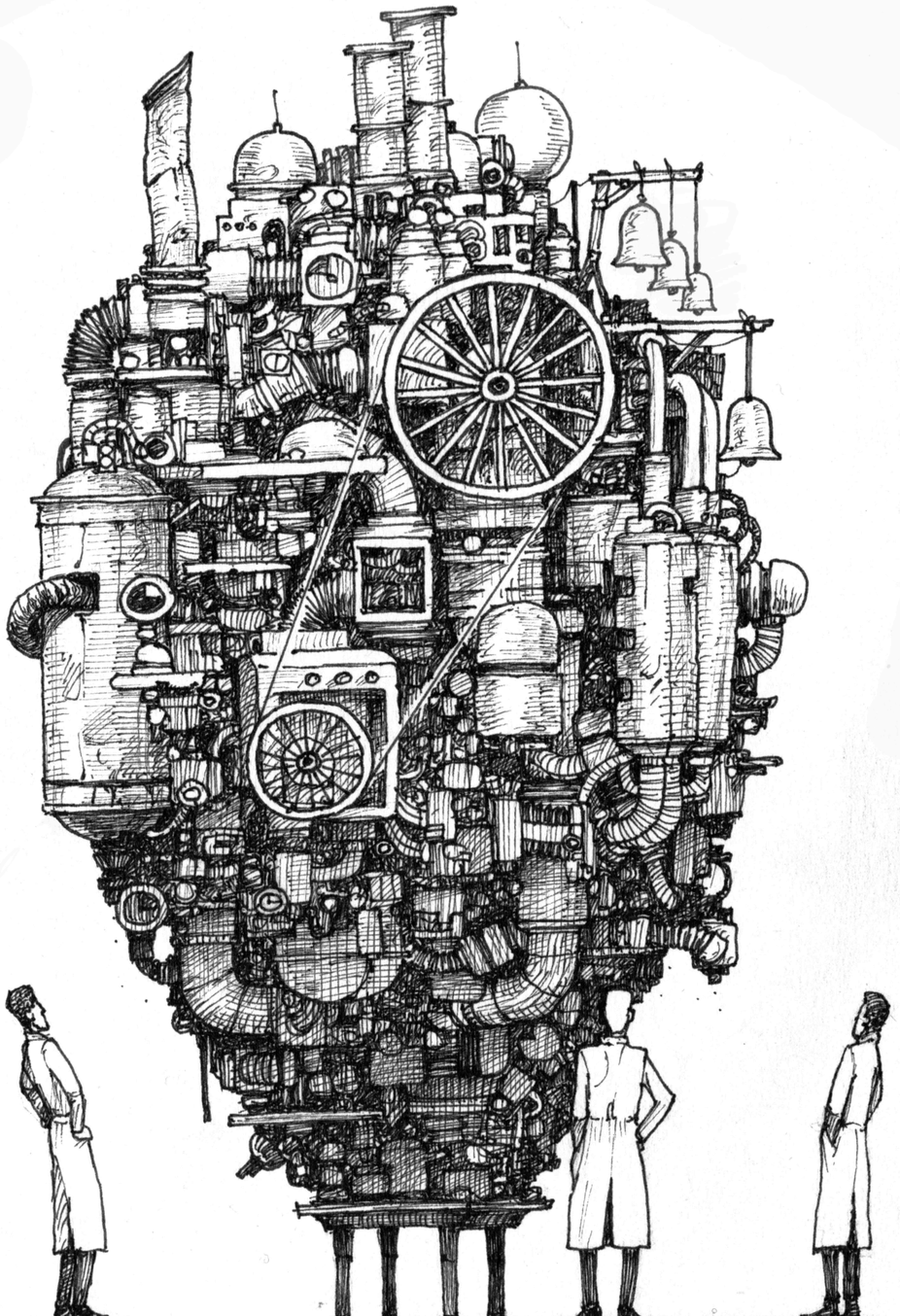
October 2022

@akshayshah



This is grpc-go.
It's complicated!

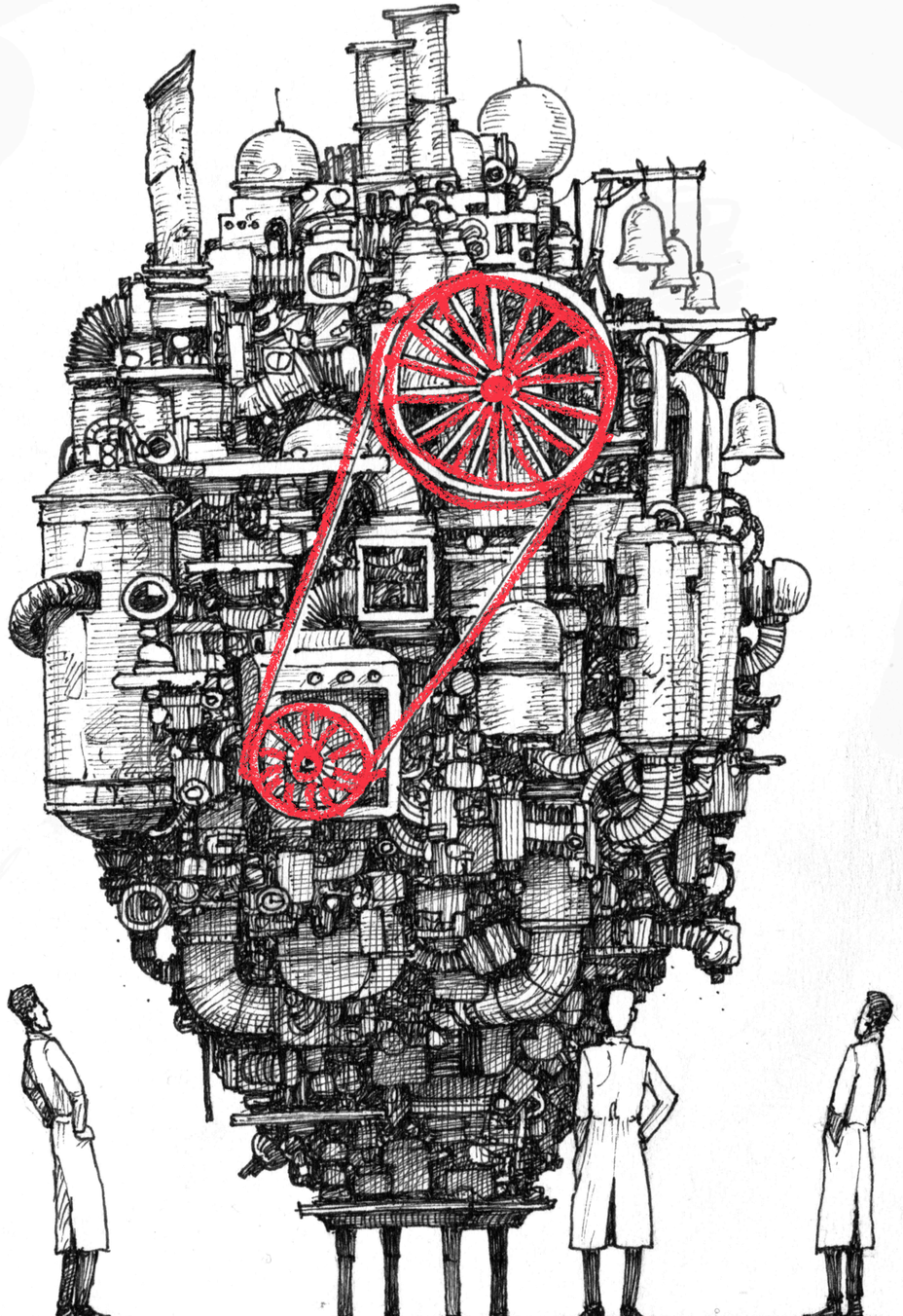
Let's write our own.



The heart of gRPC is the wire protocol.
It's just HTTP.

It supports:

- streaming
- pluggable encodings



Start with a REST handler to create a pet.

model data with a struct

validation

marshaling

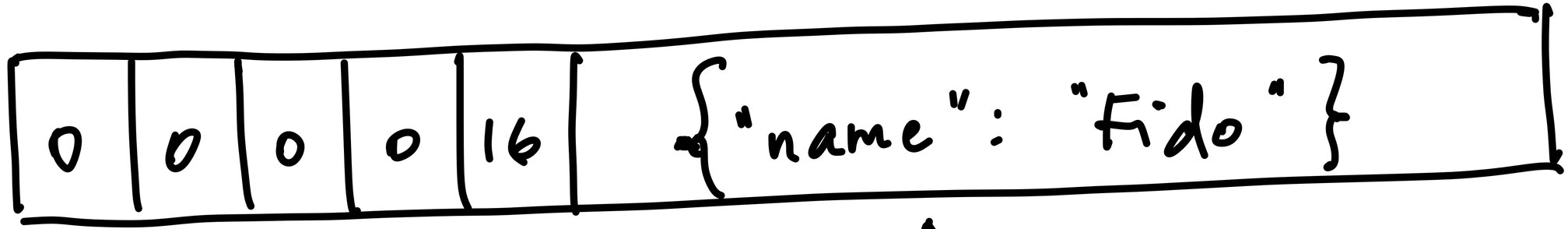
```go
package main

import (
  "encoding/json"
  "net/http"
)

type Pet struct {
  Name string `json:"name"`
}

func restHandler(w http.ResponseWriter, r *http.Request) {
  const ctype = "application/json"
  if r.Method != http.MethodPost {
    w.Header().Set("Allow", http.MethodPost)
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  if r.Header.Get("Content-Type") != ctype {
    w.Header().Set("Accept", ctype)
    w.WriteHeader(http.StatusUnsupportedMediaType)
    return
  }
  var pet Pet
  json.NewDecoder(r.Body).Decode(&pet)
  // ✨ save to imaginary DB ✨
  w.Header().Set("Content-Type", ctype)
  json.NewEncoder(w).Encode(pet)
}
```

It works!

But what about streaming and pluggable encodings?

```
$ curl --json '{"name": "Fido"}' https://localhost:8080/v1/pets
{"name": "Fido"}
{"name": "Rex"}
{"name": "Daisy"}
```

We envelope messages by adding a 5-byte prefix.



8 bitwise flags

uint32 for message length

raw message

We can modify our REST handler to use gRPC-style envelopes.

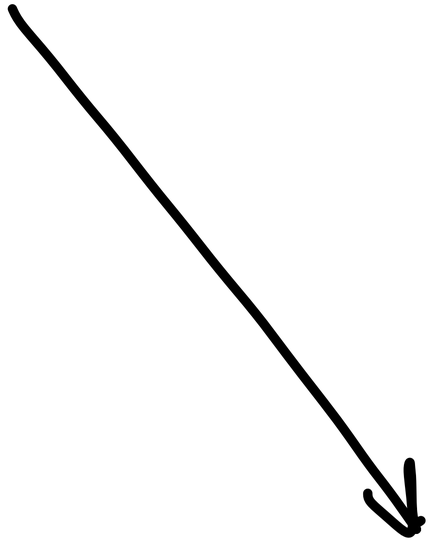
utility type for prefixes

```go
package main

import (
  "encoding/binary"
  "encoding/json"
  "net/http"
)

type prefix [5]byte

func (p prefix) Size() int {
  return int(binary.BigEndian.Uint32(p[1:5]))
}

func (p *prefix) SetSize(n int) {
  binary.BigEndian.PutUint32(p[1:5], uint32(n))
}

func grpcHandler(w http.ResponseWriter, r *http.Request) {
  const ctype = "application/grpc+json"
  if r.Method != http.MethodPost {
    w.Header().Set("Allow", http.MethodPost)
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  if r.Header.Get("Content-Type") != ctype {
    w.Header().Set("Accept", ctype)
    w.WriteHeader(http.StatusUnsupportedMediaType)
    return
  }
  var pre prefix
  r.Body.Read(pre[:])
  input := make([]byte, pre.Size())
  r.Body.Read(input)
  var pet Pet
  json.Unmarshal(input, &pet)
  // ✨ save to imaginary DB ✨
  w.Header().Set("Content-Type", ctype)
  w.Header().Set("Te", "trailers") // flush out incompatible proxies
  out, _ := json.Marshal(pet)
  pre.SetSize(len(out))
  w.Write(pre[:])
  w.Write(out)
  w.Header().Set(http.TrailerPrefix+"Grpc-Status", "0")
  w.Header().Set(http.TrailerPrefix+"Grpc-Message", "")
}
```

After enveloping, we're no longer sending plain JSON.

```go
package main

import (
  "encoding/binary"
  "encoding/json"
  "net/http"
)

type prefix [5]byte

func (p prefix) Size() int {
  return int(binary.BigEndian.Uint32(p[1:5]))
}

func (p *prefix) SetSize(n int) {
  binary.BigEndian.PutUint32(p[1:5], uint32(n))
}

func grpcHandler(w http.ResponseWriter, r *http.Request) {
  const ctype = "application/grpc+json"
  if r.Method != http.MethodPost {
    w.Header().Set("Allow", http.MethodPost)
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  if r.Header.Get("Content-Type") != ctype {
    w.Header().Set("Accept", ctype)
    w.WriteHeader(http.StatusUnsupportedMediaType)
    return
  }
  var pre prefix
  r.Body.Read(pre[:])
  input := make([]byte, pre.Size())
  r.Body.Read(input)
  var pet Pet
  json.Unmarshal(input, &pet)
  // ✨ save to imaginary DB ✨
  w.Header().Set("Content-Type", ctype)
  w.Header().Set("Te", "trailers") // flush out incompatible proxies
  out, _ := json.Marshal(pet)
  pre.SetSize(len(out))
  w.Write(pre[:])
  w.Write(out)
  w.Header().Set(http.TrailerPrefix+"Grpc-Status", "0")
  w.Header().Set(http.TrailerPrefix+"Grpc-Message", "")
}
```

Use prefixes when marshaling and unmarshaling.



```go
package main

import (
  "encoding/binary"
  "encoding/json"
  "net/http"
)

type prefix [5]byte

func (p prefix) Size() int {
  return int(binary.BigEndian.Uint32(p[1:5]))
}

func (p *prefix) SetSize(n int) {
  binary.BigEndian.PutUint32(p[1:5], uint32(n))
}

func grpcHandler(w http.ResponseWriter, r *http.Request) {
  const ctype = "application/grpc+json"
  if r.Method != http.MethodPost {
    w.Header().Set("Allow", http.MethodPost)
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  if r.Header.Get("Content-Type") != ctype {
    w.Header().Set("Accept", ctype)
    w.WriteHeader(http.StatusUnsupportedMediaType)
    return
  }
  var pre prefix
  r.Body.Read(pre[:])
  input := make([]byte, pre.Size())
  r.Body.Read(input)
  var pet Pet
  json.Unmarshal(input, &pet)
  // ✨ save to imaginary DB ✨
  w.Header().Set("Content-Type", ctype)
  w.Header().Set("Te", "trailers") // flush out incompatible proxies
  out, _ := json.Marshal(pet)
  pre.SetSize(len(out))
  w.Write(pre[:])
  w.Write(out)
  w.Header().Set(http.TrailerPrefix+"Grpc-Status", "0")
  w.Header().Set(http.TrailerPrefix+"Grpc-Message", "")
}
```

How should we handle errors after we've sent the HTTP status?

```
$ curl --json '{"name": "Fido"}' https://localhost:8080/v1/pets
{"name": "Fido"}
{"name": "Rex"}
{"name": "Daisy"}
```

error!

Send a gRPC-specific status code as an HTTP trailer.

```go
package main

import (
  "encoding/binary"
  "encoding/json"
  "net/http"
)

type prefix [5]byte

func (p prefix) Size() int {
  return int(binary.BigEndian.Uint32(p[1:5]))
}

func (p *prefix) SetSize(n int) {
  binary.BigEndian.PutUint32(p[1:5], uint32(n))
}

func grpcHandler(w http.ResponseWriter, r *http.Request) {
  const ctype = "application/grpc+json"
  if r.Method != http.MethodPost {
    w.Header().Set("Allow", http.MethodPost)
    w.WriteHeader(http.StatusMethodNotAllowed)
    return
  }
  if r.Header.Get("Content-Type") != ctype {
    w.Header().Set("Accept", ctype)
    w.WriteHeader(http.StatusUnsupportedMediaType)
    return
  }
  var pre prefix
  r.Body.Read(pre[:])
  input := make([]byte, pre.Size())
  r.Body.Read(input)
  var pet Pet
  json.Unmarshal(input, &pet)
  // ✨ save to imaginary DB ✨
  w.Header().Set("Content-Type", ctype)
  w.Header().Set("Te", "trailers") // flush out incompatible proxies
  out, _ := json.Marshal(pet)
  pre.SetSize(len(out))
  w.Write(pre[:])
  w.Write(out)
  w.Header().Set(http.TrailerPrefix+"Grpc-Status", "0")
  w.Header().Set(http.TrailerPrefix+"Grpc-Message", "")
}
```

https://connect.build

https://github.com/
akshayjshah/
grpc-demystified