Package multipart go1.15.2 Latest

Published: **Sep 9, 2020** | License: BSD-3-Clause | Standard library

Overview

Package multipart implements MIME multipart parsing, as defined in RFC 2046.

The implementation is sufficient for HTTP (RFC 2388) and the multipart bodies generated by popular browsers.

Variables

```
var ErrMessageTooLarge = errors.New("multipart: message too large")
```

ErrMessageTooLarge is returned by ReadForm if the message form data is too large to be processed.

type File

```
type File interface {
    io.Reader
    io.ReaderAt
    io.Seeker
    io.Closer
}
```

File is an interface to access the file part of a multipart message. Its contents may be either stored in memory or on disk. If stored on disk, the File's underlying concrete type will be an *os.File.

type FileHeader

```
type FileHeader struct {
    Filename string
    Header   textproto.MIMEHeader
    Size     int64
    // contains filtered or unexported fields
}
```

A FileHeader describes a file part of a multipart request.

func (*FileHeader) Open

```
func (fh *FileHeader) Open() (File, error)
```

Open opens and returns the FileHeader's associated File.

type Form

```
type Form struct {
    Value map[string][]string
    File  map[string][]*FileHeader
}
```

Form is a parsed multipart form. Its File parts are stored either in memory or on disk, and are accessible via the *FileHeader's Open method. Its Value parts are stored as strings. Both are keyed by field name.

func (*Form) RemoveAll

```
func (f *Form) RemoveAll() error
```

RemoveAll removes any temporary files associated with a Form.

type Part

```
type Part struct {
    // The headers of the body, if any, with the keys canonicalized
    // in the same fashion that the Go http.Request headers are.
    // For example, "foo-bar" changes case to "Foo-Bar"
    Header textproto.MIMEHeader
    // contains filtered or unexported fields
}
```

A Part represents a single part in a multipart body.

func (*Part) Close

```
func (p *Part) Close() error
```

func (*Part) FileName

```
func (p *Part) FileName() string
```

FileName returns the filename parameter of the Part's Content-Disposition header.

func (*Part) FormName

```
func (p *Part) FormName() string
```

FormName returns the name parameter if p has a Content-Disposition of type "form-data". Otherwise it returns the empty string.

func (*Part) Read

```
func (p *Part) Read(d []byte) (n int, err error)
```

Read reads the body of a part, after its headers and before the next part (if any) begins.

type Reader

```
type Reader struct {
    // contains filtered or unexported fields
}
```

Reader is an iterator over parts in a MIME multipart body. Reader's underlying parser consumes its input as needed. Seeking isn't supported.

func NewReader

```
func NewReader(r io.Reader, boundary string) *Reader
```

NewReader creates a new multipart Reader reading from r using the given MIME boundary.

The boundary is usually obtained from the "boundary" parameter of the message's "Content-Type" header. Use mime.ParseMediaType to parse such headers.

func (*Reader) NextPart

```
func (r *Reader) NextPart() (*Part, error)
```

NextPart returns the next part in the multipart or an error. When there are no more parts, the error io.EOF is returned.

As a special case, if the "Content-Transfer-Encoding" header has a value of "quoted-printable", that header is instead hidden and the body is transparently decoded during Read calls.

func (*Reader) NextRawPart

```
func (r *Reader) NextRawPart() (*Part, error)
```

NextRawPart returns the next part in the multipart or an error. When there are no more parts, the error io.EOF is returned.

Unlike NextPart, it does not have special handling for "Content-Transfer-Encoding: quoted-printable".

func (*Reader) ReadForm

```
func (r *Reader) ReadForm(maxMemory int64) (*Form, error)
```

ReadForm parses an entire multipart message whose parts have a Content-Disposition of "form-data". It stores up to maxMemory bytes + 10MB (reserved for non-file parts) in memory. File parts which can't be stored in memory will be stored on disk in temporary files. It returns ErrMessageTooLarge if all non-file parts can't be stored in memory.

type Writer

```
type Writer struct {
    // contains filtered or unexported fields
}
```

A Writer generates multipart messages.

func NewWriter

```
func NewWriter(w io.Writer) *Writer
```

NewWriter returns a new multipart Writer with a random boundary, writing to w.

func (*Writer) Boundary

```
func (w *Writer) Boundary() string
```

Boundary returns the Writer's boundary.

func (*Writer) Close

```
func (w *Writer) Close() error
```

Close finishes the multipart message and writes the trailing boundary end line to the output.

func (*Writer) CreateFormField

```
func (w *Writer) CreateFormField(fieldname string) (io.Writer, error)
```

CreateFormField calls CreatePart with a header using the given field name.

func (*Writer) CreateFormFile

```
func (w *Writer) CreateFormFile(fieldname, filename string) (io.Writer, error)
```

CreateFormFile is a convenience wrapper around CreatePart. It creates a new form-data header with the provided field name and file name.

func (*Writer) CreatePart

```
func (w *Writer) CreatePart(header textproto.MIMEHeader) (io.Writer, error)
```

CreatePart creates a new multipart section with the provided header. The body of the part should be written to the returned Writer. After calling CreatePart, any previous part may no longer be written to.

func (*Writer) FormDataContentType

```
func (w *Writer) FormDataContentType() string
```

FormDataContentType returns the Content-Type for an HTTP multipart/form-data with this Writer's Boundary.

func (*Writer) SetBoundary

```
func (w *Writer) SetBoundary(boundary string) error
```

SetBoundary overrides the Writer's default randomly-generated boundary separator with an explicit value.

SetBoundary must be called before any parts are created, may only contain certain ASCII characters, and must be non-empty and at most 70 bytes long.

func (*Writer) WriteField

```
func (w *Writer) WriteField(fieldname, value string) error
```

WriteField calls CreateFormField and then writes the given value.