Blackfriday build passing GO reference ===========

Blackfriday is a Markdown processor implemented in Go. It is paranoid about its input (so you can safely feed it user-supplied data), it is fast, it supports common extensions (tables, smart punctuation substitutions, etc.), and it is safe for all utf-8 (unicode) input.

HTML output is currently supported, along with Smartypants extensions.

It started as a translation from C of Sundown.

Installation

Blackfriday is compatible with any modern Go release. With Go and git installed:

```
go get -u gopkg.in/russross/blackfriday.v2
```

will download, compile, and install the package into your `$GOPATH` directory hierarchy.

Versions

Currently maintained and recommended version of Blackfriday is `v2`. It's being developed on its own branch: https://github.com/russross/blackfriday/tree/v2 and the documentation is available at https://godoc.org/gopkg.in/russross/blackfriday.v2.

It is `go get`-able via gopkg.in at `gopkg.in/russross/blackfriday.v2`, but we highly recommend using package management tool like dep or Glide and make use of semantic versioning. With package management you should import `github.com/russross/blackfriday` and specify that you're using version 2.0.0.

Version 2 offers a number of improvements over v1:

- Cleaned up API
- A separate call to `Parse`, which produces an abstract syntax tree for the document
- Latest bug fixes
- Flexibility to easily add your own rendering extensions

Potential drawbacks:

- Our benchmarks show v2 to be slightly slower than v1. Currently in the ballpark of around 15%.
- API breakage. If you can't afford modifying your code to adhere to the new API and don't care too much about the new features, v2 is probably not for you.
- Several bug fixes are trailing behind and still need to be forward-ported to v2. See issue #348 for tracking.

If you are still interested in the legacy `v1`, you can import it from `github.com/russross/blackfriday`. Documentation for the legacy v1 can be found here: https://godoc.org/github.com/russross/blackfriday

Known issue with `dep`

There is a known problem with using Blackfriday v1 *transitively* and `dep`. Currently `dep` prioritizes semver versions over anything else, and picks the latest one, plus it does not apply a `[[constraint]]` specifier to transitively pulled in packages. So if you're using something that uses Blackfriday v1, but that something does not use `dep` yet, you will get Blackfriday v2 pulled in and your first dependency will fail to build.

There are couple of fixes for it, documented here: https://github.com/golang/dep/blob/master/docs/FAQ.md#how-do-i-constrain-a-transitive-dependencys-version

Meanwhile, `dep` team is working on a more general solution to the constraints on transitive dependencies problem: https://github.com/golang/dep/issues/1124.

Usage

v1

For basic usage, it is as simple as getting your input into a byte slice and calling:

```
output := blackfriday.MarkdownBasic(input)
```

This renders it with no extensions enabled. To get a more useful feature set, use this instead:

```
output := blackfriday.MarkdownCommon(input)
```

v2

For the most sensible markdown processing, it is as simple as getting your input into a byte slice and calling:

```
output := blackfriday.Run(input)
```

Your input will be parsed and the output rendered with a set of most popular extensions enabled. If you want the most basic feature set, corresponding with the bare Markdown specification, use:

```
output := blackfriday.Run(input, blackfriday.WithNoExtensions())
```

Sanitize untrusted content

Blackfriday itself does nothing to protect against malicious content. If you are dealing with user-supplied markdown, we recommend running Blackfriday's output through HTML sanitizer such as Bluemonday.

Here's an example of simple usage of Blackfriday together with Bluemonday:

```
import (
    "github.com/microcosm-cc/bluemonday"
    "gopkg.in/russross/blackfriday.v2"
)

// ...
unsafe := blackfriday.Run(input)
html := bluemonday.UGCPolicy().SanitizeBytes(unsafe)
```

Custom options, v1

If you want to customize the set of options, first get a renderer (currently only the HTML output engine), then use it to call the more general `Markdown` function. For examples, see the implementations of `MarkdownBasic` and `MarkdownCommon` in `markdown.go`.

Custom options, v2

If you want to customize the set of options, use `blackfriday.WithExtensions`, `blackfriday.WithRenderer` and `blackfriday.WithRefOverride`.

`blackfriday-tool`

You can also check out `blackfriday-tool` for a more complete example of how to use it. Download and install it using:

```
go get github.com/russross/blackfriday-tool
```

This is a simple command-line tool that allows you to process a markdown file using a standalone program. You can also browse the source directly on github if you are just looking for some example code:

- http://github.com/russross/blackfriday-tool

Note that if you have not already done so, installing `blackfriday-tool` will be sufficient to download and install blackfriday in addition to the tool itself. The tool binary will be installed in `$GOPATH/bin`. This is a statically-linked binary that can be copied to wherever you need it without worrying about dependencies and library versions.

Sanitized anchor names

Blackfriday includes an algorithm for creating sanitized anchor names corresponding to a given input text. This algorithm is used to create anchors for headings when `EXTENSION_AUTO_HEADER_IDS` is enabled. The algorithm has a specification, so that other packages can create compatible anchor names and links to those anchors.

The specification is located at https://godoc.org/github.com/russross/blackfriday#hdr-Sanitized_Anchor_Names.

`SanitizedAnchorName` exposes this functionality, and can be used to create compatible links to the anchor names generated by blackfriday. This algorithm is also implemented in a small standalone package at `github.com/shurcooL/sanitized_anchor_name`. It can be useful for clients that want a small package and don't need full functionality of blackfriday.

Features

All features of Sundown are supported, including:

- **Compatibility**. The Markdown v1.0.3 test suite passes with the `--tidy` option. Without `--tidy`, the differences are mostly in whitespace and entity escaping, where blackfriday is more consistent and cleaner.

- **Common extensions**, including table support, fenced code blocks, autolinks, strikethroughs, non-strict emphasis, etc.

- **Safety**. Blackfriday is paranoid when parsing, making it safe to feed untrusted user input without fear of bad things happening. The test suite stress tests this and there are no known inputs that make it crash. If you find one, please let me know and send me the input that does it.

  NOTE: "safety" in this context means *runtime safety only*. In order to protect yourself against JavaScript injection in untrusted content, see this example.

- **Fast processing**. It is fast enough to render on-demand in most web applications without having to cache the output.

- **Thread safety**. You can run multiple parsers in different goroutines without ill effect. There is no dependence on global shared state.
- **Minimal dependencies**. Blackfriday only depends on standard library packages in Go. The source code is pretty self-contained, so it is easy to add to any project, including Google App Engine projects.
- **Standards compliant**. Output successfully validates using the W3C validation tool for HTML 4.01 and XHTML 1.0 Transitional.

Extensions

In addition to the standard markdown syntax, this package implements the following extensions:

- **Intra-word emphasis supression**. The `_` character is commonly used inside words when discussing code, so having markdown interpret it as an emphasis command is usually the wrong thing. Blackfriday lets you treat all emphasis markers as normal characters when they occur inside a word.
- **Tables**. Tables can be created by drawing them in the input using a simple syntax:

  ```
  Name    | Age
  --------|------
  Bob     | 27
  Alice   | 23
  ```

- **Fenced code blocks**. In addition to the normal 4-space indentation to mark code blocks, you can explicitly mark them and supply a language (to make syntax highlighting simple). Just mark it like this:

  ~~~
  ``` go
  func getTrue() bool {
      return true
  }
  ```
  ~~~

  You can use 3 or more backticks to mark the beginning of the block, and the same number to mark the end of the block.

  To preserve classes of fenced code blocks while using the bluemonday HTML sanitizer, use the following policy:

  ```
  p := bluemonday.UGCPolicy()
  p.AllowAttrs("class").Matching(regexp.MustCompile("^language-[a-zA-Z0-
  9]+$")).OnElements("code")
  html := p.SanitizeBytes(unsafe)
  ```

- **Definition lists**. A simple definition list is made of a single-line term followed by a colon and the definition for that term.

  ```
  Cat
  : Fluffy animal everyone likes

  Internet
  : Vector of transmission for pictures of cats
  ```

Terms must be separated from the previous definition by a blank line.

- **Footnotes**. A marker in the text that will become a superscript number; a footnote definition that will be placed in a list of footnotes at the end of the document. A footnote looks like this:

  ```
  This is a footnote.[^1]

  [^1]: the footnote text.
  ```

- **Autolinking**. Blackfriday can find URLs that have not been explicitly marked as links and turn them into links.

- **Strikethrough**. Use two tildes (`~~`) to mark text that should be crossed out.

- **Hard line breaks**. With this extension enabled (it is off by default in the `MarkdownBasic` and `MarkdownCommon` convenience functions), newlines in the input translate into line breaks in the output.

- **Smart quotes**. Smartypants-style punctuation substitution is supported, turning normal double- and single-quote marks into curly quotes, etc.

- **LaTeX-style dash parsing** is an additional option, where `--` is translated into `&ndash;` , and `---` is translated into `&mdash;` . This differs from most smartypants processors, which turn a single hyphen into an ndash and a double hyphen into an mdash.

- **Smart fractions**, where anything that looks like a fraction is translated into suitable HTML (instead of just a few special cases like most smartypant processors). For example, `4/5` becomes `<sup>4</sup>&frasl;<sub>5</sub>` , which renders as $^4⁄_5$.

Other renderers

Blackfriday is structured to allow alternative rendering engines. Here are a few of note:

- github_flavored_markdown: provides a GitHub Flavored Markdown renderer with fenced code block highlighting, clickable heading anchor links.

  It's not customizable, and its goal is to produce HTML output equivalent to the GitHub Markdown API endpoint, except the rendering is performed locally.

- markdownfmt: like gofmt, but for markdown.

- LaTeX output: renders output as LaTeX.

- bfchroma: provides convenience integration with the Chroma code highlighting library. bfchroma is only compatible with v2 of Blackfriday and provides a drop-in renderer ready to use with Blackfriday, as well as options and means for further customization.

TODO

- More unit testing
- Improve Unicode support. It does not understand all Unicode rules (about what constitutes a letter, a punctuation symbol, etc.), so it may fail to detect word boundaries correctly in some instances. It is safe on all UTF-8 input.

License