handlebars

Reference

Base

`Handlebars.compile(template, options)`

Compiles a template so it can be executed immediately.

```
var template = Handlebars.compile('{{foo}}');
template({});
```

Supports a variety of options that alter how the template executes

- `data`: Set to false to disable `@data` tracking.
- `compat`: Set to true to enable recursive field lookup.
- `knownHelpers`: Hash containing list of helpers that are known to exist (truthy) at template execution time. Passing this allows the compiler to optimize a number of cases. Builtin helpers are automatically included in this list and may be omitted by setting that value to `false`.

- `knownHelpersOnly`: Set to true to allow further optimzations based on the known helpers list.
- `noEscape`: Set to true to not HTML escape any content.
- `strict`: Run in strict mode. In this mode, templates will throw rather than silently ignore missing fields. This has the side effect of disabling inverse operatins such as `{{^foo}}{{/foo}}` unless fields are explicitly included in the source object.
- `assumeObjects`: Removes object existence checks when traversing paths. This is a subset of `strict` mode that generates optimized templates when the data inputs are known to be safe.
- `preventIndent`: By default an indented partial-call causes the output of the whole partial being indented by the same amount. This can lead to unexpected behavior when the partial writes `pre`-tags. Setting this option to `true` will disable the auto-indent feature.
- `ignoreStandalone`: Disables standalone tag removal when set to `true`. When set, blocks and partials that are on their own line will not remove the whitespace on that line.
- `explicitPartialContext`: Disables implicit context for partials. When enabled, partials that are not passed a context value will execute against an empty object.

`Handlebars.precompile(template, options)`

Precompiles a given template so it can be sent to the client and executed without compilation.

```
var templateSpec = Handlebars.precompile('{{foo}}');
```

Supports all of the same options parameters as the `Handlebars.compile` method. Additionally may pass:

- `srcName`: Passed to generate the source map for the input file. When run in this manner, the return structure is `{code, map}` with `code` containing the template definition and `map` containing the source map.
- `destName`: Optional parameter used in conjunction with `srcName` to provide a destination file name when generating source maps.

`Handlebars.template(templateSpec)`

Sets up a template that was precompiled with `Handlebars.precompile`

```
var template = Handlebars.template(templateSpec);
template({});
```

`Handlebars.registerPartial(name, partial)`

Registers partials accessible by any template in the environment.

```
Handlebars.registerPartial('foo', partial);
```

Also supports registering multiple partials at once.

```
Handlebars.registerPartial({
  foo: partial,
  bar: partial
});
```

If loading the whole library, then the partials may be string values which will be compiled on demand. If only loading the runtime, then the partials must be a precompiled template that has been properly setup using the `Handlebars.template` method.

`Handlebars.unregisterPartial(name)`

Unregisters a previously registered partial.

```
Handlebars.unregisterPartial('foo');
```

`Handlebars.registerHelper(name, helper)`

Registers helpers accessible by any template in the environment.

```
Handlebars.registerHelper('foo', function() {
});
```

Also supports registering multiple helpers at once.

```
Handlebars.registerHelper({
  foo: function() {
  },
  bar: function() {
  }
});
```

`Handlebars.unregisterHelper(name)`

Unregisters a previously registered helper.

```
Handlebars.unregisterHelper('foo');
```

`Handlebars.registerDecorator(name, helper)`

Registers a decorator accessible by any template in the environment.

```
Handlebars.registerDecorator('foo', function() {
});
```

Also supports registering multiple decorators at once.

```
Handlebars.registerDecorator({
  foo: function() {
  },
  bar: function() {
  }
});
```

`Handlebars.unregisterDecorator(name)`

Unregisters a previously registered decorator.

```
Handlebars.unregisterDecorator('foo');
```

`Handlebars.SafeString(string)`

Prevents `string` from being escaped when the template is rendered.

```
new Handlebars.SafeString('<div>HTML Content!</div>')
```

When constructing the string that will be marked as safe, any external content should be properly escaped using the `Handlebars.escapeExpression` method to avoid potential security concerns.

`Handlebars.escapeExpression(string)`

HTML escapes the passed string, making it safe for rendering as text within HTML content.

```
Handlebars.Utils.escapeExpression(string)
```

Replaces `&`, `<`, `>`, `"`, `'`, `` ` ``, `=` with the HTML entity equivalent value for string values. `SafeString` values are left untouched.

The output of all expressions except for triple braced expressions are passed through this method. Additionally helpers should use this method when returning HTML content via a `SafeString` instance to prevent possible code injection.

This method is aliased at `Handlebars.Utils.escapeExpression`.

`Handlebars.createFrame(data)`

Used by block helpers to create a child data object.

```
if (options.data) {
  var data = Handlebars.createFrame(options.data);
  data.foo = 'bar';
  options.data = data;
}
```

Helpers that modify the data state should create a new frame when doing so to isolates themselves and avoid corrupting the state of any parents. Generally only one frame needs to be created per helper execution, i.e. the `each` iterator only creates one frame which is reused for all child execution.

`Handlebars.create()`

Creates an isolated Handlebars environment.

```
var OtherHandlebars = Handlebars.create();
```

Each environment has it's own helpers and partials. This is only necessary for use cases that demand distinct helpers or partials. Most use cases can use the root `Handlebars` environment directly.

Templates created for a given environment are bound to that environment. This means that templates that need to run in multiple environments will need to be recompiled or reconstructed via `Handlebars.template` for each environment. This applies to partials as well.

`Handlebars.noConflict()`

Removes this Handlebars instance from the global space, restoring any libraries that may have been previously registered.

```
var myHandlebars = Handlebars.noConflict();
```

This allows for distinct versions of the library to be loaded into one global space without concern for potential version conflicts.

`Handlebars.log(level, message)`

Logger utilized by the `log` helper.

May be overriden if desired.

Utilities

Handlebars offers a variety of utility methods that are exposed through the `Handlebars.Utils`

object.

`Handlebars.Utils.isEmpty(value)`

Determines if a given value is empty.

```
Handlebars.Utils.isEmpty(value)
```

This is used by the native `if` and `with` helpers control their execution flow. Handlebar's definition of empty is any of:

- Array with length 0
- falsy values other than 0

Which is intended to match the Mustache behavior

`Handlebars.Utils.extend(obj, value)`

Simple utility method to augment `obj` with all keys defined on `value`

```
Handlebars.Utils.extend(foo, {bar: true})
```

Will set the key `bar` on object `foo` with the value `true`

`Handlebars.Utils.toString(obj)`

Generic `toString` method.

`Handlebars.Utils.isArray(obj)`

Determines if an obj is an array.

`Handlebars.Utils.isFunction(obj)`

Determines if an obj is a function.

@data Variables

The following `@data` variables are implemented by Handlebars and its builtin helpers.

`@root`

Initial context that the template was executed with.

```
{{#each array}}
  {{@root.foo}}
{{/each}}
```

Unless explicitly modified this value is consistent across all portions of the page rendering, meaning it's be be used within partials where depthed parameters are unable to reference their parent template.

`@first`

Set to true by the `each` helper for the first step of iteration.

```
{{#each array}}
  {{#if @first}}
    First!
  {{/if}}
{{/each}}
```

`@index`

Zero-based index for the current iteration step. Set by the `each` helper.

```
{{#each array}}
  {{@index}}
{{/each}}
```

`@key`

Key name for the current iteration step. Set by the `each` helper when iterating over objects.

```
{{#each array}}
  {{@key}}
{{/each}}
```

`@last`

Set to true by the `each` helper for the last step of iteration.

```
{{#each array}}
  {{#if @last}}
    Last :(
  {{/if}}
{{/each}}
```

`@level`

Assigned log level.

```
template({}, {data: {level: Handlebars.logger.WARN}})
```

May be set to one of `Handlebars.logger.DEBUG` , `Handlebars.logger.INFO` , `Handlebars.logger.WARN` , or `Handlebars.logger.ERROR`

When set the logger will only output when the `Handlebars.logger.level` value is set to that value or more verbose. This value defaults to logging only error mode.