CGIKit ユーザーガイド

目次

5章 セッション管理

6章 アプリケーションの運用

インストール

動作環境

- UNIX 系の OS
- Ruby 1.8.0 以上

以下は必須ではありませんが、合わせて使うと高速化が可能です。

- mod_ruby

インストール

`install.rb` を使うか、ライブラリを Ruby のパスが通っているディレクトリに手動でコピーしてください。インストールしなくても CGIKit は使うことができます。

リスト 1-1：install.rb でインストール

```
% tar xzf cgikit-xxx.tar.gz
% cd cgikit-xxx
% ruby install.rb config
% sudo ruby install.rb install
```

リスト 1-2：ライブラリをコピー

```
% tar xzf cgikit-xxx.tar.gz
% cd cgikit-xxx
% sudo cp lib/* /usr/local/lib/ruby/lib/site_ruby/1.8
```

ライブラリのロード

CGIKit を使うには、`cgikit.rb` をロードします。インストールしない場合は事前にライブラリのパスに `cgikit.rb` のあるディレクトリを追加して下さい。

リスト 1-3：CGIKit をインストールしてある場合

```
require 'cgikit'
```

リスト 1-4：CGIKit をインストールしない場合

```
# CGIKit のパス
$LOAD_PATH << './cgikit'
require 'cgikit'
```

アーキテクチャ

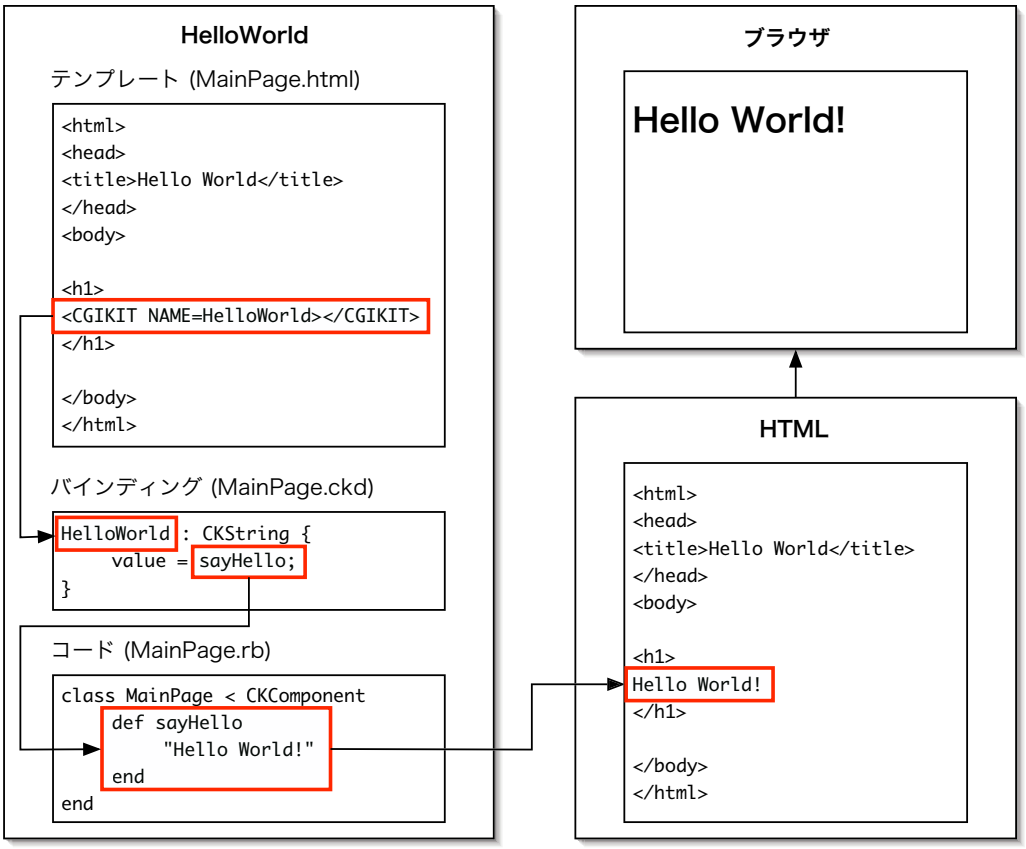
基本的な概念

CGIKit ではコンポーネントとエレメントが中心となります。

コンポーネントとは HTML ファイルと Ruby ソースファイルをまとめたものです。通常の CGI プログラムでは HTML がプログラムに埋もれてしまうことがありますが、CGIKit のコンポーネントでは別々のファイルに分離しています。

エレメントとは CGIKit 独自のタグのようなもので、コンポーネント内の HTML ファイルに埋め込んで使います。埋め込んだエレメントは Ruby プログラムのメソッドと結び付けることで（バインディング）、ページ表示時にメソッドの内容を HTML として出力します。

図 2-1：アーキテクチャ



エレメントとメソッドを結び付けながらコンポーネントを開発するのが CGIKit を使った基本的な開発スタイルです。エレメントにはメソッドの実行結果を表示するものから条件判断をするものまで、様々な機能のものを用意しています。

コンポーネント

コンポーネントは Web ページに相当するものです。クライアントからリクエストが来ると、CGIKit はコンポーネントを HTML に変換して Web ページを表示します。複数のコンポーネントを使って 1 つの Web ページを組み立てることもできます。

構成

コンポーネントは「テンプレート (.html)、バインディング (.ckd)、コード (.rb)」の 3 つのファイルで構成します。テンプレートは HTML ファイル、コードは Ruby ソースファイルです。バインディングとは、テンプレートとコードの内容を結び付けるファイルです。エレメントの定義を行います。

また、コンポーネントのファイルは 3 つとも同じディレクトリに置きます。コンポーネント名のディレクトリを作り、拡張子のみが異なる 3 つのファイルを置きます。

リスト 2-1：ディレクトリ構成

```
HelloWorld.cgi
/MainPage
  MainPage.html
  MainPage.ckd
  MainPage.rb
```

テンプレート (.html)

HTML の記述と、エレメントの配置をします。エレメントは CGIKIT 要素を用いて配置します。CGIKIT 要素は大文字小文字を問いません。

リスト 2-2：HelloWorld(MainPage.html)

```
<html>
<head>
<title>Hello World</title>
</head>
<body>

<h1>
<cgikit name="HelloWorld"></cgikit>
</h1>
```

```
</body>
</html>
```

CGIKIT 要素の属性は、name 属性 1 つだけです。ここで指定した名前は、次の定義ファイルで使用します。name 属性のデータはダブルクォーテーション ("") もしくはシングルクォーテーション (') で囲むことができます（囲まなくても構いません）。また、内容の必要ないエレメントは空要素タグにすることもできます。CKString や CKSubmitButton などのエレメントは内容を必要としませんから、空要素タグで書くと簡単です。

リスト 2-3：空要素タグ

```
<cgikit name="HelloWorld"/>
```

コメント

CGIKIT 要素を無効にしたいときや、実行時に残したくないコメントを書きたいときは <!--- ... ---> を使用します。HTML のコメント (<!-- ... -->) と違い、ハイフンを 3 つ並べます。このタグは実行時にはすべて削除されます。

HTML のコメント中の CGIKIT 要素は通常通り変換されるので、スタイルシートや JavaScript にもエレメントを使うことができます。

国際化

テンプレートを用意することで、コンポーネントを複数のロケール（言語）に対応させることができます。ロケールごとのテンプレートのファイル名は、「テンプレート + "_" + ロケール名」です。ロケールはブラウザの言語設定によって自動的に判断されます（国際化の仕様は今後のバージョンで変更する可能性があります)。

バインディング (.ckd)

このファイルでエレメントの内容を定義します。バインディングファイルの文法は HTML や Ruby と異なるので注意してください。エレメントの属性全体を大カッコ ({}) で囲み、各属性を 1 行に書くか、セミコロン (;) で区切ります。シャープ (#) から行末までコメントになります。

リスト 2-4：文法

```
定義名 : 種類 {
  # コメント
  属性 = 値

  # セミコロンで区切る
  属性 = 値 ; 属性 = 値 ;
}
```

値には、メソッドまたは文字列を指定します。文字列を定義するには、ダブルクォーテーション (") もしくはシングルクォーテーション (') で囲みます。エレメントの中には `true/false` を指定する属性もありますが、これらはダブルクォーテーションで囲む必要はありません。

リスト 2-5：HelloWorld(MainPage.ckd)

```
HelloWorld : CKString {
  value = sayHello;
}
```

このバインディングでは、CKString というエレメントを定義しています。`value` 属性に `sayHello()` メソッドを指定しています。`sayHello()` は `HelloWorld` クラスのメソッドです。

リテラル

バインディングファイルでは、属性の値に以下のリテラルを使うことができます。

表 2-1：リテラル

リテラル	書式例
文字列	'abc', "string", ...
数字	1, 2, 3, 4, 5, ...
true/false, nil	true, false, nil, ...
配列	[array], [0], [1], ...
ハッシュ	[key], ['abc'], ["string"], [0], [1], ...

コード (.rb)

Ruby ソースファイルです。クラス名はコンポーネント名と同じ名前にし、必ず `CKComponent` クラスを継承します。また、インスタンス変数をバインディングするには、アクセサを定義しておきます。

リスト 2-6：HelloWorld(MainPage.rb)

```
class MainPage < CKComponent
  def sayHello
    "Hello World!"
  end
end
```

コード内で初期化を行う場合、通常の `initialize()` ではなく `init()` メソッドを使用します。`init()` は引数のいらない初期化用メソッドです。`initialize()` をオーバーライドしても初期化はできますが、引数が多いことと `super()` をすぐに呼ぶ必要があるなど、多少面倒になります。

バインディングの流れ

CGIKit は実行時にテンプレート内のエレメントをコードと結びつけ、実行結果を HTML に変換して表示します。コンポーネントは以下の順序で HTML を出力します。

1. テンプレートを読み込む
2. `CGIKIT` 要素を見つけたら、バインディングから該当する定義を取得する
3. 定義されたエレメントをコードとバインディングする
4. エレメントを HTML に変換し、出力する

アクセサ

エレメントにバインディングしたインスタンス変数のアクセサメソッドを定義する必要はありません。CGIKit は定義されていればアクセサを使って操作しますが、そうでなければインスタンス変数を直接操作します。

フォームデータ

フォームのテキストフィールドやボタンにバインディングしたインスタンス変数には、自動的にフォームデータが代入されます。

エレメント

エレメントとは、インスタンス変数やメソッドを HTML として表示する仕組みです。エレメントを使い分けることが CGIKit による開発の核になります。

コンポーネントはエレメントの一部なので、コンポーネントをネストすることができます。この場合、複数のコンポーネントから 1 つの Web ページを構成することになります。トップレベルのコンポーネントは各コンポーネント・エレメントをすべて HTML に変換した後に展開されます。

属性

各エレメントには、動作を指定する属性があります。これらの属性にインスタンス変数やメソッドを指定（バインディング）することで、プログラムを HTML に埋め込むことができるようになります（以降、バインディングするメソッドのことを「アクション」とします）。

エレメント一覧

エレメントは全部で 19 種類あります。最も多用するエレメントに CKString があり、このエレメントはバインディングしたアクションの内容（実行結果）を文字列に変換して表示します。ほかにもバインディングの結果によって CGIKIT タグで囲んだデータの表示を制御する CKConditional やバインディングした配列データを繰り返し表示する CKRepetition など、様々なエレメントがあります。

表 2-2：一般

エレメント	概要
CKString	バインディングしたアクションの結果を表示する。
CKHyperlink	他コンポーネントやメソッドにリンクを張る。
CKImage	リソースディレクトリ内の画像を表示する。

表 2-3：条件判断・繰り返し

エレメント	概要
CKConditional	設定した条件の結果によって HTML を表示する。
CKRepetition	指定した範囲の内容を繰り返す。

表 2-4：フォーム

エレメント	概要
CKForm	フォームを用意する。送信データはそれぞれエレメントにバインディングした変数に代入される。
CKTextField	テキストフィールドを表示する。
CKRadioButton	ラジオボタンを表示する。
CKCheckbox	チェックボックスを表示する。
CKPopUpButton	ポップアップボタンを表示する。
CKText	テキストエリアを表示する。
CKBrowser	複数選択可能なリストを表示する。
CKFileUpload	ファイルアップロードフィールドを表示する。

表 2-4：フォーム

エレメント	概要
CKSubmitButton	送信ボタンを表示する。
CKResetButton	リセットボタンを表示する。

表 2-5：コンポーネントの再利用

エレメント	概要
CKFrame	フレームにコンポーネントを設定する。
CKComponent	コンポーネント内に別のコンポーネントを設定する。
CKContent	ネスティングしたコンポーネントにて、親コンポーネントを表示する。
CKGenericElement	一般的な HTML タグを生成する。

リソースの扱い

CGIKit アプリケーションで使う画像や設定ファイルなどのリソースは、リソースディレクトリに配置します。リソースディレクトリには通常のリソースディレクトリと Web サーバリソースディレクトリがあり、それぞれ `CKApplication` クラスの `resources` 属性と `web_server_resources` 属性で設定します。

通常のリソースと Web サーバリソース

通常のリソースは設定ファイルなどのブラウザに送る必要のないファイル、Web サーバリソースは画像ファイルなどブラウザに直接送るファイルです。リソースディレクトリはどこに配置しても構いませんが、Web サーバリソースディレクトリは Web サーバのドキュメントルート下に配置してください

リソースへのアクセス

リソースへアクセスするには `CKResourceManager` オブジェクトを使います。`CKResourceManager`オブジェクトは`CKApplication#resource_manager`メソッドで取得できます。

`CKResourceManager` の主なメソッドは `url()` と `path()` です。`url()` はリソースの URL を、`path()` は絶対ファイルパスを返します。ただし、`url()` は Web サーバリソースディ

レクトリにあるファイルのみが対象となります。通常のリソースディレクトリにあるファイルへの URL は取得できません。

表 2-6：CKResourceManager のメソッド

メソッド	説明
`url(name)`	Web サーバリソースの URL を返す。通常のリソースを指定した場合は `nil` を返す。
`path(name)`	リソースの絶対ファイルパスを返す。
`bytedata(name)`	リソースを CKByteData オブジェクトとして返す。
`content_type(path)`	リソースの Content-Type を返す。Content-Type は拡張子から判断する。

例：画像ファイルを表示する

CKImage エレメントを使ってリソースファイルを表示することができます。付属するサンプルアプリケーション Examples の `ImapePage` で画像ファイル (`cgikit.png`) を表示するには、以下の手順で設定します。

1. `resources` ディレクトリを画像の表示できるパスに移動する
2. `CKAppication#web_server_resources` 属性に `resources` ディレクトリのパスを設定する
3. CKImage の `file` 属性に画像ファイルのパス（`resources` ディレクトリからの相対パス）を設定する

リスト 2-7：`resources` ディレクトリを移動する

```
[localhost:/var/www/cgi-bin/Examples] user% mv resources ../../
htdocs
```

リスト 2-8：`resources` ディレクトリのパスを設定する (`Example.cgi`)

```
app = CKApplication.new
app.web_server_resources = '../../htdocs/resources'
app.run
```

リスト 2-9：file 属性の画像ファイル名を設定する (ImagePage.ckd)

```
FileInResource : CKImage {
  alt  = "File in resource direcory";
  file = "cgikit.png";
}
```

ここではWebサーバリソースディレクトリから静的コンテンツを表示していますが、CKImageのdata属性を使うことで動的に画像を生成することも可能です。

実行の仕組み

起動プログラム

CGIKitを起動するには、コンポーネントとは別にCGIプログラムを用意します。通常、コンポーネント内のファイルに直接アクセスすることはありません。

起動プログラムで行うことは3つあります。

1. CKApplicationのインスタンスを生成する
2. 環境変数を設定する
3. CKApplication#run()を実行する

CKApplicationのインスタンスを生成する

CKApplicationは、メインページやCGIプログラムのパスなど実行に必要な環境変数を持つ、プログラムを起動するために必要なクラスです。プログラムを起動するには、まずCKApplicationのインスタンスを生成します。

リスト2-10：CKApplicationのインスタンスを生成する

```
app = CKApplication.new
```

環境変数を設定する

CKApplicationには以下の環境変数があります（すべてアクセサを用意しています）。

表2-7：環境変数

環境変数	説明
baseurl	Webサーバ上のCGIプログラムのパス。デフォルトは環境変数SCRIPT_NAME。
path	ファイルシステム上のCGIプログラムのパス。デフォルトは起動プログラムのあるディレクトリ。
component_path	コンポーネントを置くディレクトリ。デフォルトは起動プログラムのあるディレクトリ。
resource	画像を置くディレクトリのパス。

表 2-7：環境変数

環境変数	説明
main	何も指定がないときに表示するページ。デフォルトは MainPage コンポーネント。
locale	ロケール。コンポーネントの表示時、設定されているロケール表記のある HTML ファイルを使う。
master_locale	主ロケール。ロケールが設定されていない場合、または設定されているロケールと主ロケールが同じ場合、ロケール表記のない HTML ファイルを使う。
tmpdir	一時ファイルのディレクトリ。セッションファイルなどを保存する。デフォルトは tmp ディレクトリ。
error_page	エラーページ。デフォルトは CKErrorPage。
manage_session	セッションの自動管理。デフォルトでは自動管理を行わない。
session_key	セッションキー。デフォルトは _session_id。
session_id	セッション ID。
store_in_url	セッション ID を URL に付加して保持する。デフォルトは有効。
store_in_cookie	セッション ID をクッキーで保持する。デフォルトは有効。
session_cookie_expires	セッション ID の保存に使うクッキーの有効期限。nil に設定したとき、セッションの有効期間はブラウザを閉じるまでになる。デフォルトは 1 週間。
timeout	セッションの有効期限（秒）。デフォルトでは 1 週間保存する（604800 秒）。
auth_by_user_agent	セッションをブラウザで認証する。デフォルトは無効。
auth_by_remote_addr	セッションを IP アドレスで認証する。デフォルトは無効。
char_code	文字コード。フォームデータをこの文字コードに変換する。"jis", "sjis", "euc" から選択する。

CKApplication#run を実行する

環境変数を設定したら、最後に CKApplication#run を実行します。

例：HelloWorld.cgi

以下は HelloWorld の起動プログラムです。

リスト 2-11：HelloWorld (HelloWorld.cgi)

```
#!/usr/local/bin/ruby

require 'cgikit'

app = CKApplication.new
app.run
```

コンポーネントの起動

ここから先は CKApplication クラスの起動処理です。通常意識する必要はありません。

`CKApplication#run` は必要な初期化の処理を行った後に、表示するコンポーネントを生成します。通常メインコンポーネントは `MainPage`（デフォルトの設定）ですが、CKHyperlink などで起動するコンポーネントが指示されていればそちらを起動します。

フォームデータのバインディング

生成されたコンポーネントはフォームデータをインスタンス変数に代入します。

アクションの実行

ここはループして実行されます。まず、指定されたアクションが実行されます。このときアクションの戻り値がコンポーネント（CKComponent）であれば再びそのコンポーネントの起動処理を行い、戻り値がコンポーネント以外のオブジェクト（`nil` など）になるまでループします。

HTTP レスポンスヘッダの出力

HTML を出力する前に HTTP レスポンスヘッダを出力します。

コンポーネントの表示

最後に `CKComponent#to_s()` を実行し、コンポーネントの表示を行います。このメソッドはコンポーネントを HTML 出力します。

エレメント

任意の HTML 属性と other 属性

すべてのエレメントには、任意の HTML 属性と other 属性を設定することができます（ただし、CKStringや CKRepetition などのエレメントでは意味がありません)。HTML の属性名と値をエレメントの属性として設定すると、出力する HTML 要素に設定した属性が追加されます。

一方 other 属性は設定した文字列をそのまま HTML要素に追加します。値を持たない属性を設定するときなど、任意の HTML 属性と組み合わせて使います。

リスト 3-1：任意の HTML 属性と other 属性を設定する

```
Link : CKHyperlink {
  href  = “http://www.foobar.com/“
  key   = “value”
  other = “anykeywords“
}

# <a href=”http://www.foobar.com/” key=”value” anykeywords></a>
```

入力値の検証

ユーザ入力を扱うエレメント（CKTextField と CKText）では、validate 属性と pass 属性を使って入力値を検証することができます。入力値が検証にパスすれば pass 属性の変数が真になりますが、そうでなければ偽が代入されます。

例：メールアドレスを検証する

例としてメールアドレスを入力するテキストフィールドを検証します。入力したメールアドレスが存在するかどうかを調べるにはコードを書かないとできませんが、フォーマットを検証すればメールアドレスではないデータが入力されたかどうかわかります。

ここでは単純に @ が含まれているかどうか検証することにします。もし @ が含まれていなければ、変数 pass_mail に false が代入されます。

リスト 3-2：メールアドレスの検証

```
Mail : CKTextField {
  value    = mail
  validate = "mail =~ /[^@]+@(.+)/"
  pass     = pass_mail
}
```

この例は Registration アプリケーションとしてサンプルに含まれています。

ルールのフォーマット

ルールの例をいくつか示します。

リスト 3-3：ルールの例

```
name == 'MyName'
(title =~ /R/) and (title.size > 10)
not (count < 20)
```

基本的に「属性　オペレータ　条件」でルールを記述します（属性はコンポーネントで定義されているアクセサかインスタンス変数です）。複数のルールをつなげるには and/or を、否定にするには not を式の前に記述します。論理演算子を使うときは、式をかっこで囲んでください。

データ型の変換

上記のうち、最後の例では属性を数値としてルールを設定しています。フォームデータは文字列として代入されますが、データの検証時は条件に合わせてデータ型が変換されます。例えば 100 以上 500 以下などのルールは以下のように記述します。

リスト 3-4：入力値を数値として検証する

```
Number : CKTextField {
  value    = number
  validate = "(number >= 100) and (number <= 500)"
  pass     = pass_number
}
```

オペレータ

使用できるオペレータを次に示します。

表 3-1：条件式に使用できるオペレータ

オペレータ	説明
==	両辺が同じ。
!=	両辺が異なる。
>	左辺が右辺より大きい。
<	左辺が右辺より小さい。
>=	左辺が右辺より大きいか、同じ。
<=	左辺が右辺より小さいか、同じ。
=~	文字列のパターンマッチを行う。

一般

CKString

バインディングしたメソッドの戻り値を表示します。

必須属性：`value`

表 3-2：CKString の属性

属性	データ型	説明
`value`	`String`	バインディングしたアクションの戻り値を表示する。
`escape`	`true/false`	HTML 制御文字のエスケープ。true ならばエスケープする。
`empty`	`String`	value 属性が nil のときに、このデータを表示する。

CKHyperlink

他のページにリンクしたり、クリックするとアクションを実行するリンクを生成します。

必須属性：action, href または page

表 3-3：CKHyperlink の属性

属性	データ型	説明
action	CKComponent	バインディングしたメソッドはクリック時に実行される。
enabled	true/false	リンクの判断。false のとき、文字は表示するがリンクはされない。
href	String	直接 URL を指定する。この属性は action・page 属性より優先される。
page	String	指定したコンポーネントへのリンクを生成する。コンポーネントは文字列で指定する。
string	String	リンクに使う文字列。設定されていなければ <CGIKIT> タグで囲んだ文字列を使う。
target	String	フレームを指定する。
secure	true/false	暗号化。true であれば生成する URL が "https" で始まる。
query	Hash	URL に付加するクエリ文字列。

CKImage

画像を表示します。画像は Web サーバリソースディレクトリのものを使います。

必須属性：file, src または data

表 3-4：CKImage の属性

属性	データ型	説明
alt	String	画像が表示されない場合のための、画像の説明。
border	Integer	画像に枠をつける。
width	Integer	画像の横幅。
height	Integer	画像の縦幅。
file	String	画像のファイル名。Web サーバリソースディレクトリのものを使う。
src	String	画像のパスを直接指定する。この属性は file 属性より優先される。

表 3-4：CKImage の属性

属性	データ型	説明
data	CKByteData	画像として表示する CKByteData オブジェクト。もし CKByteData オブジェクトがリソースマネージャーから取得したものでなければ、mime 属性を同時に設定する必要がある。
mime	String	画像の MIME タイプ。data 属性を使うときに設定する。

条件判断・繰り返し

CKConditional

条件の結果によって、タグで囲んだ内容の表示を制御します。特定の箇所を条件に応じて表示したり隠す場合に使います。

必須属性：condition

表 3-5：CKConditional の属性

属性	データ型	説明
condition	true/false	内容の表示。true であれば表示する。
negate	true/false	condition の動作を逆にする。つまりこの属性が true のとき、condition 属性が false であれば内容を表示する。

表 3-6：condition 属性と negate 属性

condition	negate	結果
true	false	表示する
false	false	表示しない
true	true	表示しない
false	true	表示する

CKRepetition

タグで囲んだ内容を繰り返し表示します。

必須属性：list と item、または count

表 3-7：CKRepetition の属性

属性	データ型	説明
count	Integer	繰り返す回数。設定した回数分繰り返す。
list	Enumerable	繰り返すデータ。この配列から要素を順に取り出し、item 属性に入れる。
item		list 属性から取り出した要素。
index	Integer	現在の要素数。list 属性を繰り返している回数。

フォーム

CKForm

フォームを生成します。フォームに関するエレメントは CKForm 内にないと動作しないので注意して下さい。必須属性はありません。

ファイルのアップロードを行うには、enctype 属性に "multipart/form-data" を設定するか、fileupload 属性を true に設定します。マルチパートフォームを設定しても、フォームデータは通常の文字列（String クラス）として扱われますが、CKFileUpload で扱うデータ（Content-Type が設定されているデータ）のみ CKByteData オブジェクトとして扱われます。

表 3-8：CKForm の属性

属性	データ型	説明
method	String	フォームデータの送信方法。POST または GET を選択する。何も設定しないと POST になる。
enctype	String	フォームデータのエンコード方法。CKFileUpload を使う場合に "multipart/form-data" を設定する。
href	String	フォームデータを送信するプログラムを直接指定する。action 属性より優先される。
target	String	フレームを指定する。
query	Hash	URL に付加するクエリ文字列。

表 3-8：CKForm の属性

属性	データ型	説明
fileupload	true/false	true を設定すると、enctype 属性に "multipart/form-data" が設定される。ファイルのアップロードをするとき、enctype 属性の代わりに使うことができる。

CKTextField

テキストフィールドを表示します。入力されたデータは value 属性に設定したインスタンス変数に代入されます。

必須属性：value

表 3-9：CKTextField の属性

属性	データ型	説明
type	String	フィールドの種類を指定する。指定できるのは「テキストフィールド (text)、パスワード (password)、隠蔽フィールド (hidden)」の 3 つ。
value	String	データの表示・代入を行う。
size	Integer	テキストフィールドの幅。
maxlength	Integer	データの最大入力文字数。
validate	String	入力値のルールを指定する。ルールに適合しなければ pass 属性に false が代入される。
pass	true/false	入力値の検証結果。ルールに適合すれば true が、そうでなければ false が代入される。
enabled	true/false	入力の可否。false であれば入力できないようにする。

CKText

テキストエリアを表示します。

必須属性：value

表 3-10：CKText の属性

属性	データ型	説明
value	String	データの表示・代入を行う。
columns	Integer	テキストエリアの横幅。
rows	Integer	テキストエリアの縦幅。
validate	String	入力値のルールを指定する。ルールに適合しなければ pass 属性に false が代入される。
pass	true/false	入力値の検証結果。ルールに適合すれば true が、そうでなければ false が代入される。
enabled	true/false	入力の可否。false であれば入力できないようにする。

CKCheckbox

チェックボックスを表示します。

必須属性：selection と value，または checked

表 3-11：CKCheckbox の属性

属性	データ型	説明
checked	true/false	チェックボックスのオン／オフ。変数をバインディングしておくと、チェックされたときに true が代入される。
value	String	チェックしたときに selection 属性に代入するデータ。value 属性と selection 属性のデータが同じならチェックボックスはオンになる。
selection	Array	チェックしたとき、value 属性のデータがこの属性に代入される。
enabled	true/false	入力の可否。false であればチェックできないようにする。

CKCheckbox と CKRadioButton は、フィールドより少し複雑になります。これらのエレメントは「checked 属性」か「value 属性と selection 属性」のどちらかの組み合わせを使います。

checked 属性

checked 属性を使う場合は、単純にオン／オフのみでチェックボックスを操作します。つまり、1 つのチェックボックスにつき 1 つの変数を割り当てることになります。

リスト 3-5：テンプレート

```
<cgikit name=Form>
<cgikit name=Checkbox1>One</cgikit>
<cgikit name=Checkbox2>Two</cgikit>
<cgikit name=Checkbox3>Three</cgikit>
<cgikit name=Submit/>
</cgikit>
```

リスト 3-6：バインディング

```
Form : CKForm {
}

Checkbox1 : CKCheckbox {
  checked = checkedOne;
}

Checkbox2 : CKCheckbox {
  checked = checkedTwo;
}

Checkbox3 : CKCheckbox {
  checked = checkedThree;
}

Submit : CKSubmitButton {
}
```

リスト 3-7：コード

```
class Checkbox < CKComponent
  attr_accessor :checkedOne, :checkedTwo, :checkedThree
end
```

このように checked 属性にそれぞれ別の変数を割り当てて、どのチェックボックスがチェックされているか判断します。チェックされると、変数には true が代入されます。

value 属性と selection 属性

value 属性にチェックボックスのデータを設定すると、そのデータがチェックしたとき selection 属性に代入されます。また、value 属性と selection 属性のデータが同じであればチェックボックスはオンになります。

基本的な動作は checked 属性を使う場合と変わりませんが、チェック時に代入されるデータが異なります。checked 属性では true が代入されますが、この場合value 属性で設定したデータが代入されることになります。

リスト 3-8：テンプレート

```
<cgikit name=Form>
<cgikit name=Checkbox1>One</cgikit>
<cgikit name=Checkbox2>Two</cgikit>
<cgikit name=Checkbox3>Three</cgikit>
<cgikit name=Submit></cgikit>
</cgikit>
```

リスト 3-9：バインディング

```
Form : CKForm {
}

Checkbox1 : CKCheckbox {
  value = "One";
  selection = checkedOne;
}

Checkbox2 : CKCheckbox {
  value = "Two";
  selection = checkedTwo;
}

Checkbox3 : CKCheckbox {
  value = "Three";
  selection = checkedThree;
}

Submit : CKSubmitButton {
}
```

リスト 3-10：コード

```
class Checkbox < CKComponent
  attr_accessor :checkedOne, :checkedTwo, :checkedThree
end
```

CKRadioButton

ラジオボタンを表示します。基本的には CKCheckbox と同じですが、CKRadioButton ではグループの各ボタンを排他的にするため name 属性を設定します。

必須属性：selection と value、または checked

表 3-12：CKRadioButton の属性

属性	データ型	説明
name	String	ラジオボタンのグループ名。同じ名前のラジオボタンを 1 つしかチェックできないようにする。
checked	true/false	ラジオボタンのオン／オフ。変数をバインディングしておくと、チェックされたときに true が代入される。
value	String	チェックしたときに代入するデータ。
selection	String	チェックしたとき、value 属性のデータがこの属性に代入される。
enabled	true/false	入力の可否。false であればチェックできないようにする。

以下は checked 属性を使った例です。

リスト 3-11：テンプレート

```
<cgikit name=Form>
<cgikit name=Radio1>One</cgikit>
<cgikit name=Radio2>Two</cgikit>
<cgikit name=Radio3>Three</cgikit>
<cgikit name=Submit></cgikit>
</cgikit>
```

リスト 3-12：バインディング

```
Form : CKForm {
}

Radio1 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}

Radio2 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}

Radio3 : CKCheckbox {
  name = "radio";
  checked = checkedOneTwoThree;
}

Submit : CKSubmitButton {
}
```

リスト 3-13：コード

```
class Checkbox < CKComponent
   attr_accessor :checkedOneTwoThree
end
```

CKPopUpButton

ポップアップメニューを表示します。

必須属性：`list`

表 3-13：CKPopUpButton の属性

属性	データ型	説明
`escape`	`true/false`	HTML 制御文字のエスケープ。
`list`	`Enumerable`	選択項目として繰り返すデータ。
`default`	`String`	何も選択されていないとき、先頭に表示するメニュー。
`selected`	`String`	選択したデータはこの属性に代入される。
`values`	`Array`	`<option>` 要素の `value` 属性に設定されるデータ。選択項目には `list` 属性の内容が表示されるが、`selected` 属性にはこの属性の内容が代入される。
`enabled`	`true/false`	入力の可否。false であれば選択を無効にする。

CKBrowser

複数選択可能なセレクトボックスを表示します。

必須属性：`list`

表 3-14：CKBrowser の属性

属性	データ型	説明
`escape`	`true/false`	HTML 制御文字のエスケープ。
`list`	`Enumerable`	繰り返すデータ。この配列から要素を順に取り出し、item 属性に入れる。
`selected`	`Array`	選択したデータはこの属性に代入される。他のフィールドと違い、データは配列になるので注意。

表 3-14：CKBrowser の属性

属性	データ型	説明
values	Array	<option> 要素の value 属性に設定されるデータ。選択項目には list 属性の内容が表示されるが、selected 属性にはこの属性の内容が代入される。
multiple	true/false	複数選択の可否。true であれば複数選択を可能にする。
size	Integer	一度に表示する選択肢の数。
enabled	true/false	入力の可否。false であればタグを無効にする。

CKSubmitButton

送信ボタンを表示します。必須属性はありません。

表 3-15：CKSubmitButton の属性

属性	データ型	説明
action	CKComponent	指定したメソッドはクリック時に実行される。
value	String	ボタンのラベル。
enabled	true/false	入力の可否。false であればタグを無効にする。

CKResetButton

リセットボタンを表示します。必須属性はありません。

表 3-16：CKResetButton の属性

属性	データ型	説明
value	String	ボタンのラベル。

CKFileUpload

ファイルをアップロードするためのフォームを生成します。このエレメントを使うときは CKForm の enctype 属性に "multipart/form-data" を設定してください。

必須属性：data と file

表 3-17：CKFileUpload の属性

属性	データ型	説明
data	CKByteData	アップロードしたファイルが CKByteData オブジェクトとして代入される。
file	String	アップロードしたファイルのパス。
enabled	true/false	入力の可否。false であればフォームを無効にする。

コンポーネントの再利用

CKFrame

フレームを生成します。

必須属性：page 属性、src 属性、value 属性のうち 1 つだけ設定してください。

表 3-18：CKFrame の属性

属性	データ型	説明
name	String	フレーム名。フレームを使う場合は、各エレメントで同じ名前を target 属性に設定する。
page	String	フレームに使うコンポーネントを設定する。
src	String	フレームに使うファイルを直接指定する。page 属性より優先される。
value	CKComponent	フレームの内容をコンポーネントで表示する。

フレームを扱うには、フレーム専用のコンポーネントを用意します。テンプレートにフレームセットを記述し、フレーム表示したいコンポーネントを CKFrame で定義します。

リスト 3-14：テンプレート

```
<frameset cols="200,*">
<cgikit name=Index></cgikit>
<cgikit name=Contents></cgikit>
</frameset>
```

リスト 3-15：バインディング

```
Index : CKFrame {
  name = "Index";
```

```
  page = "IndexPage";
}

Contents : CKFrame {
  name = "Contents";
  page = "IntroductionPage";
}
```

CKComponent

コンポーネントは再利用が可能です。コンポーネント内に別のコンポーネントをネストすることができます。バインディングファイルにて、エレメントの代わりにコンポーネントを指定してください。

リスト 3-16：MainPage コンポーネントをネストする

```
OtherComponent : MainPage {}
```

コンポーネントはエレメント属性を持ちません。その代わりに、インスタンス変数を属性として扱うことができます。

リスト 3-17：コード (MainPage)

```
class MainPage < CKComponent
  attr_accessor :title
end
```

リスト 3-18：バインディング（親コンポーネント）

```
OtherComponent : MainPage {
  title = "Example for CKComponent";
}
```

上記の例では親コンポーネントが MainPage コンポーネントをネストし、その属性として title を設定しています。

title 属性は MainPage コンポーネントのインスタンス変数であるため、文字列 "Example for CKComponent" が代入されることになります。

CKPartsMaker

コンポーネントは単独で使うこともネストすることもできますが、各コンポーネントを構成する「部品」としてネスティングのみに使うコンポーネントには単独表示させたくないことがあります。このような場合には CKPartsMaker モジュールを使います。

CKPartsMaker はコンポーネント単体で表示を行わないためのモジュールです。このモジュールをインクルードしたコンポーネントは、単独表示のリクエストが来ると代わりの

ページを表示します。部品コンポーネントは、ほかの Web ページコンポーネントと区別するため名前の最後に "Parts" や "Component" などとつけるといいでしょう。

表 3-19：CKPartsMaker モジュールのオブジェクト属性

オブジェクト属性	説明
substitute_page	代替ページ。指定しないときはメインページを表示する。

CKContent

親コンポーネントの位置を指定します。このエレメントは、コンポーネントをネストするときにのみ使います。CKContent は何も属性を持ちません。

リスト 3-19：テンプレート（親コンポーネント）

```
<cgikit name=OtherComponent>Content of parent</cgikit>
```

リスト 3-20：バインディング（親コンポーネント）

```
OtherComponent : MainPage {}
```

リスト 3-21：テンプレート（子コンポーネント）

```
<b><cgikit name=Content></cgikit></b>
```

リスト 3-22：バインディング（子コンポーネント）

```
Content : CKContent {}
```

リスト 3-23：出力

```
<b>Content of parent</b>
```

CKGenericElement

CKGenericElement は一般的な HTML タグを生成します。

必須属性：tag

表 3-20：CKGenericElement の属性

属性	データ型	説明
tag	String	HTML タグ名。nil のときタグは生成されず、要素で囲んだ文字列か "string" 属性が表示される。

表 3-20：CKGenericElement の属性

属性	データ型	説明
enabled	true/false	タグを表示するか否か。false のときタグは生成されず、要素で囲んだ文字列か "string" 属性が表示される。
string	String	<cgikit> 要素で囲んだ内容がないときに表示される文字列。
option	String	開始タグに追加する文字列。"checked" や "selected" などを設定する。
form_value	String	フォームの場合、この変数にフォームデータの文字列を代入する。
form_values	Array	フォームの場合、この変数にフォームデータの配列を代入する。
invoke_action	CKComponent	エレメントが実行可能な場合（リンクやボタンなど）、クリック時にメソッドを実行する。

CKGenericElement には、以上のほかに任意の属性を定義することができます。定義した属性は「属性 = 値」の形式でタグに追加されます。

クッキー

CGIKitでは、CKCookieクラスによりクッキーを扱います。CKCookieクラスで作成するか、またはCKRequestオブジェクトから取得したCKCookieオブジェクトをCKResponseオブジェクトに設定するのが基本的な流れです。CKResponseオブジェクトに設定されたクッキーは、実行時にブラウザへ送信されます。

CKCookieクラス

CKCookieオブジェクトは1組の名前とパラメータ（文字列）を持ちます。1つのクッキーに複数のパラメータを持たせたい場合は、自分でエンコード・デコードするコードを書く必要があります。

表4-1：属性

属性	説明
`name`	クッキーの名前
`value`	パラメータ（文字列）
`path`	Webサイトのパス
`domain`	ドメイン
`expires`	クッキーの有効期限（Timeオブジェクト）
`secure`	HTTPS接続の指定

クッキーの操作

クッキーを作成する

クッキーの名前とパラメータを与えてCKCookieオブジェクトを作成します。クッキー名は必須ですが、パラメータは省略しても構いません。

リスト4-1：クッキーを作成する

```
cookie = CKCookie.new( name, value )
```

クッキーを取得する

ブラウザに設定されているクッキーは CKRequest オブジェクトから取得することができます。CKRequest オブジェクトは、CKApplication#request 属性で取得できます。

表 4-2：CKRequest クラスのクッキー取得メソッド

メソッド	説明
`cookie(key)`	`key` の名前のクッキーを返す。クッキーが複数ある場合は、最初に見つかったものを返す。
`cookies`	すべてのクッキーを配列で返す。
`cookie_value(key)`	名前が `key` のクッキーのパラメータを返す。クッキーが複数ある場合は、最初に見つかったものを返す。
`cookie_values(key)`	名前が `key` のすべてのクッキーのパラメータを配列で返す。`key` を指定しない場合は、すべてのクッキーのパラメータを返す。

クッキーを設定する

生成したクッキーは CKResponse#add_cookie で CKResponse オブジェクトに追加します。クッキーを置き換えるのではなく、以前に設定したクッキーはそのままに新しいクッキーを追加していくことになります。

リスト 4-2：クッキーを追加する

```
cookie = CKCookie( 'name' )
application.response.add_cookie( cookie )
```

クッキーをブラウザから削除する

クッキーをブラウザから削除するには、同じ名前を持つクッキーを送信して上書きします。その際に有効期限を過去に設定しておくと、ブラウザから完全にクッキーが削除されます。

リスト 4-3：クッキーをブラウザから削除する

```
cookie = CKCookie "name"
cookie.expire = Time.new - 60
response.add_cookie cookie
```

クッキーを CKResponse オブジェクトから削除する

CKResponse オブジェクトに設定したクッキーは CKResponse#remove_cookie で削除します。

リスト 4-4：一度セットしたクッキーを削除する

```
application.response.remove_cookie( 'name' )
```

セッション管理

セッション管理を行うには、CKApplication・CKSession クラスを使います。CKSession オブジェクトは、任意のオブジェクトを格納したハッシュとブラウザ名や IP アドレスなどの情報を持ちます。ただしデフォルトのデータベースマネージャ CKSessionStore::FileStore では、Marshal できないオブジェクト（IO、Proc など）を格納することはできません。

自動管理

セッションは自動的に管理することが可能です。自動管理時は CGIKit 側でセッションの読み込みと保存が行われます。自動管理を設定すると、CGIKit アプリケーションにアクセスしたときに必ずセッションが生成されます。

自動管理を有効にするには、CKApplication#manage_session 属性を true に設定にしてください（デフォルトは false です)。

基本操作

セッションを取得する

セッションは CKApplication#session で取得します。セッションが存在しなければ新しいセッションが生成されます。

リスト 5-1：セッションを取得する

```
session = application.session
p session #-> <CKSession:0x....>
```

セッションデータの取得と設定

ハッシュ形式でデータの取得と設定を行います。

リスト 5-2：セッションデータの取得と設定

```
session['key']     =   'value'
session['array']   =   [1,2,3,4,5]
p session['key']   #-> 'value'
p session['array'] #-> [1,2,3,4,5]
```

セッションを保存する

CKApplication#save_session を実行します。ただし、セッションの自動管理を有効にしている場合は必要ありません。

セッションを終了、または削除する

セッションの削除は手動管理時と自動管理時で方法が異なるので注意してください。手動管理時は CKApplication#clear_session を、自動管理時は CKSession#clear を実行します。

CKSession#clear はセッション削除用のフラグを立てます。実行するとセッションデータは空になりますが、セッション自体は削除されません。自動管理によってセッションが保存されるときに完全に削除されます。

セッション ID の保管

セッション ID は URL またはクッキーを使って保管します。このとき発行されるクッキーの有効期限は `session_cookie_expires` で設定できます。

保管方法は以下の属性で設定します。

表 5-1：セッション ID の保管方法（CKApplication クラス）

属性	デフォルト	説明
`store_in_url`	`true`	セッション ID を URL に埋め込む。
`store_in_cookie`	`true`	セッション ID をクッキーに埋め込む。
`session_cookie_expires`	`604800`（1 週間）	セッション ID の保存に使うクッキーの有効期限。nil に設定したとき、セッションの有効期間はブラウザを閉じるまでになる。

認証

有効期限による認証

セッションには有効期限を設定することができます。有効期限が経過したセッションでアクセスすると、例外 SessionTimeoutError を発生します。

有効期限は CKApplication#timeout 属性に秒数で指定します。セッションに最後にアクセスした時間から timeout 秒後がタイムアウトになります。セッションを無期限に有効にしたい場合は、timeout 属性に 0 を設定してください。また、指定されたセッション ID が存在しない場合もタイムアウト扱いになります。

ブラウザと IP アドレスによる認証

ユーザーのブラウザと IP アドレスで認証を行うことができます。セッション生成時と異なるブラウザや IP アドレスでセッションにアクセスすると、例外 SessionAuthorizationError を発生します。

認証方法は以下の属性で設定します。true のとき有効になります。

表 5-2：セッションの認証方法（CKApplication クラス）

属性	デフォルト	説明
auth_by_user_agent	false	ブラウザによる認証を行う。
auth_by_remote_addr	false	IP アドレスによる認証を行う。

セッションエラーの捕捉

認証エラー時に処理を行うには CKApplication#handle_session_error をオーバーライドし、表示したいコンポーネントを返してください。このメソッドはセッションの例外処理のためのフックメソッドで、タイムアウトかブラウザ・IP による認証エラーが発生したときに実行されます。

リスト 5-3：CKApplication#handle_error をオーバーライドする

```
class CKApplication
  def handle_error( error )
    if error.class == CKSession::SessionTimeoutError then
      # ... code for timeout
    elsif error.class == CKSession::SessionAuthorizationError then
      # ... code for authorizaion error
    end

    error_page       = page @error_page
```

```
    error_page.error = error
    error_page.debug = @debug
    error_page
  end
end
```

データベースマネージャ

セッションの保存は CKSessionStore::FileStore などのデータベースマネージャが行います。データベースマネージャは、インターフェースとして以下の３つのメソッドを持ちます。データベースマネージャを開発またはカスタマイズするときは、これらのメソッドを実装してください。

表 5-3：データベースマネージャのメソッド

メソッド	説明
save	セッションを保存する。
clear	セッションを削除する。
restore	復旧したセッションを返す。

その他の注意事項

パーミッションエラー

セッションを生成するときにパーミッションエラーが発生することがあります。これはセッションを保存するファイルや一時ディレクトリを生成しようとしても保存先のディレクトリが書き込み可能になっていないためです。このエラーが発生したら、セッションを保存するディレクトリのパーミッションを変更してください。

セッションファイルの削除

セッションを保存するファイルはセッション ID の数だけ生成されます。タイムアウトが発生したときやセッションを削除したときはセッションファイルも一緒に削除されますが、たまってしまったセッションファイルは手動で削除して下さい。

クックブック

アプリケーション

アプリケーションをインストールする

CGIKit アプリケーションの運用は一般的な CGI アプリケーションと同じです。ここではサンプルとして付属の Examples アプリケーションを取り上げます。サーバの設定は以下のものと仮定します。

表 6-1：サーバの設定

設定項目	設定内容
ホスト名	`localhost`
ドキュメントルート	`/var/www/htdocs`
CGI ディレクトリ	`/var/www/cgi-bin`

Web サーバが CGI を実行できるディレクトリにコピーし、パーミッションを実行可能に変更します。

リスト 6-1：Examples ディレクトリをコピーし、パーミッションを変更する

```
[localhost:samples] user% cp -R Examples /var/www/cgi-bin
[localhost:samples] user% cd /var/www/cgi-bin/Examples
[localhost:/var/www/cgi-bin/Examples] user% chmod 755 Examples.cgi
```

アプリケーションにアクセスするための URL

CKApplication#run を実行する起動プログラムのパスがアプリケーションの URL です。例えばインストールした Examples アプリケーションにアクセスするための URL は、**`http://localhost/cgi-bin/Examples/Examples.cgi`** などのようになります。

アプリケーションの URL を取得する

`CKApplication#baseurl()` でアプリケーションの URL を取得できます。引数に `true` を指定すると、セッション ID を含めた URL を返します。この URL は `SCRIPT_NAME` ヘッダの値を元に生成されます。

CGI

CGIKit は CGI ライブラリとしての基本的な機能も備えています。Ruby には CGI ライブラリが標準添付されていますが、通常 CGIKit と併用する必要はありません。

フォームデータを取得する

フォームデータは CKRequest オブジェクトから取得することができます（CKRequest オブジェクトは `CKComponent#request()` メソッドで取得できます）。クエリ、標準入力のどちらの送信方法でも使用するメソッドは同じです。フォームデータはハッシュとして扱われ、各キーに対して配列がセットされています。また、マルチパートのフォームデータは文字列ではなく CKByteData オブジェクトになります。

表 6-2：フォームデータを取得するメソッド

メソッド	説明
`form_values()`	フォームデータのハッシュを返す。
`form_value(key),[key]`	キー `key` の配列の最初のオブジェクト（文字列）を返す。キーが存在しなければ `nil` を返す。

リスト 6-2：フォームデータを取得する

```
class MainPage < CKComponent
  def get_form_value
    value = request['key']
    ...
  end
end
```

HTML テキスト・URL をエスケープする

CKUtilities モジュールに HTML テキストと URL の特殊文字をエスケープするメソッドがあります。`escape_html()`、`escape_url()` メソッドでエスケープを、`unescape_html()`、`unescape_url()` メソッドでエスケープした文字列を元に戻すことができます。

環境変数を取得する

CKRequest オブジェクトには HTTP ヘッダの値がセットされており、ヘッダ名を指定して環境変数を取得することができます。また、特定の環境変数にはアクセスメソッドも用意してあります。

表 6-3：環境変数を取得するためのメソッド

メソッド	説明
`headers()`	HTTP ヘッダのハッシュを返す。
`accept()`	`HTTP_ACCEPT`
`accept_charset()`	`HTTP_ACCEPT_CHARSET`
`accept_language()`	`HTTP_ACCEPT_LANGUAGE`
`auth_type()`	`AUTH_TYPE`
`content_length()`	`CONTENT_LENGTH`
`content_type()`	`CONTENT_TYPE`
`from()`	`HTTP_FROM`
`gateway_interface()`	`GATEWAY_INTERFACE`
`path_info()`	`PATH_INFO`
`path_translated()`	`PATH_TRANSLATED`
`query_string()`	`QUERY_STRING`
`raw_cookie()`	`HTTP_COOKIE`
`referer()`	`HTTP_REFERER`
`remote_addr()`	`REMOTE_ADDR`
`remote_host()`	`HTTP_HOST`
`remote_ident()`	`REMOTE_IDENT`
`remote_user()`	`REMOTE_USER`
`request_mothed()`	`REQUEST_METHOD`
`script_name()`	`SCRIPT_NAME`
`server_name()`	`SERVER_NAME`
`server_port()`	`SERVER_PORT`
`server_protocol()`	`SERVER_PROTOCOL`
`server_software()`	`SERVER_SOFTWARE`
`uri(), url()`	`REQUEST_URI`
`user_agent()`	`HTTP_USER_AGENT`

HTTP ヘッダのパラメータを取得する

CKRequest#headers() メソッドでHTTP ヘッダのパラメータのハッシュを取得できます。

HTTP レスポンスヘッダを設定する

HTTP レスポンスヘッダは CKResponse オブジェクトに設定します。headers 属性にレスポンスヘッダの内容をハッシュの形式で設定してください。

リスト 6-3：HTTP レスポンスヘッダを設定する

```
class MainPage < CKComponent
  def set_http_response_header
    response.headers['Content-Type'] = 'text/html'
  end
end
```

CKRequest・CKResponse オブジェクトを取得する

CKRequest・CKResponse オブジェクトはそれぞれ CKComponent#request()、response() メソッドで取得できます。

コンポーネント

他の Web ページ（コンポーネント）を表示する

CGIKit では Web ページ = コンポーネントです。基本的に、メソッドの戻り値にコンポーネントのオブジェクトを返すことで Web ページを表示します。新しいコンポーネントオブジェクトは CKComponent#page() で取得できます。引数にはコンポーネント名を指定します。

リスト 6-4：アクション実行後、他のコンポーネントを表示する

```
class MainPage < CKComponent
  def do_any_action
    # 何か処理を行う
    ...

    # 次に表示するページ（コンポーネント）を生成する
    next_component = page(“NextPage“)
    return next_component
  end
end
```

CKHyperlink を使ってリンクを張る

他のコンポーネントにリンクを張るだけ（実行するアクションがない）ならば CKHyperlink エレメントを使います。エレメントの page 属性にリンクするコンポーネント名を設定してください。

リスト 6-5：CKHyperlink でリンクを張る

```
Link : CKHyperlink {
  page = "OtherPage";
}
```

他のコンポーネントにデータを渡す

「フォームから入力されたデータを処理し、次のページで入力したデータの確認を行う」など、他のコンポーネントに必要なデータを渡さなければならないことがあります。このようなときはデータを渡すコンポーネントにアクセサを定義しておき、そのアクセサを使ってデータを設定します。

例として、MainPage コンポーネントでユーザーの名前を入力し、HelloPage コンポーネントでその名前を表示するアプリケーションを考えます。MainPage にはユーザー名を入力するためのテキストフィールドがあり、インスタンス変数 user_name にデータが代入されるものとします。

HelloPage では MaiPage で入力されたユーザー名を表示します。MainPage からデータを受け取るためのアクセサ user_name を定義します。

リスト 6-6：HelloPage コンポーネント

```
class HelloPage < CKComponent
  attr_accessor :user_name
end
```

MainPage では入力されたユーザー名を HelloPage に渡します。

リスト 6-7：ユーザー名を HelloPage に渡し、表示する

```
class MainPage < CKComponent
  def set_user_name_to_hellopage
    # HelloPage を生成し、ユーザー名を渡す
    hellopage = page("HelloPage")
    hellopage.user_name = @user_name

    # HelloPage を表示する
    return hellopage
  end
end
```

コンポーネントの初期化

CKComponent クラスでは専用の初期化メソッド init() が用意されています。初期化をするには initialize() ではなく init() をオーバーライドします。

init() の実行時、フォームデータはまだコンポーネントに代入されていません。エレメントにバインディングした変数を使おうとしてもデータが代入されていませんので注意して下さい。変数にデータが代入された状態での初期化は pre_action() メソッドをオーバーライドして記述します。

アクション実行前後に処理を行う

アクション実行前後に処理を行うには CKComponent#pre_action()、post_action() メソッドをオーバーライドします。これらのフックメソッドはそれぞれアクション実行前、実行後に呼ばれます。特定のアクションではなく、どのアクションに対しても呼ばれることに注意してください。また pre_action() が呼ばれるときには、フォームデータはバインディングファイルの設定通りに代入されています。

エラーページのカスタマイズ

アプリケーションで例外が発生すると CGIKit のデフォルトエラーページ（CKErrorPage）が表示されますが、アプリケーション専用のエラーページを作成することもできます。カスタムエラーページを作成・設定すると、例外が発生したときデフォルトエラーページの代わりに使われるようになります。

カカスタムエラーページは CKErrorPage のサブクラスになります。それ以外は普通のコンポーネントと同じように作成します。

リスト 6-8：エラーページのクラス定義

```
class CustomErrorPage < CKErrorPage
end
```

以下に CKErrorPage クラスの主なメソッドを示します。

表 6-4：CKErrorPage クラスの主なメソッド

メソッド	説明
error()	発生した例外オブジェクトを返す。
reason()	エラーメッセージを返す。
backtrace()	バックトレースを返す。
error_class()	発生した例外クラスを返す。

次に、アプリケーション環境設定の error_page 属性にカスタムエラーページ名を設定します。

リスト 6-9：カスタムエラーページ名を設定する

```
app = CKApplication.new
app.error_page = 'CustomErrorPage'
```

以上でカスタムエラーページを使うことができるようになりますが、カスタムエラーページで例外が発生した場合はデフォルトエラーページが表示されます。

コンポーネントが出力する HTML をファイルに保存する

CGI の負荷を軽減するためなど、表示される HTML をファイルに保存しておくには CKComponent#to_s() を使います。このメソッドはコンポーネントを HTML に変換した文字列を返します。

リスト 6-10：コンポーネントを HTML に変換し、ファイルに出力する

```
class MainPage < CKComponent
  def save_to_file( filename )
    open(filename, 'w+') do |f|
      f.write to_s()
    end
  end
end
```

フォームデータの文字コードを変換する

CGIKit はフォームデータの文字コードを自動的に変換してからコンポーネントに代入します。文字コードを設定するには、アプリケーション環境設定の char_code 属性に「jis、sjis、euc」のいずれかを指定します。デフォルトは nil で、文字コードを変換しない設定になっています。

他の URL にリダイレクトする

コンポーネントを表示する代わりに他の URL へリダイレクトするには、CKResponse#set_redirect() でリダイレクト先の URL を設定します。リダイレクト先を設定すると、コンポーネントを返す・返さないに関わらず指定した URL にリダイレクトされます。

リスト 6-11：他の URL にリダイレクトする

```
class MainPage < CKComponent
  def redirect
    # リダイレクト先を設定
    response.set_redirect('http://www.ruby-lang.org/')
```

```
    # コンポーネントを返しても上記の URL にリダイレクトされる
    nextpage = page('NextPage')
    return nextpage
  end
end
```

最初にアクセスしたときに表示するコンポーネントを変更する

アプリケーション環境設定の main 属性を使うと、アプリケーションに最初にアクセスしたときに表示するコンポーネントを変更することができます。デフォルトでは MainPage コンポーネントに設定されています。

次にコンポーネントを OtherPage に変更するコードを示します。こうすると、最初にアプリケーションにアクセスしたときに MainPage ではなく OtherPage が表示されるようになります。

リスト 6-12：表示するコンポーネントを OtherPage に設定する

```
app = CKApplication.new
app.main = "OtherPage"
```

エレメント

任意の HTML 属性を設定する

任意の HTML 属性をそのままエレメントに設定することで、エレメントが出力する HTML に属性を追加することができます。"checked" や "disabled" などの属性値を持たない文字列を追加するには other 属性を使います。

リスト 6-13：任意の HTML 属性を設定する

```
Link : CKHyperlink {
  href = "http://www.*****.com/";
  id = "link";
  other = "hello";
}
```

(出力)

```
<a href="http://www.*****.com" id="link" hello>Go to ******</a>
```

デバッグ

コマンドラインで実行する

アプリケーションをコマンドラインで起動すると、オフラインモードで実行します。実行すると下記のような文が表示され、入力待ちになります。ここでアプリケーションに渡すフォームデータを name=value の形式で指定します。最後に Ctrl-D を押すと実行します。

リスト 6-14：オフラインモードで実行する

```
[localhost:/cgi-bin/Examples] user% ./Examples.cgi
(offline mode: enter name=value pairs on standard input)
# Ctrl-D
Content-Type: text/html

<html>
<head>
<title>Examples</title>
</head>
<frameset cols="200,*">
    <frame name="Index" src="?element_id=IndexPage">
    <frame name="Contents" src="?element_id=IntroductionPage">

       <body>
            Use other browser.
        </body>
    </noframes>
</frameset>
```

ログを出力する

CKLog クラスを使うと簡単なロギングを行うことができます。CKLog には 5 つの出力レベルがあり、設定されたレベルより優先度の高いログ情報のみを出力します。出力レベルは低いほうから DEBUG < INFO < WARN < ERROR < FATAL となります。

表 6-5：ログ出力メソッド

メソッド	説明
debug(message)	メッセージを DEBUG レベルで出力する。
info(message)	メッセージを INFO レベルで出力する。
warn(message)	メッセージを WARN レベルで出力する。
error(message)	メッセージを ERROR レベルで出力する。
fatal(message)	メッセージを FATAL レベルで出力する。

オプション

ロギングに関するオプションは以下のものがあります。直接 CKLog オブジェクトに設定することもできますが、オプションを CKApplication の log_options 属性に設定しておき、CKLog オブジェクトを生成するときに使ってください。

ログの出力先をファイルに設定すると、アプリケーションで発生したエラー（例外）もファイルに出力するようになります。

表 6-6：ロギングオプション

オプション	説明
`level`	出力レベル。
`name`	プログラム名。
`out`	出力先。デフォルトでは標準エラーに出力する。
`file`	出力ファイル名。このオプションか `out` オプションのどちらかを設定する。
`max_file_size`	ファイルサイズの指定（出力先にファイルを指定したときのみ有効)。出力先のファイルサイズが指定したサイズを超えると、例外 `FileSizeError` を発生する。

リスト 6-15：ロギングオプションを設定する

```
options = {'level'         => CKLog::DEBUG,
           'name'          => 'CGIKit Application',
           'file'          => 'log.txt',
           'max_file_size' => 1000000}

app = CKApplication.new
app.log_options = options
app.run
```

リスト 6-16：ログ出力を行う

```
class MainPage < CKComponent
  def logging
    log = CKLog.new(application.log_options)
    log.debug 'log message'
  end
end
```

アプリケーションの高速化

mod_ruby

mod_ruby は Apache に Ruby インタプリタを組み込むモジュールです。CGIKit アプリケーションで mod_ruby を利用するためにはちょっとした作業が必要になります。このほか mod_ruby の設定に合わせて起動スクリプトの拡張子を変更するなどしてください。

なお、mod_ruby への対応は実験的機能です。安定して動作する保証はありません。

コンポーネントの名前空間を保護する

mod_ruby では複数のスクリプトで 1 つの Ruby インタプリタを共有するため、コンポーネントの名前空間を保護しないと CGIKit アプリケーションに影響が出ることがあります。そこで、CKApplication のサブクラスを作ってコンポーネントの名前空間を保護します。

まず、起動スクリプトとは別にファイルを作成し、CKApplication のサブクラスを定義します（起動スクリプトでこのファイルをロードします)。サブクラス名はアプリケーションに応じて変更してください。

リスト 6-17："application.rb" ファイルに CKApplication のサブクラスを定義する

```
class Application < CKApplication
end
```

次に、各コンポーネントをこのサブクラスの内部に定義します。

リスト 6-18：Application::MainPage コンポーネントを定義する

```
class Application
  class MainPage < CKComponent
    ...
  end
end
```

CKApplication のサブクラスは mod_ruby の対応のみに限りません。CKApplication オブジェクトはすべてのコンポーネント間で共有されるので、CKApplication のサブクラスにアプリケーション全体に関係するメソッドをまとめておくと有用です。

アダプタを mod_ruby に変更する

CGIKit は「アダプタ」を使ってブラウザとデータ送受信を行います。通常 CGI と mod_ruby のアダプタは自動的に判断されますが、それでも判断されないときやカスタマイズしたアダプタを使う場合は、`CKApplication#interface` 属性でアダプタを指定してください。

リスト 6-19：アダプタを mod_ruby に変更する

```
#!/usr/local/bin/ruby

require 'cgikit'
require 'application'

app = Application.new
app.interface = CKAdapter::ModRuby
app.run
```

WEBrick

WEBrick は Web サーバ用のツールキットです。WEBrick で CGIKit を動かすには、アプリケーションのインスタンスを生成し、サーブレットとしてマウントします。

CGIKit 用のハンドラは 3 種類あります。

表 6-7：ハンドラの種類

ハンドラ	説明
`WEBrick::CGIKitServlet::PathHandler`	第 2 引数にコンポーネントパスを定義するハンドラ
`WEBrick::CGIKitServlet::HashHandler`	第 2 引数に、ハッシュで CKApplication へのアクセサを定義するハンドラ
`WEBrick::CGIKitServlet::ApplicationHandler`	第 2 引数に CKApplication オブジェクトを渡すハンドラ

以下は `ApplicationHandler` を使った起動スクリプトです（付属サンプルの HelloWorld に添付してあります)。このスクリプトは、コンポーネントのパスとポート番号を指定して起動します。

```
% webrick-app.rb '.' 8080
```

リスト 6-20：ApplicationHandler を使った起動スクリプト（webrick-app.rb）

```
# webrick-app.rb [component_path [port]]
require 'webrick'
require 'cgikit'

path = ARGV.shift || Dir.pwd
port = (ARGV.shift || 8080).to_i

app = CKApplication.new
app.component_path = path

server = WEBrick::HTTPServer.new({:Port => port})
server.mount('/', WEBrick::CGIKitServlet::ApplicationHandler, app)
```

```
trap("INT"){ server.shutdown }
server.start
```

その他

CGIKit のバージョンを調べる

CGIKit のバージョンは CKApplication クラスのクラスメソッド `version()` で知ることができます。

リスト 6-21：CGIKit のバージョンを調べる

```
version = CKApplication.version() #-> '1.2.0'
```

ファイルをロックする

CKFileLock クラスを使うと、ファイルの排他・共有ロックを行うことができます。デフォルトの実装は flock を使ったファイルロックです。flock が使えない環境や flock 以外のロックを使いたいときは CKFileLock クラスを再定義してください。

リスト 6-22：ファイルをロックする

```
# ファイルロック（排他）
CKFileLock.exclusive_lock('file.txt', 'rw+') { |f|
  # ファイルロック中の処理
}

# ファイルロック（共有）
CKFileLock.shared_lock('file.txt', 'r') { |f|
  # ファイルロック中の処理
}
```

Emacs を使って開発する

CGIKit では多くのファイルを同時に編集する必要がありますが、Emacs の CGIKit モードを使えば CGIKit の開発が楽になります。CGIKit モードのファイルは `misc/cgikit-el` ディレクトリにあります。インストールや使い方はドキュメントを参照してください（ドキュメントの HTML 版は API リファレンスにもあります）。

CGIKit モードはるびきちさんによって開発されています。最新版は `http://www.rubyist.net/~rubikitch/computer/cgikit-el/` を参照してください。