WebObjects とバックトラック

WR@Csus4.net

2004-01-08

1 はじめに

Web アプリケーションの開発を行う場合、バックトラック（Web ブラウザの戻るボタンを押して前のページ戻ること）に適切に対処する必要がある。WebObjects 開発におけるバックトラックの扱いについて調査した。

2 Backtracking and Cache Management[1] 和訳

WebObjectcs において Web アプリケーションの開発を行う際の入門書的な位置づけの Apple 公式ドキュメント「Inside WebObjects : Web Application」においてバックトラックに関する章があるため、これを和訳する。
なお、「Inside WebObjects : Web Application」は Apple から和訳したバージョンが提供されているが、和訳バージョンにはこの章が含まれていない。

2.1 凡例

引用

> 訳
> 《訳者によるおせっかい否、補足》
> (?) 訳して意味が取りにくかった点

> 補足

2.2 ここから

Backtracking, client-side page caching, and web component caching are three closely related issues that cause many headaches for web application developers. Fortunately, WebObjects offers a number of mechanisms that help you deal with the collective problem of managing page state.

[1] http://developer.apple.com/documentation/WebObjects/Web_Applications/BacktrackingAndCache/chapter_6_section_1.html

バックトラッキング、クライアントサイドのページキャッシュ、Webコンポーネントのキャッシュは、Webアプリケーション開発者を悩ます、密接に関連しあっている問題である。幸運なことに、WebObjectsはページ状態を管理する共同の(?)問題に対処することを助けるいくつかのメカニズムを提供する。

Dynamic web applications are possible due to, among other things, server-side state persistence and state management. HTTP, the protocol of the web, is inherently stateless. However, storing state in an application server makes persistence management in web applications possible. In WebObjects, the Session object holds state but is not solely responsible for state management. The Session object tracks sessions, flags WOComponent and WOElement objects with special identifiers, and uses other mechanisms to hold and manage state. WOComponent objects manage the state of their internal instance variables and dynamic elements.

ダイナミックWebアプリケーションは、サーバサイドでの状態永続化と状態管理により可能となる《実現される》。
Webのプロトコルであ るHTTPは、本質的にステートレスである。しかしながら、アプリケーションサーバに状態を保管することで、Webアプリケーションにおける永続化管理を可能としている。WebObjectsにおいて、Sessionオブジェクトは状態を保持するが、状態管理の全ての責任を負っているわけではない。Sessionオブジェクトはセッションをトラッキングし、WOコンポーネントとWOエレメントを特殊な識別子を用いて識別し、状態を維持、管理するために、他の機構を使用する。WOコンポーネント オブジェクトは内部のインスタンス変数とダイナミックエレメントの状態を管理する。

Along with these mechanisms, caching plays an important role in managing the state of visual components. Caching allows a user to view a previously viewed webpage (even a dynamically generated one) without the application needing to regenerate the page. Caching also plays a crucial role in providing a good user experience in web applications. Caching lets users backtrack using their web browser’s Back button, which often allows for instantaneous loading of pages from the client-side cache rather than requesting a previously viewed page from the application server. However, due to the diverse implementations of the HTTP protocol in web browsers, backtracking behavior is inconsistent and requires considerable attention when developing web applications.

これらのメカニズムの他に、キャッシュはビジュアルコンポーネントの状態を管理する点で、重要な役割を担う。キャッシュはアプリケーションがページを再生成する必要なく、先に見ていたWebページ（動的に生成されていたものでさえも）を見ることを可能とする。キャッシュはユーザーにWebブラウザのバックボタンを使用してバックトラックすることを可能にする。（しばしがアプリケーションサーバから先に見ていたページを要求するのではなく、クライアントサイドのキャッシュからページを即座にロードすることを可能にする）
しかし、Webブラウザにおける多様なHTTPプロトコル実装のため、バックトラックの振る舞いは一貫性がなく、Webアプリケーションの開発においては、多くの注意が必要である。

In addition to client-side page caching, WebObjects also caches components in a server-side cache. If used correctly, this is a valuable feature that can improve performance and user

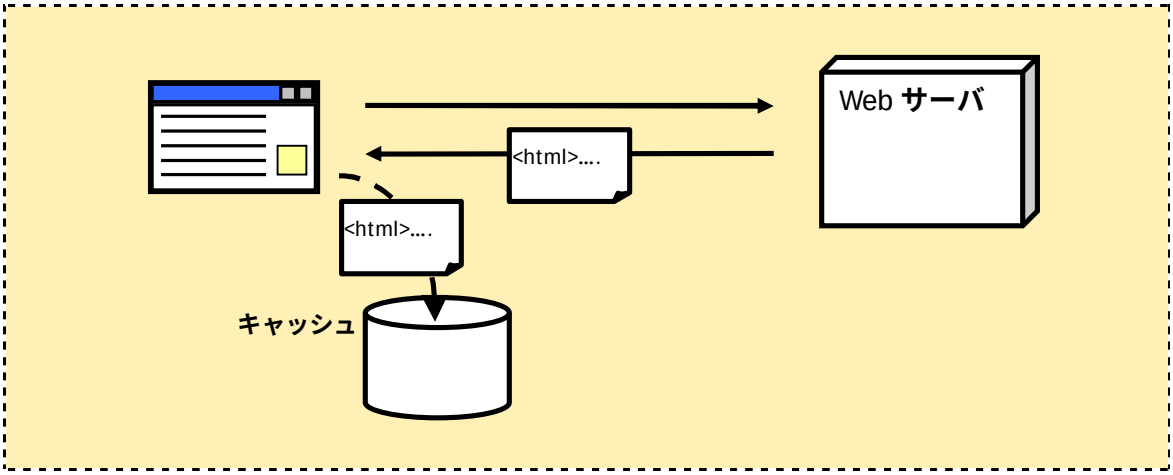
experience. But you must be conscious of the relationship between server-side component caching and client-side page caching, and how inconsistencies in backtracking behavior affect the result when either or both caching features are active.

クライアントサイドのページキャッシュに加えて、WebObjectsはサーバサイドキャッシュにコンポーネントをキャッシュする。正しく使用すれば、パフォーマンスとユーザーの使用感を改善できる有効な特性である。しかし、サーバサイドのコンポーネントキャッシュとクライアントサイドのページキャッシュの間の関連に注意し、そして、キャッシュ機能のどちらかまたは両方が有効である場合に、《Webブラウザの》バックトラックの振る舞いに一貫性がないことが、処理結果にどのように影響するか？について注意しなければならない。

2.3 Client-Side Page Cache

A web component is the aggregate of WebObjects elements and subcomponents. When a web browser caches a webpage from a WebObjects application, it caches the HTML code of a generated page (which does not include a web component's programmatic entities, such as instance variables). In contrast, server-side component caching caches a web component's definition and state.

クライアントサイド ページキャッシュ
WebコンポーネントはWebObjectsエレメントとサブコンポーネントの集合である。Webブラウザは、Webアプリケーションからの Webページをキャッシュするとき、生成されたページのHTMLコードをキャッシュする。（これには、Webコンポーネントのプログラム的な実体、たとえばインスタンス変数とか、は含まれていない）。
対照的に、サーバサイド コンポーネントキャッシュは、Webコンポーネントの定義と状態をキャッシュする。



Client-side page caching is a feature implemented by web browsers to improve performance and user experience. Although WebObjects applications primarily publish dynamic webpages, many websites serve static pages: They do not change as rapidly as content-driven dynamic sites.

クライアントサイド ページキャッシュは、パフォーマンスとユーザーの使用感を改善するためにWebブラウザに実装された特性である。WebObjectsアプリケーションは主に動的なWebページを生成するが、多くのWebサイトは、コンテントドリブンな動的なサイトほどすぐに変化しないような、静的なページを提供する。

For instance, consider a website that publishes news stories and other articles. Although the front page of the site probably changes a few times each day, it likely would not change in the few minutes an average user spends browsing headlines and reading a few articles.

With client-side page caching active, the front page of the news website is cached on the client's computer upon the first visit. The first page could be large, containing images, banner adds, and text. The user could select an article, read part of it, and access other articles through URLs in the first article. Then, having visited five or six pages within the website, she could backtrack to the main page. Since the content of that page is not likely to change in the time the user took to peruse the five or six pages, the page should be reloaded from the local cache. So the web browser—instead of requesting and downloading the main page from the web server again—would retrieve it from the local cache, avoiding a round trip over the network to the web server. In this case, page caching serves a sensible and user-friendly function.

たとえば、ニュースや記事を発行するするWebサイトを考えてみてほしい。サイトのフロントページは1日に2,3回変更するかもしれないが、平均的なユーザーが見出しの閲覧と2,3の記事を読むことに費やす数分間の間には変更しない。
クライアントサイド ページキャッシュが有効である場合、ニュースWebサイトのフロントページは、最初に訪れたときにクライアントコンピュータにキャッシュされる。最初のページは、大きく、画像、広告バナー、テキストを含むかもしれない。ユーザーは記事を選択し、その一部分を読み、最初の記事に含まれたURLを介して、別の記事にアクセスする。
Webサイト内の5,6のページを訪れ、メインページにバックトラックするかもしれない。そのページのコンテンツは、5,6のページを精読するためにかかった時間内に変更される可能性は少ないため、ページはローカルキャッシュからロードされるべきである。
したがって、Webブラウザは、ネットワークを介したWebサーバへのラウンドトリップをさけるために、―Webサーバから再度メインページを要求してダウンロードする代わりに― ローカルキャッシュからそれを取得する。
この場合、ページキャッシュは、道理にかなったユーザーに優しい機能である。

Now, consider the case of an online store: A user chooses items to buy and adds them to a shopping cart. It's generally not a good idea for the user to view a cached webpage representing the shopping cart as it likely does not contain the most up-to-date information. If client-side page caching is active, however, this is a real possibility.

では、オンラインストアの場合を考えてみよう。ユーザーが購入したいアイテムを選択し、ショッピングカートに追加する。一般的に、最新の情報を含まないであろう（ショッピングカードを表示する）キャッシュされたWebページを、ユーザーに見せるのはよいアイディアではない。しかし、クライアントサイド ページキャッシュが有効である場合、このことは本当に起こりうるのである。

WebObjects offers a number of mechanisms to deal with the problems of backtracking and

client-side caching. The first one you should use is a flag on the Application object that you set using the setPageRefreshOnBacktrackEnabled method of the WOApplication class (com.webobjects.appserver). When pageRefreshOnBacktrackEnabled is true, a number of HTTP headers are added to each response generated by the WebObjects application to disable client-side page caching. Table 6-1 shows these headers and their values.

WebObjectsはバックトラックとクライアントサイドキャッシュの問題に対処するいくつかのメカニズムを提供する。使用すべき最初の方法は、WOApplicationクラスのsetPageRefreshOnBacktrackEnabledメソッドを使用するようにApplicationオブジェクトのフラグである。pageRefreshOnBacktrackEnabledがtrueである場合、クライアントサイド ページキャッシュを無効にするために、WebObjectsアプリケーションによって生成されるレスポンスは、いくつかのHTTPヘッダーを付加される。表6-1にこれらのヘッダーと値を示す。

Table 6-1 HTTP response headers that deactivate client-side page caching

Header	Value
date	The time the response page was generated.
expires	The time the response is to expire. (Same as date.)
pragma	no-cache
cache-control	private, no-cache, no-store, must-revalidate, max-age = 0

See section 14.9 of the HTTP 1.1 specification (RFC 2616) for more details on each of these headers.

これらのヘッダーの詳細については、HTTP1.1仕様(RFC2616)の14.9章をみられたい。

The pageRefreshOnBacktrackEnabled property affects all responses generated by an application. If you want to restrict the behavior to a specific response, invoke the disableClientCaching method of the WOResponse object (com.webobjects.appserver). WOResponse also includes the methods setHeader and setHeaders, which allow you to explicitly set the HTTP headers for a particular response.

pageRefreshOnBacktrackEnabledプロパティはアプリケーションによって生成された全てのレスポンスに影響する。この振る舞いを、特定のレスポンスに限定したい場合は、WOResponseオブジェクトのdisableClientCachingメソッドを起動せよ。また、WOResponseは、特定のレスポンスに対するHTTPヘッダーを明示的に設定できるsetHeaderとsetHeadersメソッドを持つ。

When a web browser receives a response page with the headers shown in Table 6-1, it should not add the page to its local cache and it should invalidate the page as soon as it is displayed. In other words, when users backtrack to retrieve previously viewed pages, the web browser should request the response page from the application server. However, not all web browsers follow this protocol, as demonstrated in “Web Browser Backtracking Behavior”. The first few times the user backtracks to previously viewed pages, most web browsers ignore the HTTP headers and render the page stored in the cache.

Webブラウザが表6-1に示されたヘッダーを持つレスポンスページを受け取った場合、ページをローカルキャッシュに追加すべきでなく、表示するとすぐにページを無効とすべきである。言い換えれば、以前に表示したページを得るために、ユーザーがバックトラックするとき、Webブラウザはアプリケーションサーバからレスポンスページを要求すべきである。
しかし、全てのWebブラウザが、"Web Browser Backtracking Behavior"で示したように、このプロトコルに従うわけではない。最初の数回は以前に見たページをバックトラックする。
たいていのWebブラウザは、HTTPヘッダーを無視し、キャッシュに保管されているページをrenderする。

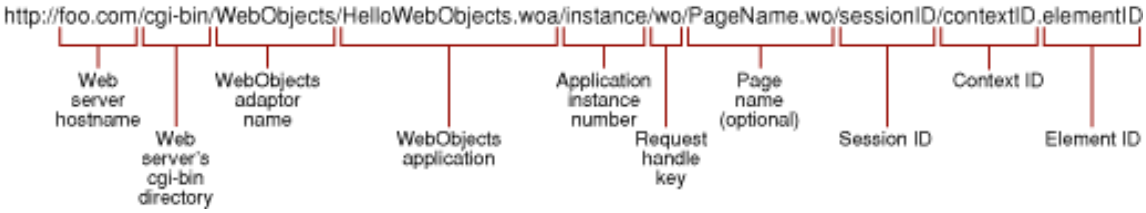
When a web browser needs to refresh an expired page, it sends a request to the application server, which accesses the server-side cache to reconstruct the page (see "Server-Side Page Cache" for more information on server-side caching). "Request Processing" explained the phases of the request-response loop in detail. The main phases are sync, action, and response. When processing a refresh request, an application does not go through the sync and action phases; it performs only the response phase.

Webブラウザが期限切れのページをリフレッシュする必要がある場合、アプリケーションサーバにリクエストする。そして、ページの再構築のためにサーバサイドキャッシュにアクセスする（サーバサイド キャッシュに関する情報は、"Server-Side Page Cache"を見てほしい）。
"Request Processing" では、リクエスト-レスポンスループを詳細に説明した。主要なフェーズは、同期化、アクション、レスポンスである。リフレッシュ リクエストを処理する場合、アプリケーションは、同期、アクションフェーズを経ずに、レスポンスフェーズのみを実行する。

So how does an application know to perform only the response phase (just returning the response page stored in the server cache, rather than regenerating it)? WebObjects assigns each response a context ID. The context ID is increased by 1 each time a web browser requests a specific page from the application server during a session. It identifies a specific instance of the corresponding WOComponent. (Figure 6-1 shows the elements of a WebObjects URL). Specifically, an application assigns the outermost component of a WOComponent a context ID each time that component is part of a response. So, if the same component is dynamically generated multiple times, each instance of the page (each response) is assigned a unique context ID.

では、レスポンス フェーズだけを実行する（それを再生成するのではなく、サーバキャッシュに保存されているレスポンスページを返すだけ）ことを、アプリケーションはどうやって知るのだろうか？ WebObjectsは各々のレスポンスにコンテキストIDを付与している。コンテキストIDはセッション中、アプリケーションサーバからWebブラウザがあるページをリクエストする毎に、1づつ増やされる。コンテキストIDは対応するWOコンポーネントのあるインスタンスを識別する。
（図6-1にWebObjectsURLの構成要素を示す。）
特に、アプリケーションはレスポンスの一部であるたびに、最も外側のコンポーネントにコンテキストIDを割り当てる。したがって、同じコンポーネントが動的に複数回生成された場合、各々のページのインスタンス（各々のレスポンス）は唯一のコンテキストIDが付与される。

Figure 6-1 Structure of a component action URL



2.4 Server-Side Component Definition Cache

When a web component is accessed for the first time, it's definition is placed on the server-side cache. Subsequent requests for the same component use the definition stored in the cache. Using the web component cache improves performance because the application looks up a component's definition only one time during the lifetime of the application. You can control web component caching at the application level and the component level.

Webコンポーネントが最初にアクセスされたとき、それの定義はサーバサイドキャッシュに置かれる。同じコンポーネントに対する、以降のリクエストはキャッシュに保存された定義を使用する。アプリケーションのライフタイムにおいて一回だけコンポーネントの定義を参照するため、Webコンポーネント キャッシュを利用することで、パフォーマンスは改善される。アプリケーションレベルとコンポーネントレベルでWebコンポーネントのキャッシュを制御できる。

You can set a caching policy for the application (either active or inactive) for all components, but can also override such policy on specific components. To set the caching policy for an application or a web component you use the setCachingEnabled method of WOApplication or WOComponent, respectively. Sending true as the argument activates web component–definition caching, while sending false deactivates it.

全てのコンポーネントへのアプリケーションのキャッシング ポリシーを（有効か、無効のいずれかに）セットできる。しかし、特定のコンポーネントにおけるポリシーをオーバライドすることも可能である。
特定のアプリケーションまたはWebコンポーネントのキャッシング ポリシーを設定するためには、WOApplicationまたはWOComponentのsetCachingEnabledメソッドを使用する。
引数にtrueを与えると、Webコンポーネント定義のキャッシュが有効となり、falseとすると、無効となる。

2.5 Server-Side Page Cache

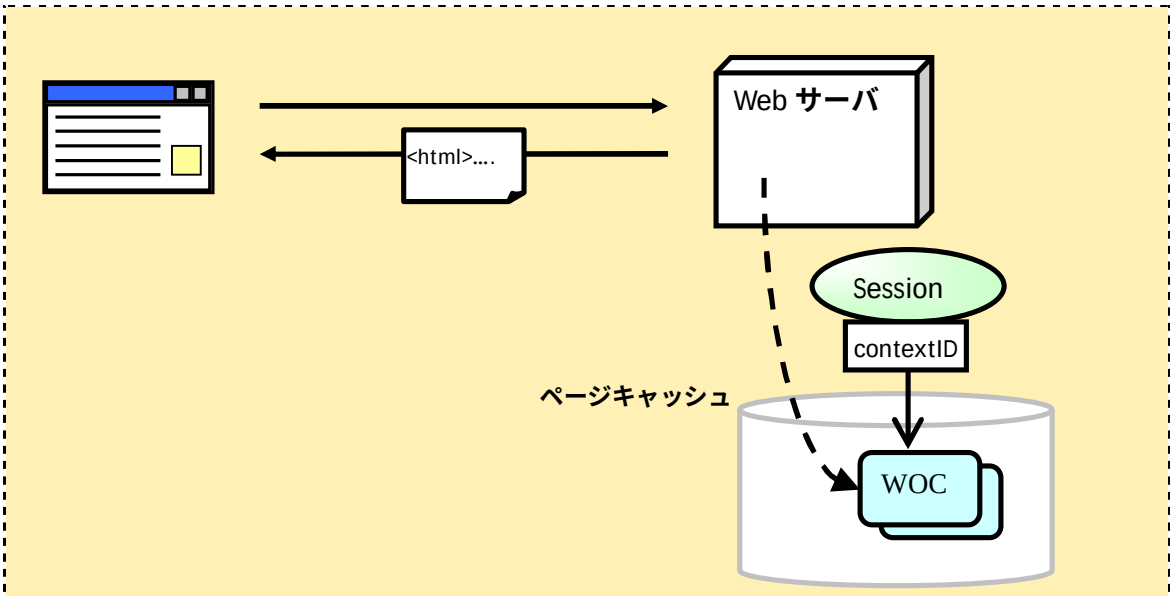
In addition to component-definition caching, WebObjects applications can also cache responses sent to a client. When an already-generated page is requested from the application server, WebObjects checks the context ID of the requested page with the context IDs of pages in its cache. If it finds a match, it performs the response phase of the request-response loop. This returns a response that has a new context ID and updated content from the invocation of the

response phase of the request-response loop (dynamic bindings are again resolved in the response phase).

コンポーネント定義のキャッシュに加えて、WebObjectsアプリケーションはクライアントへ送るレスポンスをキャッシュすることができる。すでに生成されたページがアプリケーションサーバから要求された場合、WebObjectsはキャッシュ内のコンテキストIDと要求されたページのコンテキストIDをチェックする。マッチする（ページを）見つけた場合、リクエストーレスポンスループのレスポンスフェーズを実行する。リクエストーレスポンスループのレスポンスフェーズの実行から、新しいコンテキストIDと更新されたコンテンツを持つレスポンスが返される。（動的なバインディングは、再びレスポンスフェーズにおいて解決される）

By default, the WebObjects application server maintains a page cache for each session. Each page a user accesses is added to the session's page cache. When a user backtracks, accesses a URL, or selects a bookmark of a page that is cached but expired in the local cache, the web browser requests a refreshed version of that page from the application server. The server-side page cache preserves resources as it hands out the result of previously generated pages. When the page the user backtracks to is no longer in the cache, WebObjects returns an error page.

デフォルトでは、WebObjectsアプリケーションサーバは、セッション毎にページキャッシュを維持する。ユーザーがアクセスする各々のページは、セッションのページキャッシュに追加される。ユーザーがバックトラックした場合や、URLにアクセスした場合や、ローカルキャッシュにキャッシュされているが期限切れとなっているページへのブックマークを選択した場合、Webブラウザは、そのページのリフレッシュされたバージョンをアプリケーションサーバへリクエストする。サーバサイド ページキャッシュは、昔生成したページの結果を配るためにリソースを保存する。ユーザーがバックトラックしたページがキャッシュにない場合は、WebObjectsはエラーページ《backtrack too far!エラー》を返す。



WebObjectsコンポーネントキャッシュの概念図。
WOコンポーネントはSessionの保持するページキャッシュにキャッシングされる。
ページキャッシュ内のWOコンポーネントインスタンスはコンテキストIDをキーにして管理され

る。

If you deactivate the server-side page cache (by passing 0 to the setPageCacheSize method of WOApplication), the application assumes that you intend to provide custom component state persistence rather than rely on WebObjects inherent support. Deactivating the component cache means that new WOComponent objects are instantiated (that is, each request for a component creates a new instance of that component) with each cycle of the request-response loop, even for component action requests that return the invoking page. This means that any nondefault instance variable values are discarded with each subsequent cycle of the request-response loop. In large applications, this redundancy and overhead could hinder performance.

（WOApplicatonのsetPageCacheSizeメソッドに0を渡すことで）サーバサイド ページキャッシュを無効にする場合、WebObjects固有のサポートに頼るのではなく、開発者が、カスタムの《＝開発者独自の意味》コンポーネント状態永続化を提供しようとしていると、アプリケーションは仮定する。
コンポーネントキャッシュを無効にすることは、コンポーネントアクションが起動元のページを返す場合においてさえも、リクエストーレスポンス ループのサイクル毎に新しいWOコンポーネント オブジェクトがインスタンス化されることを意味する。
このことは、デフォルトがないインスタンス変数の値は、以降のリクエストーレスポンスループのサイクル内で捨てられることを意味する。大規模アプリケーションにおいては、このことは冗長さと負荷がパフォーマンスを妨げる可能性がある。

WebObjects also provides a permanent page cache that is useful for storing subcomponents such as navigation bars or page headers, or when using frame sets. You have to explicitly add components to it using the savePageInPermanentCache method of WOSession (com.webobjects.appserver). See the API reference for details.

また、WebObjectsはナビゲーションバーや、ページのヘッダーなどのサブコンポーネントを保管するとき、フレームセットを使用するときに便利な恒久的なページキャッシュを提供する。
WOSessionのsavePageInPermanentCacheメソッドを使用して、明示的にコンポーネントをそれに追加する必要がある。詳細はAPIリファレンスを参照されたい。

2.6 Web Browser Backtracking Behavior

To better understand the concepts of backtracking, client-side page caching, and component-definition caching, perform the tasks described in the following sections.

バックトラック、クライアントサイド ページキャッシュ、コンポーネント定義キャッシュの概念をより理解するためには、以下のセクションに記述したタスクをせよ。

2.6.1 Viewing the HTML Headers

Open the TimeDisplay project described in “Developing Dynamic Content”.

In Main.java, add a method called outgoingHeaders:

“Developing Dynamic Content”に記述したTimeDisplayプロジェクトを開くこと。
Main.javaに、outgoingHeadersと呼ばれるメソッドを追加すること。

```
public String outgoingHeaders() {
    return context().response().headers().toString();
}
```

This gets the headers that are attached to each outgoing WOResponse. To view these headers, override the sleep method in the Main class so that it prints the headers to the console:

これは、外に向かう各々のWOResponseに加えられたヘッダを取得する。これらのヘッダーを見るためには、Mainクラス内のsleepメソッドをオーバライドすること、すると、コンソールにヘッダーがプリントされる。

```
public void sleep() {
    System.out.println("<Main.sleep> headers=" + outgoingHeaders());
}
```

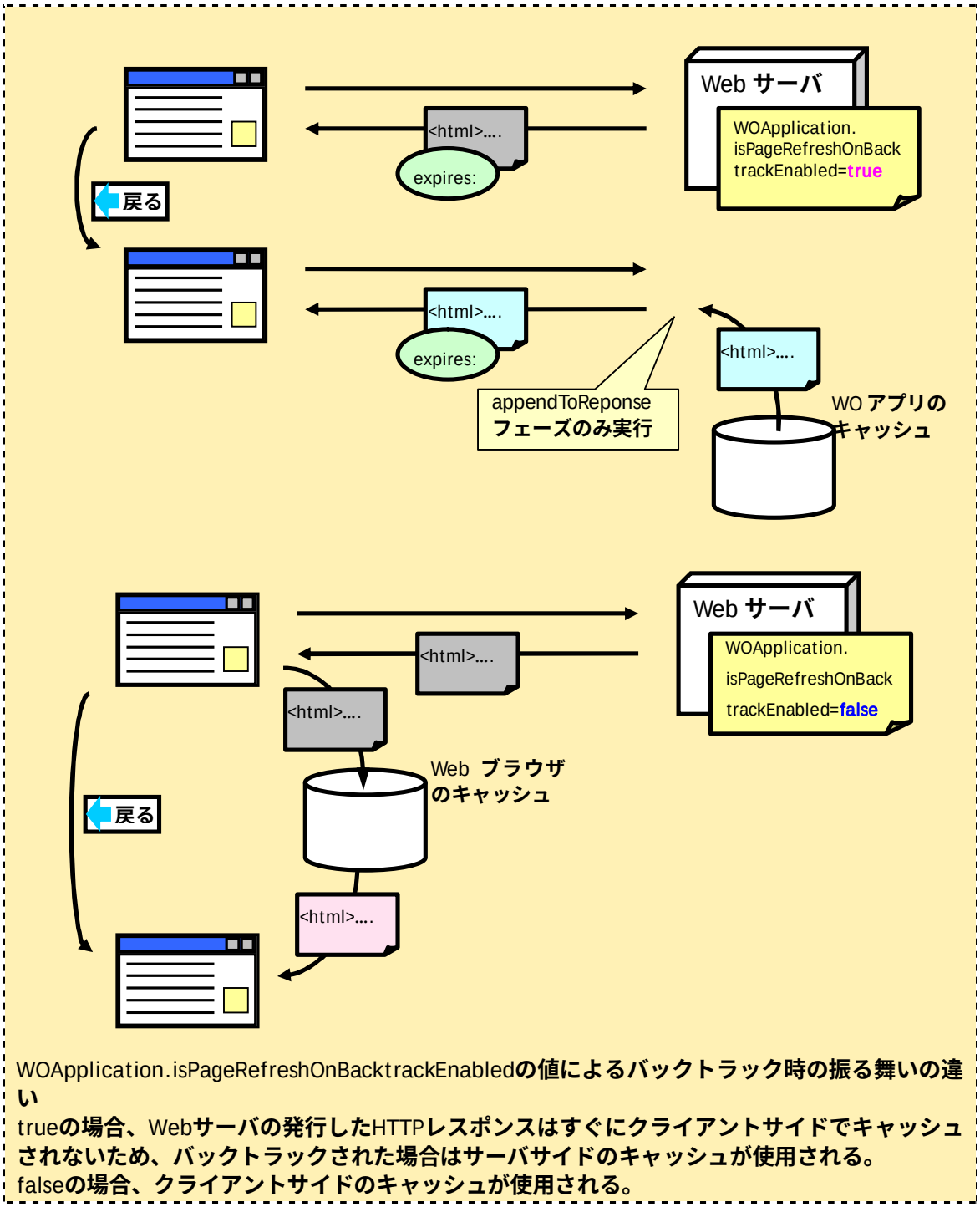
Build and run the application. You should see output similar to this in the console:

ビルドして、アプリケーションを実行せよ。コンソールにこのような出力を見るだろう

```
Welcome to TimeDisplay!
[2003-01-08 17:53:56 PST] <main> Opening application's URL in browser:
http://17.203.33.19:8888/cgi-bin/WebObjects/TimeDisplay.woa
[2003-01-08 17:53:56 PST] <main> Waiting for requests...
<Main.sleep> headers={cache-control = ("private", "no-cache", "no-store", "must-revalidate",
"max-age=0"); expires = ("Thu, 09-Jan-2003 01:53:54 GMT"); date = ("Thu, 09-Jan-2003 01:53:54
GMT"); pragma = ("no-cache"); content-type = ("text/html"); }
```

The expires header is set to the time the component is generated, so that when the web browser receives the webpage, it is already expired in the web browser’s cache. These headers (except content-type) are appended to the response when the isPageRefreshOnBacktrackEnabled method of WOApplication returns true, which it does by default.

expiresヘッダーはコンポーネントが生成された時間を指している。よって、Webブラウザが Webページを受け取ったとき、Webブラウザのキャッシュの期限はすでに切れている。これらのヘッダー（content-typeを除く）は、WOApplicationのisPageRefreshOnBacktrackEnabledメソッドがtrueを返す（デフォルト）ときに、付加される。



WOApplication.isPageRefreshOnBacktrackEnabledの値によるバックトラック時の振る舞いの違い
trueの場合、Webサーバの発行したHTTPレスポンスはすぐにクライアントサイドでキャッシュされないため、バックトラックされた場合はサーバサイドのキャッシュが使用される。
falseの場合、クライアントサイドのキャッシュが使用される。

In Application.java, set the pageRefreshOnBacktrackEnabled property to false in the constructor:

```
public Application() {
   super();
   System.out.println("Welcome to " + this.name() + "!");
   setPageRefreshOnBacktrackEnabled(false);
}
```

Build and run the application. You should see output similar to the following in the console:

```
Welcome to TimeDisplay!
[2003-01-08 17:57:15 PST] <main> Opening application's URL in
browser:http://17.203.33.19:8888/cgi-bin/WebObjects/TimeDisplay.woa
[2003-01-08 17:57:15 PST] <main> Waiting for requests...<Main.sleep> headers={content-type =
("text/html"); }
```

Notice that the headers disabling client-side caching are not generated in the response.

クライアントサイド キャッシュを無効とするヘッダーがレスポンスに生成されていないことに注意せよ。

2.6.2 Standard Webpage Backtracking

So, how does the pageRefreshOnBacktrackEnabled property of WOApplication affect user backtracking? You need to add some more code to trace what WebObjects does behind the scenes. Modify Main's constructor to look like this:

したがって、WOApplication**の**pageRefreshOnBackTrackEnabled**プロパティが、ユーザーのバックトラックにどうやって影響するのか？**
WebObjects**が背後でやっていることをトレースするためのコードを追加する必要がある。これを見るために、**Main**のコンストラクタを修正すること**

```
public Main(WOContext context) {
    super(context);
    System.out.println("<Main> context ID="+ context().contextID());
}
```

Each time an instance of Main is created, this code outputs the context ID of the WOResponse associated with the new instance. This allows you to see when user actions like clicking the Refresh hyperlink on the webpage or the web browser's Back button produce a new instance of the Main component. While this is useful information, you may also want to know when a user action causes the application to send a new response page to the client web browser. You can trace this by adding similar code to the refreshTime method:

毎回、Main**のインスタンスが生成されるとき、このコードは新しいインスタンスに関連した**WOResponse**のコンテキスト**ID**を出力する。ユーザーが、**Web**ページのリフレッシュ ハイパーリンクや、**Web**ブラウザのバックボタンをクリックしたときに、**Main**コンポーネントの新しいインスタンスが生成されるのをみることを可能とする**(?)
これは有用な情報であるが、

```
public WOComponent refreshTime() {
  System.out.println("<Main.refresh> contextID=" + context().contextID());
  loadCount++;
  return null;
}
```

Now, remove the sleep and outgoingHeaders methods and build and run the application.

Click Refresh Time three times. This prints the incremental context ID of the instance of Main through which you navigate. When you click Refresh Time, the application invokes the refreshTime method, which outputs the context ID of the outgoing response to the console:

Refresh Time[2]を3回クリックしてほしい。ナビゲートしているMainのインスタンスの増加するコンテキストIDをプリントする。Refrech Timeをクリックするとき、アプリケーションはコンソールにレスポンスのコンテキストIDを出力するrefrechTimeメソッドを実行する。

```
Welcome to TimeDisplay!
[2003-01-08 18:56:18 PST] <main> Opening application's URL in browser:
http://17.203.33.19:8888/cgi-bin/WebObjects/TimeDisplay.woa
[2003-01-08 18:56:18 PST] <main> Waiting for requests...
<Main> context ID=0
<Main.refreshTime> context ID: 1
<Main.refreshTime> context ID: 2
<Main.refreshTime> context ID: 3
```

Now, click your browser's Back button three times. Notice that nothing is printed to the console. This is because, when pageRefreshOnBacktrackEnabled is set to false, backtracking does not result in a request to the application; the page is simply rendered using the copy in the browser's cache. Similarly, choosing the bookmark of a page cached in the web browser does not result in a request to the application.

では、ブラウザのバックボタンを3回クリックしてほしい。何もコンソールにプリントされないことに注意せよ。
これは、pageRefreshBacktrackEnabledがfalseに設定されている場合、バックトラックは、アプリケーションに対するリクエストに帰着しない。
ページはブラウザのキャッシュにあるコピーを使用してレンダリングされる。
同様に、Webブラウザにキャッシュされたページのブックマークを選択すると、アプリケーションへのリクエストに帰着しない。

2.6.3 Refreshing Pages When Backtracking

When pageRefreshOnBacktrackEnabled is set to true, backtracking should result in a request to the application (you should see a context ID line with a new context ID) when a user backtracks, although the actual behavior differs among various web browsers.

pageRefreshOnBacktrackEnabledがtrueに設定されている場合、バックトラックはアプリケーションへのリクエストへ帰着する。（新しいコンテキストIDと共にコンテキストIDを見る???）
ユーザーがバックトラックしたとき、
しかし、実際の振る舞いは様々なブラウザによって異なる。

In Mac OS X, web browsers that use the Gecko HTML rendering engine (such as Chimera and Mozilla), comply most closely to the HTTP specification. Clicking the Back button causes the browser to ask for an updated version of an expired webpage. Other browsers, such as Internet Explorer and OmniWeb, behave differently: The first few clicks (two to three, depending on the

[2] 自ページに戻ってくるリンク。loadCount++する。

browser) of the Back button reload the page from the cache. Subsequent clicks cause the browser to send a request to the application.

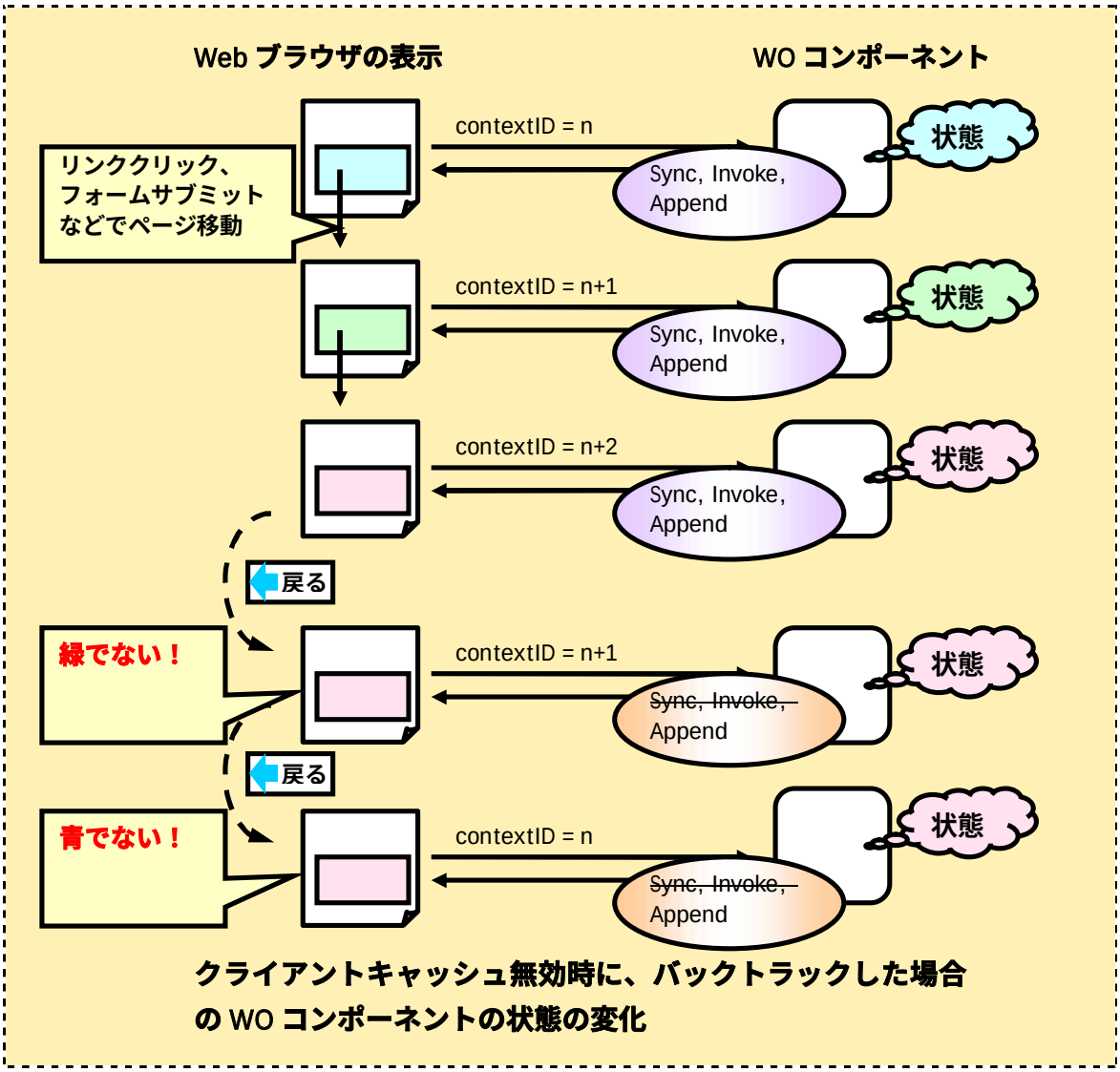
Mac OS Xにおいて、Gecko HTMLレンダリングエンジンを使用するWebブラウザ（Chimera、Mozilla）は、HTTP仕様に準拠している。バックボタンをクリックすると、期限切れのWebページの更新されたバージョンを要求する。
他のブラウザ、例えばIEやOmniWebは異なる振る舞いをする。
バックボタンをクリックする最初の2,3回は（2か、3、ブラウザに依存する）、ページをキャッシュから再読み込みする。以降のクリックは、アプリケーションにリクエストを要求する。

上記の現象は、IEのバグとして報告されていたような・・・

Notice that when the browser requests the updated version of the webpage from the application, the page-load counter doesn't decrease, but the time is updated.

ブラウザがアプリケーションからWebページの更新されたバージョンを要求するとき、ページロード カウンタは減らないが、時間は更新される。

「ページロードカウンタは減らない」の意味について
静的なWebページの場合、「戻るボタン＝昔のページの表示」となる。これとの類推で考えると、このWOアプリケーションはリンクをクリックする毎にカウンタをアップするものであるため、戻るボタンを押すとカウンタは戻されるように思われるかもしれない。しかし、クライアントキャッシュが無効となっているため、単純に昔のページが表示されることはなく、戻るボタンを押すたびにWebブラウザからWOアプリケーションにHTTPリクエストが発行され、ページの再生成が行われてappendToResponse()が実行される。このため、時間は更新される。しかし、アクションは実行されないため、ロードカウンタのインスタンス変数はそのままとなる。
カウンタを状態として一般化すると、下のように図式化できる。



You must test your application on many configurations to ensure that it provides a good user experience.

良いユーザーの操作感を提供するできるように、アプリケーションの多くの設定をテストしなければならない。

2.6.4 Disallowing Server-Side Caching

A WebObjects application can hand back only the response of a previously generated page when server-side page caching is active, which is the default. When this feature is inactive, the println statement in the constructor of the Main class (of the TimeDisplay application described earlier in this chapter) is invoked each time you click the Refresh Time link. This indicates that the application instantiates a Main object each time the refreshTime method of Main is invoked, instead of returning the current Main object.

WebObjectsアプリケーションは、サーバ・サイド・ページキャッシュが有効（デフォルト）である場合、以前に生成されたページのレスポンスを返すことができる。
無効である場合、（本章の最初に記述したTimeDisplayアプリケーションの）Mainクラスのコンストラクタのprintlnステートメントは、Refrech Timeリンクをクリックするたびに実行される。これは、アプリケーションはMainオブジェクトをMainのrefrechTmeメソッドが起動されるたびに生成している。現在のMainオブジェクトを返す代わりに。

Modify Main's constructor by adding a call to setPageCacheSize:

```
public Application() {
   super();
   System.out.println("Welcome to " + this.name() + "!");
   setPageRefreshOnBacktrackEnabled(true);
   setPageCacheSize(0);
}
```

Build and run the application. After clicking Refresh Time three times, you should see the following console output:

setPageCacheSizeへの呼び出しをMainコンストラクタに追加する。
アプリケーションをビルド後実行せよ。Refrech Timeを3回クリックした後、以下のコンソールアウトプットを確認できるはずである。

```
Welcome to TimeDisplay!
[2003-01-08 20:31:58 PST] <main> Opening application's URL in browser:
http://17.203.33.19:8888/cgi-bin/WebObjects/TimeDisplay.woa
[2003-01-08 20:31:57 PST] <main> Waiting for requests...
<Main> context ID=0
<Main> context ID=1
<Main.refreshTime> context ID: 1
<Main> context ID=2
<Main.refreshTime> context ID: 2
<Main> context ID=3
<Main.refreshTime> context ID: 3
```

Notice that Main's constructor is invoked each time you click Refresh Time, before the refreshTime method is executed. An instance of Main is created during each cycle of the request-response loop. Also notice that the page-view counter does not increase. The primary consequence of deactivating server-side page caching is that the values of variables in components are lost after each response is generated.

Refrech Timeをクリックするたび、refrechTimeメソッドが実行される前にMainのコンストラクタがに起動されていることに注意されたい。Mainのインスタンスはリクエストーレスポンス ループの度に生成されている。また、page-viewカウンタが増えていないことに注意されたい。サーバサイド ページキャッシュを無効にした主要な結果は、コンポーネント内の変数値がレスポンス生成された後に失われることである。

2.6.5 Setting the Size of the Server-Side Cache

Instead of completely disallowing server-side caching, you can use the setPageCacheSize method of WOApplication to define the number of instances of a component an application is

to keep in its cache. For example, if you want to maintain state between cycles of the request-response loop (that is, to ensure that state is transferred between user actions), set the pageCacheSize to 1.

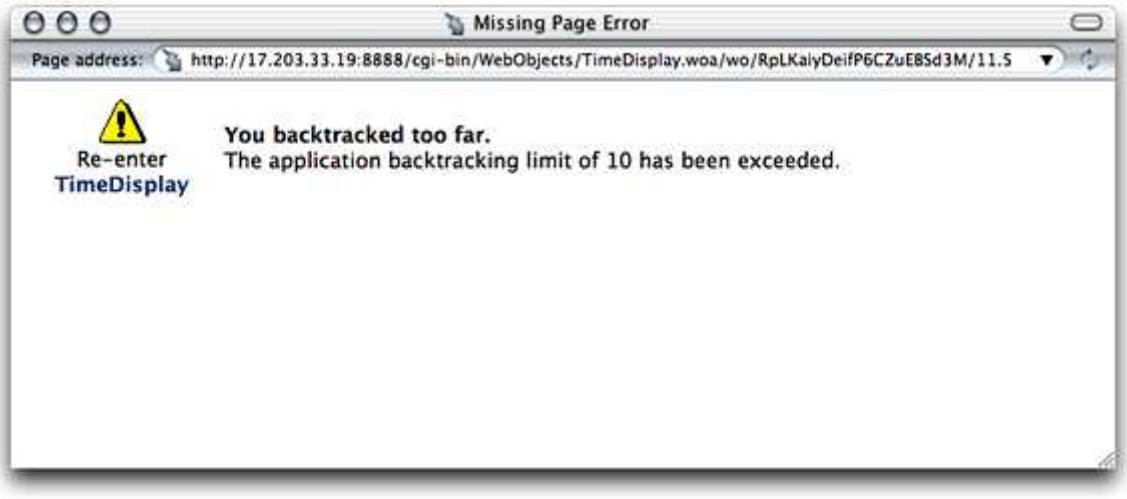
サーバサイドキャッシュのサイズを設定する。
サーバサイドキャッシュを完全に無効とする代わりに、アプリケーションがキャッシュに保存するコンポーネントのインスタンスの数を定義するためにWOApplication**の**setPageCacheSize**メソッドを使用することができる。例えば、リクエストーレスポンス ループのサイクルの間で状態を維持したい場合（すなわち、ユーザーアクションの間で状態を転送することを保証するため）、**pageCacheSize**を**1**に設定せよ。**
Modify Application's constructor by adding a call to setPageCacheSize, setting the pageCacheSize property to 10.

```
public Application() {
  super();
  System.out.println("Welcome to " + this.name() + "!");
  setPageRefreshOnBacktrackEnabled(true);
  setPageCacheSize(10);
}
```

Figure 6-2 shows the page an application sends to a web browser when a user backtracks too far (the page is no longer in the cache).

Applicationのコンストラクタに**setPageCacheSize**の呼び出しを追加して、**pageCacheSize**プロパティを**10**に設定する。図**6-2**は、ユーザーがバックトラックしすぎ（ページがキャッシュにない）場合に、アプリケーションが**Web**ブラウザに送るページを示す。

Figure 6-2 Backtracking-error webpage



You can customize the error page users receive by implementing the handlePageRestorationErrorInContext method in the Application class:

```
public WOResponse handlePageRestorationErrorInContext(WOContext aContext) {
```

```
    WOComponent nextPage;
    nextPage = (Error)pageWithName("Error", aContext);
    return nextPage.generateResponse();
}
```

In this code listing, a page is instantiated from a web component named Error, which you must build. The contents of the component are completely up to you, but should include the name of the application, your company's name, and a friendly message that tells the user that something went wrong and suggests ways they can return to normal operation.

ApplciationクラスのhandlePageRestrorationErrorInContext**メソッドを実装することで、ユーザーが受け取るエラーページをカスタマイズできる。**
このコードに示されている、Error**なる名前の**Web**コンポーネント（作成する必要がある）からページが生成される。コンポーネントのコンテンツは、完全に開発者に任されている。しかし、アプリケーションの名前、会社の名前、ユーザーが行ったミスと、通常の操作に戻ることができる適切な方法を示すわかりやすいメッセージを含むべき。**

3 リファレンスマニュアルから

3.1 WOApplication.setPageRefreshOnBacktrackEnabled()

public void setPageRefreshOnBacktrackEnabled(boolean aFlag)

When aFlag is true, disables caching of pages by the client by setting the page's expiration-time header to the current date and time. By default, this attribute is set to true. Disabling of client caching affects what happens during backtracking. With client caching disabled, the browser resends the URL to the server for the page requested by backtracking. The application will then return a new page to the browser (corresponding to a new WOComponent instance). This behavior is desirable when you do not want the user to backtrack to a page that might be obsolete because of changes that have occurred in the session.

aFlagが**true**である場合、ページの**expiration-time**ヘッダーを現在の日時に設定することで、クライアントのページキャッシュを無効にする。デフォルトでは、この属性は**true**である。クライアントベージキャッシュを無効とすることは、バックトラックしたときに何が起こるか？に影響する。クライアントキャッシュが無効である場合、ブラウザは、バックトラックで求められたページへのURLを再送する。《HTTPリクエストを再送するということ》
アプリケーションは、ブラウザへ（新しいWO**コンポーネントインスタンスに相当する）新しいベージを返すだろう。この振る舞いは、セッション内で起きた変更により、時代遅れとなっているであろうページへユーザーをバックトラックさせたくない場合、望ましいものである。**

When this flag is turned on and a request corresponding to a client backtrack occurs, the retrieved page will only be asked to regenerate its response. The first two phases of a normal request-response loop (value extraction from the request and action invocation) do not occur.

このフラグが立てられて、クライアントのバックトラックに対応するリクエストが発生した場合、得られるページは、レスポンスを再生成することのみが求められる。通常のリクエストーレスポンスループの最初の2フェーズ（リクエストからの値取得と、アクションの起動）は発生しない。

Parameters: aFlag - when true caching of pages by the client is disabled

See Also: isPageRefreshOnBacktrackEnabled()

3.2 WOApplication.setpageCacheSize

public void setPageCacheSize(int anUnsigned)

Sets the number of page instances the cache will hold. Disable page caching by passing 0 for anUnsigned. Pages that require state must be cached in order to access that state for the very next request. If you have components that hold state (which is the usual case), disable page caching by passing 1 for anUnsigned.

キャッシュが保持すべきページインスタンスの数を設定する。0を与えると、ページキャッシュが無効となる。状態が必須のページは、まさにその次のリクエストで状態にアクセスするため、キャッシュされる必要がある。
コンポーネントに状態を維持させたい場合は、1を与えて、ページキャッシュを無効とすること。

Parameters:

anUnsigned - the number of pages the cache will hold

3.3 WOApplication.setCachingEnabled

public void setCachingEnabled(boolean aBool)

Sets whether or not component caching is enabled. If this is enabled, changes to a component will be reparsed after being saved, assuming the project is under the NSProjectSearchPath. Note that this has no effect on page caching.

コンポーネントキャッシュが有効とするかどうかを設定する。
有効である場合、保存された後でコンポーネントへの変更は再パースされる。プロジェクトはNSProjectSearchPath下にあることを仮定する。
ページキャッシュには影響を与えないことに注意せよ。

ページキャッシュではなく、ページ定義（＝Webコンポーネント定義）のキャッシュに影響するということ。

Parameters:

aBool - boolean specifying whether or not component caching is enabled

4 ブラウザの実装

4.1 InternetExplorer

マイクロソフトサポート技術情報 [HOWTO] Internet Explorer でキャッシュを無効にする[3] より

-- begin --

概要

この資料では、HTTP ヘッダーを使って、Internet Explorer で Web ページのキャッシュをコントロールする方法について説明します。

Microsoft Internet Information Server (IIS) を使用すると、特定の Active Server Pages (ASP) ページの一番最初に次のスクリプトを使用することで、非常に動的なページを簡単にマークすることができます。 <% Response.CacheControl = "no-cache" %>

<% Response.AddHeader "Pragma", "no-cache" %>

<% Response.Expires = -1 %>

詳細

Expiration および Expires ヘッダー

すべての Web サーバーで、全 Web ページの有効期限に関するスキームを使用することを強くお勧めします。Web サーバーは、リクエストを実行するクライアントに返すすべてのリソースで、HTTP Expires 応答ヘッダーを使って有効期限情報を提供する必要があります。ほとんどのブラウザおよび中間に存在するプロキシでは、この有効期限情報を重要視し、この情報を使ってネットワーク上の通信の効率化を図るようになっています。

常に Expires ヘッダーを使用して、クライアントがサーバー上の特定のファイルを更新する必要のある最適な時間を指定する必要があります。ページが定期的に更新される場合、次の更新までの期間が最適な時間指定になります。たとえば、毎朝 5 時に更新されるデイリー ニュースが Web ページに掲載されているとします。このニュースページが設置されているサーバーは、次の日の朝 5 時の値を持つ Expires ヘッダーを返す必要があります。この動作が行われた場合、そのページが実際に変更されるまで、ブラウザは Web サーバーにアクセスする必要がありません。

変更が予定されていないページには、約 1 年の期限の値を指定する必要があります。多くの場合、Web サーバーには、すぐに変更する必要のある情報を含むページが 1 つ、または複数含まれています。こうしたページでは、サーバーにより、Expires ヘッダー

[3] マイクロソフト サポート技術情報 - 234067 [HOWTO] Internet Explorer でキャッシュを無効にする (http://support.microsoft.com/default.aspx?scid=kb;ja;234067&Product=ie)

の値を "-1" に設定する必要があります。ユーザーによるリクエストが発生した場合、Internet Explorer は通常、If-Modified-Since の条件リクエストを使って、そのページを更新するために Web サーバーにアクセスします。しかし、そのページはディスク キャッシュ ("インターネット一時ファイル") に保持され、リモート Web サーバーにアクセスすることなく、適切な状況で使用されます。たとえば、[戻る]、[進む] ボタンを使って履歴にアクセスする場合やブラウザがオフライン モードの場合などです。

HTTP1.1仕様(RFC2616)ではExpiresヘッダーは以下のフォーマットと規定されており、Expiresに-1を指定すると、RFC違反となってしまう。が、特に-1としなくてもキャッシュの破棄は可能なようである。

```
Expires: wkday "," SP date1 SP time SP "GMT"
   wkday = "Mon" | "Tue" | "Wed"
           | "Thu" | "Fri" | "Sat" | "Sun"
   date1 = 2DIGIT SP month SP 4DIGIT
   time  = 2DIGIT ":" 2DIGIT ":" 2DIGIT
   DIGIT = <any US-ASCII digit "0".."9">
   month = "Jan" | "Feb" | "Mar" | "Apr"
           | "May" | "Jun" | "Jul" | "Aug"
           | "Sep" | "Oct" | "Nov" | "Dec"
```

例:
Tue, 29 Feb 2000 12:00:00 GMT
※ "GMT" とはグリニッジ標準時に換算した時刻であることを意味します。HTTP の規定では、すべての時刻は GMT タイムゾーンで指定します。

Cache-Control ヘッダー

Web ページの中には非常に動的なものが存在し、こうしたページの場合、ディスク キャッシシュの必要がありません。このため、Internet Explorer では HTTP 1.1 Cache-Control ヘッダーをサポートしています。このヘッダーの働きにより、HTTP 1.1 サーバーで [no-cash] の値が指定されている場合、特定の Web リソースのキャッシシュをまったく行わないようにすることができます。

キャッシュに保存されていないページにアクセスするには、ブラウザが Web サーバーに再接続する必要があるため、サーバーは、必要な場合にのみ Cache-Control ヘッダーを使用しなければなりません。通常は、"Expires: -1" の使用をお勧めします。

Pragma : No-Cache ヘッダー

従来の HTTP 1.0 サーバーでは Cache-Control ヘッダーを使用することができません。HTTP 1.0 サーバーとの下位互換性を維持するため、Internet Explorer では、HTTP Pragma: no-cache ヘッダーの特別な利用方法をサポートしています。クライアントが、セキュリティで保護された状態 (https://) でサーバーと通信し、サーバーが、Pragma: no-cache ヘッダーを持つ応答を返した場合、Internet Explorer はその応答をキャッシュしません。

しかし、Pragma: no-cache ヘッダーは、こうした用途を想定したものではないことに注意してください。HTTP 1.0 と 1.1 の仕様によれば、このヘッダーは、リクエストのコンテキストにおいてのみ定義されており、応答に関する定義はありません。このヘッダーは実際にはプロキシ サーバーで使用され、特定の重要なリクエストが対象となる Web サーバーに届かないようにすることを目的としています。将来的に適用する場合は、Cache-Control ヘッダーを使ってキャッシュをコントロールする必要があります。
-- end --

5 I_love_my より

From http://homepage.mac.com/I_love_my/webobjects.html より

WebObjects is Apple's pure Java application server. NeXT pioneered the application server market with this excellent piece of software long before Java even existed. If this is news to you, check out the WebObjects product pages.

Even as the application server market expands, Apple still offers the most versatile solution.

The persistence layer offered by the Enterprise Objects Framework (EOF) is a very powerful tool that compares well to the much hyped EJBs, by actually providing object persistence - not just hooks for you to plug into - even in complex object-to-relational mapping sitations. In fact, EJBs focus mainly on object distribution and end up being a non-portable equivalent to CORBA with virtually no object persistence. EOF however specialises in fine grained object persistence, making it the more likely candidate for OO applications that need database access.

While WebObjects supports Java clients and standalone applications, it is mostly used to create thin client web applications. Again it can outperform its closest competitor, JSP, by providing a clean architecture that separates HTML files from Java code. In fact, WebObjects uses an architecture of reusable components for an object oriented approach to interface building.

Well, there is so much more to say, but some day you will have to check it out on your own. I will, of course, be glad to engage in further discussion. Just email me.

5.1.1 Client-side backtracking - sample code

Client-side backtracking - sample code

This code snippet provides ways to handle client-side backtracking (the evil back button).

The main problems related to backtracking are:

* Wrong items being selected from WORepetitions after the user backtracked
* A component being asked to perform an action that is inappropriate in the current state, e.g. cancel an edit after committing it

このコード断片は、クライアント サイド バックトラックを処理する方法を提供するものである。（邪悪なバックボタンめ！）
バックトラックに関する主要な問題は、
*** ユーザーがバックトラックした後、WORepetitionから誤ったアイテムが選択されること**
*** 現在の状態では適切でないアクションの実行が、コンポーネントに求められること**
（たとえば、コミットした後に、編集をキャンセルする）

The first problem is in fact caused by a more general problem: a component always exits with only one state, i.e. the latest one. The user may however backtrack to a cached page that matches a previous state: showing components that should be hidden, displaying an earlier batch of a display group,... .

The second problem is closely related, but differs by the fact that the former state of the component is permanently lost in that the moving to the new state has non reversible effects like deleting from a database.

実際、最初の問題は、より一般的な問題により引き起こされる。コンポーネントは常に１つの状態（ ＝ 最新の状態）だけを持つ形で存在する。しかし、ユーザーは、以前の状態に対応するキャッシュされたページにバックトラックするかもしれない。：ディスプレイグループの先のバッチ（＝ページ）を表示するとか、隠されているべきコンポーネントを表示するとか・・・

2番目の問題は、密接に関連している(?)が、先のコンポーネントの状態は、新しい状態は、データベースから削除することのように､逆に戻すことができない効果をもつ状態に移行するなどの点で(?)恒久的に失われている点で(?)異なっている。

Another problem is that some actions modifying a display group may get repeated, causing unwanted behaviour like deleting the wrong item. This situation fits the 2nd problem: the state change is so dramatic that backtracking is not tolerable.

もうひとつの問題は、ディスプレイグループを修正するアクションは、繰り返される可能性があり、誤った項目を削除してしまうような望まない振る舞いを引き起こすなど、
この状況は、２番目の問題にフィットする。状態変化が、劇的であるために、我慢できない。ということ。

The workaround I suggest tries to address these two problems. I tried to come up with a mechanism that is very general and should apply to many situations. Most notably, the business

decision of what to do when the user has backtracked is left to the afffected component. There will however be situations that can not be addressed by my approach. This fix does not address problems with users reloading a page.

これらの２つの問題に答えるために、試行錯誤した、私の提案するworkaround
多くの状況に適用できる、一般的な機構を提案しようとした。
明白に、ユーザーがバックトラックしたときに、何をすべきかというビジネス上の決定は影響するコンポーネントに残される。しかし、これらは、私のアプローチによって答えられる状況ではない。このfixは、ユーザーのページのリロードの問題に答えるものではない。

The MYBackTrackComponent class detects backtracking by comparing context IDs between requests. Deciding what to do in that event is up to the subclass. Such behavior may be provided by overriding the following methods:

```
protected abstract boolean needsBackTrackDetection();
protected void sleepInContext(String contextID)
protected void awakeFromContext(String contextID)
```

MYBackTrackComponentクラスはリクエストの間のコンテキストIDを比較することで、バックトラックを検出する。起こったときに、何をすべきかを決定することは、サブクラスのやるべきことである。以下のメソッドをオーバライドすることで、このような振る舞いを提供できる。

Two common situations may be handled by simply implementing needsBackTrackDetection():

* The component may consider backtracking inappropriate an refuse it by throwing an exception that should be handled by the exception handler on the application level. This is likely the appropraite behavior of pages that modify EOs or are part of a sequential workflow. All that needs to be done to get this behaviour is to implement needsBackTrackDetection() to return true.

* The component may accept the request without any special behavior. By implementing needsBackTrackDetection() to return false, MYBackTrackComponent may be configured to behave exactely like a standard WOComponent. This is useful if your architecture forces a component to extend MYBackTrackComponent even though it does not require its behavior. This is likely to be the appropriate behavior that do not change state. E.g. inspect pages.

2つの共通の状況は、単にneedsBackTrackDetection()を実装することで処理できる。

* コンポーネントがバックトラックを不適切なものとみなし、アプリケーションレベルの例外ハンドラによって処理されるべき例外をスローして、拒否する(?)
これは、一連のワークフローの一部をなすか、EOを修正するなどのページの適切な振る舞いで

あるとようだ。この振る舞いを実現するためにやるべきことは、needsBackTrackDetection() がtrueを返すように実装することである。
* コンポーネントは特別な振る舞いなしにリクエストを受け付けなければならない。needsBackTrackDetection()がfalse返すように実装することで、MyBackTrackComponentは通常のWOコンポーネントとまったく同じように振舞うよう、コンフィギュアすることができる。
これは、この振る舞いが必要でないが、コンポーネントがMyBackTrackComponentを継承せざるを得ない場合。
これは、状態を変化しない振る舞いの場合適切である。（閲覧ページ）

MYBackTrackComponent.DefaultImplentation is a sublcass of MYBackTrackComponent that offers utilities to implement another commonly usefull reaction to backtracking:

* The component may also attempt to restore the state it was in at the time the page the user is seeing was generated. This is likely to be the appropriate behavior for list pages: the current batch index of the WODisplayGroup needs to be restored in order to respect the user's selection

MyBackTrackComponent.DefaultImplementationは、バックトラックに対する一般的に有用な反応を実装したユーティリティを提供するMyBackTrackComponentのサブクラスである。

* ユーザーが見ているページを

Subclasses configure these behaviors by overriding the following methods:
protected NSArray getPersistentKeys()
The getPersistentKeys() returns the list of keys to access via key-value coding for storing/restoring component state.
A list page would override this to return an array containing the String "batchIndex" and implement the _getBatchIndex() and _setBatchIndex(int) methods. With this and a needsStatePersistence() that returns true the second behavior is activated.

This code plays nicely with the sever-side backtracking code sample posted on my web site. You merely need to modify MYBackTrackComponent to extend the MYBackButtonComponent class from the server-side backtracking example.

BTW, this code was written with component actions in mind and tested with WebObjects 5.1.x. It will probably need to be amended to work with direct actions. Obviously, this code is meant to be used with application().setPageRefreshOnBacktrackEnabled(false). I would be glad to hear your opinion on this code snippet and will gladly help you with more detailed information than provided above. Mail me at: I_love_my@mac.com

last edited: May 14, 2002

5.1.2 Server-side backtracking - sample code

Server-side backtracking - sample code

This code snippet provides an implementation of a server-side backtracking mechanism similar to the browser's back button.

The server-side implementation offers several benefits though:

- It guarantees that the destination component is still available
- It gives a more graceful, i.e. application like behavior, where backtracking brings up the previous page instead of a previous state of the current one. E.g. if a form validation error is sanctioned by redisplaying the form augmented with an error message, the browser back button would return to the form as it was before validation. Server-side backtracking would bring up the page that let to the form.

The main idea behind this code is that the constructor of top-level components is invoked at the page creation. That is when pageWithName is called. This is done in the request-response loop preceeding the one for which the page is destined. Thus we get a chance to access the preceding context.

Thus the only component that needs special code, i.e. extend MYComponent rather than WOComponent, is the component that has the back button. It can backtrack to any other page.

BTW, this code was written with component actions in mind and tested with WebObjects 5.1.x. It will probably need to be amended to work with direct actions.

I would be glad to hear your opinion on this code snippet and will gladly help you with more detailed information than provided above. Mail me at: I_love_my@mac.com

---

last edited: May 14, 2002