Practical WebObjects
Chapter 6 (Page 159-185):

The Secret Life of Components

WR

WR at Csus4.net

http://www.csus4.net/WR/

目次詳細

- The Hypertext Transfer Protocol
  - Spying on HTTP
- The Request-Response Loop, Briefly
  - The Role of the WebObjects Framework
  - The Big Picture
- The Request-Response Loop, Deeply
  - The Beginning: The Request Is Made
  - The Web Server Receives the Request
  - The woadaptor and wotaskd Forward the Request
  - The Application Receives the Request
  - The Application Forwards the Request to the Request Handler
  - The Request Handler Forwards the Request to the Component
  - The Main Event
  - Detour: Component Templates
  - The Take Values Phase
    - The Connection Between HTML Inputs and Elements
    - Text Input Elements Component
  - Perfoming the Request Action
    - Hyperlink Example Component
  - Generating the Response
    - Reprise of the Text Input Example Component
  - The End of the World As WO Knows It
  - … And Beyond
  - Complications
- Summary

本プレゼンテーションに関する注意点

- 書籍の記述はComponent Actionに限定されている
  - Direct Actionの振る舞い・仕組みについては記述されていない
- フレームワーク自体の挙動をカスタマイズするための記述については場合に応じて省略した
  - Sessionの代わりにユーザーが独自に定義した別のクラスを使うとか
  - WOContextの代わりに別のコンテキストクラスを使うとか
  - 詳細を理解したい方は、書籍を参照してください
- 資料内のクラス図らしきものは、概念的なレベルのもの
  - 利用関係、包含関係の「雰囲気」を表現していると理解してください。
  - 特に包含関係は怪しい。（コンテナクラスの省略など）

ひとこと

P159. The Secret Life of Components

- リクエストーレスポンスループを理解することはオブジェクトグラフ管理を理解するのと同じくらい重要
- コレを理解しておかないと、

いずれ壁にぶち当たる

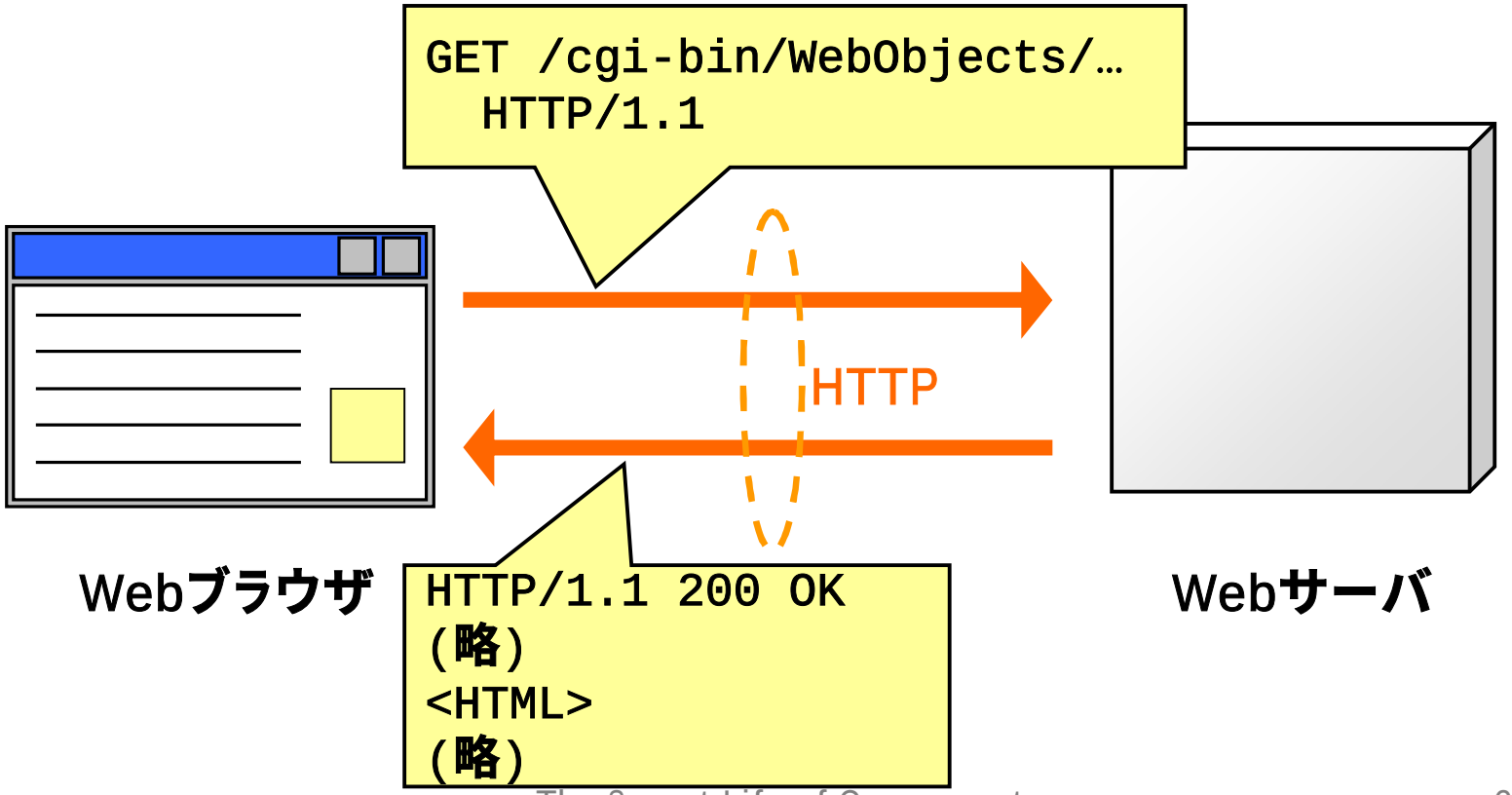
目次

- The Hypertext Transfer Protocol
  - Spying on HTTP
- The Request-Response Loop, Briefly
- The Request-Response Loop, Deeply
- Summary

The Hypertext Transfer Protocol

P159-160. The Hypertext Transfer Protocol

- HTTPを理解する意義
  - 基本的には理解する必要がない・・・
  - が、HTTPに対する知識は、トラブルに対処するための助けとなる



HTTPをトレースしてみる

P161-164. Spying on HTTP

- HTTPを直接見てみるのも有益
  - 学習のため
  - デバッグのため
- それにはいくつか方法がある
  - Proxy方式：WebScarab, etc...
    - HTTPに特化
    - クライアントにProxy設定必要
  - スニファ方式：Ethereal, etc...
    - 汎用（プロトコルを問わない）
    - クライアント設定不要

Tips

P164. Spying on HTTP

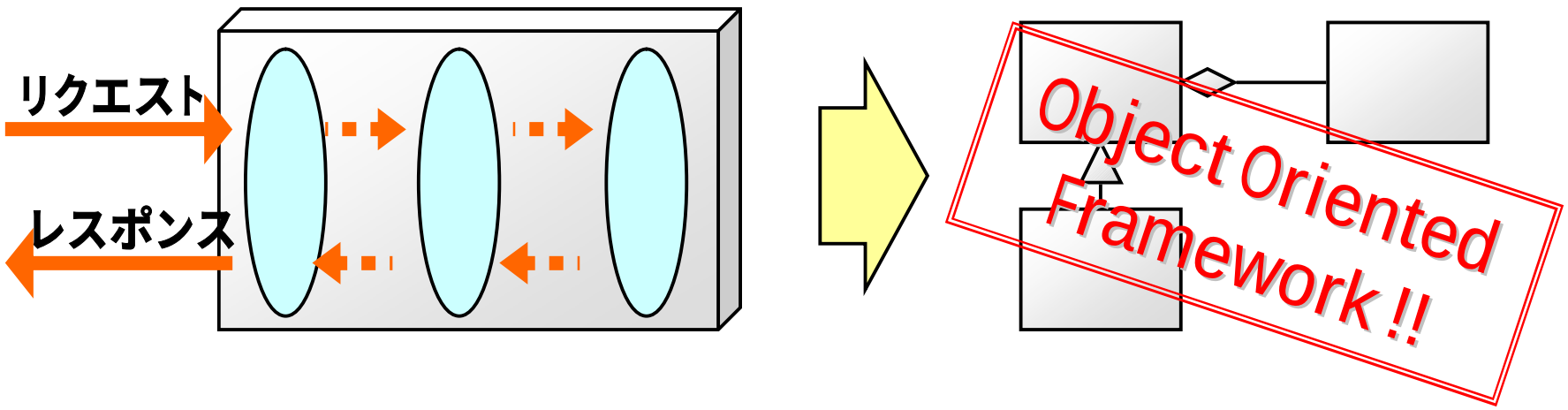
- **開発中でも、Webサーバを介してWebObjectsアプリケーションを動作させたほうが良い**
  - Direct Connectモードではなく
- **理由**
  - HTTP通信の正確な様子を見ることができる
  - Direct Connectモードと、Webサーバを介した動作モードでは、リソースへのアクセス方法が異なる

目次

- The Hypertext Transfer Protocol
- The Request-Response Loop, Briefly
  - The Role of the WebObjects Framework
  - The Big Picture
- The Request-Response Loop, Deeply
- Summary

WebObjectsフレームワークの役割

P164-165. The Role of the WebObjects Framework



- リクエスト処理をオブジェクト指向的に抽象化
  - リクエスト受け取り→処理実行→レスポンス生成＆発行みたいな・・・
- 処理の共通部分をフレームワーク化
  - アプリケーションに固有の部分を修正（オーバライド）してやればよい

- 以後、Component Actionリクエストの処理を中心に解説する
  - Direct Actionリクエスト、Webサービスリクエストはおいておく

Request-Responseループの処理をざっくり捉える

P165. The Big Picture

- Figure 6-6: Request-Responseループのざっくりした処理イメージ
  - Webサーバ内のwoadaptorがWOアプリケーションにリクエストを振り分ける
  - WOアプリケーション内では、4つのフェーズで処理が実行される
- Figure 6-7: WebObjectsフレームワークの主要4オブジェクトの処理イメージ
  - 基本的にApplication → Session → Page → SubComponentsなる処理順序
  - sleepメソッドは例外

目次

- The Hypertext Transfer Protocol
- The Request-Response Loop, Briefly
- The Request-Response Loop, Deeply
  - The Beginning: The Request Is Made
  - The Web Server Receives the Request
  - The woadaptor and wotaskd Forward the Request
  - The Application Receives the Request
  - The Application Forwards the Request to the Request Handler
  - The Request Handler Forwards the Request to the Component
  - The Main Event
  - Detour: Component Templates
  - The Take Values Phase
    - The Connection Between HTML Inputs and Elements
    - Text Input Elements Component
  - Perfoming the Request Action
    - Hyperlink Example Component
  - Generating the Response
    - Reprise of the Text Input Example Component
  - The End of the World As WO Knows It
  - ... And Beyond
  - Complications
- Summary

Request-Responseループ詳解

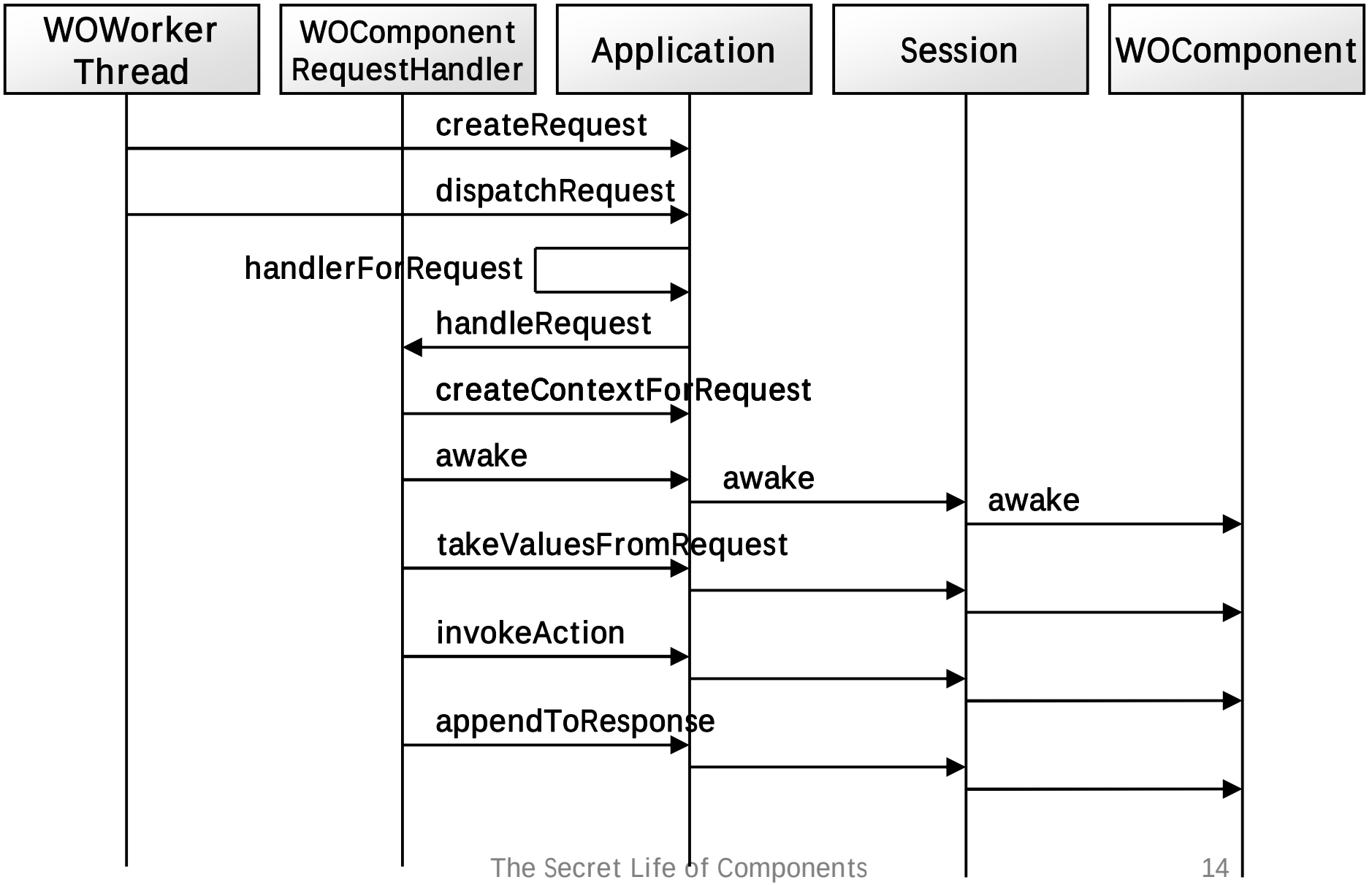
P166-167. The Request-Response Loop, Deeply

- 本章のゴール
  - 処理がどのように動作するかを理解する
  - 処理をコンフィギュアする箇所、方法を理解する
  - 効率的なデバッグのため、ある事象が発生する箇所を理解する
  - コンフィギュアにまつわるトラブルシュートができるように、失敗しやすいポイントを見ておく

- 以降Figure 6-8の処理を追ってゆく

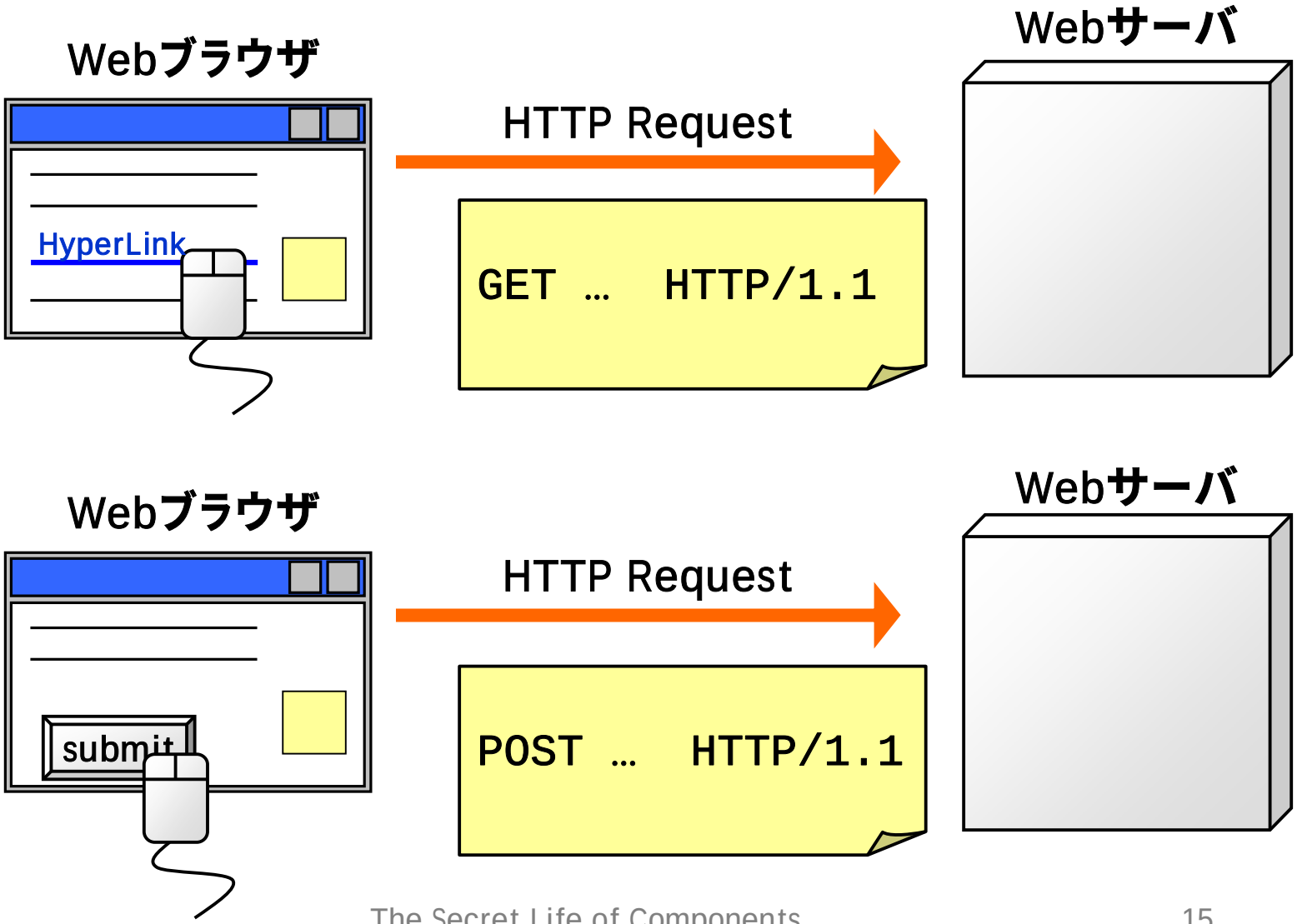
Request-Responseループ

P166-167. The Request-Response Loop, Deeply



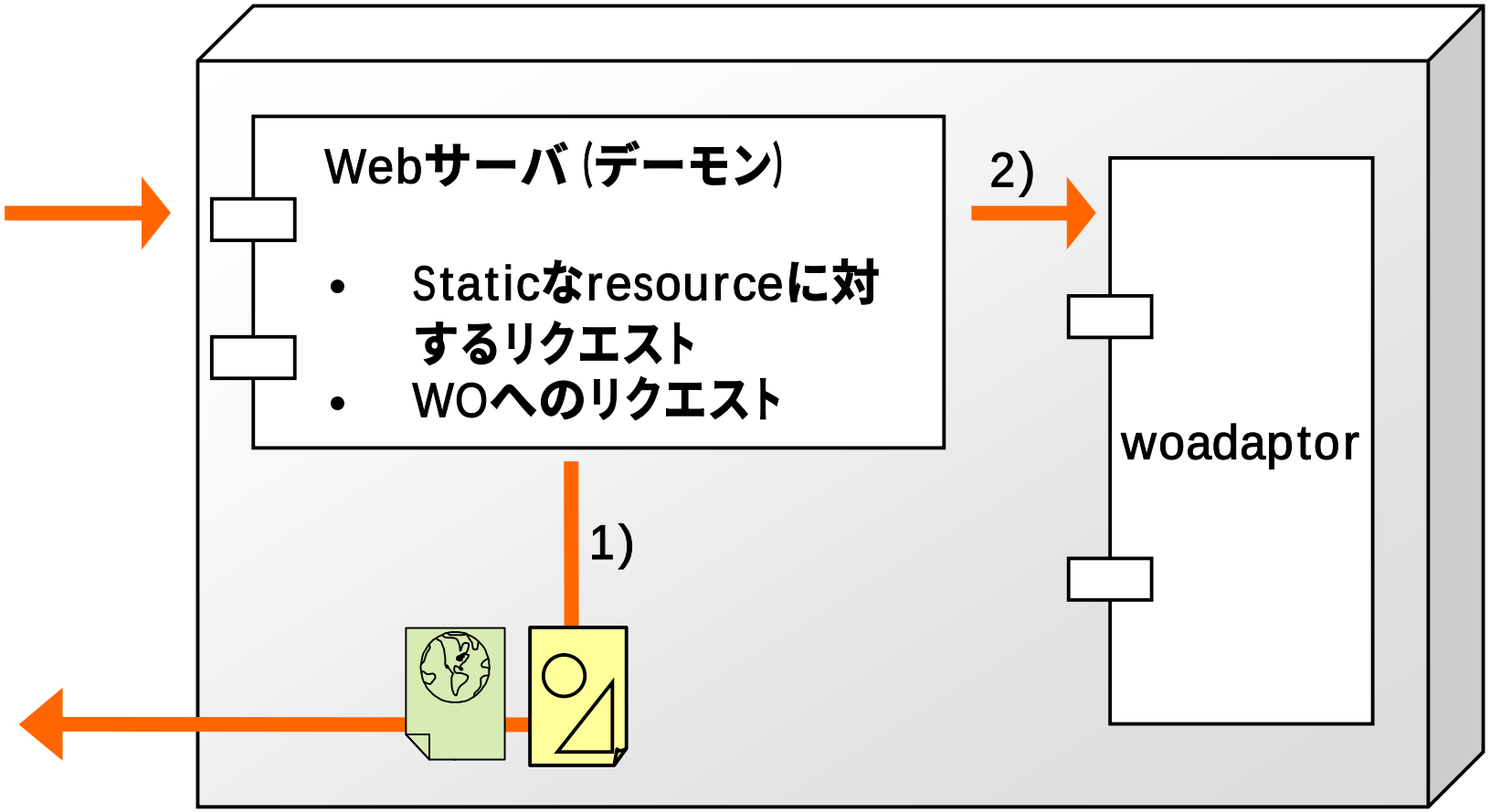
リクエストの生成

P167-. The Beginning: The Request Is Made



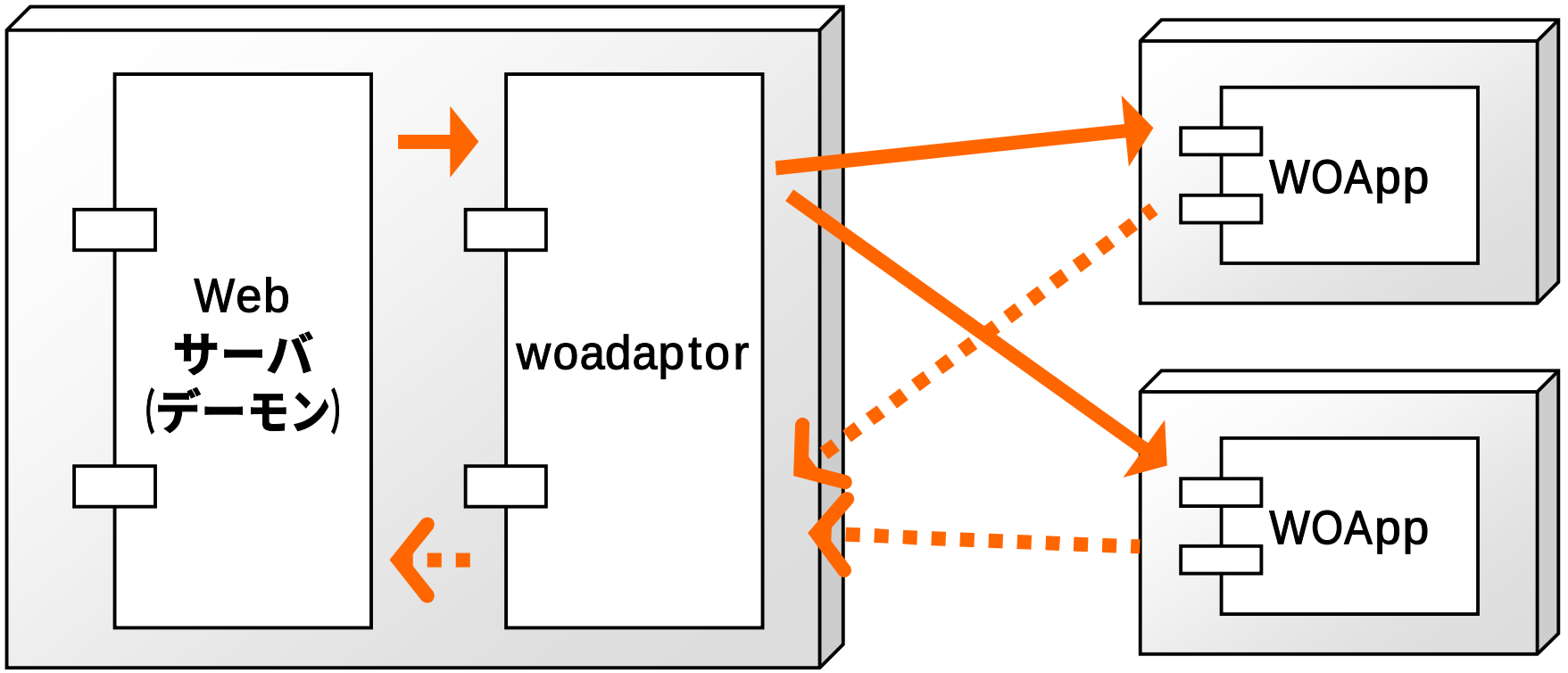
Webサーバのリクエスト処理

P168. The Web Server Receives the Request



WOリクエストの転送 - woadaptor

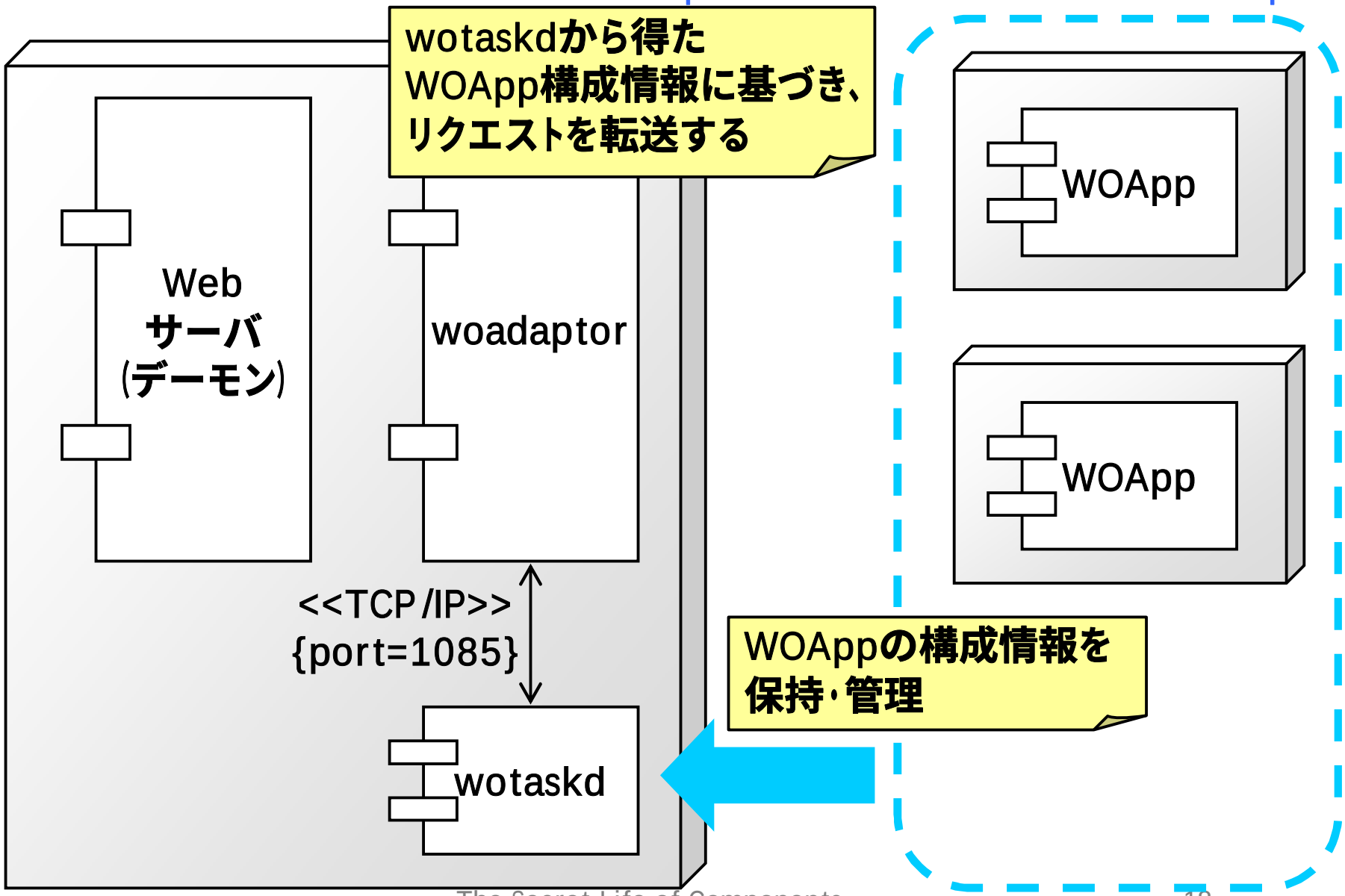
P168-169. The woadaptor and wotaskd Forward the Request



- standard WebObjects deployment with JavaMonitor and WebObjects Task Daemon (wotaskd)

WOリクエストの転送 - wotaskd

P168-169. The woadaptor and wotaskd Forward the Request



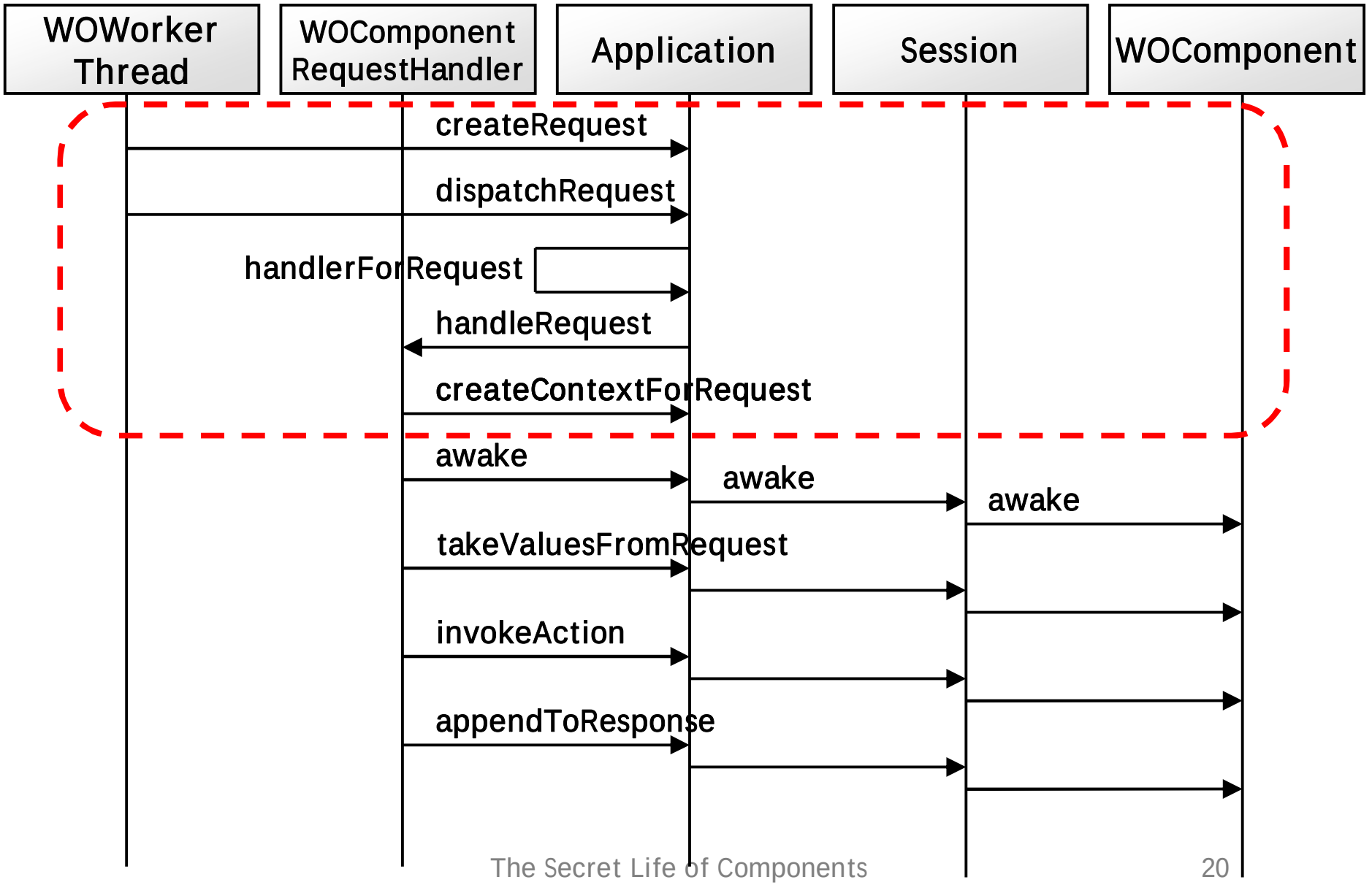
WOApp構成に関するトラブルシュート方法

P168-169. The woadaptor and wotaskd Forward the Request

- http://<<hostname>>:1085 **にアクセスしてみる**
  - WOApp**構成情報が**XML**フォーマットで返される･･･はず。**
  - **エラーの場合は、エラーメッセージが返される**

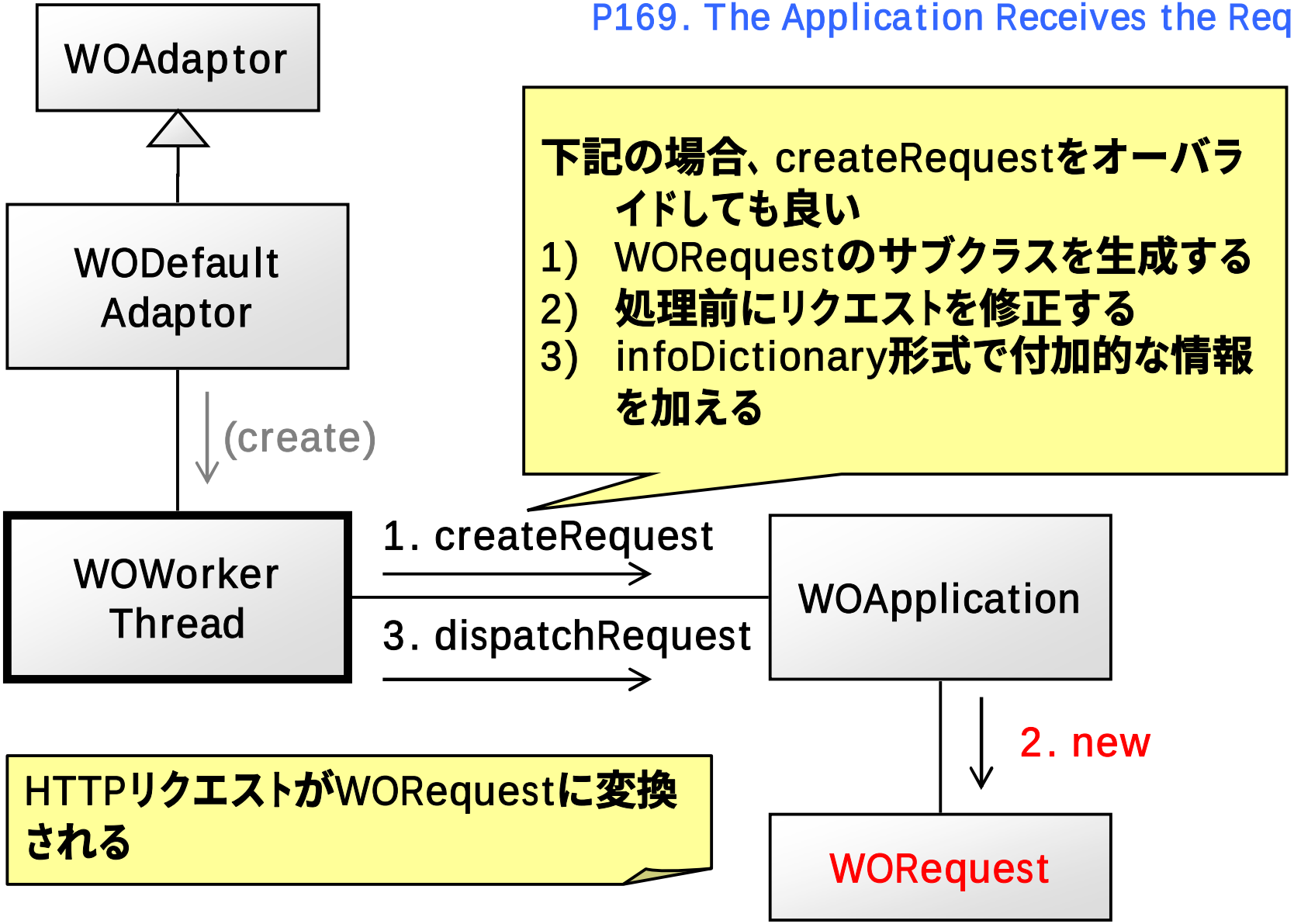
- **ログを見る**
  - /tmp/logWebObjects or C:¥Temp¥logWebObjects
    - **あらかじめ空のファイルを作成しておく必要アリ**
    - **ファイルを削除すると、ログの出力停止**

Request-Responseループ



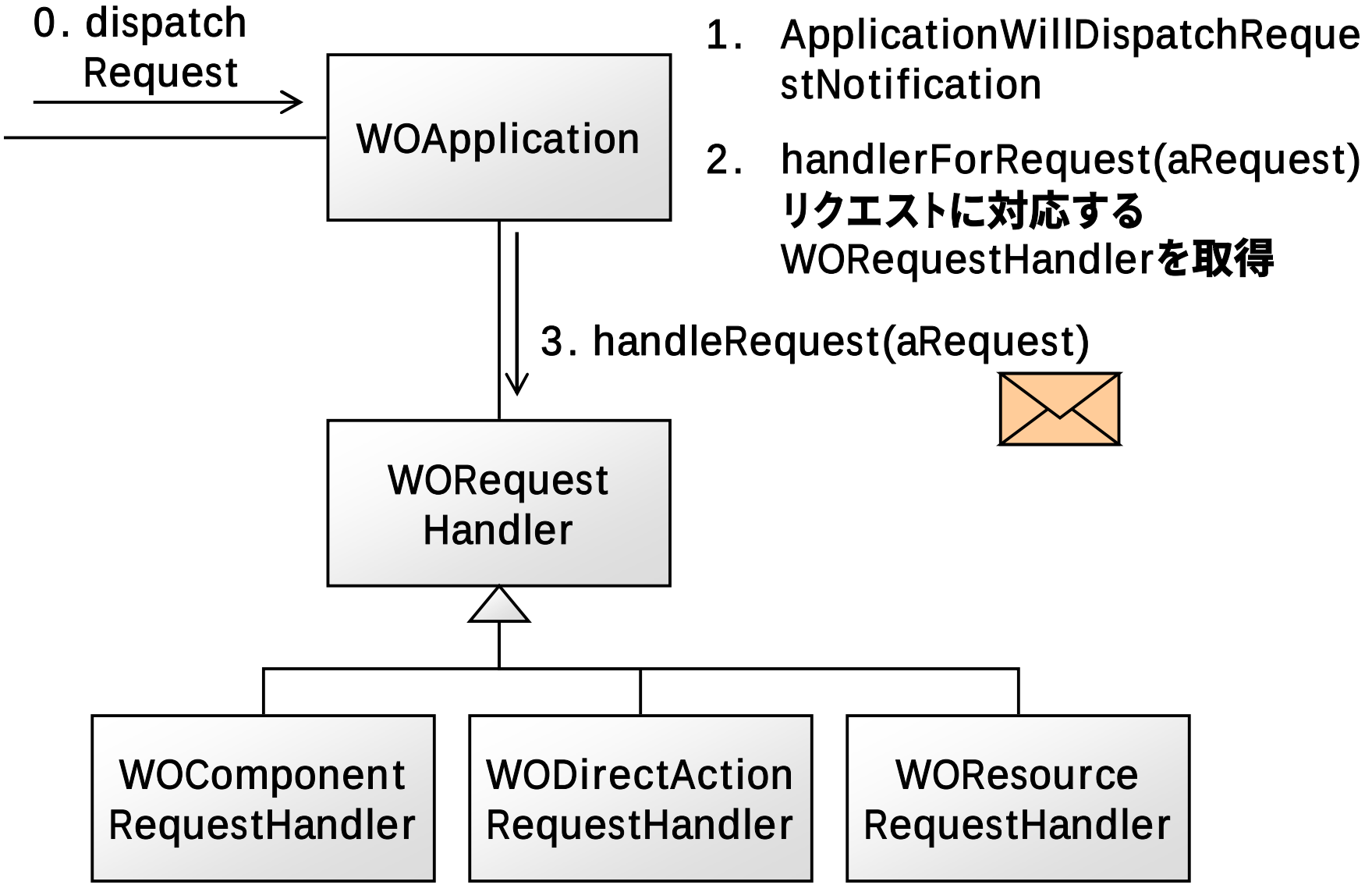
リクエスト受信処理 - WOApp

P169. The Application Receives the Request



リクエストハンドラへリクエストを転送

P169-170. The Application Forwards the Request to the Request Handler



キーとリクエストハンドラ

P169-170. The Application Forwards the Request to the Request Handler

- http://<hostname>/.../<woa>/.../wo/...

リクエスト ハンドラクラス	機能	Key
WOComponent RequestHandler	WOComponentに実装されたアクションのためリクエストを処理する	wo
WODirectAction RequestHandler	WODirectActionに実装されたアクションのためリクエストを処理する	wa
WOResource RequestHandler	リソース リクエストを処理する	wr

カスタムリクエストハンドラ

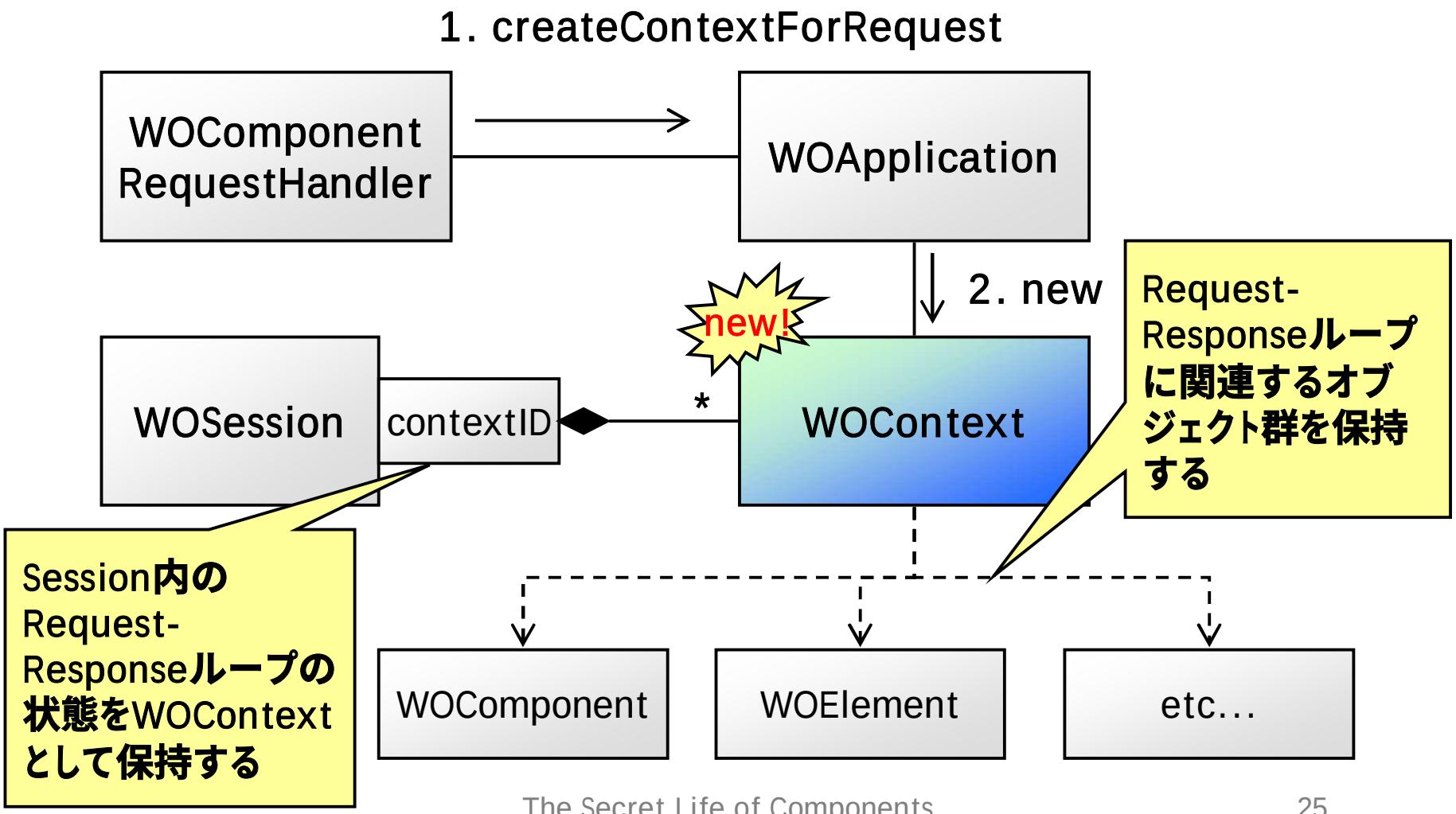
P169-170. The Application Forwards the Request to the Request Handler

- wo/wa/wr以外のカスタムリクエストハンドラを作成できる
  - 要件：handleRequestを実装すること
  - 作成例) RSS feed用リクエストハンドラ
    - http://www.wodeveloper.com/omniLists/webobjects-dev/2002/August/msg00010.html
- カスタムリクエストハンドラの登録
  - WOApplicationのregisterRequestHandlerを実行する
- URLの生成
  - WOConextの該当メソッドを呼べ！ (省略)

コンポーネントへのリクエストをハンドラへ転送

P170-171. The Request Handler Forwards the Request to the Component

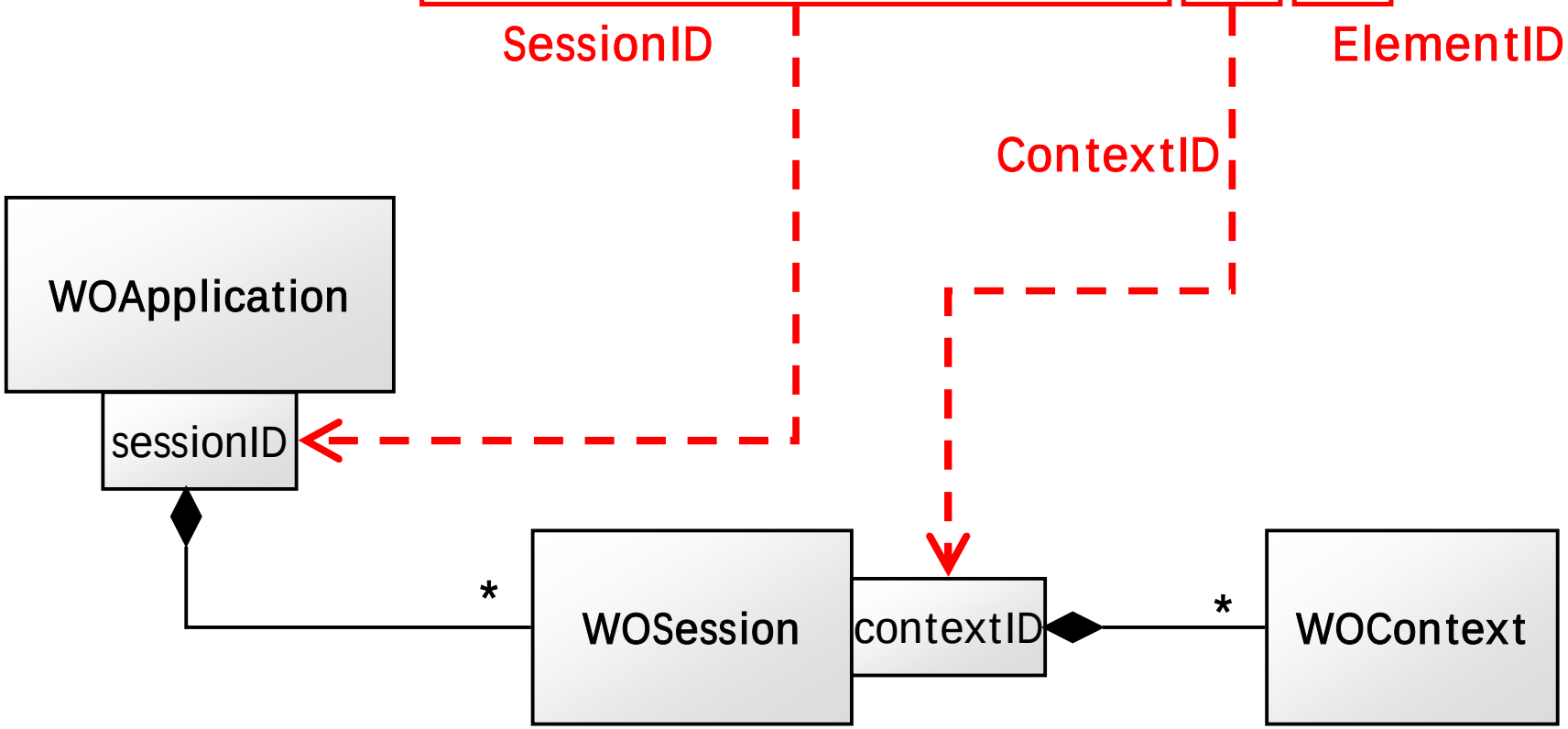
- **以下、WOComponentRequestHandlerに限定して解説する**



URLとContextID

P170-171. The Request Handler Forwards the Request to the Component

```
http://localhost/cgi-bin/WebObjects/App.woa/-
    1234/wo/EgcbgQmbDaRCNP856aQj0/27.1
```



※：上記のクラス図は概念的なもので、正確ではない。

WOContextをカスタムContextクラスへ置き換える

P170-171. The Request Handler Forwards the Request to the Component

- 目的
  - URL生成ロジックをカスタマイズする(see 4章)
  - とか・・・
- 方法
  - 一時的な置き換え
    - ApplicationクラスのWOContextClassNameをカスタムContextクラスに設定する
  - 恒久的な置き換え
    - ApplicationクラスのsetContextClassNameを実行する
    - ApplicationクラスのcontextClassNameをオーバライドする
  - コンストラクタにパラメータを追加など
    - ApplicationクラスのcreateContextForRequestをオーバライドする

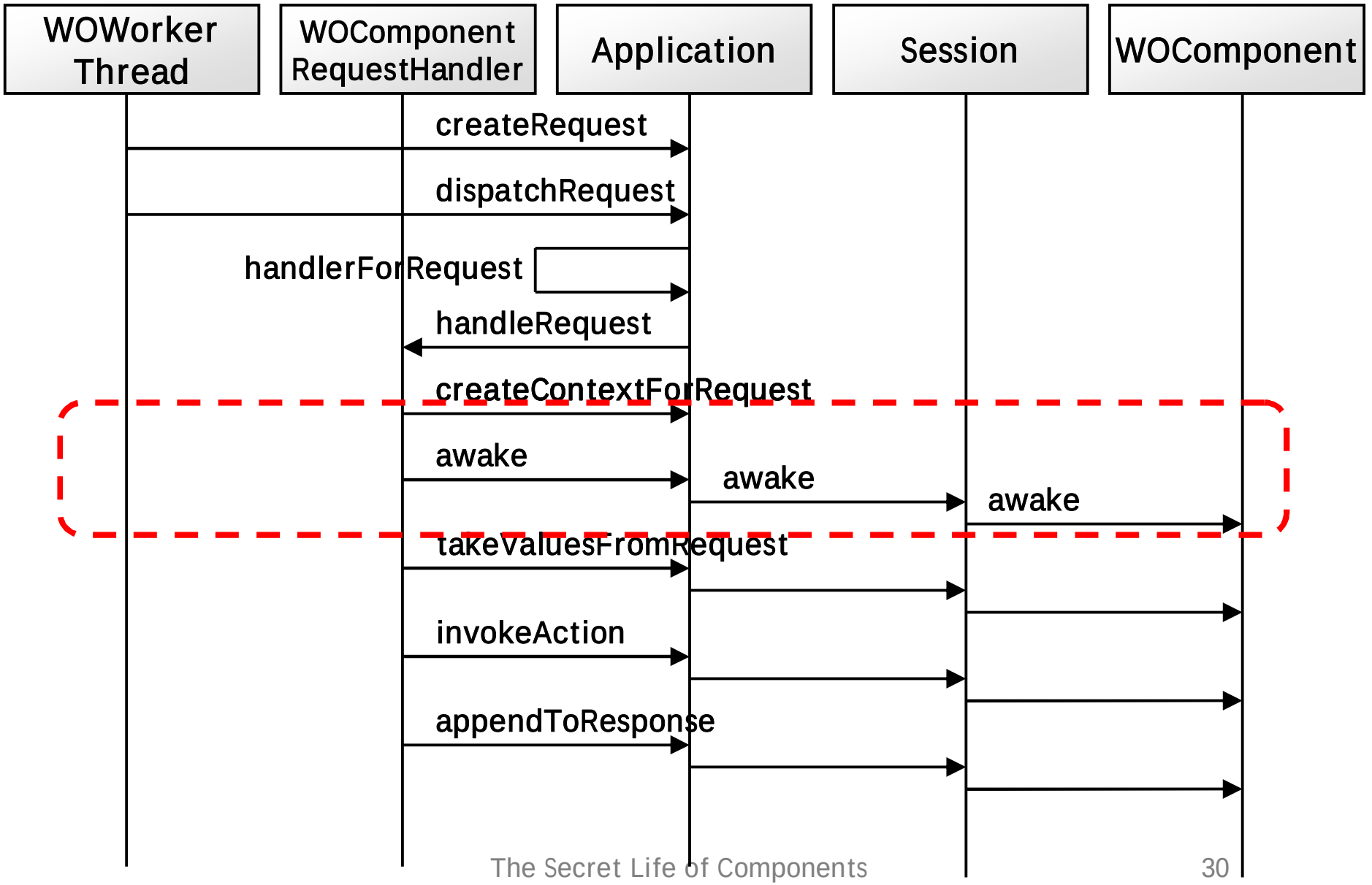
本編開始

P171-173. The Main Event

- Application::awake
- Sessionの生成または復元 from sessionStore
- Session::awake
- Pageの生成または復元

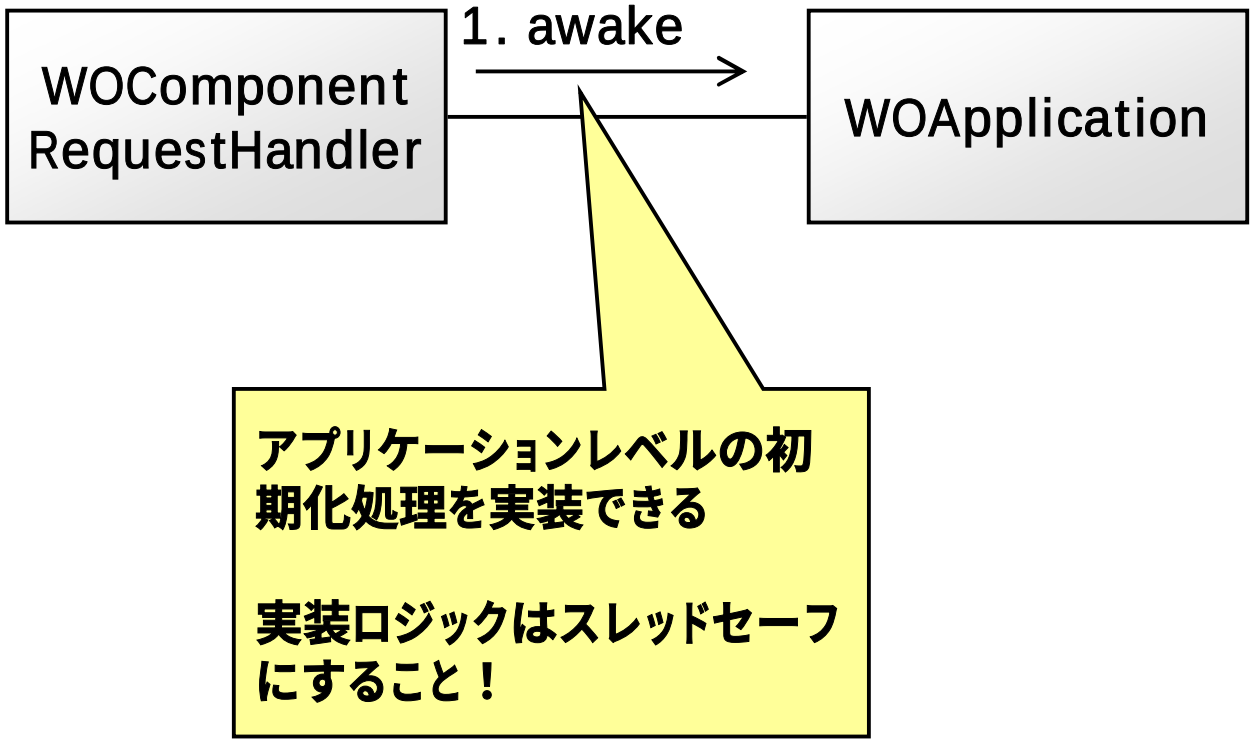
- 注意点
  - 処理が進むに従い・・・
    - オブジェクト(Application, Session, ...)が生成・復元される
    - WOContextの状態が変化する

Request-Responseループ



Applicationのawake

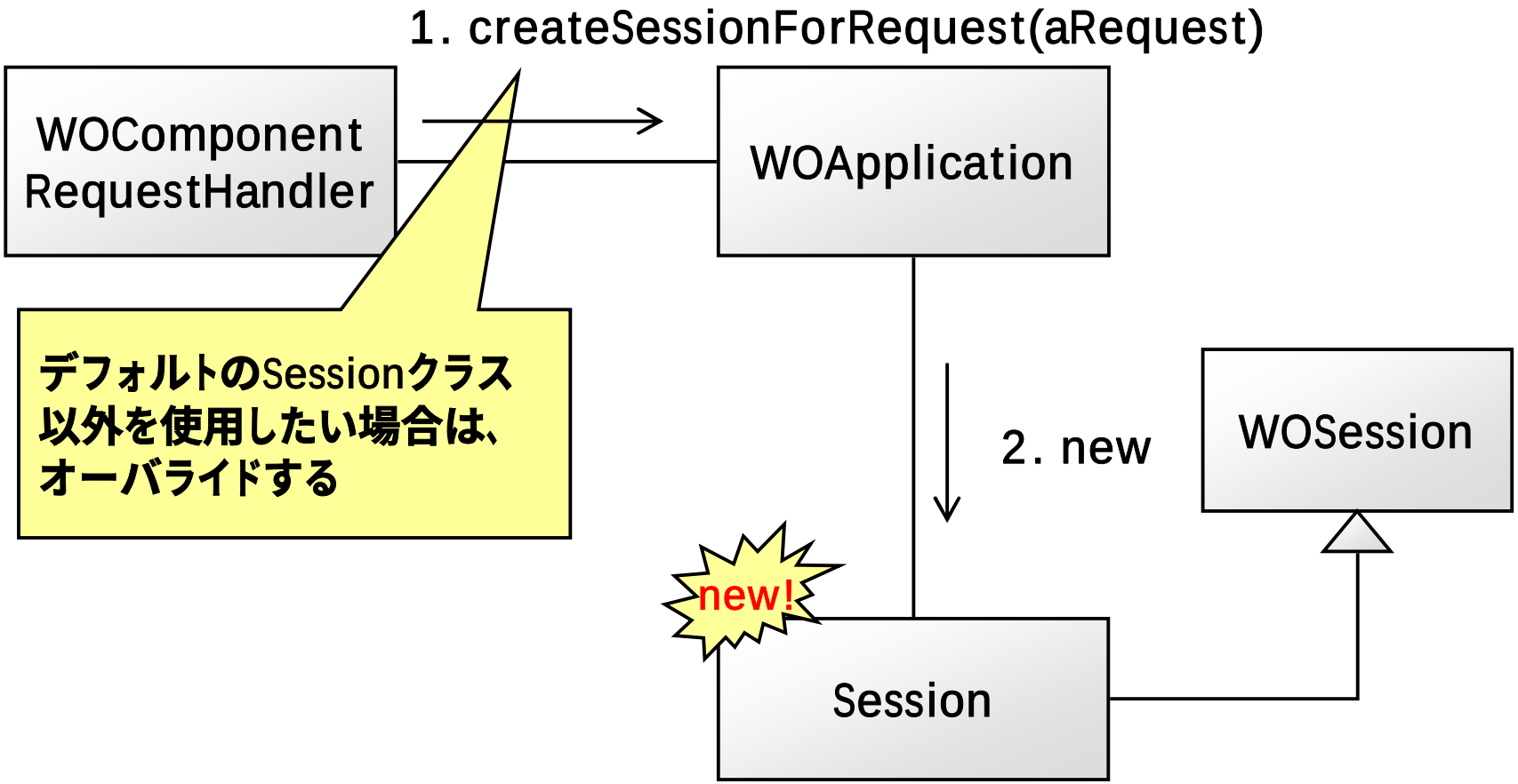
P171-173. The Main Event



Sessionの作成

P171-173. The Main Event

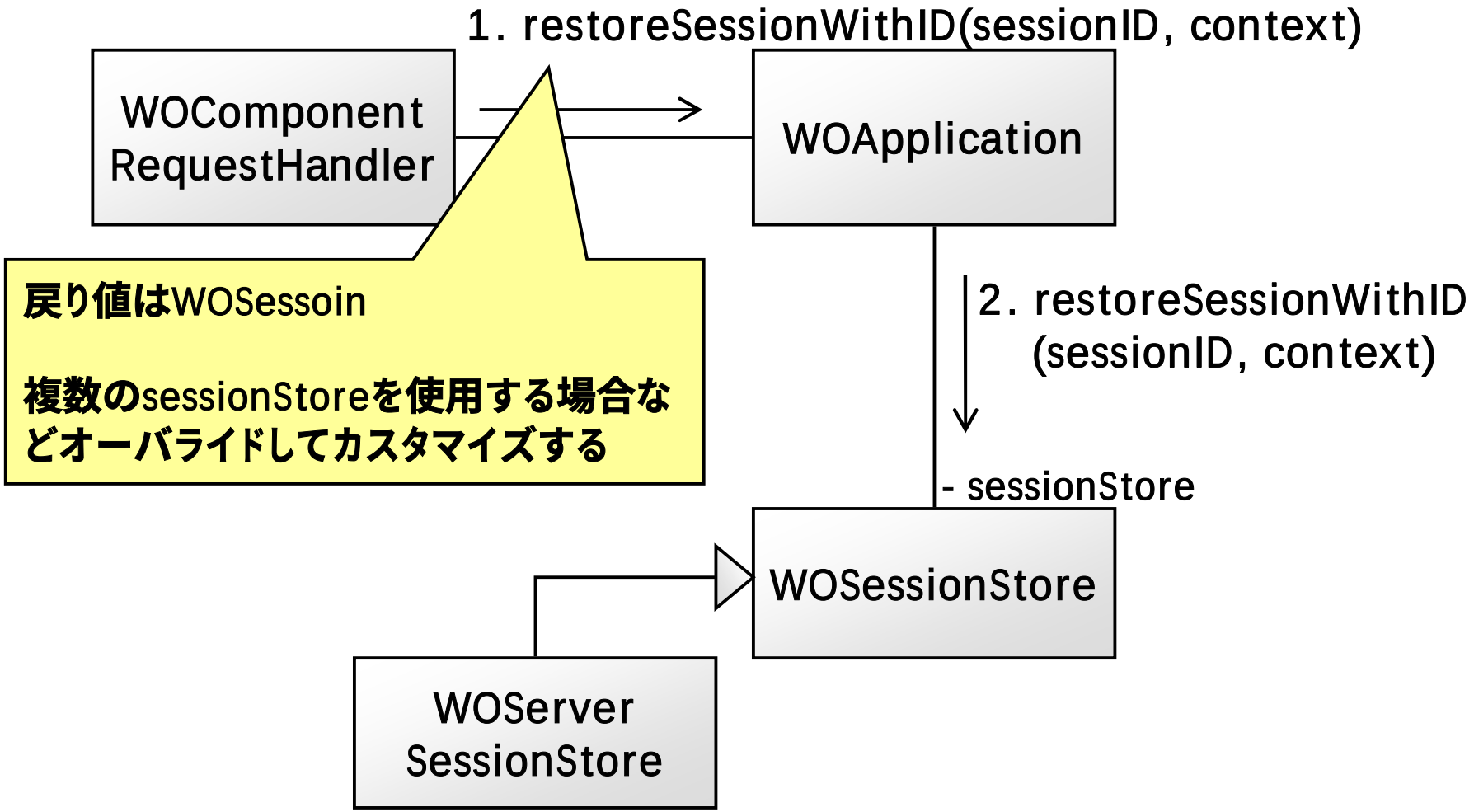
- Sessionが存在しない場合



Sessionの復元

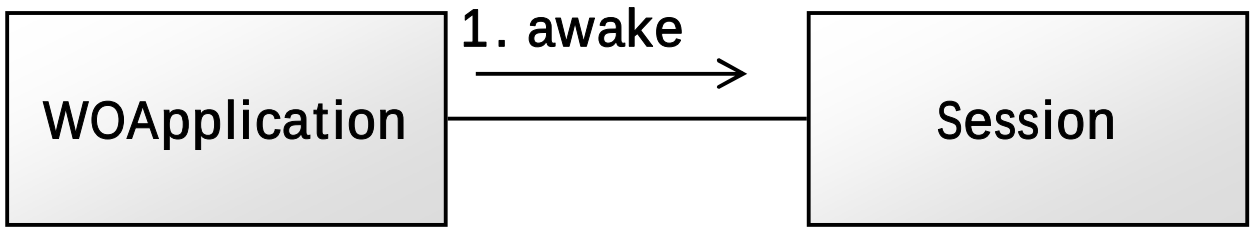
P171-173. The Main Event

- Sessionが存在する場合



Sessionのawake

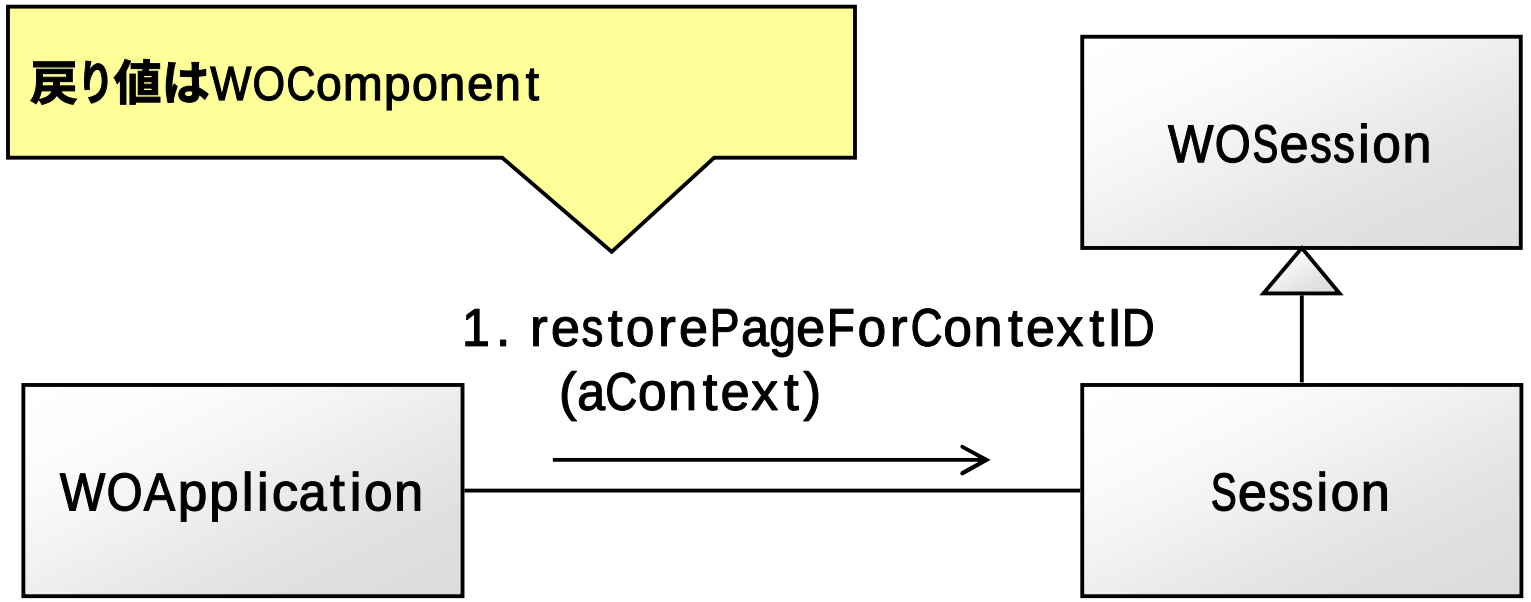
P171-173. The Main Event



- Session::awake
  - **セッションレベルの初期化処理を実装可能**
  - **実装ロジックは非スレッドセーフでよい**
  - **デフォルトの実装**
    - defaultEditingContextをロックしている
  - **その他**
    - 3章で紹介したMultiECLockManagerもココでecをロックしている

Pageの復元

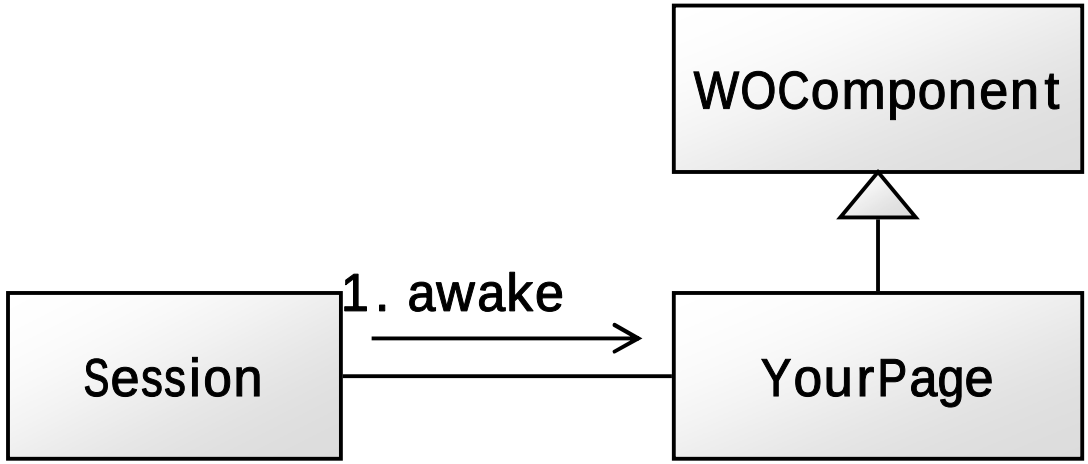
P172. The Main Event



- WOSessionはPageの保存･復元機能を持つ
- 保存するPageの数はWOApplication::pageCacheSizeで定義される
- 該当Pageがキャッシュに無い場合は WOApplication::handlePageRestrationErrorInContextが呼ばれる
- restorePageForContextIDのオーバライドは難しいので注意せよ

Pageのawake

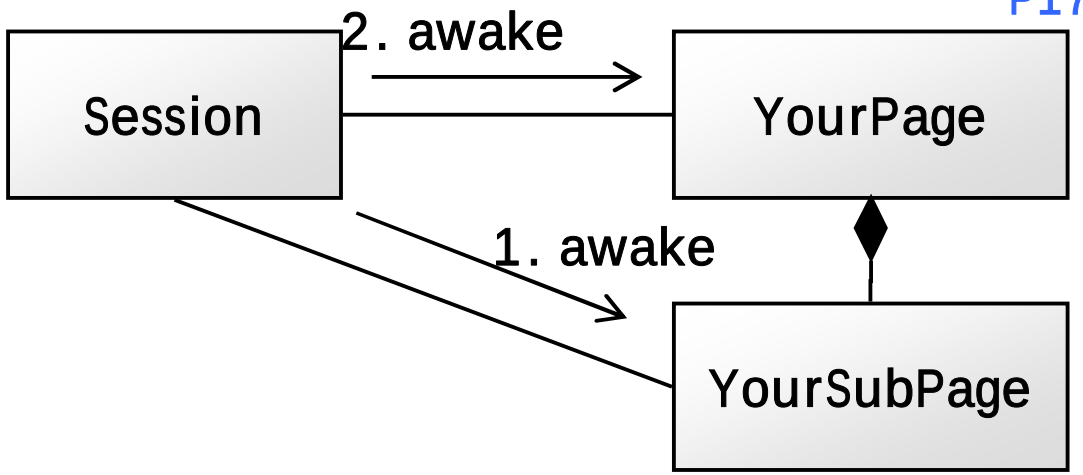
P172. The Main Event



- WOComponent::awake
  - **ページレベルの初期化ロジックを実装できる**
    - **以後使用しないオブジェクトへの参照はsleepで廃棄しなさい**
  - awake**メソッド実行時は、バインディングが実行されていないことに注意**

入れ子Pageのawake

P172. The Main Event



- SubPage → Pageの順序でawakeが実行される
- WODynamicElementのサブクラスにはawakeが実行されない
- サンプルアプリケーションの”Components In Repetition”を参照のこと
  - ListPage - ChattyElementなる入れ子構造になっている

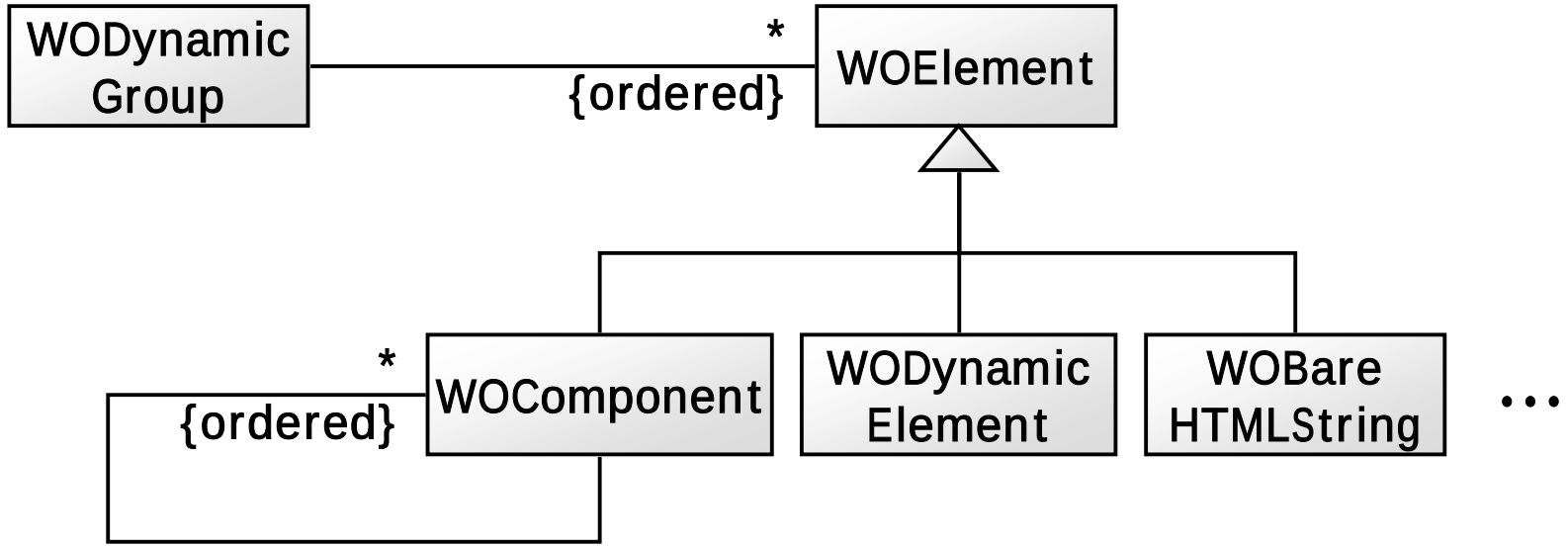
準備完了

P173. The Main Event

- **準備完了**
  - Application, Session, Context, Page**の生成/復元と初期化処理が完了した**
- **残された仕事**
  - **残りは**Response**の生成のみ**
  - WOApplication::createResponseInContext**で空の**WOResponse**が生成される**
    - **本メソッドをオーバライドして、カスタマイズ可能**
- **以後、残りの3つのフェーズを見てゆく**
  - takeValuesFromRequest
  - invokeAction
  - appendToResponse

コンポーネント・テンプレート

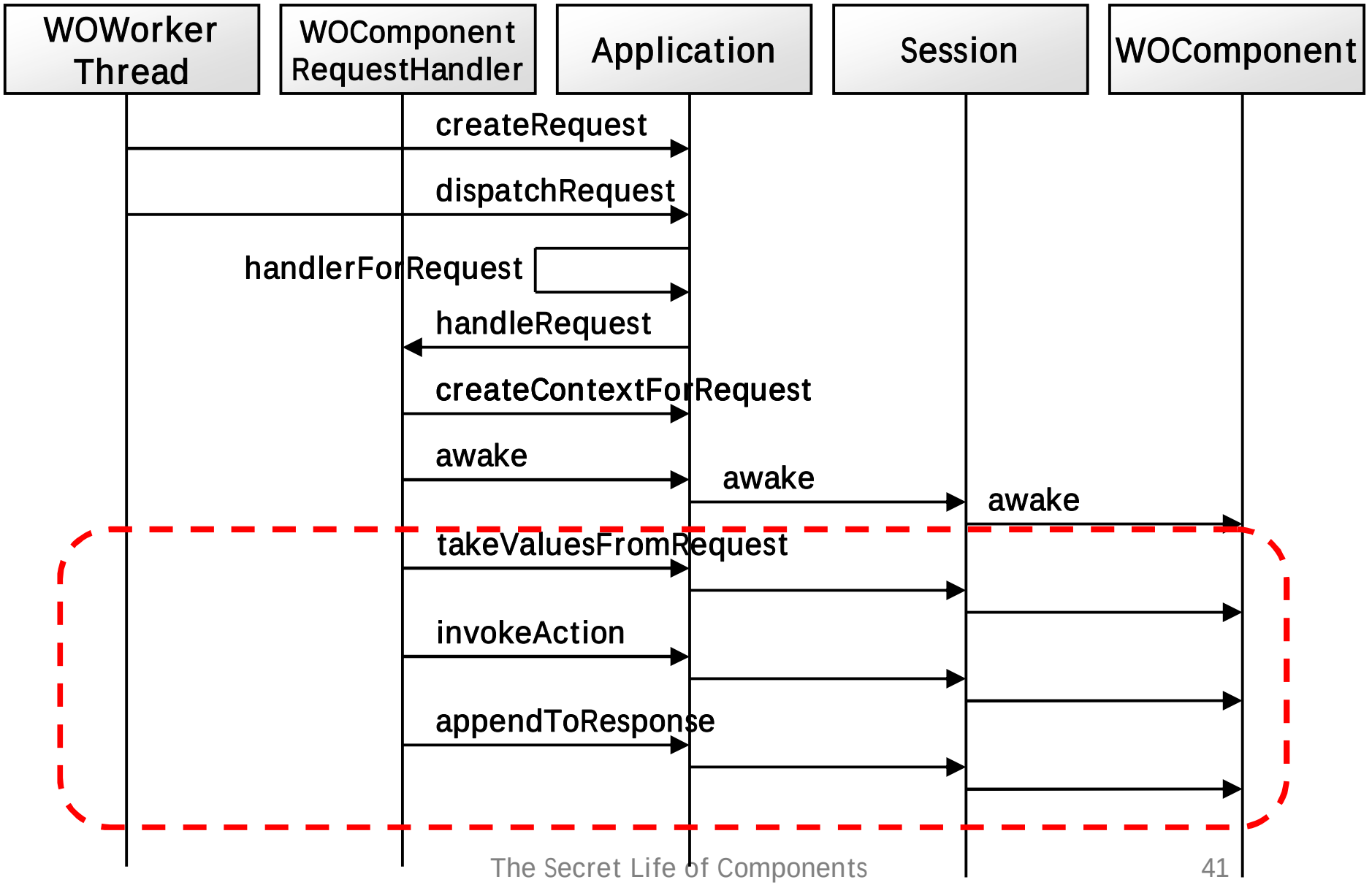
P173. Detour: Component Templates



- WOCompoent::template()を実行すると、テンプレートが取得できる
  - WODynamicGroup

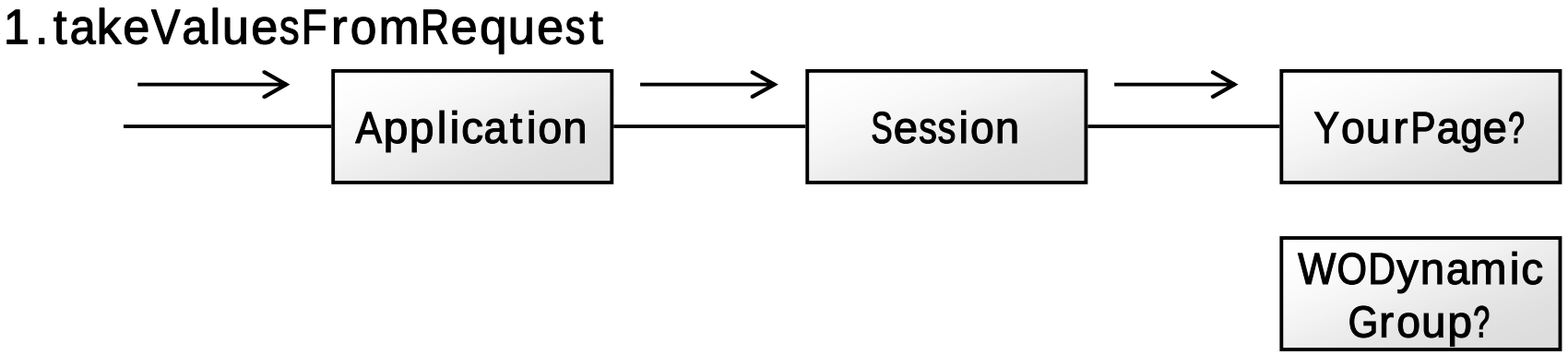
- 詳しくは次章・・・

Request-Responseループ



Take Values フェーズ

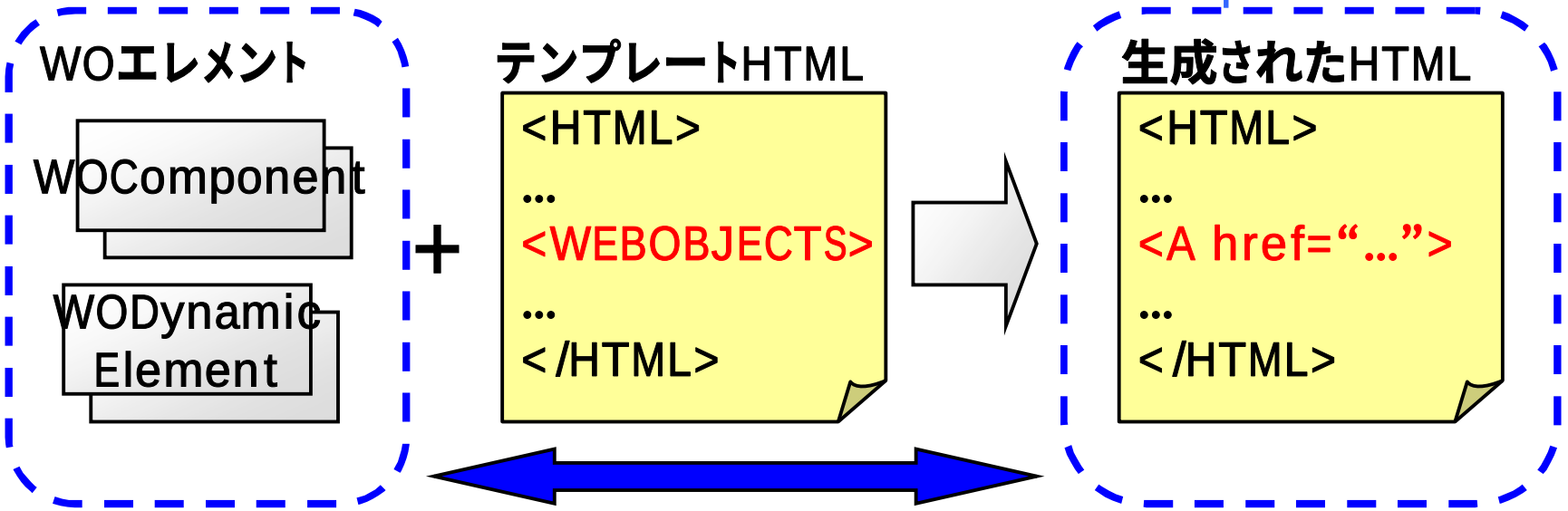
P173-. The Take Values Phase



- **本フェーズは実行されない場合あり**
  - **パラメータなしのWOHyperlinkがクリックされた場合など**
- **WODynamicGroupが子エレメントへメッセージをフォワードする？**
  - **よくわからない**

WOエレメントと生成されたHTMLの関係

P174-176. The Connection Between HTML Inputs and Elements



- WOエレメントと生成されたHTMLの関係を見てゆく
- 以下を参照せよ
  - サンプルアプリケーションの “Links and Simple Form” ページ
  - Listing 6-2 : “Links and Simple Form”ページのHTML

Listing 6-2

P174-176. The Connection Between HTML Inputs and Elements

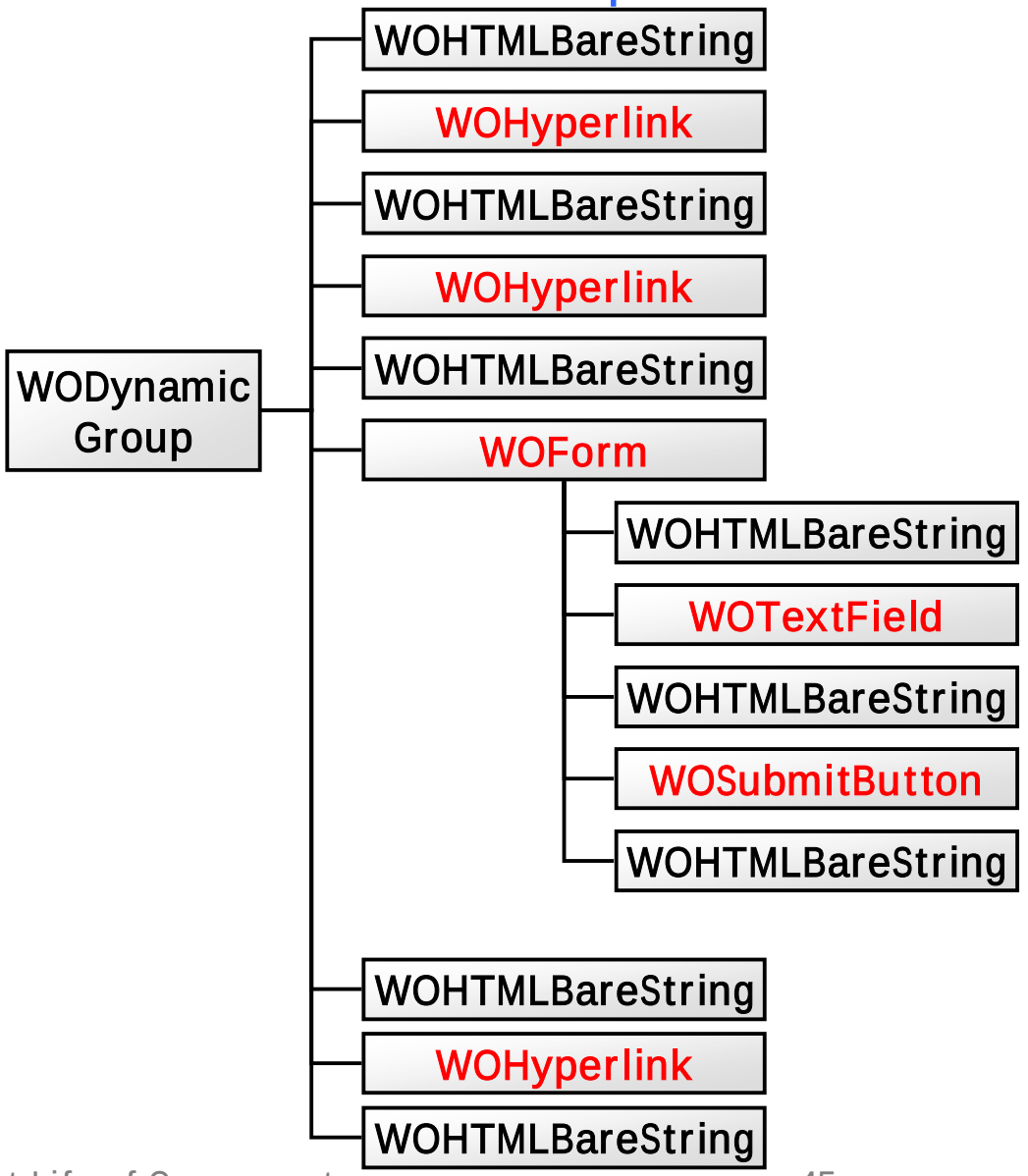
URL	http://.../wo/<sid>/0.1
0	<DOCTYPE
1	<a href="…/wa/RRLoop?wosid=…
2	
3	<a href="…/wo/<sid>/1.3"> …
4	
5	<form … action="…/wo/<sid>/1.5">
5.0	Text:
5.1	<input type=text name="5.1">
5.2	
5.3	<input type=submit … name="5.3"
5.4	
5	</form>
6	
7	<a href="…/wo/<sid>/1.7">…
8	<body></html>

- **9パートに分割**
- **2種類の**HTML
  - **静的(元のテンプレートからコピー)**
  - **動的生成**
- **動的生成**HTMLと Numbering
  - action names
    - "<contextID>.<part#>"
  - element names
    - "<part#>.<subpart#>"

HTMLとWOエレメントツリーの対応関係

P174-176. The Connection Between HTML Inputs and Elements

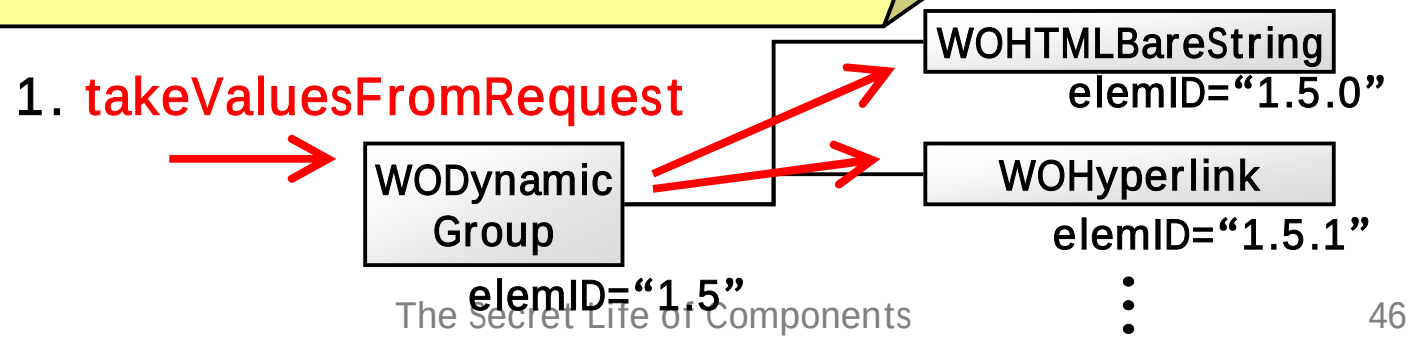
URL	http://.../wo/<sid>/0.1
0	<DOCTYPE
1	<a href="…/wa/RRLoop?wosid=…
2	
3	<a href="…/wo/<sid>/1.3"> …
4	
5	<form … action="…/wo/<sid>/1.5">
5.0	Text:
5.1	<input type=text name="5.1">
5.2	
5.3	<input type=submit … name="5.3"
5.4	
5	</form>
6	
7	<a href="…/wo/<sid>/1.7">…
8	<body></html>



WODynamicGroup::takeValuesFromRequest実装イメージ

P175-176. Listing6-3

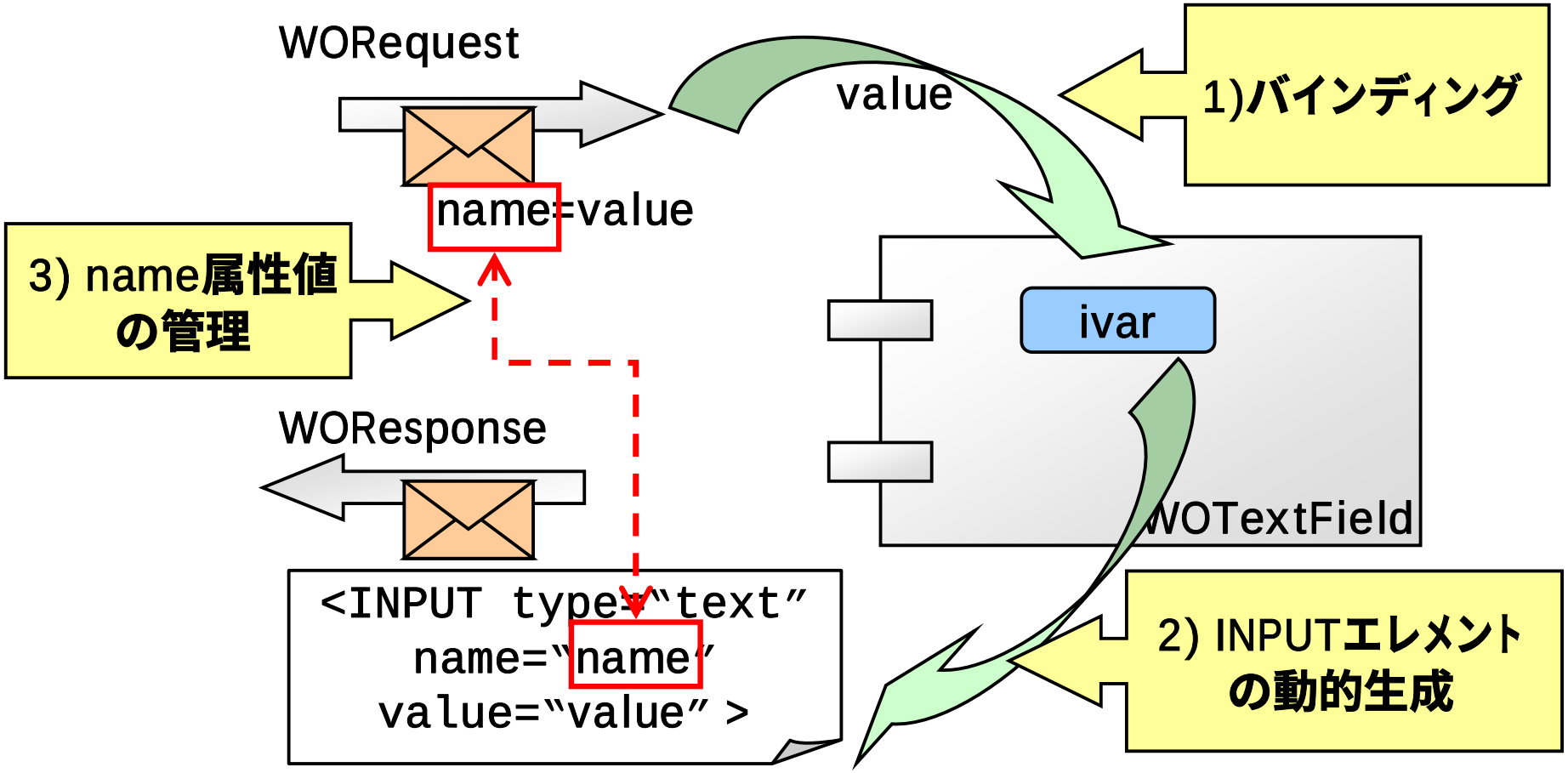
```
public void takeValuesFromRequest(
  WORequest aRequest, WOContext aContext) {

  aContext.appendZeroElementIDComponent();

  Enumerator childElems = childEnumerator();
  while(childElems.hasMoreElements()) {
    WOElement elem= childElems.nextElements();

    elem.takeValuesFromRequest(aRequest, aContext);

    aContext.incrementLastElementIDComponent();
  }
  aContext.deleteLastElementIDComponent();
}
```



WOTextFieldの代替品(TextInput)を作る

P176-177. Text Input Elements Component

- そもそもWOTextFieldの機能って？



TextInputの設計

P176-179. Text Input Elements Component

1. **バインディング**
   - WO組み込みのバインディング機構を利用せず、takeValuesFromRequestを自作し、この中でバインディング機能を実装する
     - synchronizesVariablesWithBindingをfalseに
2. INPUTエレメントの動的生成
   - HTMLリテラルとWOStringで動的に生成する
3. name属性値の管理
   - 通常のWOエレメントと同様の値割り当てスキーマに従う。
   - いわゆる”X.Y.Z”のドットつなぎのスキーマ

```
<INPUT type="text" name="  " value="  ">
```

※ 上記はイメージ。WOBではレンダリング不可。Orz...

TextInput::takeValuesFromRequestの実装

P176-179. Text Input Elements Component

```
public void takeValuesFromRequest(
  WORequest aRequest, WOContext aContext){

  if (aContext._wasFormSubmitted()){
     aContext.appendElementIDComponent("1");

     String formValue
        = aRequest.stringFormValueForKey(
            aContext.elementID());
     setValueForBinding(formValue, "value")

     aContext.deleteLastElementIDComponent();
  }
}
```

【"1"である理由】
Name**属性に対応する**WOString**が**0**から数えた**1**番目のエレメント**
であるため
P176**下の**HTML**断片を見よ。**

invokeActionフェーズ

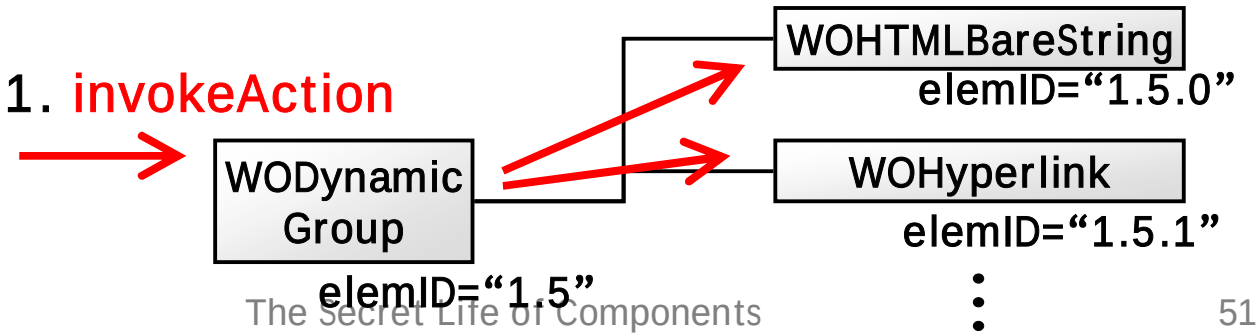
P179. Perfoming the Request Action

- invokeActionフェーズは必ずしも実行されない
  - 最初のページ(Main.wo)ロード時は実行されない
- 実行時の内部処理はtakeValuesFromRequestフェーズに似ている

WODynamicGroup::invokeAction実装イメージ

P179. Listing6-6.

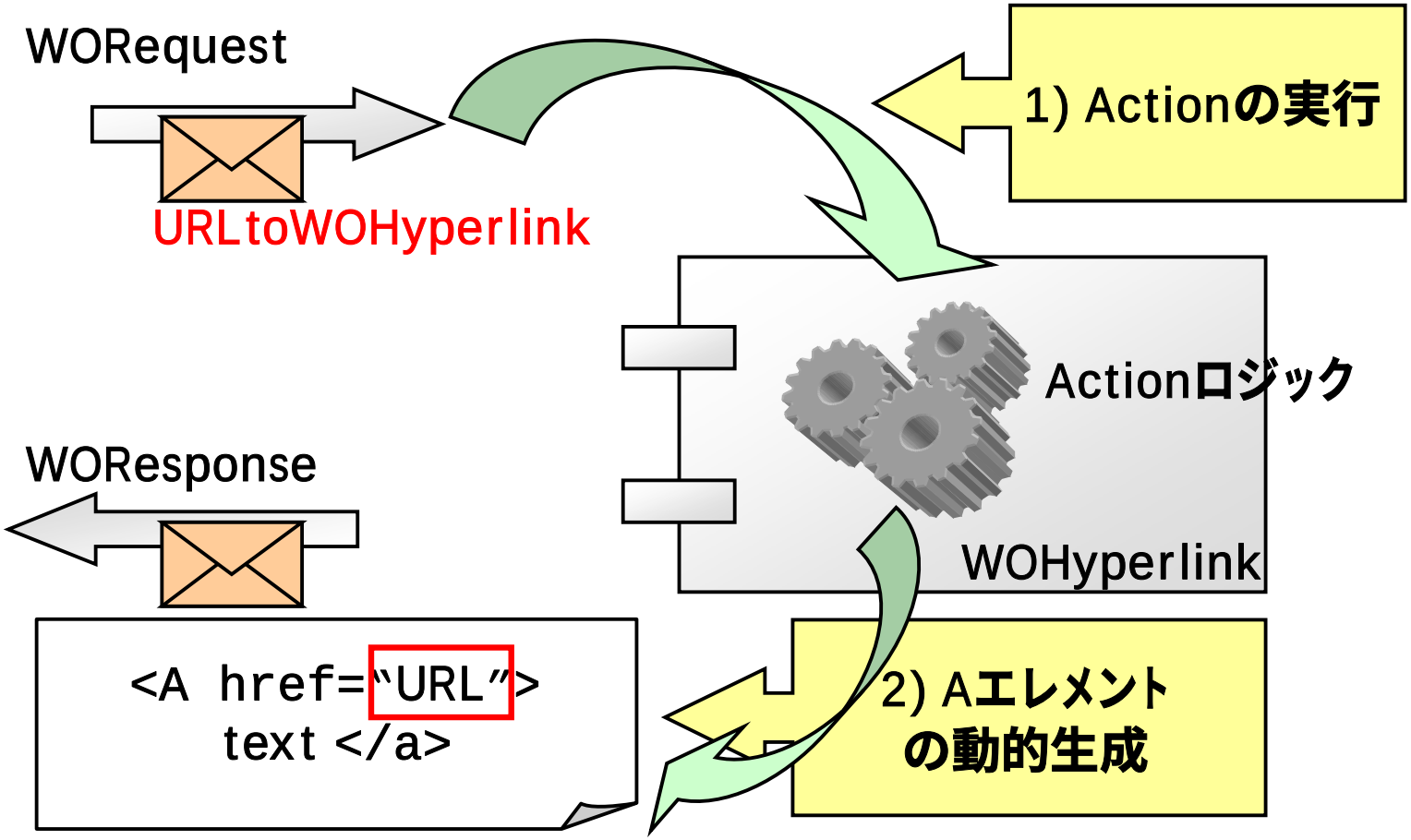
```
public void invokeAction(
  WORequest aRequest, WOContext aContext) {

  aContext.appendZeroElementIDComponent();

  Enumerator childElems = childEnumerator();
  WOActionResults resultPage = null;
  while(childElems.hasMoreElements() && resultPage==null) {
    WOElement elem= childElems.nextElements();

    resultPage = elem.invokeAction(aRequest, aContext);

    aContext.incrementLastElementIDComponent();
  }
  aContext.deleteLastElementIDComponent();
}
```



WOHyperlinkの代替品(Hyperlink)を実装する

P179-181. Hyperlink Example Component

- **そもそもWOHyperlinkの機能って？**



Hyperlinkの設計

P179-181. Hyperlink Example Component

1. Actionの実行
   - WORequestが自エレメントに対するものである場合、Actionを実行する
     - senderIDと自分のelementIDが等しい場合
2. Aエレメントの動的生成
   - テンプレートを使用せず、Javaロジックのみで HTMLを生成する
     - Hyperlink.wo (Hyperlink.html, Hyperlink.wod)は不要

Hyperlink::invokeActionの実装

P181. Listing6-7. Implementation of the Hyperlink Component

```
public WOActionResults invokeAction(
  WORequest aRequest, WOContext aContext){

  if (aContext.elementID().equals(aContext.senderID())){
     return (WOActoinResults)valueForBinding("action");
  }
  return null;
}
```

Hyperlink::appendToResponseの実装

P181. Listing6-7. Implementation of the Hyperlink Component

```
public void appendToResponse(
  WORequest aResponse, WOContext aContext){

  aResponse.appendContentString("<a href=¥"");
  aResponse.appendContentString(
     aContext.componentActionURL());
  aResponse.appendContentString("¥">");
  aResponse.appendContentString(
     (String)valudForBinding("linkText"));
  aResponse.appendContentString("</a>");
}
```

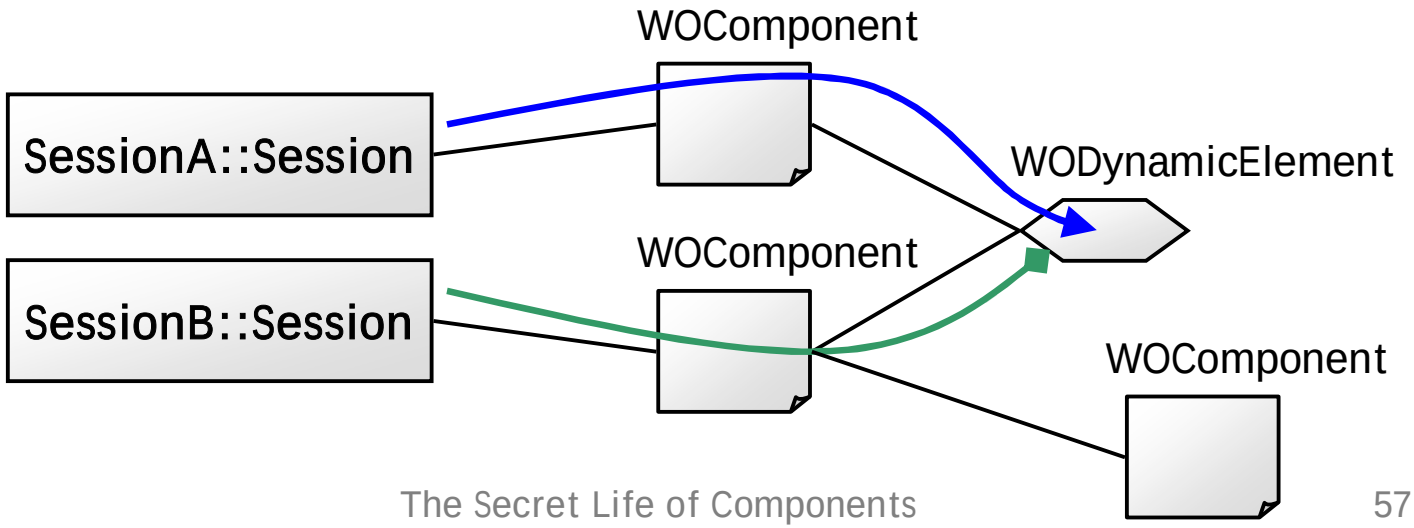
appendToResponseフェーズ

P181. Generating the Response

- Component Requestの場合、必ず実行される
- やるべきこと
  - HTMLやDHTMLやXMLや・・・を生成する
  - 上記コンテンツの**正しい場所に**エレメントIDを埋め込む
- 注意点
  - context.pageが、生成対象のページを返す

WODynamicElementとWOComponentのサブクラス

	WODynamicElementのサブクラス	WOComponentのサブクラス
インスタンスの共有	される	されない
スレッドセーフ性	必要	不要



WOTextFieldの代替品その2 (TextInput2)

P182-183. Reprise of the Text Input Example Component

1. バインディング
   - WO組み込みのバインディング機構を利用せず、takeValuesFromRequestを自作し、この中でバインディング機能を実装する
     - synchronizesVariablesWithBindingをfalseに　TextInputと同じ
2. INPUTエレメントの動的生成
   - テンプレートを使用せず、Javaロジックのみでを生成する。
     - TextInput2.wo (TextInput2.html, TextInput2.wod)は不要
3. name属性値の管理
   - 通常のWOエレメントと同様の値割り当てスキーマに従う。
   - いわゆる”X.Y.Z”のドットつなぎのスキーマ　TextInputと同じ
     - ただし、WOFのテンプレート機構を使用しないため、付与される属性値は異なる

name属性値の違い

```
<form method="post" action="/<pathToWoa>/wo/<sid>/2.1">
        Text Input:
        <input type=text name="1.1.1" value="">          TextInput
        <br />
        <p><input type=submit value="Submit" name="1.3"></p>
</form>Value submitted:
<p> </p>
<hr>
<form method="post" action="/<pathToWoa>/wo/<sid>/2.5">
        Text Input2:
        <input type="text" name="5.1" value="">          TextInput2
        <br />
        <p><input type=submit value="Submit" name="5.3"></p>
</form>Value submitted:
```

- **上記HTMLは一部省略箇所あり**

appendToResponseフェーズより後

P183. The End of the World As WO Knows It

- PageをSessionのページキャッシュに保存
  - Session::savePage()
  - ページを保存不要な場合は、オーバライドして保存しないようにしてもよい
    - ComponentActionへのリンクを含まないDirectActionページなど
- SessionをsessionStoreへ保存
  - Application::saveSessionForContext()
- sleepフェーズ

ハンドラ終了～。

P183-184. ... And Beyond

- DispatchRequestメソッドが終了したらApplicationDidDispatchRequestNotificationをブロードキャストする
- ワーカースレッドにレスポンスを返す

同一のリクエストに何を返すか？

P184-185. Complications

- 同一のリクエストが投げられるケース
  - すでにダウンロード済みのコンテンツの2番目のコピーをブラウザ・プラグインが要求した
  - ウィンドウをリサイズしたとき、ブラウザがリフレッシュリクエストを投げる
  - ユーザがリフレッシュ/リロード ボタンを押した
  - ユーザがリンクまたはサブミットボタンをダブルクリックした
  - ユーザが戻るボタンを押した

同一のリクエストが投げられた場合、どうするか？

P184-185. Complications

- 基本的に、以下の2つの選択肢が存在する
  - 以前表示したのと同じページ（コンテンツ）を返す
  - 全く別のもの（エラーページ、ログインページなど）を返す
- WebObjectsは両方をサポートする

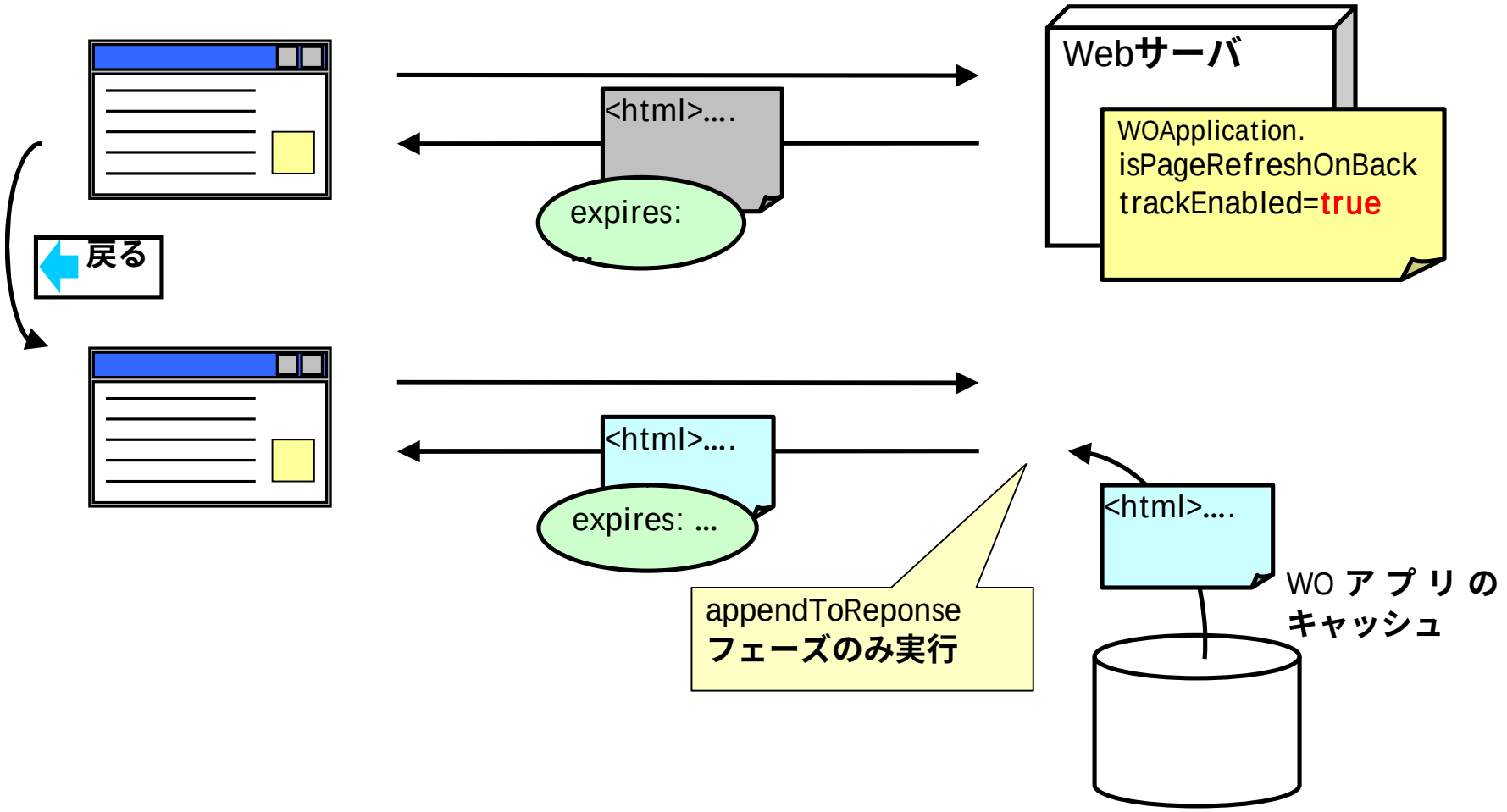
WebObjectsのデフォルト処理

P184-185. Complications

- 最後に返したコンポーネントから、レスポンスを再生成する
  - ステートフルなアプリケーションのような振る舞い
    - うーん、イマイチわかりにくい表現だ。
- デフォルト処理はカスタマイズ可能
  - WOApplication::setPageRefreshOnBackTranckEnabledメソッドを利用する

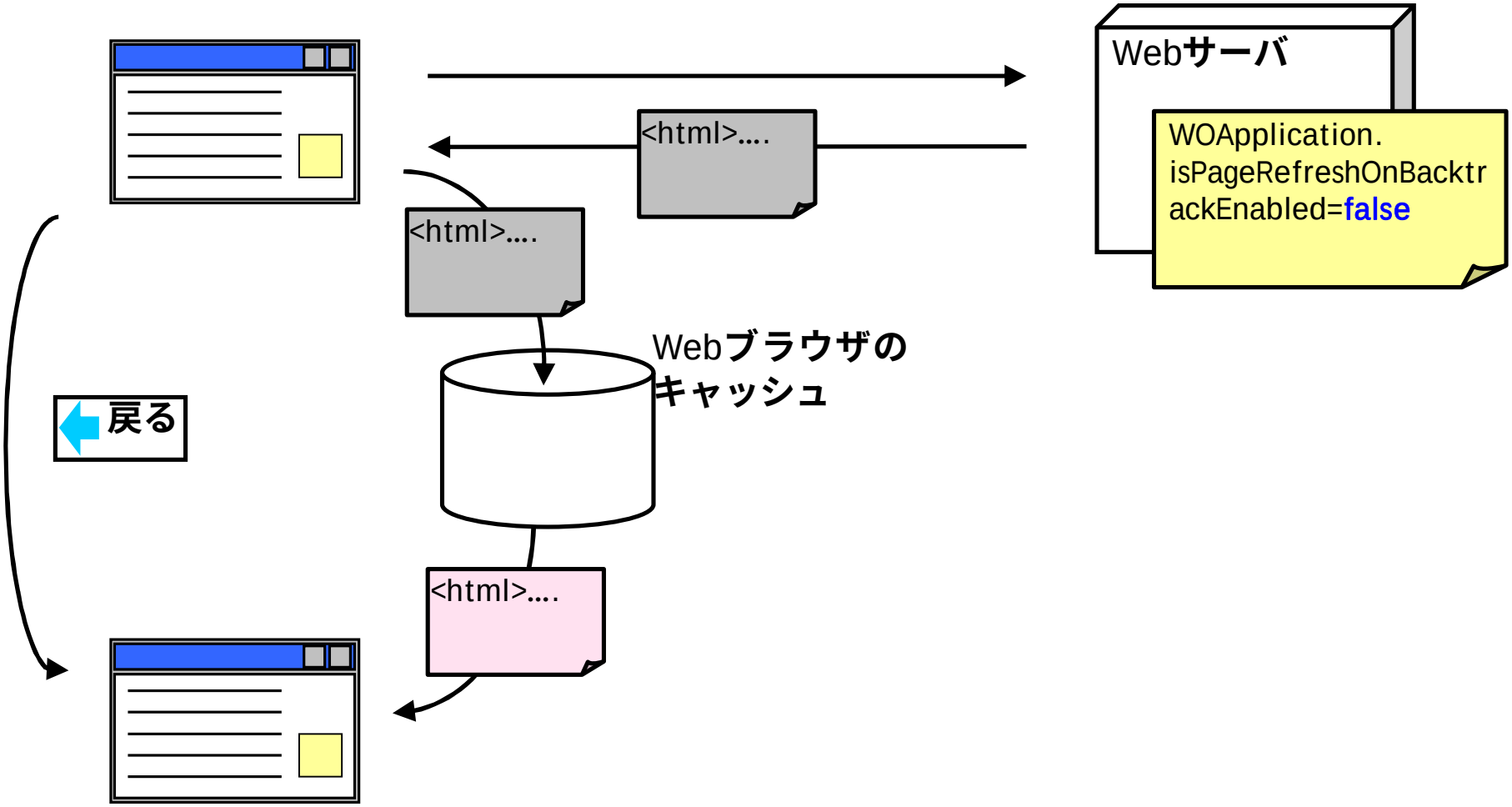
isPageRefreshOnBacktrackEnabled=true

P184-185. Complications



isPageRefreshOnBacktrackEnabled=false

P184-185. Complications



第3の方法

P184-185. Complications

- **基本的に**
  setPageRefreshOnBackTrackEnabled=false
  を使用し、特定のレスポンスで
  disableClientCaching**を実行する**

結局・・・

P184-185. Complications

- **「これでオールOK！」という方法は無い**
- **デフォルト処理**
  (isPageRefreshOnBackTrackEnebled=true)
  が最もトラブルがすくないようだ

以上！

- **来月は・・・**
  - Chapter 7. Components and Elements by WR