Practical WebObjects
Chapter 7 (Page 187-220):

Components and Elements

WR

WR at Csus4.net

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

目次詳細

- The Case of the Missing Element
  - Request-Response and Component Debugging
  - The Golden Rule
  - The Mystery of the Missing Submit
- Dynamic Elements
  - Dynamic Elements, Components, and Associations
  - How Dynamic Elements Work
  - An Example Dynamic Element
    - Constructing and Validating a Dynamic Element
    - Generating HTML from a Dynamic Element
    - Handling Actions in a Dynamic Element
  - Replacing the WebObjects Dynamic Elements
- Mysteries of Binding Synchronization
  - Timing
  - A Binding Synchronization Example
  - Binding Instance Variables and Methods
  - Debugging Binding Synchronization
  - Manual Synchronization
  - The Carat Notation
- Stateless Components
  - The Mapped List Popup Stateless Component
- Debugging and Optimization with WOEvent
  - Preparing for WOEvent Logging
  - Displaying the Logged Events
  - Some Practice Reading the Event Log
  - Creating Custom Events
    - Creating a Subclass of WOEvent
    - Describing the New Event
    - Registering the Event Class
    - Implementing the Rest of the Event
    - Instrumenting Code with Events
- Summary

本章の概要

- バインディング設定のデバッグについて
- WODynamicElementとWOComponent
  - 比較・優位点
- 入れ子Componentにおける同期
  - Automatic Synchronizationの仕組み
  - Manual Synchronizationの利点、実現方法
- WODynamicElementの代用品としてのStatelessコンポーネント
- WOEventの代替品の作成

目次

- The Case of the Missing Element
  - Request-Response and Component Debugging
  - The Golden Rule
  - The Mystery of the Missing Submit
- Dynamic Elements
  - Dynamic Elements, Components, and Associations
  - How Dynamic Elements Work
  - An Example Dynamic Element
  - Replacing the WebObjects Dynamic Elements
- Mysteries of Binding Synchronization
  - Timing
  - A Binding Synchronization Example
  - Binding Instance Variables and Methods
  - Debugging Binding Synchronization
  - Manual Synchronization
  - The Carat Notation
- Stateless Components
  - The Mapped List Popup Stateless Component
- Debugging and Optimization with WOEvent
  - Preparing for WOEvent Logging
  - Displaying the Logged Events
  - Some Practice Reading the Event Log
  - Creating Custom Events

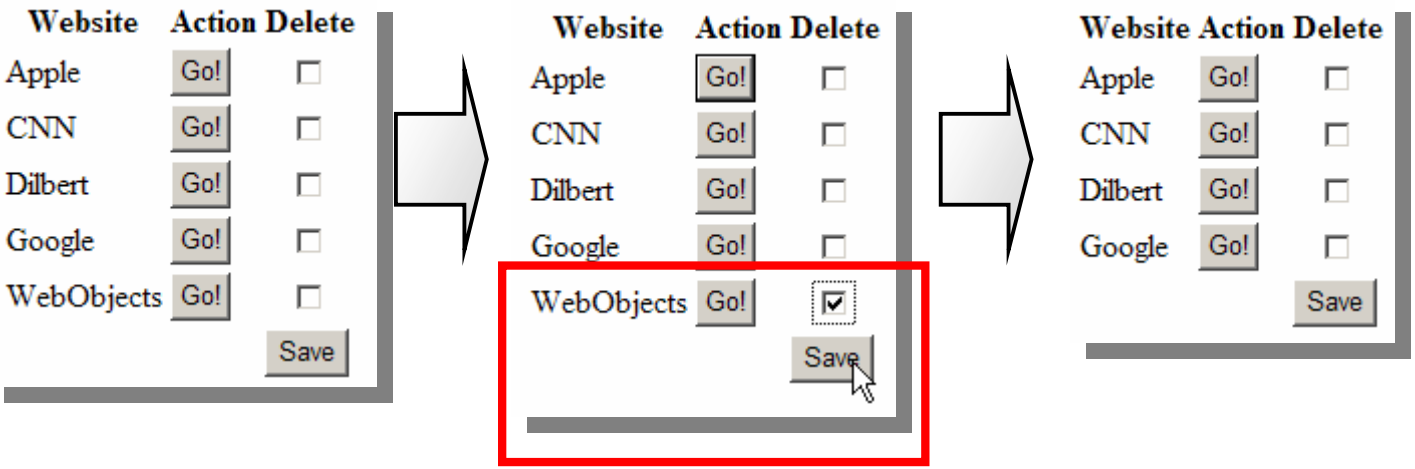
本セクションの内容

- バインディング絡みのありがちな失敗事例と、そのデバッグ方法

- 事例1：ChangedCollection.wo
  - 繰り返し表示に用いるコレクションを、不適切なタイミングで修正してしまった事例
- 事例2: BadConditinal.wo
  - チェックボックスにバインドしたBooleanをビュー制御に使用してハマった事例

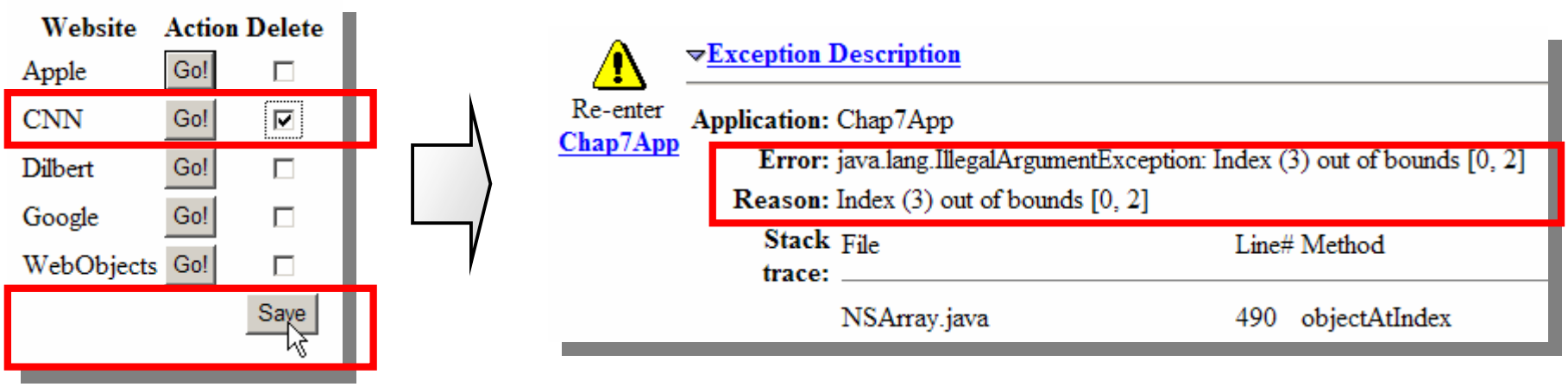
事例1: ChangedCollection.wo

- **機能**
  - Actionをクリックすると、対応するWebサイトに飛ぶ
  - DeleteにチェックしてSaveすると、エントリが削除される



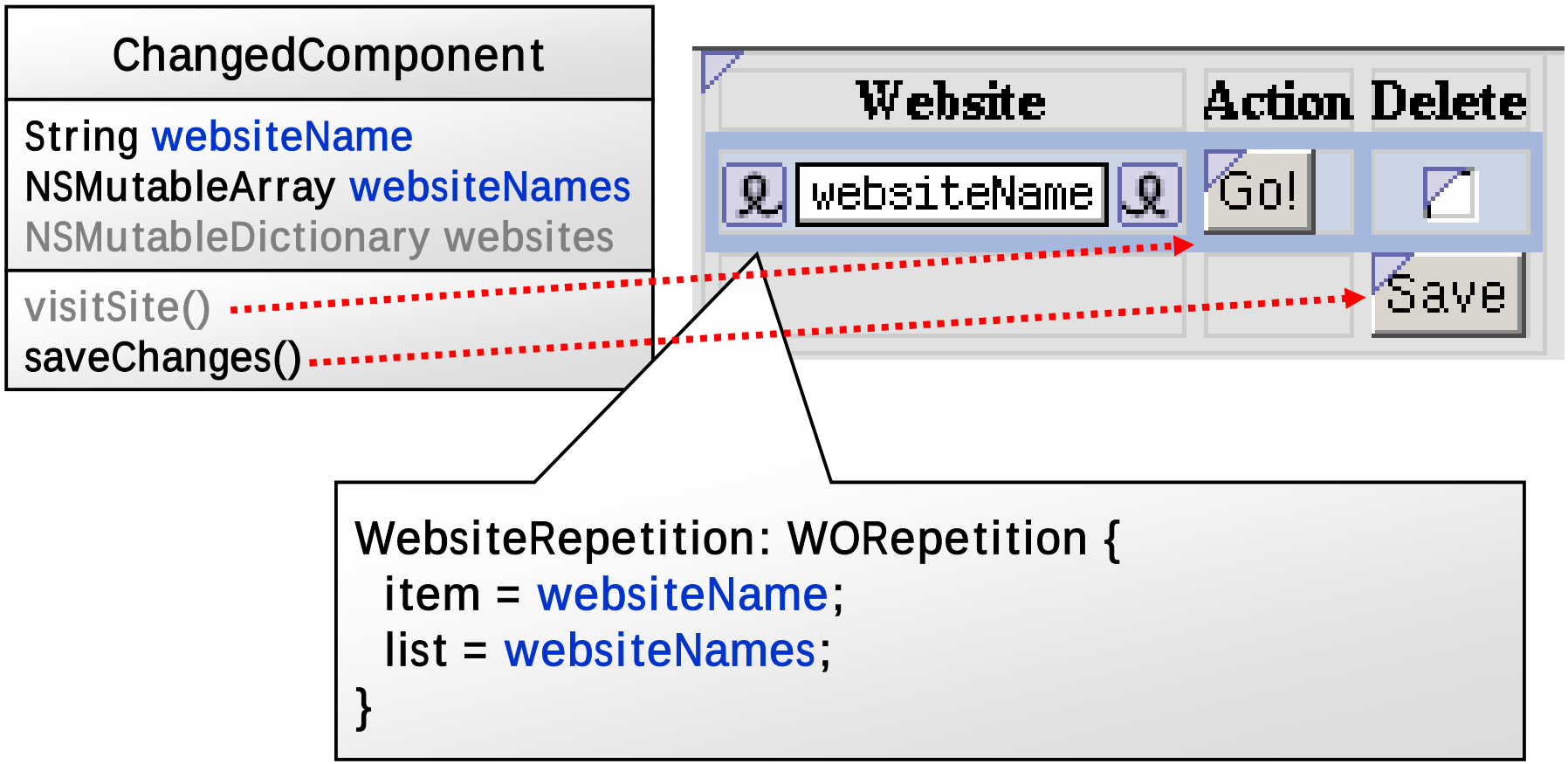
ChangedColletion.woのバグ

- 2番目の”CNN”のDeleteにチェックして、クリックすると・・・



- あらら・・・・バグじゃん・・・。
  - 削除機能は最後のエントリ (“WebObjects”) のみを選択した場合しか、正しく動作しない・・・

ChangedColletion.woの構成



バグ解析

P188-190．Request-Response and Component Debugging

- エラー発生条件のおさらい
  - ”Delete”チェックボックスにチェックしてsubmit→エラー
  - 最後の”Delete”チェックボックスにチェックした時はエラーとならない
- スタックトレース (Listing 7-1.)からわかること
  - エラーはtakeValuesFromRequestフェーズで発生している
    - まだ、invokeActionフェーズには到達していない
  - ということは、HTTPリクエストから値を取得(take values from Request!)しているコンポーネントが怪しい
  - ChangedCollection.woでは、WOCheckBox (checkedアトリビュートにキーisSiteDeletedがバインドされている)のみ！

isSiteDeletedのアクセッサを見る

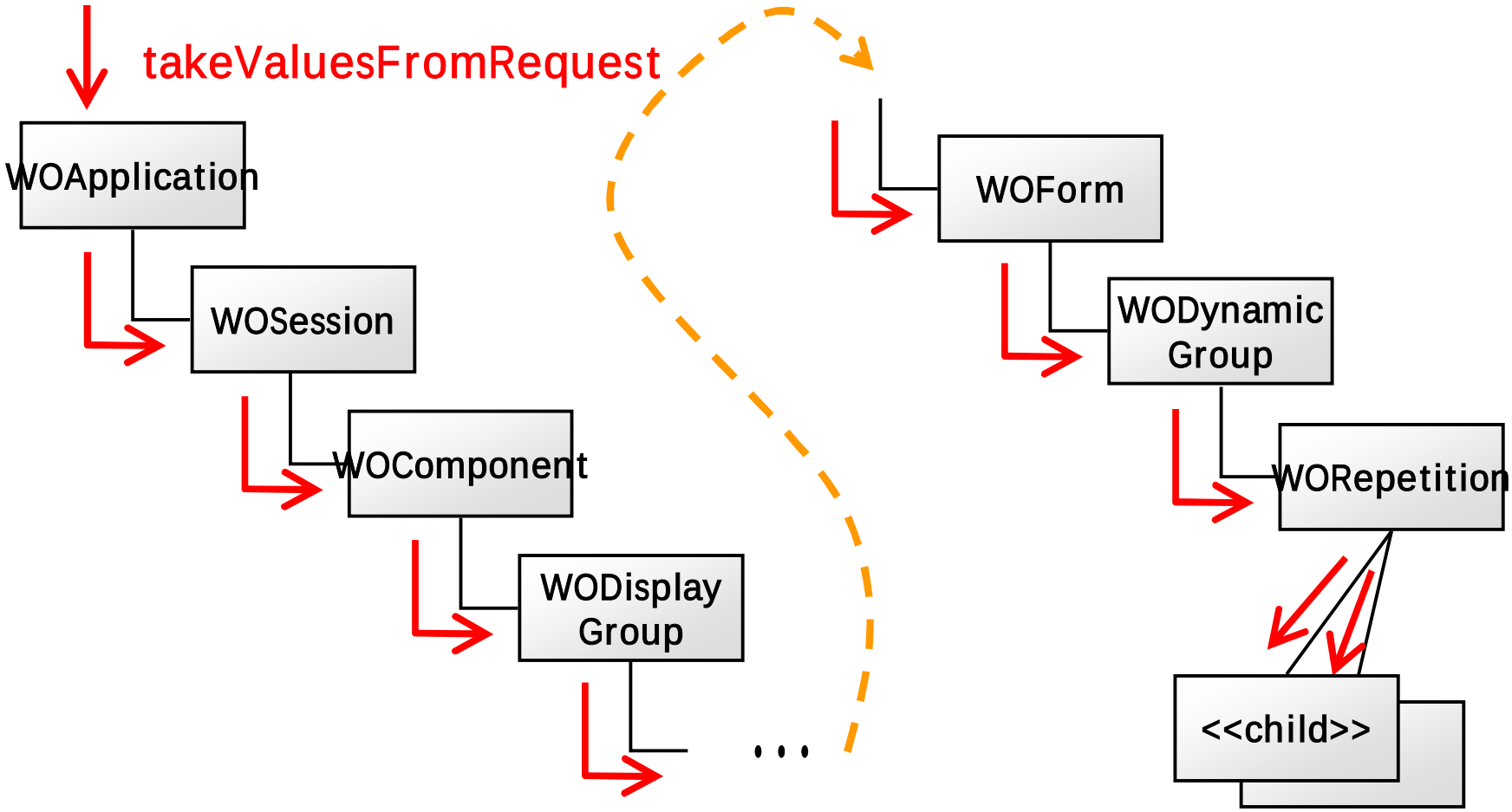
P188-190．Request-Response and Component Debugging

- <<getter>> isSiteDeleted()
  - falseを返すだけ
  - 明らかに問題なし
- <<setter>> setIsSiteDeleted()
  - 引数isSiteDeleIted==trueのときのみ、処理が実行される

```
public void setIsSiteDeleted(boolean isSiteDeleted){
        if (isSiteDeleted){
            websites.removeObjectForKey(websiteName);
            websiteNames.removeObject(websiteName);;
        }
}
```

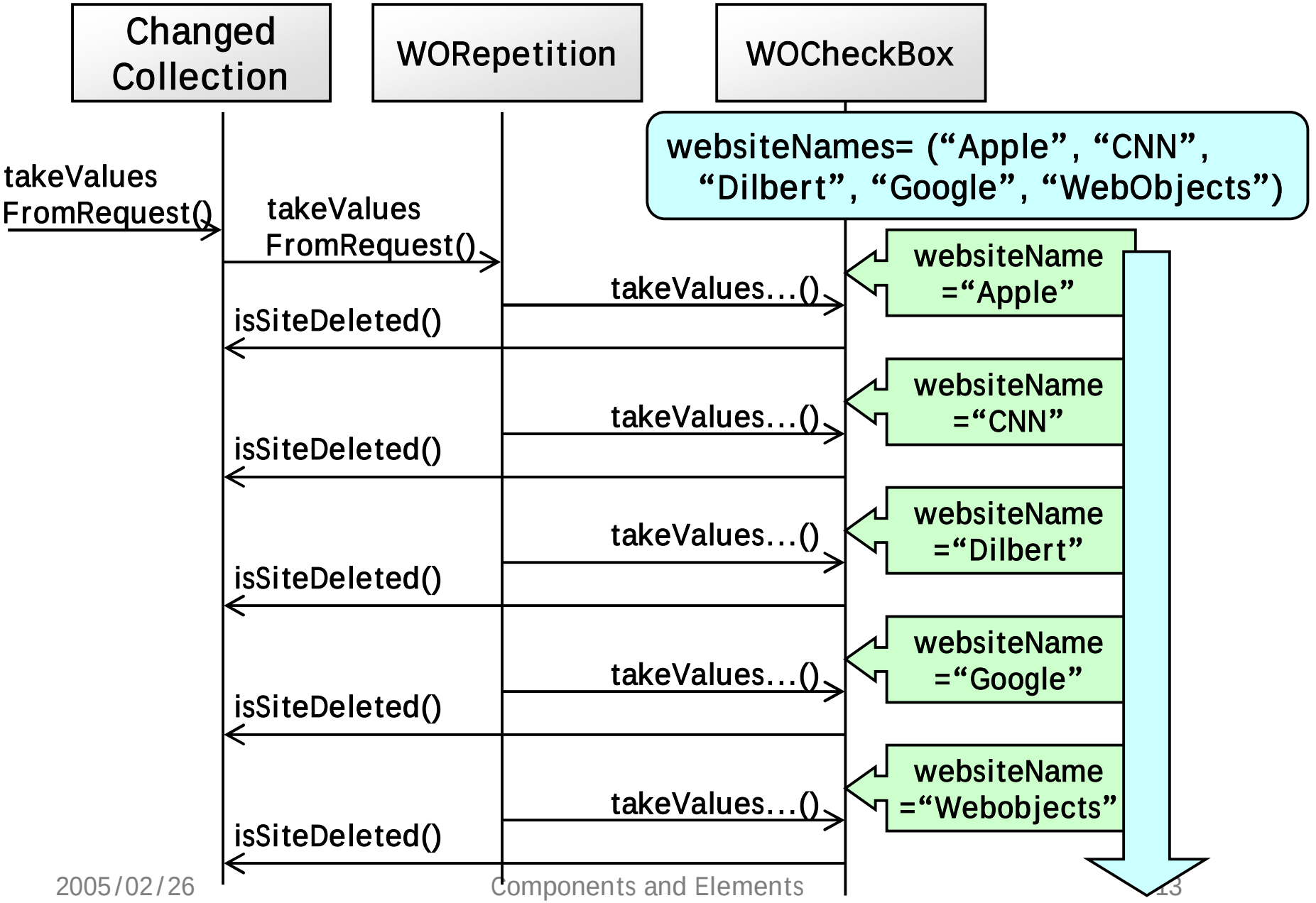
コレが問題

takeValuesFromRequestのドリルダウン

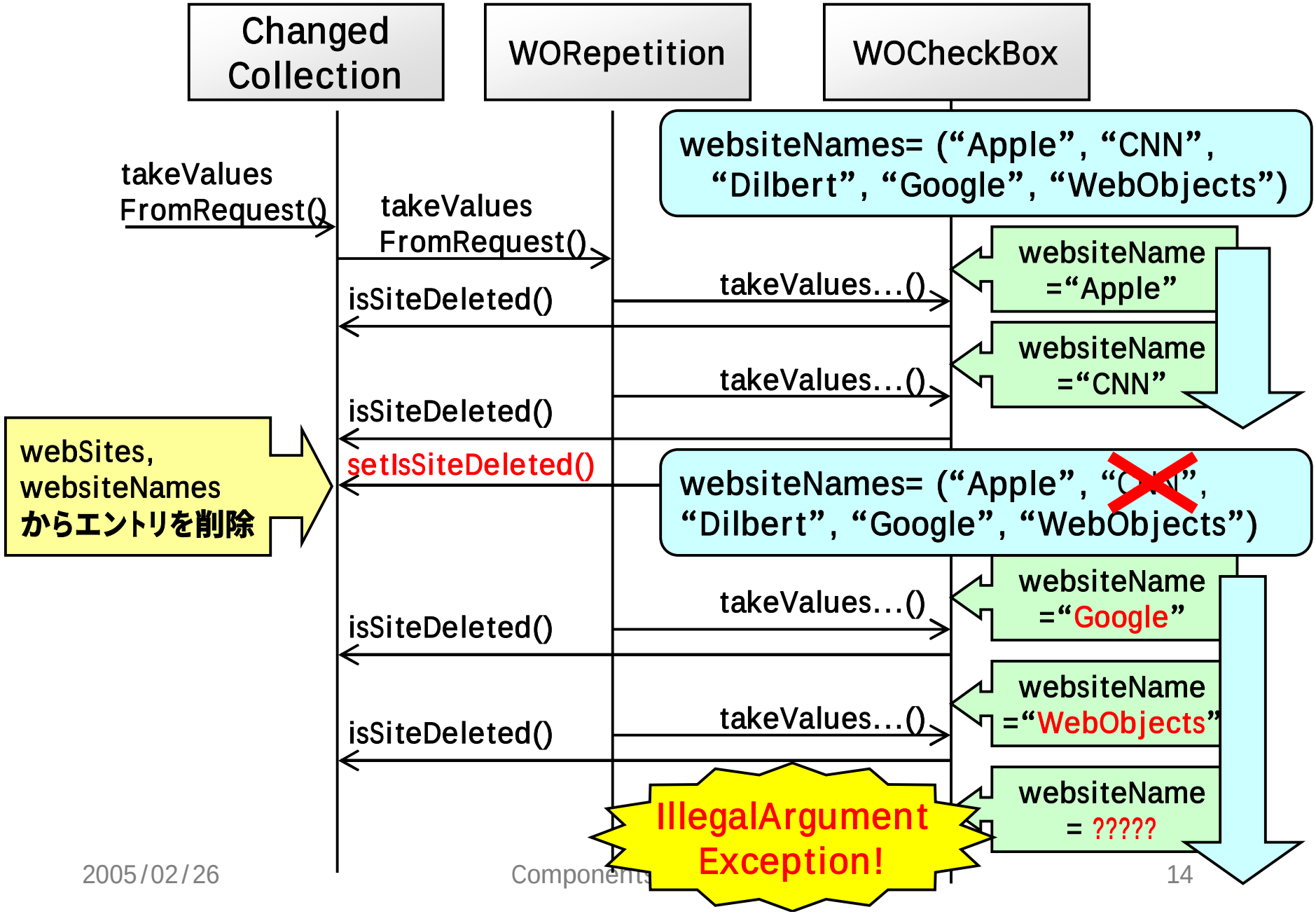


- 上記のコラボレーション図は概念的なレベルであり、かつ、メッセージを省略している。
- 実際のメッセージのやり取り (の一部) については、Listing 7-1.のスタックトレースを参照のこと。

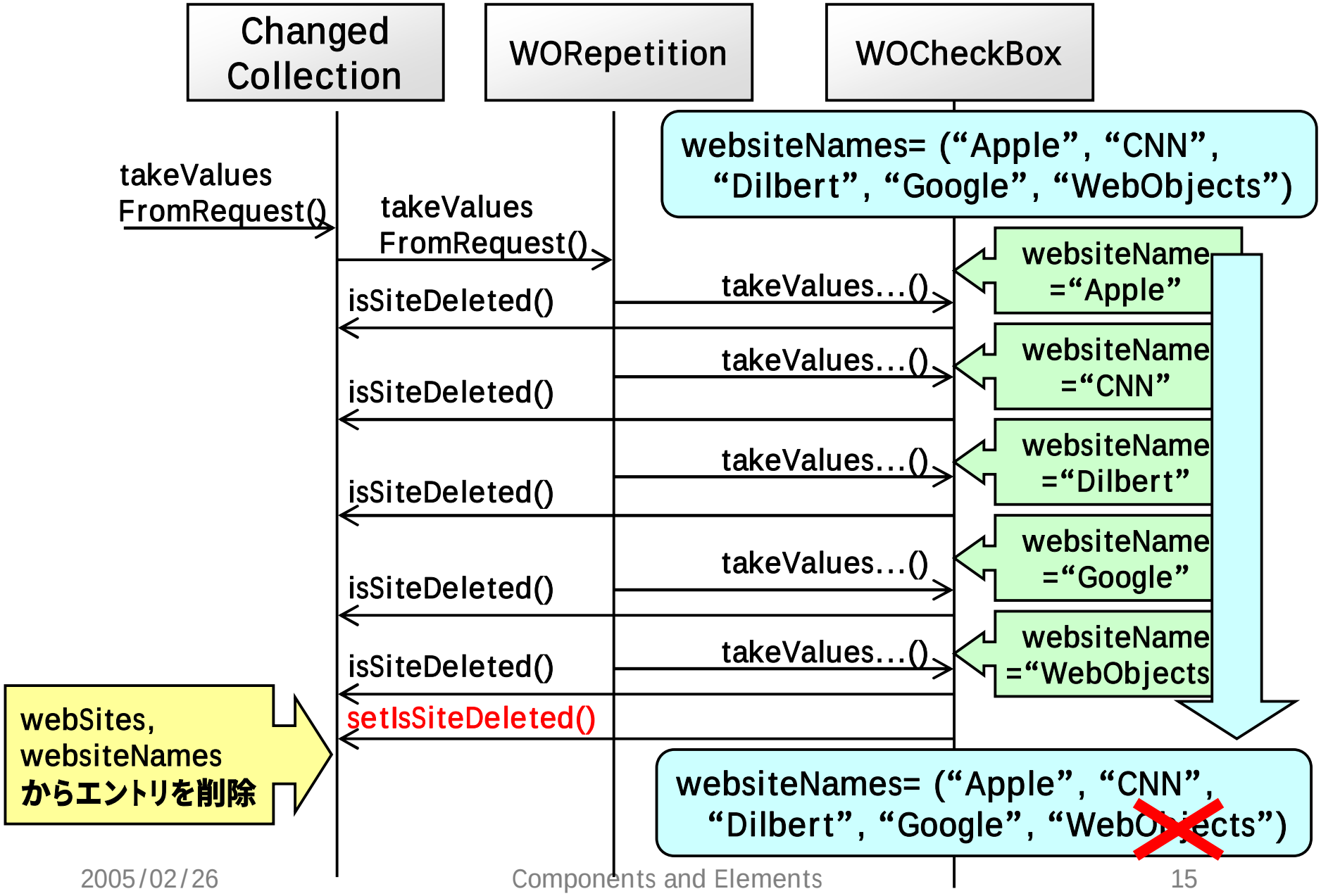
Deleteにチェックしないで Save



“CNN”のDeleteにチェックしてSave



“WebObjects”のDeleteにチェックしてSave



黄金律

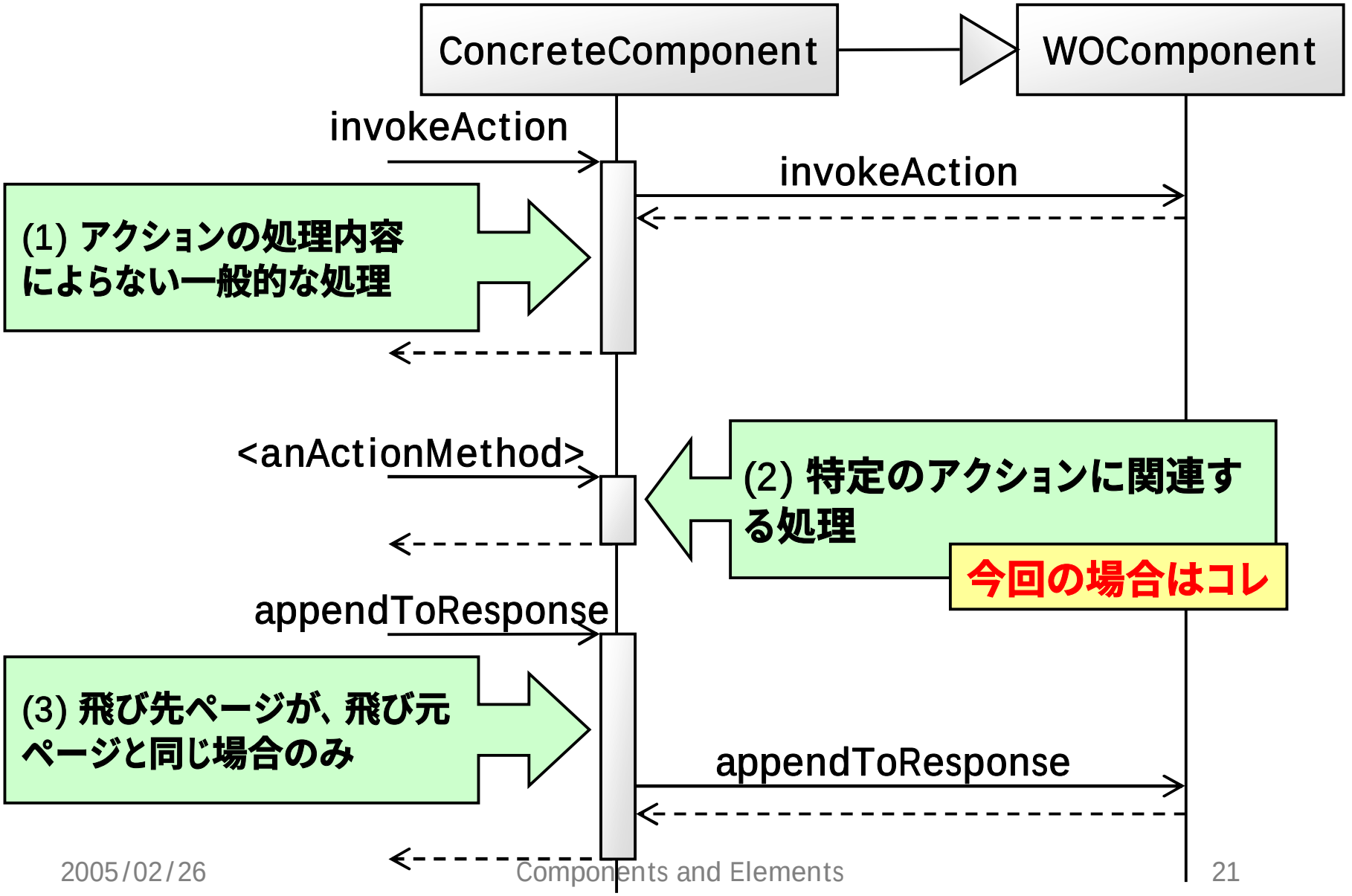
P190-191. The Golden Rule

- *Don't Modify anything that the component or dynamic elements depends on while in the sequence appendToResponse, takeValuesFromRequest, and invokeAction*
  - appendToResponse, takeValuesFromRequest, invokeAction **フェーズのシーケンス(???)において、コンポーネントとダイナミックレメントが依存している全てのものを修正してはいけない**

- **では、**ChangedCollection.wo**の場合はどうか**?
  - 「takeValuesFromRequest**フェーズにおいて、**WORepetition**ダイナミックエレメントが依存している**websites, websiteNames**から要素を削除している」ため、黄金律に違反している！**

コレクション編集処理の実装候補

P190-191. The Golden Rule



今回の解決方法

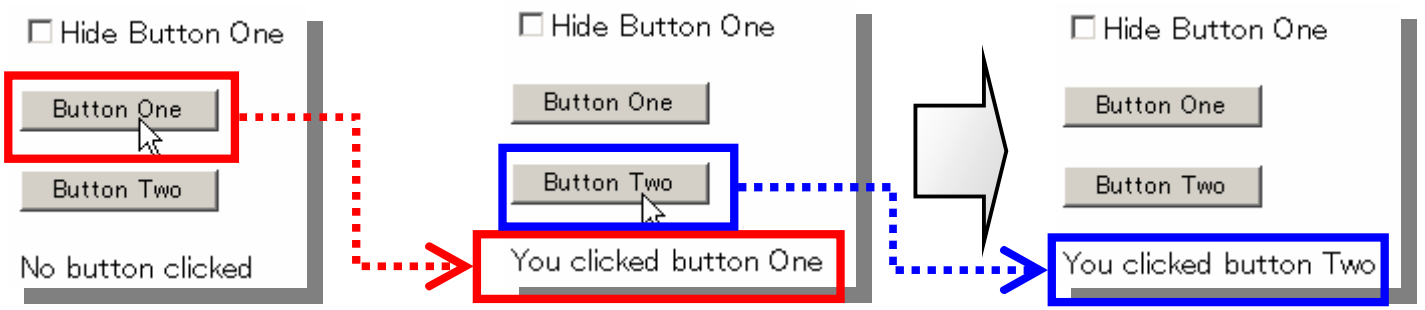
P190-191. The Golden Rule

- アクションメソッドsaveChanges()内でwebsites, websiteNamesから要素を削除する
  - 詳細は
    - P190-191のソースコード
    - サンプルコードのGoodChangedCollection.wo
  - を参照

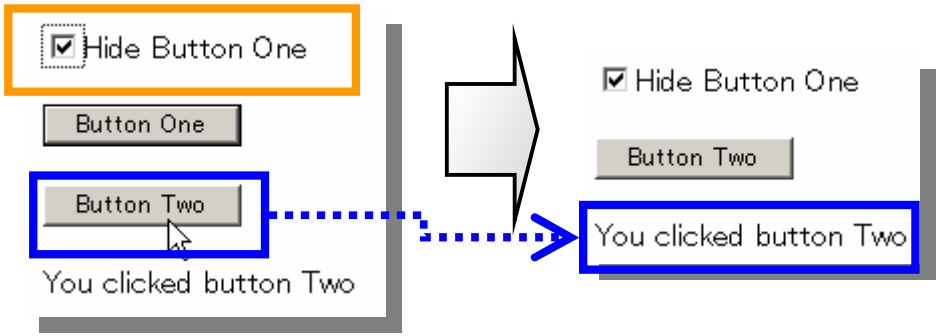
- ちなみに、サンプルコードは http://www.apress.com/book/bookDisplay.html?bID=248からダウンロード可能

事例2: BadConditional.wo

- 機能
  - クリックしたボタンが下部に表示される

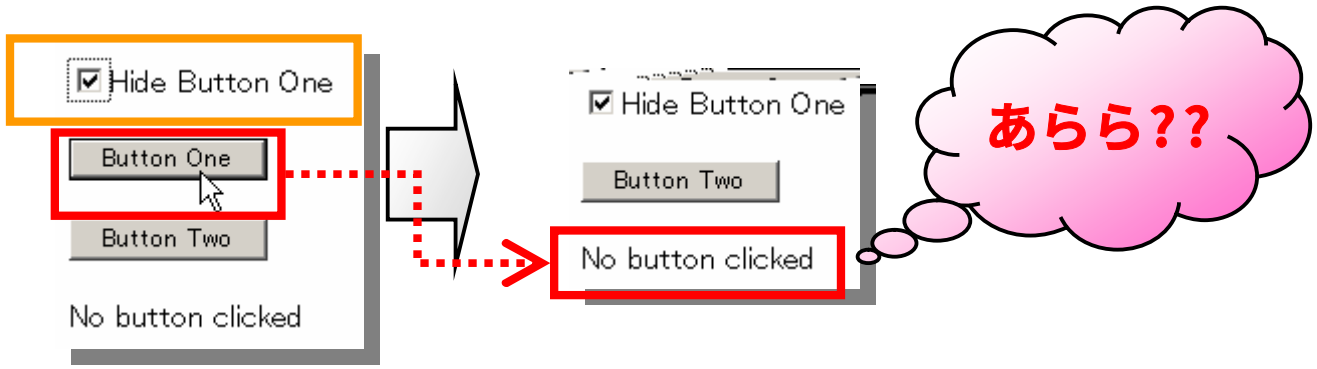


  - “Hide..”にチェックすると、ボタン1が消える



BadConditional.woのバグ

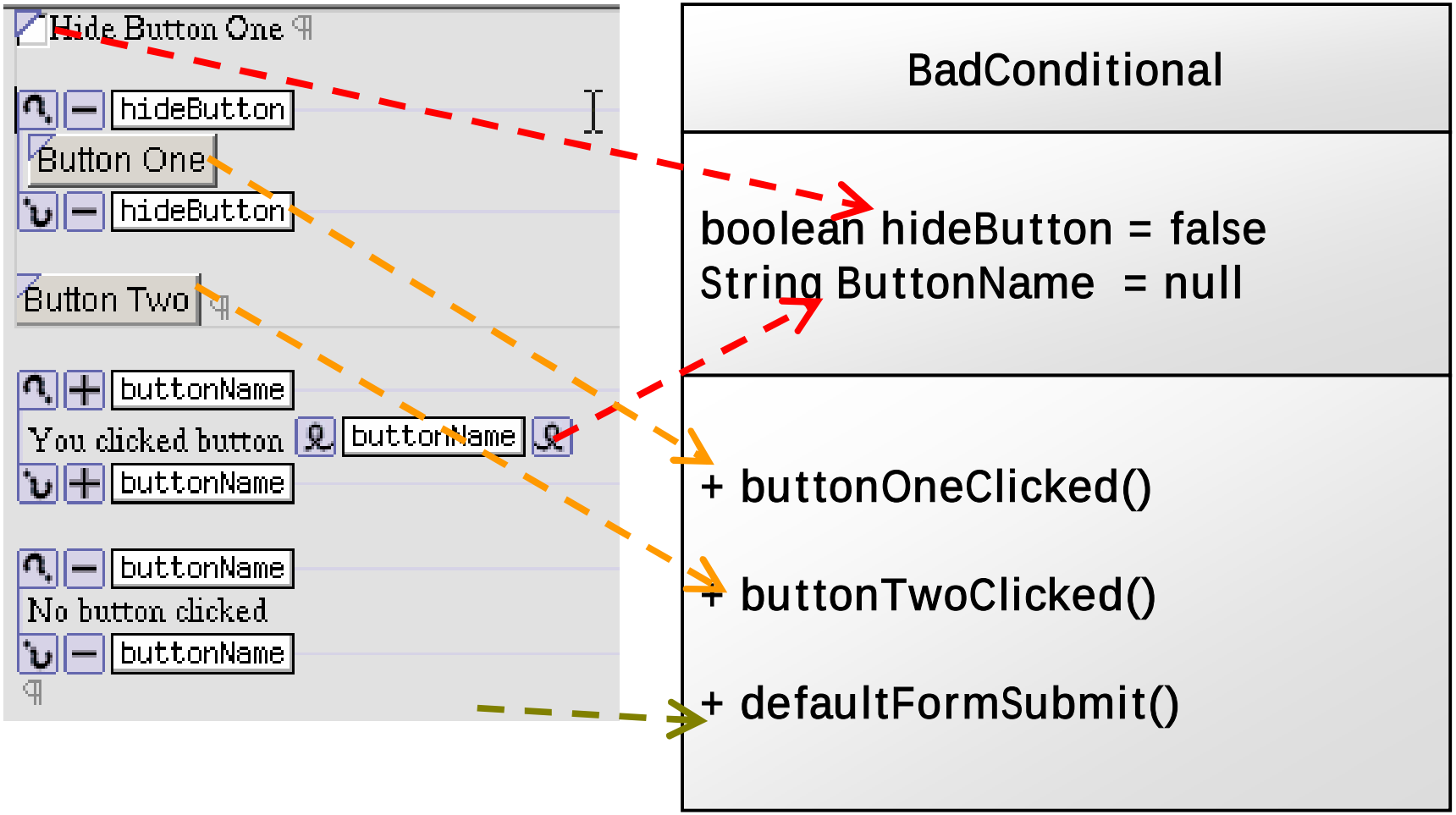
- “Hide…”にチェックしてボタン1をクリックすると・・・



- あらら・・・バグじゃん・・・。
  - 本当は、“You clicked button One”と表示して欲しいところ

BadConditional.woの構成

P191. The Mystery of the Missing Submit



バグ解析 ～ボタン1押下時の処理を見る～

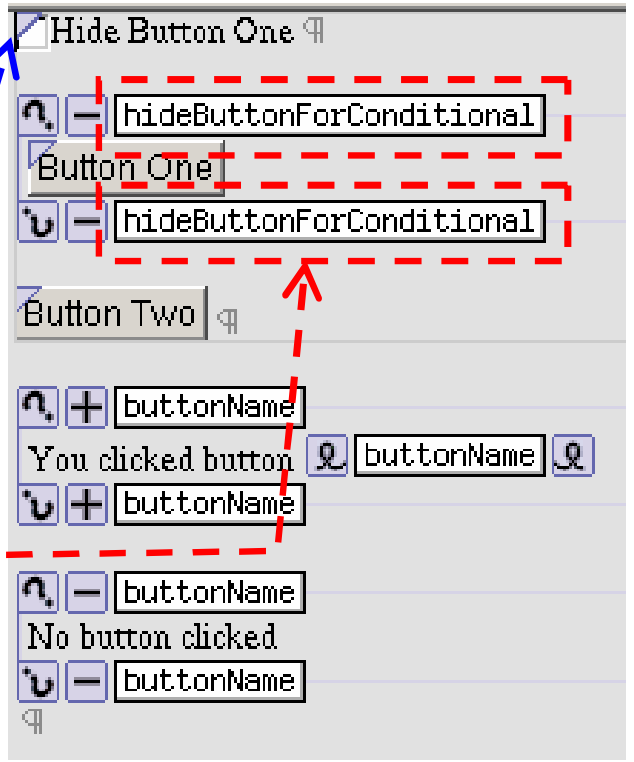
P192. The Mystery of the Missing Submit

- チェックボックスにチェックしてボタン1をsubmit
- takeValuesFromRequest
  - WOFormの子要素のうち、takeValuesFromRequestに反応するのはWOCheckBoxのみ
  - フォーム値とキーhideButtonのバインドが実行される
- invokeAction
  - WOFormの子要素のうち、invokeActionに反応するのはボタン1に相当するWOButtonのみ
  - しかし、ボタン1に相当するWOButtonはWOConditonalに囲まれているため、WOButtonは存在しないものとみなされる
  - 結局、WOFormにバインドされたアクションdefaultFormSubmitが実行される

解決方法

P192-193 The Mystery of the Missing Submit

- hideButtonForConditionalを導入して・・・
  - フォーム値を受け取る機能(=hideButton)
  - ビューを制御する機能(=hideButtonForConditional)
  - を分離する




```
public void appendToResponse
        (WOResponse response, WOContext context)
    {
        hideButtonForConditional = hideButton;
        super.appendToResponse(response, context);
    }
```

目次

- The Case of the Missing Element
  - Request-Response and Component Debugging
  - The Golden Rule
  - The Mystery of the Missing Submit
- Dynamic Elements
  - Dynamic Elements, Components, and Associations
  - How Dynamic Elements Work
  - An Example Dynamic Element
  - Replacing the WebObjects Dynamic Elements
- Mysteries of Binding Synchronization
  - Timing
  - A Binding Synchronization Example
  - Binding Instance Variables and Methods
  - Debugging Binding Synchronization
  - Manual Synchronization
  - The Carat Notation
- Stateless Components
  - The Mapped List Popup Stateless Component
- Debugging and Optimization with WOEvent
  - Preparing for WOEvent Logging
  - Displaying the Logged Events
  - Some Practice Reading the Event Log
  - Creating Custom Events

本セクションの内容

- Dynamic Elementの特性を理解して、是非、カスタムDynamic Elementを作りましょう
  - スレッドセーフ要件
  - WOAssociationと状態
  - WOComponentとの違い
- 事例) カスタムDynamic Elementとして、WOHyperlinkの代替品を実装する

Dynamic Elements

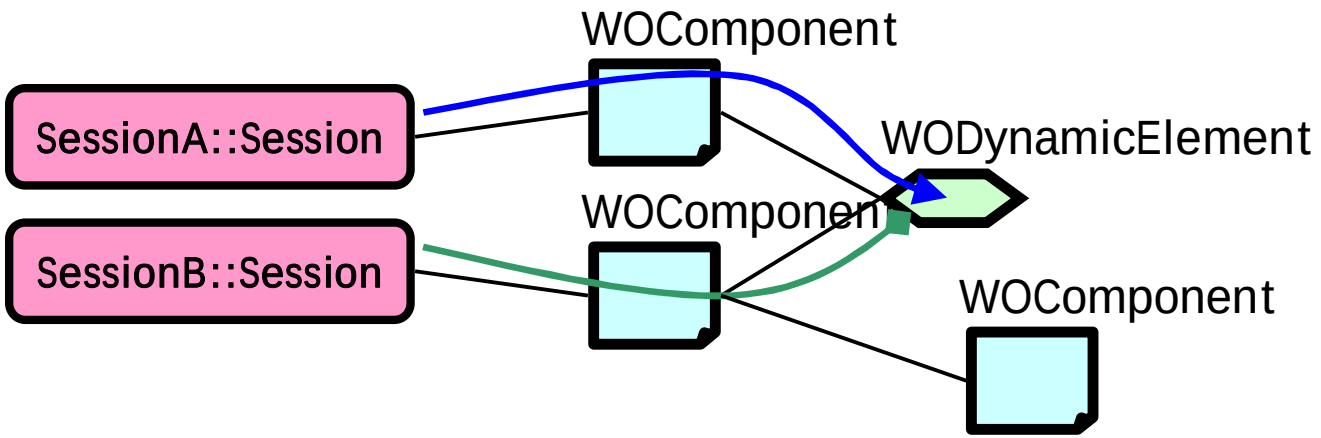
P193. Dynamic Elements

- 現状
  - カスタムdynamic elementは便利であるが、あまり使われていない
- 使われていない理由
  - ドキュメントが整備されていない、わかりにくい
- 使う理由
  - WOComponentと異なり、インスタンスが共有される。(see chap6)
  - バインディング
    - 前バージョンでは、WOComponentのバインディング実装が面倒だった。
- まとめると
  - 大量に使われるエレメントの場合、メモリーとスピードの点でカスタムdynamic elementが有利

Dynamic Elementとスレッドセーフ要件

P193. Dynamic Elements, Components, and Associations

- **カスタムdynamic elementはスレッドセーフにしなくてはならない。**
  - **複数のコンポーネント間でインスタンスが共有されるため**



スレッドセーフ性の実現とWOAssociation

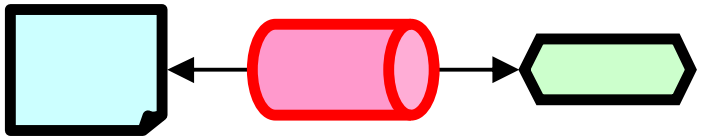
P193. Dynamic Elements, Components, and Associations

- スレッドセーフ性を実現するための一般的な方法
  - 1: 変化する状態(=インスタンス変数)を持たせない
  - 2: 変化する状態を持たせるが、同期機構を設け、状態に対する同時アクセスを排他する
- Dynamic Elementの解決策
  - インスタンス変数に「値(=状態)」を保持しない。
  - インスタンス変数には、プレースホルダとして機能するWOAssociationをのみをもち、これを介して外部の「値」を参照する。
- わかりにくいと思うので、後でまた戻ってきましょう・・・

WOAssociation

P194. Dynamic Elements, Components, and Associations

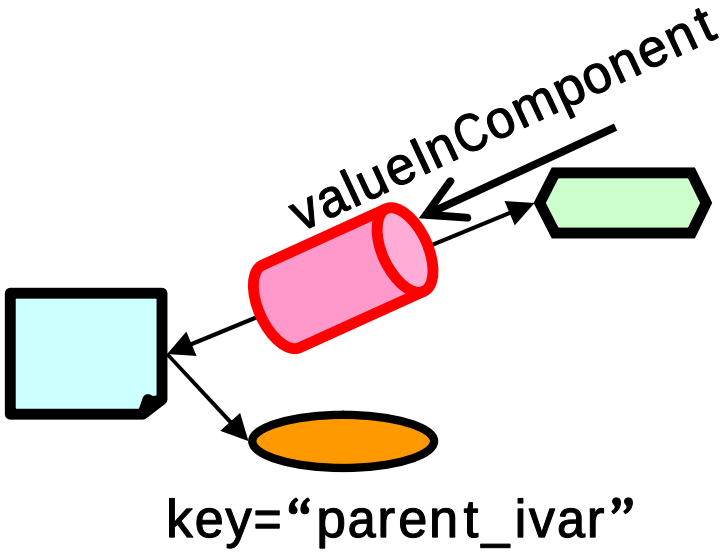
- WOAssociationの責務
  - 上位WOComponentと下位WODynamicElement 間のデータ転送のためのパイプライン



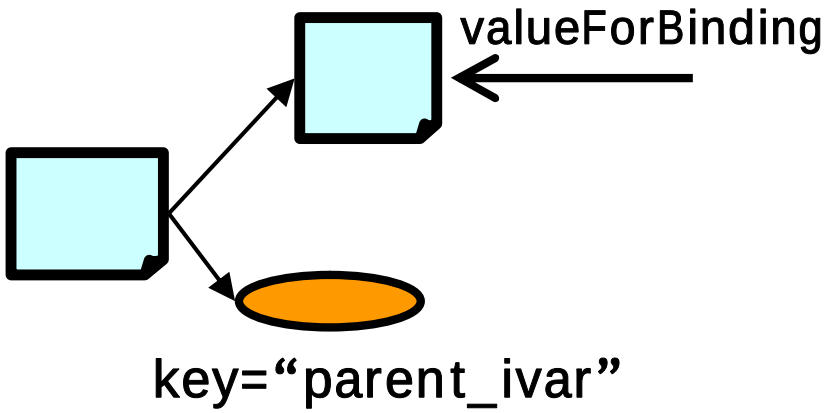
  - WR:「パイプライン」よりも「プレースホルダ」と理解したほうがよいかも
- 主要なメソッド
  - valueInComponent(aComponent)
  - setValue(aValue, aComponent)
    - WOComponentにおけるvalueForBinding(), setValueForBinding()に対応
    - `Object someVal = anAssociation.valueInComponent(aContext.component());`
- 特徴
  - Immutable (不変) なオブジェクトである
  - wodファイルと密接な関係がある (後述)

【余談】ValueForBindingとValueInComponent

WOAssociation::valueInComponent
(aComponent)



WOComponent::valueForBinding
(aBindingName)



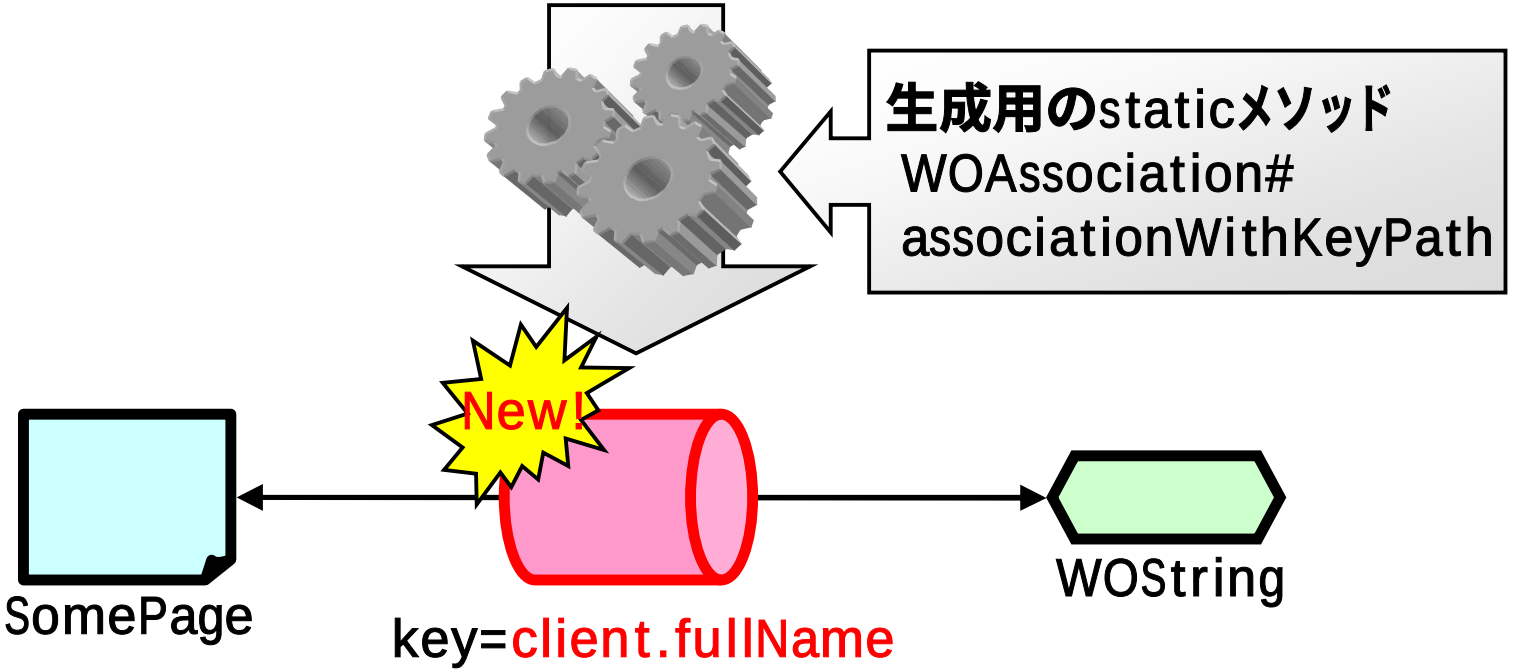
※ 引数に注意しましょう（大きなお世話？）

*.wodからWOAssociationが生成される

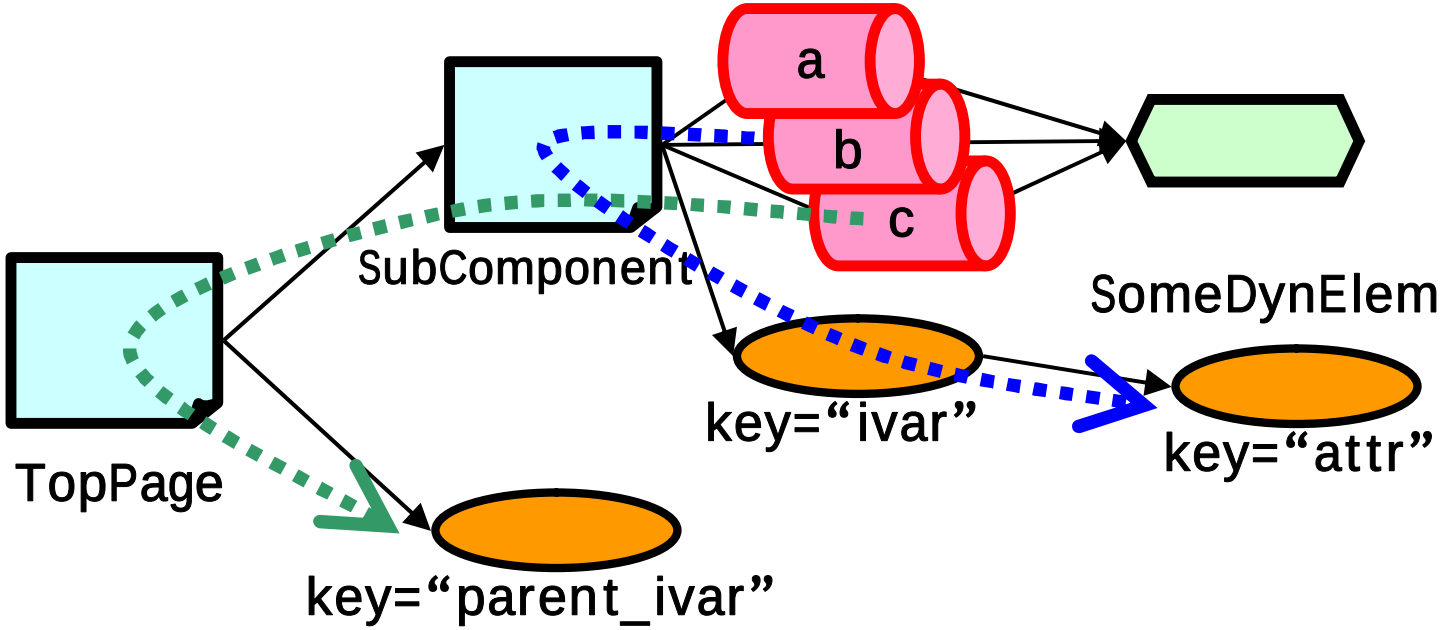
P194. Dynamic Elements, Components, and Associations

SomePage.wod

```
FullName : WOString {
  value= client.fullName
}
```



wod記述とWOAssociationの関係を具体的に・・・

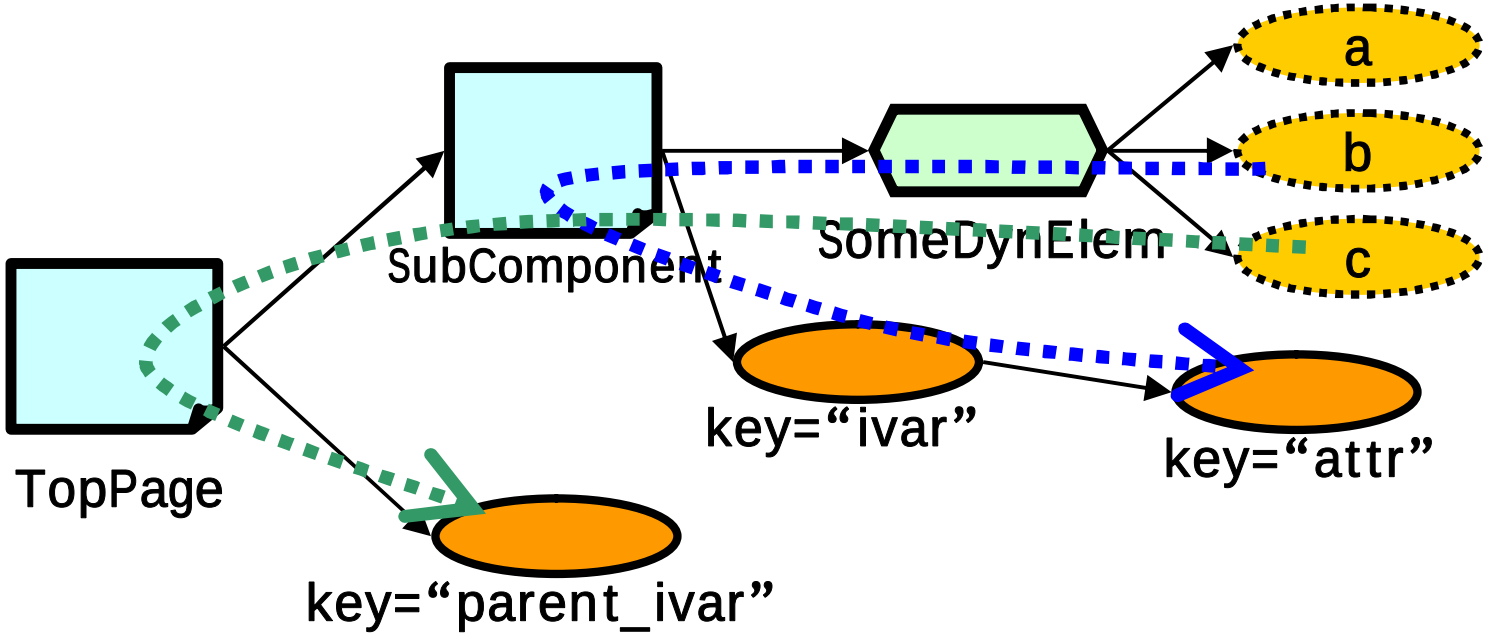


WOAssociationインスタンス	対応するSubComponent.wodの記述 (※ SomeDynElemにbindingを仮定)
a	key_a = “literalValue”
b	key_b = ivar.attr
c	key_c = ^parent_ivar

独り言：WOAssociationは、値or参照を保持するプレースホルダーと捉えたほうが理解しやすいかも・・・

【余談】WOAssociationの概念表現

- こっちの方が良い?



- どっちでもよい？ あ、そう・・・。

WODynamicElementのコンストラクタ

P194-195 How Dynamic Elements Work

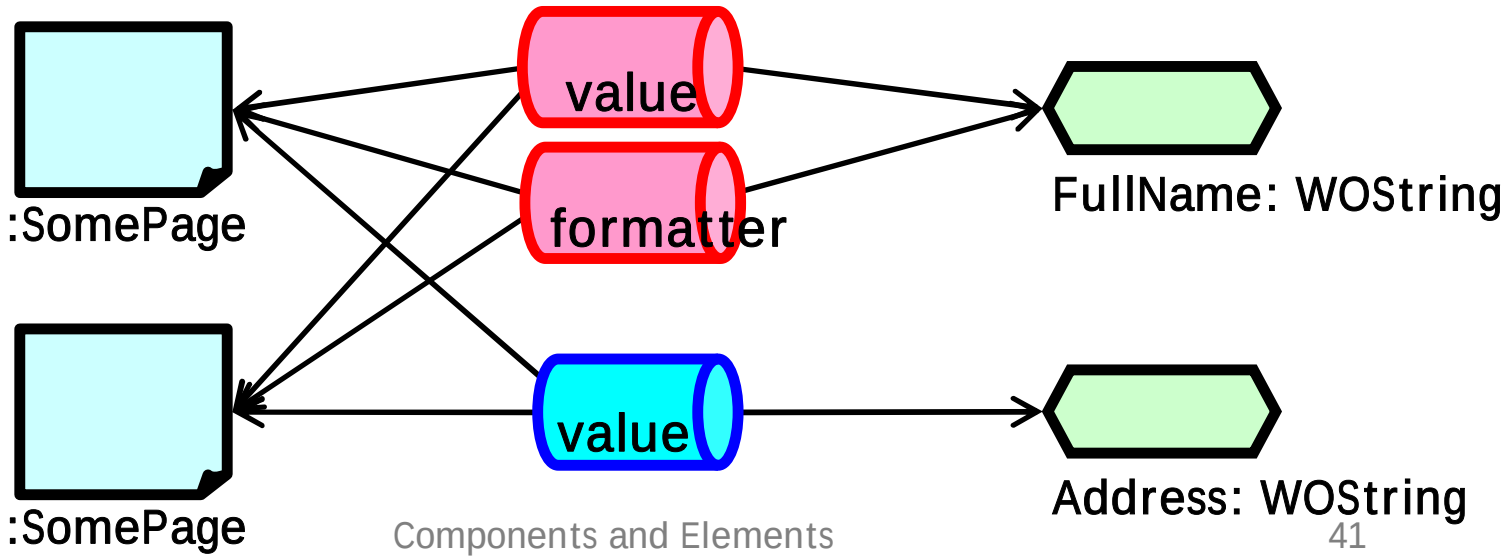
- WODynamicElement(String name, NSDictionary associations, WOElement children)

引数	意味
name	ダイナミックエレメントの名前
associations	wodで定義されたバインディング情報
children	ダイナミックエレメント内部の静的/動的HTMLまたはコンポーネント

DynamicElement, WOAssociationのインスタンス関係について

- DynamicElementはシングルトンではない
  - あるWOComponentクラスの1回の利用につき1つ生成される
  - (最大) インスタンス数＝wodエントリの数
- WOAssociationのインスタンスの数は<key>=<value>の数
- 例) SomePage.wod

```
FullName : WOString {value=...; formatter=...; }
Address  : WOString {value=...;}
```



WODynamicElementのコンストラクタですべきこと

P194-195 How Dynamic Elements Work

- associationとchildrenのバリデーション
  - 具体的な処理は、該当DynamicElementの仕様に依存する
- (Request-Responseループの間で使う) associationを保管する
- (Request-Responseループの間で使う) childrenを保管する

- (大きなお世話)
  - Immutableにしたいので、初期化はコンストラクタのみで行い、以後は初期化関連処理を行わない

WODynamicElementのサブクラスでオーバライドすべきメソッド

P194-195 How Dynamic Elements Work h

- **以下の3つのうち、すくなくとも1つをオーバライドすべき**
  - takeValuesFromRequest
  - invokeAction
  - appendToResponse

- **例えば・・・**
  - **表示専用エレメント** → appendToResponse
  - **フォーム入力用エレメント**→takeValuesFromRequestとappendToResponse
  - **アクションを起動するエレメント** → invokeActionとappendToResponse
- **必要に応じて、childrenに対してメソッド(メッセージ)を転送する**

さあ！ カスタムDynamic Elementを実装してよう！

P195- An Example Dynamic Element

- またかい・・・(笑)。
  - 先の章で実装したWOHyperlinkの代替品は、WOComponentのサブクラス版
  - 今回は、WODynamicElementのサブクラスとして実装してみる。
    - クラス名はHyperlink (see Listing 7-2,7-3,7-4)

- 実際の実装はサンプルコードを参照

Hyperlinkのコンストラクタ

P195-196 Constructiong and Validating a Dynamic Element

- 詳細はListing 7-2 (p196)を参照・・・

- やっていることをざっくり
  - 引数associationsからWOAssociationを取得＆インスタンス変数に設定
  - バインディングのバリデーション
    - 必須･オプション
    - 組み合わせ
    - その他チェック
  - 未定義バインディング情報の取得＆保管
  - 引数children (WODynamicElementのコンテンツ) の保管

HTMLの生成 - appendToResponseの実装

P197-198. Generating HTML from a Dynamic Element

- **詳細はListing 7-3を参照**
  - WOComponentサブクラス版のappendToResponse(Listing 6-x)に酷似

- **やっていること**
  - <A>エレメントの開始タグ生成
    - <A>エレメントの前半部分の生成
    - HREF属性生成 (WOContext::componentActionURL())
    - 未定義バインディング情報の<A>エレメントへの埋め込み
    - extraBindingのkey&objectを<A>エレメントの属性値として埋め込む
  - <A>エレメントの内部コンテンツ生成
    - childrenのappendToResponse()実行
  - <A>エレメントの終了タグ生成

アクションメソッドの実行 - invokeActionの実装

P198. Handling Actions in a Dynamic Element

- **詳細は**Listing 7-4**を参照**

- **やっていること** (**例外処理は省略していることに注意！**)
  - WOContext::senderID() == WOContext::elementID**である場合に限り・・・**
  - **インスタンス変数**action**から**WOAssociaton**を取得**
  - valueInComponent()**を使用して**Hyperlink**が置かれている**WOComponent**から値**(=WOActionResults)**を取得して返す**

さて、これで代替品が出来たわけだが・・・

P199．Replacing the WebObjects Dynamic Elements

- **イマイチな点**
  - WebObjectsBuilder**がレンダリングしてくれない**
- **回避するアイディア**
  - WebObjectsBuilder**には**WOHyperlink**を使用していると思わせつつ・・・**
  - **実行時は**WOHyperlink**の代わりに代替品**(Hyperlink)**を生成する**
- **その方法**
  - Application::dynamicElementWithName()
    (DynamicElement**のファクトリメソッド) をオーバライドする**
    - **参)** Chapter6 Page182.

エレメント名⇔クラス名の対応の管理

P200．Replacing the WebObjects Dynamic Elements

- 1個だけなら良いけど、たくさんあると大変
  - ElementMap.propertiesを使え！

```
com.webobjects.appserver._private.WOHyperlink = ¥
  com.apress.practicalwo.chap7app.Hyperlink
```

- 同じアイディアはコンポーネントの差し替えにも使える
  - pageWithNameをオーバライド
  - pageWithName内でコンポーネントを生成
  - コンポーネント名と生成するコンポーネントの対応はJava Propertyファイルで管理

目次

- The Case of the Missing Element
  - Request-Response and Component Debugging
  - The Golden Rule
  - The Mystery of the Missing Submit
- Dynamic Elements
  - Dynamic Elements, Components, and Associations
  - How Dynamic Elements Work
  - An Example Dynamic Element
  - Replacing the WebObjects Dynamic Elements
- Mysteries of Binding Synchronization
  - Timing
  - A Binding Synchronization Example
  - Binding Instance Variables and Methods
  - Debugging Binding Synchronization
  - Manual Synchronization
  - The Carat Notation
- Stateless Components
  - The Mapped List Popup Stateless Component
- Debugging and Optimization with WOEvent
  - Preparing for WOEvent Logging
  - Displaying the Logged Events
  - Some Practice Reading the Event Log
  - Creating Custom Events

本セクションの内容

- Synchronization機構の理解
  - Automatic
  - Manual

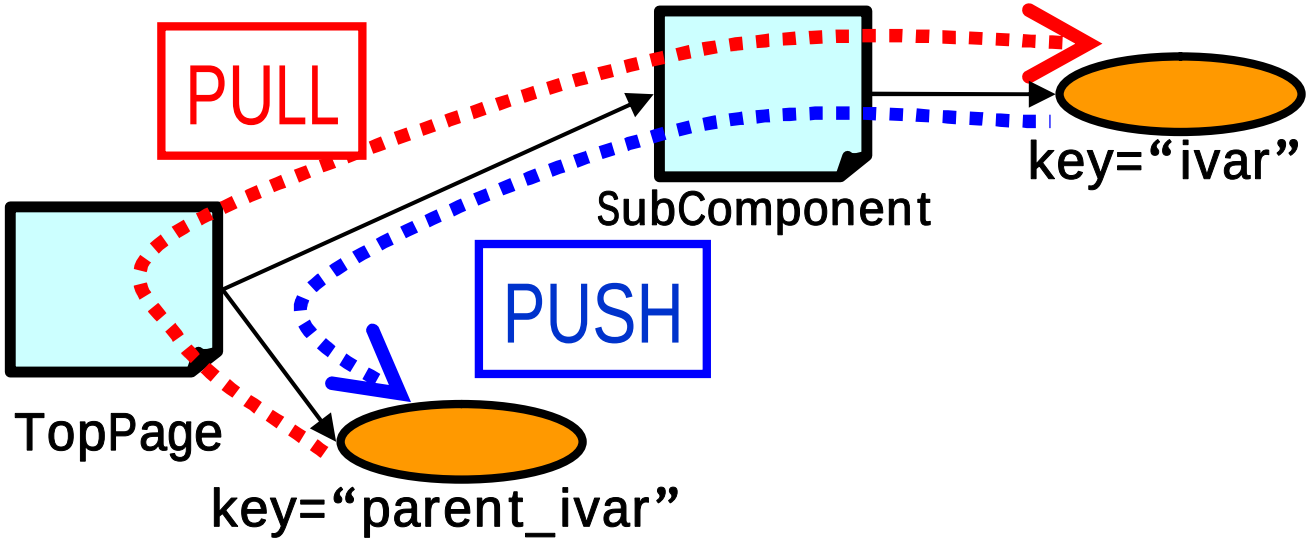
Mysteries of Binding Synchronization

P201. Mysteries of Binding Synchronization

- Synchronization**とは**?
  - Binding**を介した、親子間の値のやり取り**
- Synchronization**の分類**
  - Automatic
    - **利用するメソッド:特になし**
      - **(フレームワーク側で自動的に行われる)**
  - Manual
    - **利用するメソッド**: valueForBinding(), setValueForBinding()

PULLとPUSH

P201. Timing



- Automatic Bindingを使用する場合・・・
  - PULLしたら、必ずPUSHされる
  - どちらか片方だけはありえない
- 片方だけにする方法
  - Manual Bindingにする
  - その理由は、後ほど・・・(WR: どこに書いてある???)

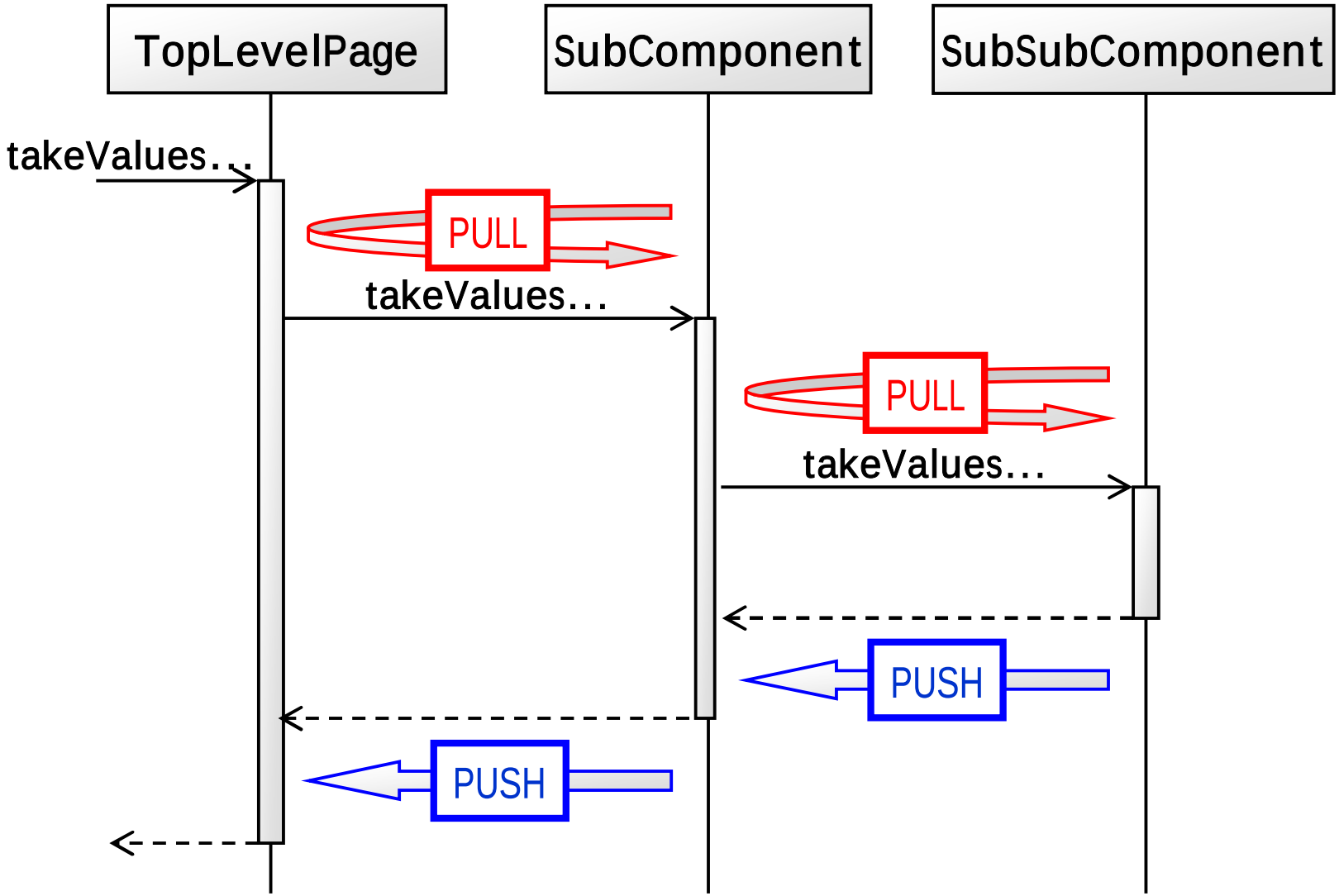
A Binding Synchronization Example

P201-202. A Binding Synchronization Example

- **では**Automatic Synchronization**で**
  PUSH/PULL**がどのように機能するか？を見てみましょう**

takeValuesFromRequestフェーズにおけるPUSH/PULL

P201-202. A Binding Synchronization Example (Listing x-x)



ポイント

- Synchronization = pull & push
- **各フェーズの前でpull、後でpushが行われている**
  - **各フェーズにおけるインスタンス変数の状態変更がキチンと伝播する**
- **コンポーネントの入れ子の階層分だけpull & pushが行われている**

バインディングとうまくやるためのルール

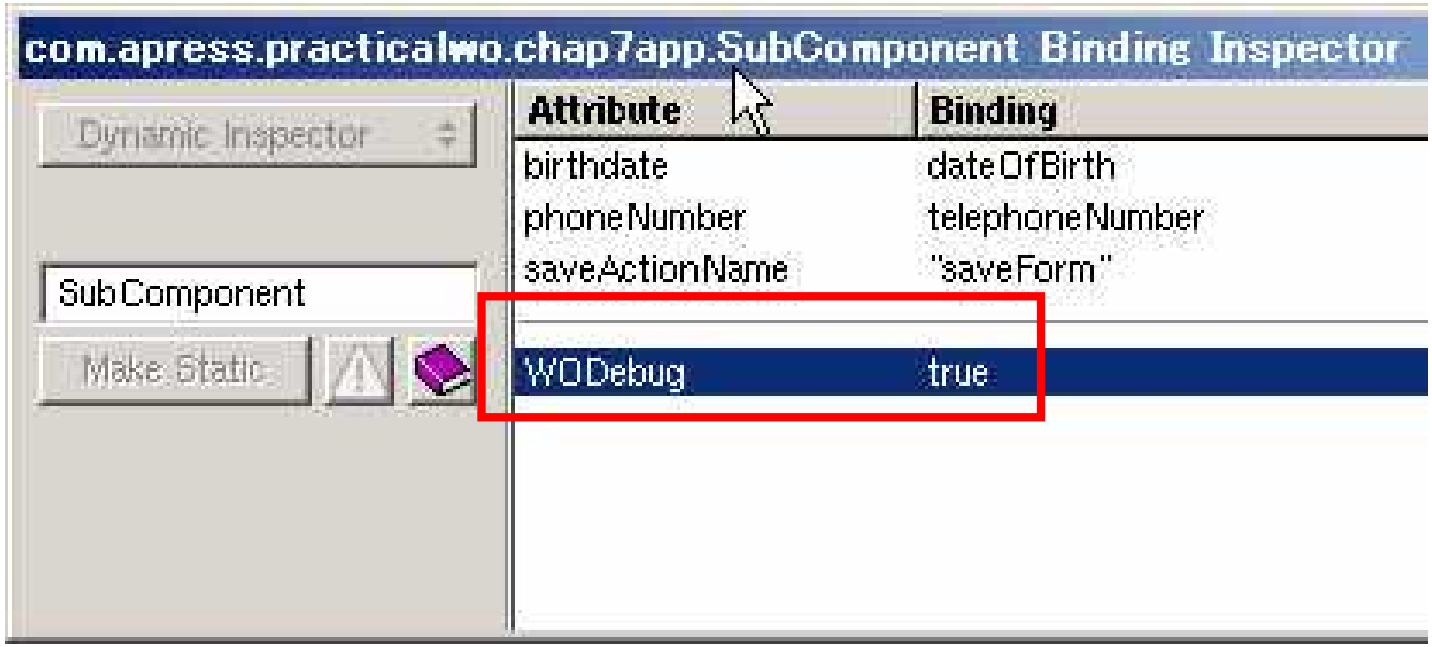
P205. Binding Instance Variables and Methods

1. publicインスタンス変数を使う
   - コンポーネント継承を使用しない場合
   - コードが少なくてすむ
   - packageを使用している場合、ほぼ必須
     - 一般的なオブジェクト設計的にはあんまり良くないけどね・・・
2. privateインスタンス変数+publicアクセッサを使う
   - コンポーネントの継承を使う場合
3. インスタンス変数なし+ publicアクセッサを使う
   - Dynamic elementのactionバインディングには適用不可(WR:????)

バインディングを介した同期のデバッグ

P205-206. Debugging Binding Synchronization

- サブコンポーネントにWODebug=trueなるバインディングを追加せよ！



Manual Synchronizationとは何か?

Pxxx. Manual Synchronization

- フレームワークによって、bindingを介した自動的なpull / pushが行われないこと
  - 前の数スライドで書いてきたこと
- → pull / pushは自分たちで行う必要がある

Manual Synchronizationの利点

- **同期機構を制御できる**
  - **不要な同期の抑制など**
- **パフォーマンス向上**
- Dynamic Element**のような「**action**へのバインディング」が可能に**
  - WR：???
- Automatic Synchronization**時に必要だった**
  インスタンス変数、アクセッサが不要に

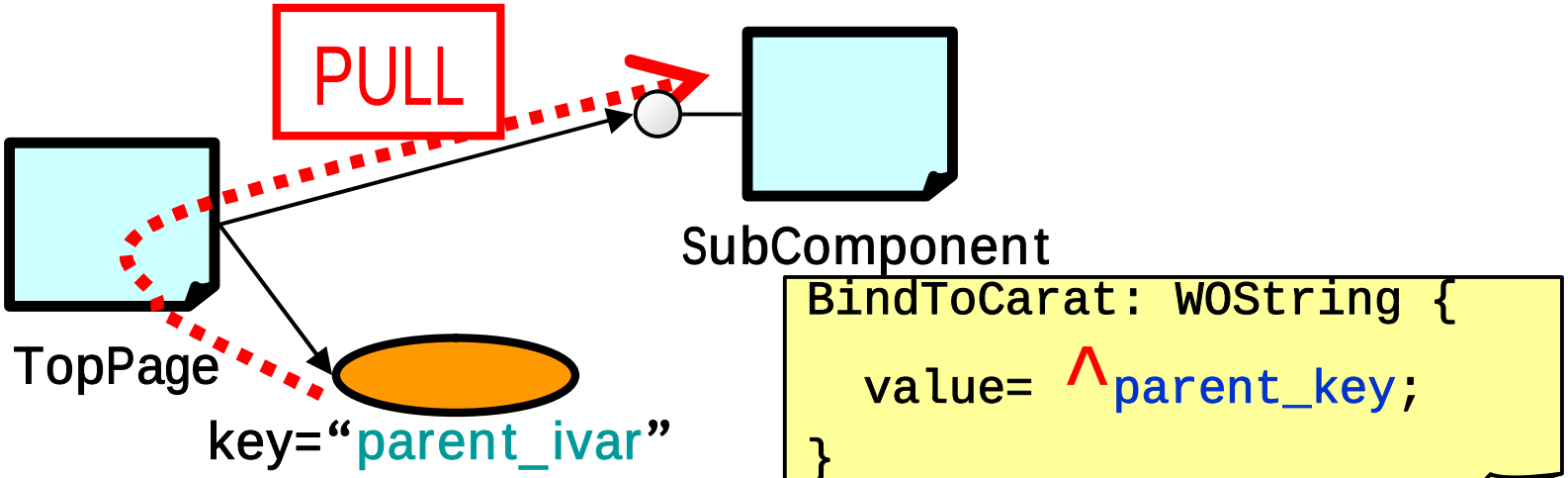
Manual Synchronizationの方法

- コンポーネントに `synchronizesVariablesWithBindings(){ return false;}`を実装
- 親コンポーネントとの値のやり取りをするコード記述
  - valueForBinding(), setValueForBinding()を利用する
  - 戻り値のキャスト
- 信頼性の高いコンポーネントのために
  - hasBinding()を用いてbindingの有無を確認
  - canSetValueForBinding()を用いてbinding先が定数でないことを確認

Manual Synch時の子→親方向の参照を簡単に！

P207. The Carat Notation

- Manual Synchronizationにすると、(当然) bindingを介した親⇔子Component間の値のやり取りが行われない
  - Manualなんだから当たり前！
- じゃ、いちいちvalueForBinding()を書くか?
  - valueForBinding() : Bindingをたどって親Componentにアクセスするメソッド
- 面倒！ そこで、Carat('^') Notation!




```
BindToCarat: WOString {
  value= ^parent_key;
}
```

```
ChildCmp: SubComponent{
 parent_key = parent_ivar;
}
```

目次

- The Case of the Missing Element
  - Request-Response and Component Debugging
  - The Golden Rule
  - The Mystery of the Missing Submit
- Dynamic Elements
  - Dynamic Elements, Components, and Associations
  - How Dynamic Elements Work
  - An Example Dynamic Element
  - Replacing the WebObjects Dynamic Elements
- Mysteries of Binding Synchronization
  - Timing
  - A Binding Synchronization Example
  - Binding Instance Variables and Methods
  - Debugging Binding Synchronization
  - Manual Synchronization
  - The Carat Notation
- Stateless Components
  - The Mapped List Popup Stateless Component
- Debugging and Optimization with WOEvent
  - Preparing for WOEvent Logging
  - Displaying the Logged Events
  - Some Practice Reading the Event Log
  - Creating Custom Events

本セクションの内容

- Stateless Componentの特性
  - Dynamic Element, Stateful Componenとの違い
- Stateless Componentの実装方法
- Stateless Componentの実装例としてのPOMappedPopupList

Stateless ComponentとDynamic Element

P207．Stateless Components

- **インスタンス数**
  - DynElm: **あるWOCクラスにおける利用回数**
  - Compnt: **参照される回数**(???)
- **動作に必要なメモリ量**
  - Dynamic Element < Stateless Component < Stateful Component
- **開発の容易さ**
  - stateful Component > Stateless Component > Dynamic Element
    - Dynamic Element**の開発は面倒**
      - WOAssociation**の処理が必要**
      - WO**のテンプレート機構が利用できない**

Stateless ComponentとDynamic Element

P207-208．Stateless Components

- **利用可能な子エレメント**
  - Stateful：**全て**OK
    - Stateful Component
    - Stateless Component
    - Dynamic Element
  - Stateless：**子に**Stateful Component**をもてない**
    - Stateless Component
    - Dynamic Element
  - (Dynamic Element：**全て**NG??)
- **スレッドセーフ要件**
  - Stateless Component: **不要**
  - Dynamic Element：**必須**

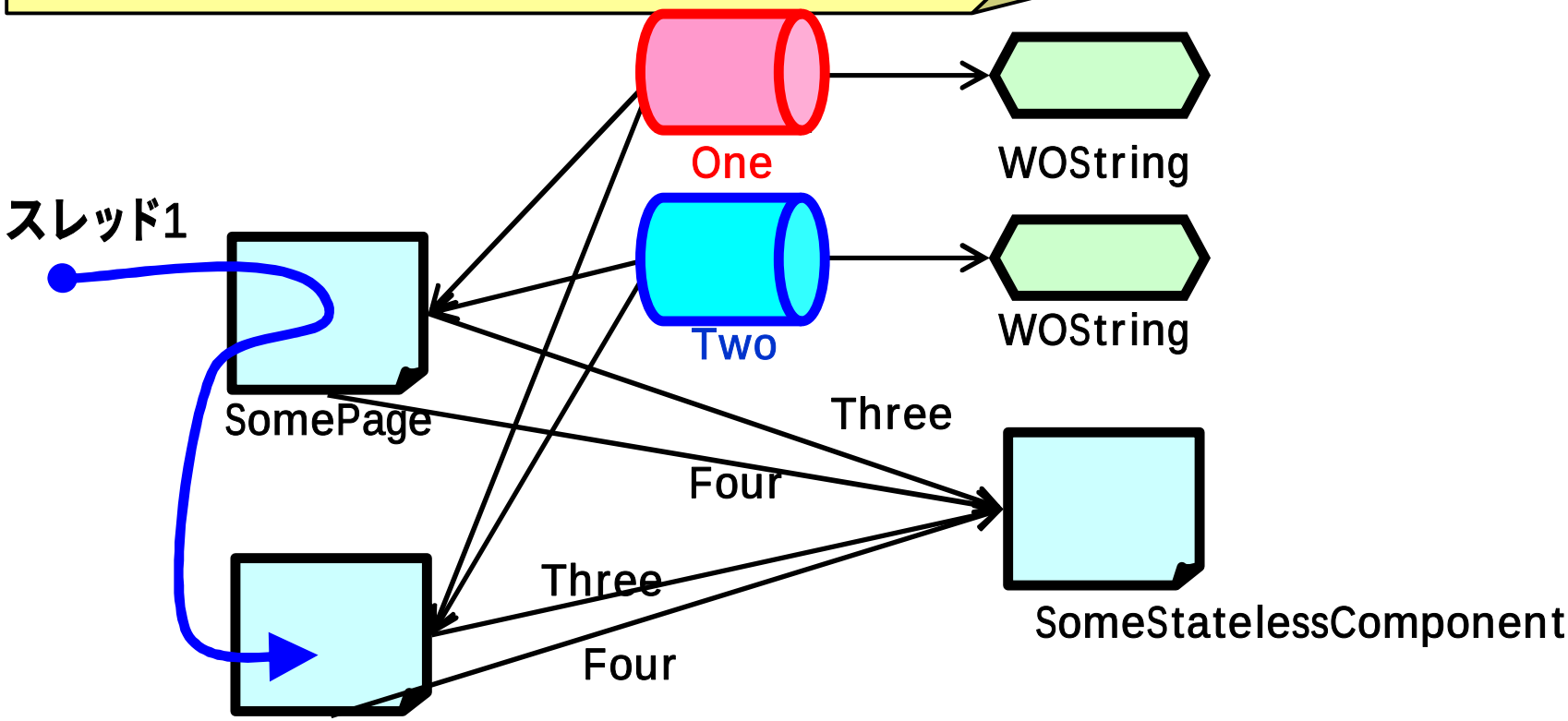
WR:ContentsとChildrenの区別が曖昧なため、注意して読んでください!!!!!

インスタンス数の比較 - シングルスレッドアクセス時

SomePage.wod

P208. Stateless Components

```
One : WOString {...}
Two  : WOString {...}
Three : SomeStatelessComponent{ ...}
Four : SomeStatelessComponent{ ...}
```

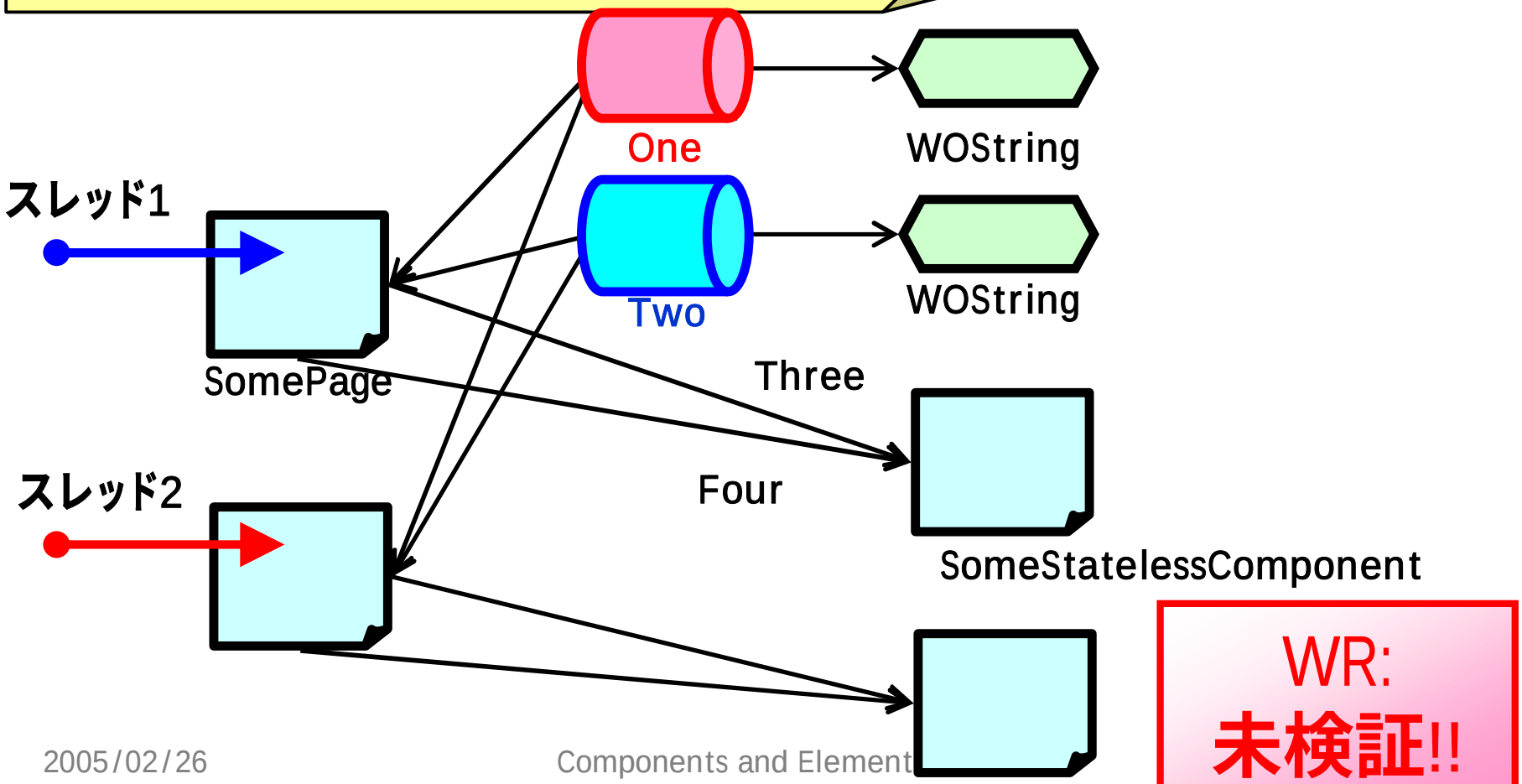


インスタンス数の比較 - マルチスレッドアクセス時

SomePage.wod

P208. Stateless Components

```
One : WOString {...}
Two  : WOString {...}
Three : SomeStatelessComponent{ ...}
Four : SomeStatelessComponent{ ...}
```



WR:
未検証!!

Stateless Componentの作成方法

P208-209．Stateless Components

- `isStateless() {return true;}`を実装すればよい
- Stateless componentはデフォルトではManual Synchronization
  - 推奨しないがAutomaticにもできる
- Stateless Componentは状態を持つ(!)
  - しかし、状態を保持できる期間はあるRequest-Responseループ内のあるフェーズ内に限定される
  - フェーズ終了後に`reset()`が呼ばれるので、ここに初期化処理を記述すること
    - 基本的な実装パターンはListing 7-8を参照

Stateless Componentの例としてのPWOMappedPopup

P209．The Mapped List Popup Stateless Component

- 表示用の文字列を格納したArray
- と、
  - 選択肢となるオブジェクトを格納したArray
    - または
  - 選択肢となるオブジェクトを格納したDictionary

- をとるPopupボタン

- 表示用の文字列を格納したArrayで表示の順序を制御できる

PWOMappedPopupの実装ポイント

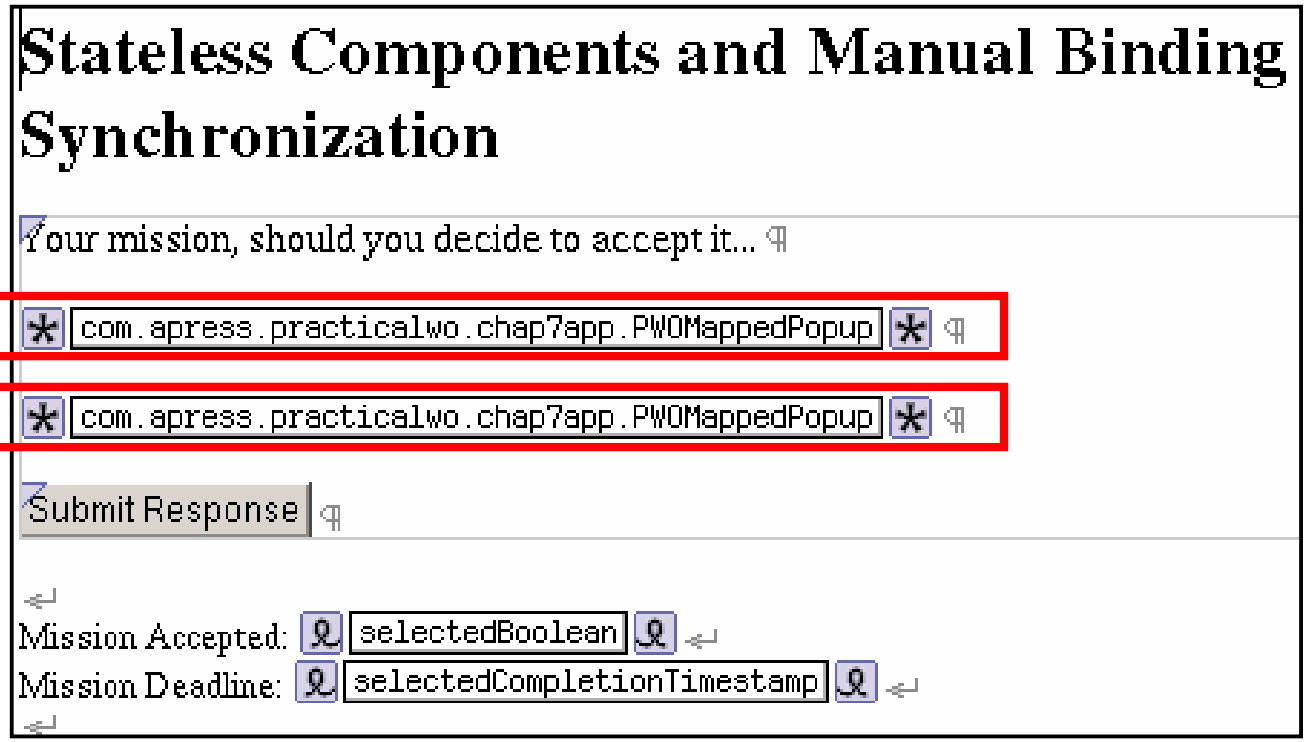
P210-211．The Mapped List Popup Stateless Component

- 関連するオブジェクトのhashCodeをロギングしている
  - インスタンス関係をわかりやすくするため
- Bindingのvalidationを一括して行っていない
  - Bindingされ方が色々あるため(???)
  - パラメータを利用する都度、validationを行っている
- 隠しAPI _setParent()をロギング
  - 同一PWOMapped..に関連するインスタンスの情報が切り替わる様をロギングしたいため
- pullValues...(), pushValues...()は実行される
  - ただし、Synchronizationは実行されない
- reset()をコメントアウトすると面白いかもね
  - → 2つのPWOMapped...の選択肢リストが同じになる

PWOMappedPopupを利用してみる

P211-212．The Mapped List Popup Stateless Component

- **利用例の構成**
  - StatelessManualBindingSynching内にPWOMappedPopupが2つ存在



ログ(Listing 7-9.)を見てみると・・・

P210-212．The Mapped List Popup Stateless Component

- StatelessManualBindingSynching内にPWOMappedPopupが2つ存在するが、コンストラクタは1回しか呼ばれない
  - インスタンスは1つのみ
- _setParent(), _awakeInContext(), awake(),take...()などは2回呼ばれる
  - _setParent() の1回目と2回目では渡されるbinding情報が異なる
- 各フェーズの終了後にreset()が呼ばれる
- 各フェーズの前後に、pullValues..(), pushValues...()が呼ばれる

今回はここまで・・・