ZK Training

MODULE 3-1

ZK BEST PRACTICE

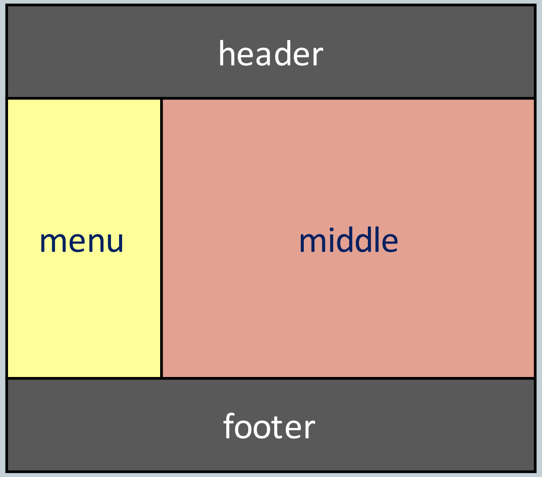
Outline

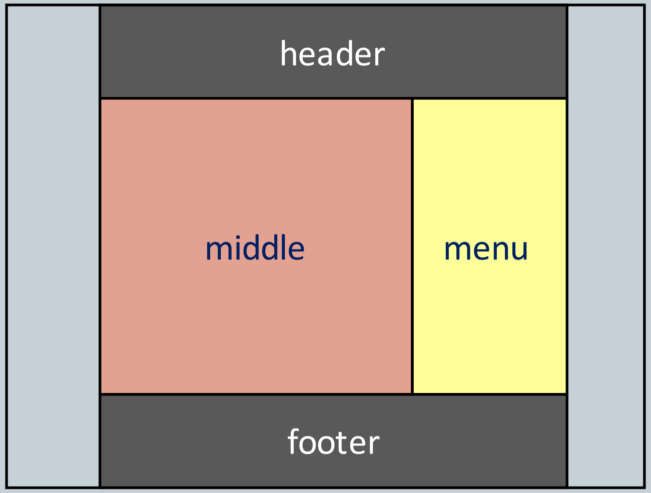
- Layout injection
- Composite Component
- Huge Data Modeling(ROD)
- Long running operations in MVVM
- ZK JSP technology
- Technology Guidelines

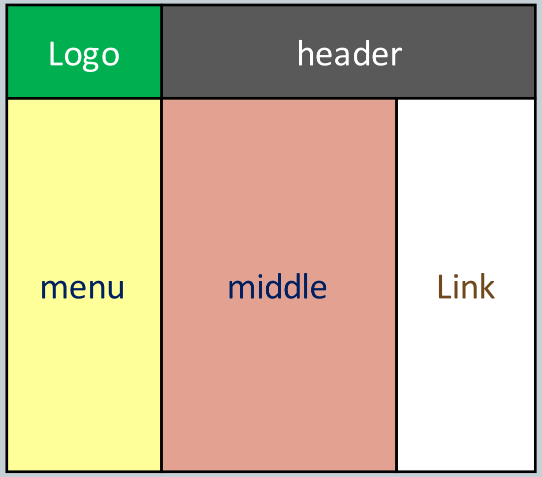
Layout Injection

Layout template

- Injecting (loading) a predefined template inside a layout







Layout injection (1)

- Create Component
  - Executions.createComponents(uri, parent, data)
- Include - Include another page
  - Instant - immediately creates components
  - Defer - sends a new Servlet request
  - Default is Auto - choses instant or defer automatically

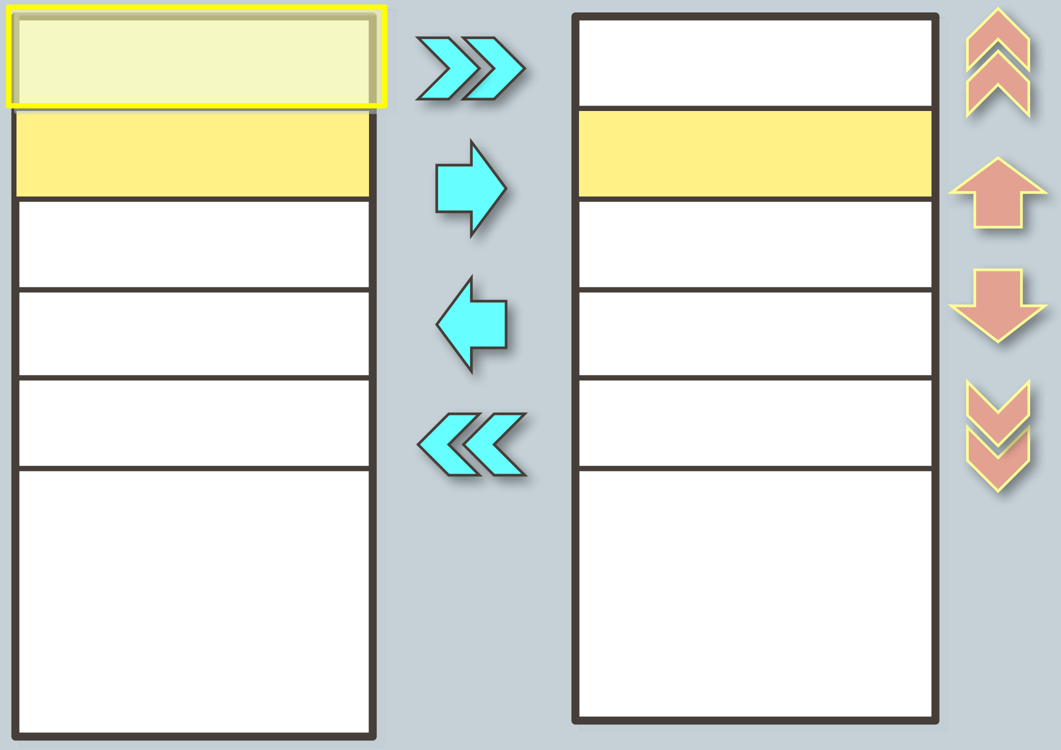
Layout injection (2)

- Macro - define a reusable component
  - Regular Macros (creates HTMLMacroComponent)
  - Inline Macros - expands inline i.e. root element becomes direct descendant
- Composition - compose UI based on smaller fragments
  - Define fragments
  - Apply template

Macro Component

- Make reusable components out of basic ZK components
- Example: DualListbox
- Code walk through

Example: DualListbox



Example: DualListbox

```
<?component name="dual-listbox" extends="div"
class="composite.DualListbox"?>

<window title="Dual Listbox" border="normal" width="100%" height="500px"
    apply="composite.ctrl.DualListboxDemoCtrl">
    <dual-listbox id="dualLBox"
        renderer="composite.PersonDualListitemRenderer" />
</window>
```

Handling Large Data

Handling Large Data

- What are the options?
  - Live data & Paging
  - Enable ZK Render on Demand (ROD)
  - Biglistbox
  - ZK Spreadsheet
  - Custom model
- Custom Model Code Walkthrough

Handling Large Data

- Use "live data"

  ```
  <listbox model="${model_name}">
  ```

  - set model instead of using:
    - forEach

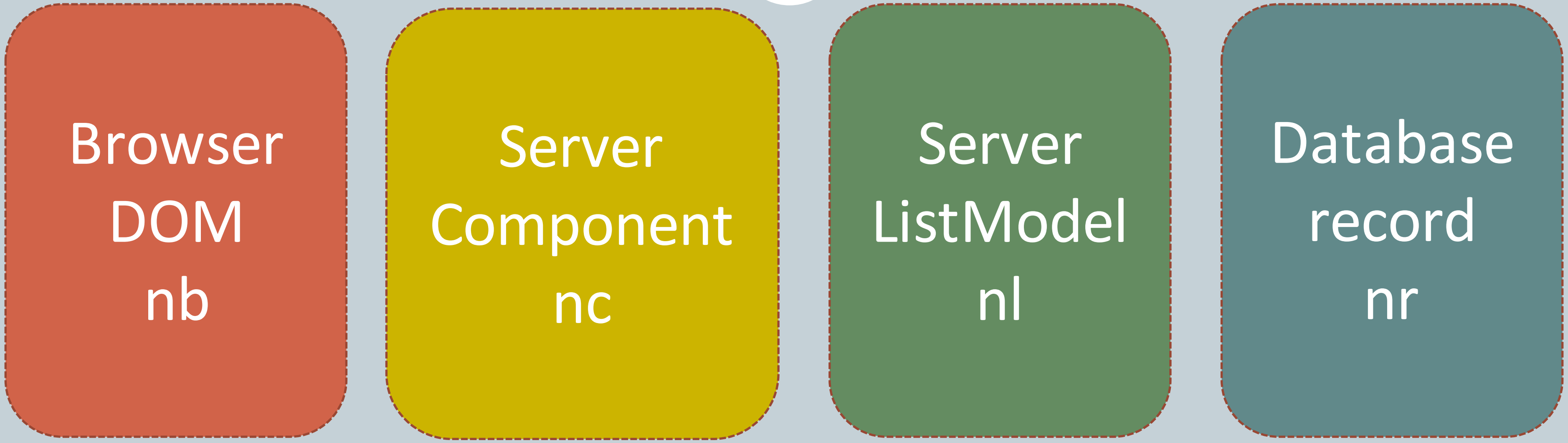
      ```
      <listitem label="${each}" forEach="A,B,C"/>
      ```

    - component manipulation Java APIs

      ```
      new Listitem().setParent(listbox)
      ```

- Benefits:
  - data pertaining to list items are sent to client only when they are required to be visible
    - note this provides only client side performance enhancement

Handling Large Data



- No ROD
  - nc = nl = nr
- ROD enabled
  - nc << nl = nb

Handling Large Data

- Enable Render on Demand (ROD)
  - Global (across app in zk.xml):

```
<library-property>
   <name>org.zkoss.zul.listbox.rod</name>
   <value>true</value>
</library-property>
```

  - Local (component or idspace):

```
<custom-attributes org.zkoss.zul.grid.rod="true" />
```

- Benefits:
  - Only the necessary data chunk associated with the visible components are fetched from the ListModel
  - list items are only created on the server and their data sent to the client when they're required to be visible
    - less memory and processing time on both the server and client

Handling Large Data

- Create Custom ListModel
  - why?
    - ZK's default implementations such as ListModelList assumes all data are available in memory; however, this may not be practical when data are large
  - how?
    - extend AbstractListModel
    - implement getElementAt(int index) such that when the desired entry is not in cache, we load its associated data range to replace the data in cache
- Benefits:
  - Further optimize ROD by minimizing the data present in the model

Handling Large Data

- Custom Model Code Walkthrough

Long Operation

Long Operation in MVVM

- Echo Event
- Timer
- Server Push
  - Wrapped as an abstract class
  - Extends everywhere (in ViewModel, SelectorComposer)

Echo Event

- When an operation takes a longer time, ZK will display the default busy message
- Let you update UI before starting a long operation
  - display more friendly, informative messages
  - Block further user interaction
- When users trigger the first event, just update UI status
  - Don’t start a long operation
- But start it at the 2nd custom event
- Will block users other server side operations

Timer

- Run a long operation in a separate thread when users trigger an event
- Update UI status through Timer events sent repeatedly by a Timer to a server

Server Push

- http://books.zkoss.org/wiki/Small_Talks/2015/January/Simplify_Long_Operation_Handlings
- Utility class `LongOperation`
  - Provide simplified API by wrapping the event queue API
  - Create an instance and override methods
  - Start a long operation easily
  - Can display operation progress
  - can abort the long operation
  - Create an event queue internally
  - Execute a long operation in a working thread

Server Push

- Start a long operation
  - doLongOperation(), implement your long operation, is invoked asynchronously
  - onFinish(), update UI with the result
- Show the progress
  - Provide a callback class that implement LongOperationCallback
  - modify UI in the callback

Modularize

Modularize UI

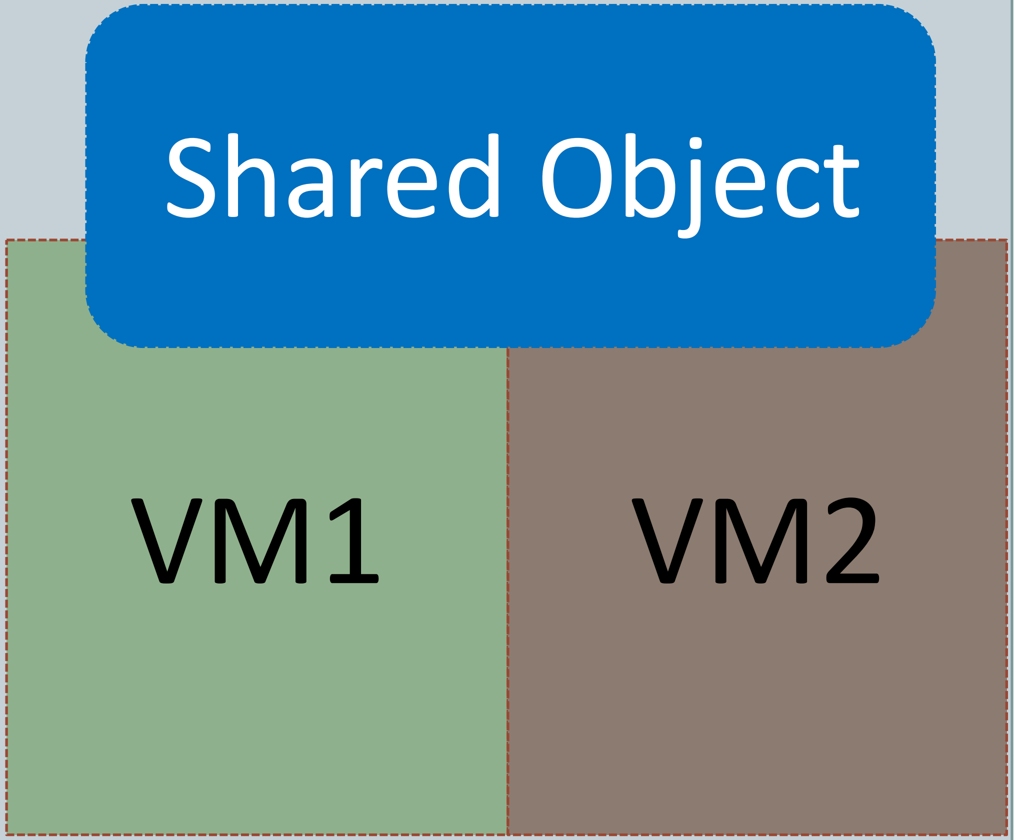
- Divide a system view into multiple, independent modules
- Advantages:
  - Reuse
  - easier to maintain
  - Bring abstraction
- Main issue is “communication” among separate views

Modularize UI

- Communication
  - Global command
    - Hard to debug if there are many calling – invoke a commend method by Java reflection
  - Scoped attribute
  - Call back
  - Observer pattern
  - other OO techniques
- 2 ViewModels relationship
  - Separate
  - Parent - Child

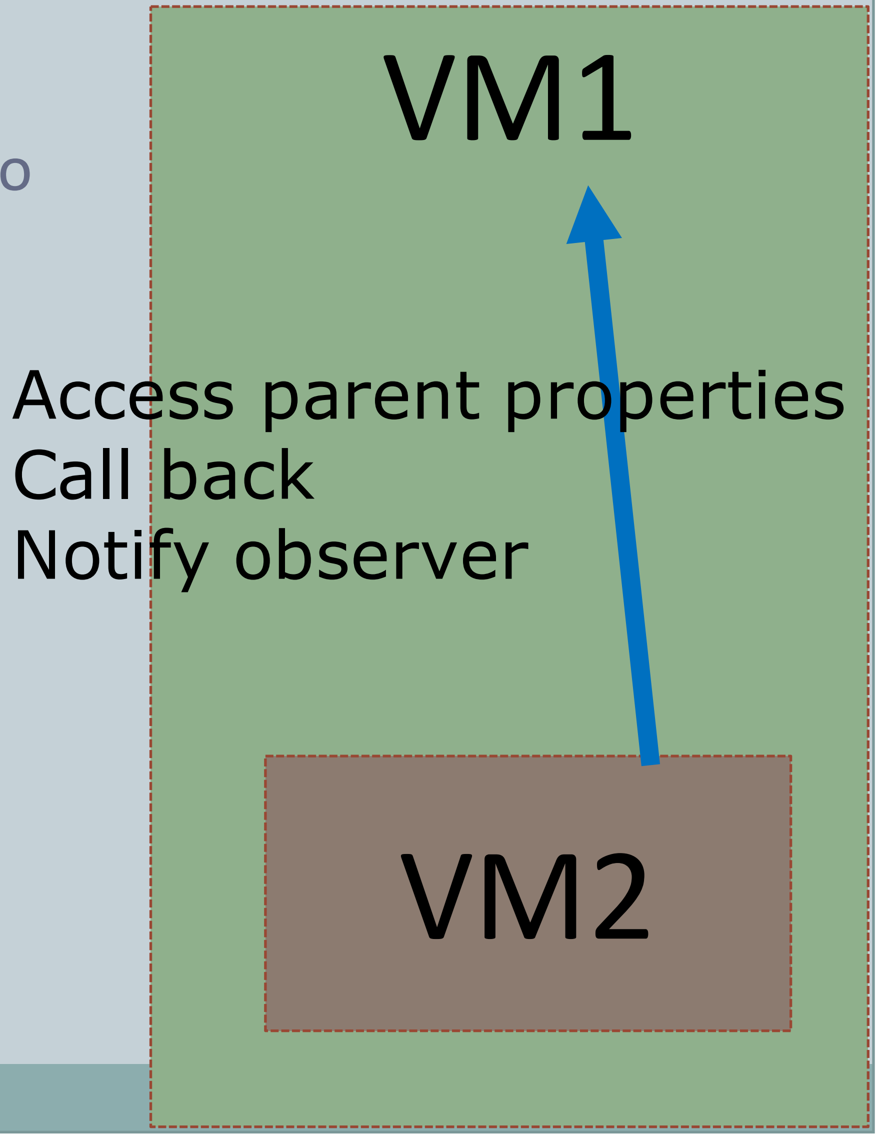
Modularization

- Separate
  - Share with desktop attribute
  - Use ”notification” to notify another VM
  - separate-vm.zul



Modularization

- Parent – Child
  - Parent VM can create child VM to get access to it
  - parent-child.zul



Performance Tuning

Performance Tuning

- Memory: reduce the number of ZK component
  - ROD
  - Use light-weighted component sets (xhtml, native)
- Time: Reduce rendering time
  - ROD
  - fulfill attribute
  - renderdefer attribute

ZK JSP

ZK JSP Tag Library

- How to configure
  - Copy zuljsp.jar to WEB-INF/lib or use maven dependency
- How to use ZK JSP Tags
  - Declare DOCTYPE
  - Body style CSS
- Pass variable from Jsp to Zul
  - via request or session scope
- Use MVC or MVVM just as you would in a ZUML

Technology Guidelines

Single or Multiple pages

- Traditional page-based Web framework
  - Split an application into multiple pages for one function
  - Push results to view
  - For example, pure JSP-Servlet, Struts, Spring MVC
- Ajax Web framework (ZK)
  - Group a set of functionality into a single page
  - Pull results from multiple controllers in one view

ZUL or Richlet

- ZUL
  - Strongly Suggested
- Richlet
  - Compose UI in pure java
  - Swing like
- ZUL + JSP
  - Design Layout with JSP (pure HTML tag for header/footer)
  - Include ZUL page
    - <jsp:include> tag
    - ZK JSP tag

Multiple Pages in Ajax Framework

- Each page has a set of functionality
  - For example, separate pages based on user role
    - admin, logged user, anonymous user

Single Page

- User browse behavior
  - Browser's back button
- Bookmarks
  - Simulate multiple pages
    - A single page with multiple states
  - Reference:
    - Browser History Management
    - HTML 5 Push State

ZScript

- Zscript
  - Embed Java code in ZUL pages
  - Used for prototyping, POC and testing
  - ***NOT*** suggested for production systems

MVC or MVVM

- MVC
  - Straightforward
  - Good for small user interfaces
    - Implementing macro or composite component
- MVVM
  - Requires extra design thinking
  - Suggested for better separation of concerns
  - Consume more memory than MVC
  - More complex when using Richlet

MVVM Data Binding

- Automates the data-copy plumbing code (CRUD) between UI components and the data source
- Easy to read and maintain
- When not to use
  - Developers don't familiar with EL expressions

Native or XHTML

- Native Namesapce
  - Allow any HTML tags
  - Suggested for static content
    - Not be able to change content through Ajax
    - Saves memory on server
- XHTML
  - ZK components to represent HTML tags (i.e., org.zkoss.zhtml.div)
  - Suggested for dynamic content
    - Consumes more memory because it is a ZK component

Include, Macro, Composition

- Include
  - Can include different pages, such as html, jsp, and so on
  - Can't encapsulate its behavior in a Java class
- Macro (Composite)
  - Reusable component to group ZK components
  - Encapsulate the behavior in a Java class
    - Optional for macro, Must for composite
- Composition
  - Assemble UI at runtime
  - Base on user role or preference

Performance and Security

- Performance Tips
- Security Tips

Q & A