Determine the layout initialization process

Determine how layout is compiled.

View\Layout::*build* = View\Layout\Builder::*build* - once

1. loadLayoutUpdates - reads layout xml files and DB updates by*current handles*, result in `_updates` array
    - event `layout_load_before`
2. generateLayoutXml - joins `_updates` into XML string, loads XML object, initiailzes `_elements` = []
    - layout.generateXml
    - no events
3. generateLayoutBlocks - layout.generateElements
    - event `layout_generate_blocks_before`
    - readerPool->interpret - every reader checks for matching xml node name (attribute, body, head etc.), prepares page structure
    - generatorPool->process
    - add top-level element as outputElement. default “root”
    - event `layout_generate_blocks_after`

```
builder.generateLayoutBlocks -> layout.generateElements
readerPool.*interpret* -- schedules
  nodeReaders[type].interpret each element -- Layout\ReaderInterface
  - html, move
  - body - own readerPool.interpret, but without 'body' reader
  - head - css,script,link,remove,meta,title,attribute
  - 'container', 'referenceContainer' --> Layout\Reader\Container
  - 'block', 'referenceBlock' --> Layout\Reader\Block
  - uiComponent
  View\Layout\ReaderInterface::interpret -- html, body, head, ui component, reader
pool, container, move, block

View\Layout\GeneratorPool.process -- generates blocks
  buildStructure
  generator[].*process*
  - head, body
  - block - creates blocks, sets layout, event `core_layout_block_create_after`,
block 'actions'
  - container - sets tag, id, class, label, display
  - uiComponent - creates wrapper element, prepareComponent recursively etc.

getOutput
- renderElement(root)
  - renderNonCachedElement(root) -- hardcoded switch
    - is uiComponent -> toHtml
    - is block -> toHtml
    - is container -> renderElement(child[])
  - event `core_layout_render_element`
```

How would you debug your layout.xml files and verify that the right layout instructions are used?

When XML object is created from string XML updates, this is a good place to examine resuls. View\Layout\Builder.generateLayoutXml or View\Layout.generateXml is a good place to dump structure.

Determine how HTML output is rendered.

First external code calls layout.addOutputElement(name) to register top level elements for output generation. When layout.*getOutput* is called, it renders each outputElement.

- View\Layout::getOutput
- for every registered `_output` element
- View\Layout::renderElement

When layout is build initially, it finds top level container and registers it as addOutputElement - default “root”.

How does Magento flush output, and what mechanisms exist to access and customize output?

Render response:

- action controller returns ResponseInterface object, e.g. View\Response\Page
- front controller returns this response
- App\Http.launch application renders result, calling response.renderResult
- controller response renders contents and assigns to response.body
- App\Http event `controller_front_send_response_before` allows to modify response before returning
- App\Bootstrap object gets response object from App\Http
- response.sendResponse - Zend implementation - send headers, send content

Customize:

1. modify any response in event `controller_front_send_response_before`
2. View\Element\AbstractBlock::toHtml event `view_block_abstract_to_html_after`

Determine module layout XML schema.

Module layouts:

- module/view/base/layout/
- module/view/frontend/layout/
- module/view/adminhtml/layout/

layout_generic.xsd

Generic layout - useful for returning Ajax response. Doesn’t have body, head, css etc., only pure structure:

```
<layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/layout_generic.xsd">
  <container
    name="" as="" after="" before="" label=""
    output=""
    htmlTag="" htmlClass="" htmlId=""
  />
  <referenceContainer
    name="" label=""
    htmlTag="" htmlClass="" htmlId=""
    display="true/false"
    remove="true/false"
  />
  <block
    class="" name="" as="" template="" before="" after="" group=""
    acl="" aclResource="" ifconfig=""
    output="" cacheable="bool" ttl="int"
  />
  <referenceBlock name="" template="" display="" remove="" />
  <update handle="name" />
  <move element="" destination="" as="" after="" before="" />
  <uiComponent
    component="" name="" as="" before="" after="" group=""
    aclResource="" ifconfig=""
    output="" cacheable="" ttl=""
  />
</config>
```

page_configuration.xsd

Same as layout plus:

- top level layout selection
- structure is wrapped in body node
- additional nodes - html attribute, head title, attribute and scripts

```
<page
  layout="1column"
  label=""
  design_abstraction="custom / page_layout"

  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd"
>
  <html>
    <attribute name="" value="" />
  </html>
  <head>
    <title>Title</title>
    <css src defer ie_condition charset hreflang media rel rev sizes target type src_type />
    <link src defer ie_condition charset hreflang media rel rev sizes target type src_type
/>
    <meta content charset http-equiv name scheme />
    <script src defer ie_condition async charset type src_type />
    <remove src="" />
    <attribute name="" value="" />
  </head>
  <update handle="" />
  <body>
    <attribute name="" value="" />
    <!-- container/block structure same as in generic layout -->
  </body>
</page>
```

layouts.xml - declare available page layouts

- <module_dir>/view//layouts.xml
- <theme_dir>/_/layouts.xml
- base:
    - empty
- frontend:
    - 1column - extends empty
    - 2columns-left - extends 1column
    - 2columns-right - same as 2columns-left
    - 3columns - same as 2columns-left
- adminhtml:
    - admin-empty
    - admin-1column
    - admin-2columns-left

page_layout.xsd

Only containers

```
<layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_layout.xsd">
  <referenceContainer />
  <container />
  <update />
  <move />
</layout>
```

layout result.renderResult:

- layout.getOutput
- layout.build
- View\Layout\Builder.build:
    - loadLayoutUpdates
    - generateLayoutXml
    - generateLayoutBlocks
- create result page - adds default handles
- page builder.build
- all normal layout XML is collected and is about to create blocks
- custom step before generating layout blocks

page result.renderResult:

- View\Result\PageFactory.create

- View\Result\Page::addDefaultHandle - `default` , `$fullActionName`

- View\Page\Config.publicBuild = build

- View\Page\Builder.build - extends View\Layout\Builder, custom readPageLayout on step generateLayoutBlocks

    - (inherit) loadLayoutUpdates

    - (inherit) generateLayoutXml

    - generateLayoutBlocks - additionally *readPageLayout*

        - right before creating actual blocks, reads and merges page layout xml files
        - new instance of View\Model\Layout\Merge::load handles = ‘1column’ – uses subDir ‘page_layout’
        - *interprets* found page nodes - schedules blocks. page layout instructions (root page template) are interpreted before others
        - original generateLayoutBlocks *interprets*, then runs *generators*
- check View\Page\Config.pageLayout, e.g. “1column”

- adds default body classes - `$fullActionName` , `page-layout-$layout`

- renders block ‘head.additional’, ‘require.js’

- assigns vars ‘requireJs’, ‘headContent’, ‘headAdditional’, ‘htmlAttributes’, ‘headAttributes’, ‘bodyAttributes’, ‘loaderIcon’

- layout.getOutput – already built

- renderPage

    - result.template defined via app/etc/di.xml
    - include Magento_Theme::root.phtml

In your custom controller don’t forget to add entity-specific IDs:

$page->addPageLayoutHandles([‘id’ => $category->getId()]);

- adds handle `$fullActionName_$param_$value` like `catalog_category_view_id_17`

- informs full page cache of *entity specific page* \Magento\Framework\View\EntitySpecificHandlesList::addHandle

    - \Magento\PageCache\Model\Layout\MergePlugin::beforeValidateUpdate - entity-specific layout handles must not contain TTL

      e.g. catalog_category_view_id_15.xml: … will throw error

    - \Magento\PageCache\Observer\ProcessLayoutRenderElement::_wrapEsi - exclude entity-specific handles from ESI URL

View\Layout\Builder.build

- loadLayoutUpdates
- generateLayoutXml
- generateLayoutBlocks

View\Page\Builder.build

- extend generateLayoutBlocks
- readPageLayout

File Collector

- View\File\CollectorInterface
- View\File\Collector\Decorator\ModuleDependency - Decorator that sorts view files according to dependencies between modules they belong to
- View\File\Collector\Decorator\ModuleOutput - Decorator that filters out view files that belong to modules, output of which is prohibited
- View\File\Collector\Base - Source of base files introduced by modules

Collector\Base - Source of base files introduced by modules

- view/base/, view/{$area}/

Collector\Override\Base - Source of view files that explicitly override base files introduced by modules

- theme/{$namespace}_{$module}/

Collector\Theme - Source of view files introduced by a theme

- theme/

Collector\Override\ThemeModular - Source of view files that explicitly override modular files of ancestor themes

- theme/{$namespace}_{$module}/{$themeVendor}/{$themeName}/

Collector\Library - Source of base layout files introduced by modules

- lib_web/
- each inherited theme[]: theme/web/

How do you add new elements to the pages introduced by a given module?

```
<block class="module\block\class" template="new module template" ... />
```

Demonstrate the ability to use layout fallback for customizations and debugging.

- View\Element\Template::getTemplateFile

- View\Element\Template\File\Resolver::getTemplateFileName(template, [module, area]) – just caches in memory

- View\FileSystem::getTemplateFileName – detects module by Module_Name::template.phtml, adds default params

    - View\Asset\Repository::updateDesignParams - adds default missing params [object themeModel, locale]
- View\Design\FileResolution\Fallback\TemplateFile.getFile(area,) – minify html

- View\Design\FileResolution\Fallback\File.getFile – calls resolver type='file'

- View\Design\FileResolution\Fallback\Resolver\Simple.resolve

- View\Design\Fallback\RulePool.getRule - by type: file, locale file, template file

  File fallback rule:

    - when missing 'module_name':
        - <theme_dir>
    - when set 'module_name':
        - <theme_dir>/<module_name>/templates
        - <module_dir>/view//templates
        - <module_dir>/view/base/templates
- View\Design\FileResolution\Fallback\Resolver\Simple::resolveFile(fileRule)

- search file in each directory

How do you identify which exact layout.xml file is processed in a given scope?

How does Magento treat layout XML files with the same names in different modules?

They are merged in module sequence order. Additionally, modules marked as "disable module output" are skipped (though deprecated).

View\File\Collector\Decorator\ModuleDependency - Decorator that sorts view files according to dependencies between modules they belong to

View\File\Collector\Decorator\ModuleOutput - Decorator that filters out view files that belong to modules, output of which is prohibited

View\File\Collector\Base - Source of base files introduced by modules

Identify the differences between admin and frontend scopes.

- customized layout builder and page builders - automatically init messages block

- custom parent \Magento\Backend\Block\AbstractBlock

    - auto inject AuthorizationInterface
- custom parent \Magento\Backend\Block\Template:

    - available some properties - authorization, mathRandom, backendSession, formKey, class nameBuilder
    - event `adminhtml_block_html_before` - fired only when non-cached render. To compare, event `view_block_abstract_to_html_after` fires even when block loaded from cache
- customized "block" layout reader (runs interpret) – reads "acl" block attribute

- customzied “block” layout generator - default block “class=Magento\Backend\Block\Template”

- controller result object with additional methods:

    - setActiveMenu
    - addBreadcrumb
    - addContent(block) - moves to ‘content’
    - addLeft(block)
    - addJs
    - moveBlockToContainer

What differences exist for layout initialization for the admin scope?

Custom layout and page builders - automatically initializes message block from message manager: \Magento\Backend\Model\View\Layout\Builder and \Magento\Backend\Model\View\Page\Builder:

- afterGenerateBlock - called in the end of generateLayoutBlocks, the end of generation - `layout->initMessages()`
- getBlock(‘messages’) or create new one
- addMessages from Message\ManagerInterface
- addStorageType(‘default’)