public class RenderSubContentCacheTransform

Line 49-50

Not useful class Javadoc specification: insufficient and too general.

“This transform cannot be called recursively (at this time).” Totally meaningless because it is talking about a class not a method.

Lines 54-55

public static final String module = RenderSubContentCacheTransform.class.getName();

public static final String [] upSaveKeyNames = {"globalNodeTrail"};

They do not follow the common standard regarding variable naming: for static final variables one has to follow this type of pattern: all uppercase with words separated by an underscore. In these cases it should be:

public static final String MODULE = RenderSubContentCacheTransform.class.getName();

public static final String [] UP\_SAVE\_KEY\_NAMES = {"globalNodeTrail"};

Line 57

@SuppressWarnings("unchecked") -> Maybe there is an unsafe type checking.

Line 58

public Writer getWriter(final Writer out, Map args) {

Map is a raw type. References to generic type Map<K, V> should be parametrized.

Line 73

Boolean nullThruDatesOnly = (strNullThruDatesOnly != null && strNullThruDatesOnly.equalsIgnoreCase("true")) ? Boolean.TRUE :Boolean.FALSE;

This line’s length exceeds the maximum 120 allowed.

Space missing between ‘:’ and Boolean.FALSE.

Literal “subContentId” is duplicated three times. Define a constant instead. Lines affected 74, 110, 220.

Line 75

final boolean directAssocMode = UtilValidate.isNotEmpty(thisSubContentId) ? true : false;

This method returns a boolean already.

Line 78

val = ContentWorker.getCurrentContent(delegator, trail, userLogin, templateRoot, nullThruDatesOnly, contentAssocPredicateId);

This line’s length exceeds the maximum 120 allowed.

Literal “contentId” is duplicated 5 times. Define a constant instead. Lines affected 92, 161, 191, 200, 201.

Line 94

Comment: “This order is taken so that the dataResourceType can be overridden in the transform arguments.” is totally meaningless: what order is referencing to? What is “dataResourceType”? No instance found about it. “transform arguments” isn’t an English well formed sentence: transform is a verb.

Line 98

subDataResourceTypeId = (String) view.get("drDataResourceTypeId");

The object “view” may be nullable, so the try-catch statement might throw a “NullPointException” here.

Moreover, few lines of code before there is an if statement checking if this object is not null and then, at line 98 it is used again without checking his potential null value.

Lines 99-100

Catch block is empty but a meaningless comment is written: “view may be ‘Content’”.

Lines 102/104

TODO task left unsolved.

Lines 115/117

@Override

public void write(char cbuf[], int off, int len) {

}

This method is empty. It should provide a comment explaining why or should throw an UnsupportedOperationException or complete the implementation.

“char cbuf[]” parameter declaration does not follow the standard java variables’ declaration pattern which is: “char[] cbuf”.

Line 135

List<Map<String, ? extends Object>> passedGlobalNodeTrail = UtilGenerics.checkList(templateRoot.get("globalNodeTrail"));

This line’s length exceeds the maximum 120 allowed.

Lines 138/140

if (view != null) {

thisView = view;

} else if (passedGlobalNodeTrail.size() > 0) {

This if-else-if block should have curly brackets after the else statement according to the standard pattern for if statements.

It should be used “!passedGlobalNodeTrail.isEmpty()” instead of “passedGlobalNodeTrail.size() > 0”, indeed “passedGlobalNodeTrail” is an List.

Line 141

Map<String, ? extends Object> map = UtilGenerics.checkMap(passedGlobalNodeTrail.get(passedGlobalNodeTrail.size() - 1));

This line’s length exceeds the maximum 120 allowed.

Lines 152-153

if (locale == null)

locale = Locale.getDefault();

No curly brackets are used according to the standard pattern for if statements.

Line 164

ContentWorker.renderContentAsText(dispatcher, delegator, contentId, out, templateRoot, locale, mimeTypeId, null, null, true);

This line’s length exceeds the maximum 120 allowed.

Literal “contentAssocTypeId” is duplicated 3 times. Define a constant instead. Lines affected: 192, 208, 209.

Literal “mapKey” is duplicated 3 times. Define a constant instead. Lines affected: 193, 212, 213.

Literal “fromDate” is duplicated 3 times. Define a constant instead. Lines affected: 194, 216, 217.

Line 195

if (Debug.infoOn()) Debug.logInfo("in Render(0), view ." + view , module);

No curly brackets are used according to the standard pattern for if statements and two statements are on the same line.

Lines 198/217

All if statements are neither following the standard pattern for if statements nor using curly brackets.

Line 222

if (Debug.infoOn()) Debug.logInfo("in Render(0), contentIdTo ." + contentIdTo , module);

If statement is neither following the standard pattern for if statements nor using curly brackets.

Line 223

String delim = "?";

Declaration out of a beginning block. Moreover, it seems that this variable is used only for a few lines highlighting the fact that it is a temporary variable.

Line 242

delim = "&";

This variable is assigned but never used after.

Line 245

if (Debug.infoOn()) Debug.logInfo("in Render(2), contentIdTo ." + contentIdTo , module);

If statement is neither following the standard pattern for if statements nor using curly brackets.

General considerations about this class

This class manages the rendering of sub contents which have already been generated by back-end modules. It also handles the transformation of the result of back-end processes in a renderable format.

We came up with this consideration because the incomplete JavaDoc about the entire class gave us an essential information: “Freemarker Transform for Content rendering”. Referencing Wikipedia: “FreeMarker is a Java-based Template Engine, originally focusing on dynamic web page generation with MVC software architecture. However, it is a general purpose template engine, with no dependency on servlets or HTTP or HTML, and so it is often used for generating source code, configuration files or e-mails.”.

Instructions at lines from 60 to 69 do use “FreeMarkerWorker” class static methods. The main used method is “getWrappedObject” which seems to retrieve environment information from some back-end modules.

Till line 111, the method “getWriter” has been involved in collecting information useful for rendering.

After that, a new “Writer” object has been returned in a very awful fashion: it uses an anonymous inner class definition (to be precise: it is a sub-class extension of “Writer” class). This anonymous inner class takes a very large number of methods and instructions; therefore, it would have been better to define a class separately.

“openEditWrap” and “closeEditWrap” methods are clearly opening, filling and closing some front-end wrapper which will contain all information involved. Indeed, there are HTML tags at lines 178 and 251 opening and closing a div tag element. In the middle there is some sort of strings generation containing interested information.

For the entire class and the anonymous inner class there are not JavaDocs specified, therefore it is very difficult to read the code.

Methods should have less lines of code in general than the ones in this class.

In “closeEditWrap” method there is a very long if statements sequence. It would have been better to use a design patter instead, in order to overcome this awful way of programming which does not exploit the object oriented programming advantages.