programcreek.com

Java Code Example org.w3c.css.sac.SimpleSelector

11-14 minutes

The following are top voted examples for showing how to use org.w3c.css.sac.SimpleSelector. These examples are extracted from open source projects. You can vote up the examples you like and your votes will be used in our system to product more good examples.

```
/**
 * Creates a sibling selector.
 *
 * @param nodeType       the type of nodes in the
siblings list.
 * @param child          the child selector
 * @param directAdjacent the direct adjacent
selector
 * @return the sibling selector with nodeType
equals to org.w3c.dom.Node.ELEMENT_NODE
 * @throws CSSException If this selector is not
supported.
 */
public SiblingSelector
createDirectAdjacentSelector( final short
nodeType,

final Selector child,

final SimpleSelector directAdjacent )
  throws CSSException {
  return new CSSSilblingSelector( nodeType,
child, directAdjacent );
}

public CSSSilblingSelector( final short nodeType,
                            final Selector
```

```
selector,
                              final SimpleSelector
silblingSelector ) {
  this.nodeType = nodeType;
  this.selector = selector;
  this.silblingSelector = silblingSelector;
}

@Override
public DescendantSelector createChildSelector(
Selector parent, SimpleSelector child )
  throws CSSException
{
  //    return new ChildSelectorImpl( parent,
child );
  String mesg = "Child selectors not supported by
RAP - ignored";
  reader.addProblem( new CSSException( mesg ) );
  return new NullDescendantSelector();
}

@Override
public DescendantSelector
createDescendantSelector( Selector parent,
SimpleSelector descendant )
  throws CSSException
{
  String mesg = "Descendant selectors not
supported by RAP - ignored";
  reader.addProblem( new CSSException( mesg ) );
  return new NullDescendantSelector();
}

@Test
    public void testConditionToXPath() {
        CssParsedSelectorList
cssParsedSelectorList =
CssSelectorParser.parseSelector("span.a.b");
        SelectorList selectorList =
cssParsedSelectorList.getSelectorList();
        ConditionalSelector selector =
```

```
(ConditionalSelector) selectorList.item(0);

//        XPathComponent cs =
conditionalCssSelector.toXPath(cssParsedSelectorList.getArgumentMap(
selector);
        ArgumentMap argumentMap =
cssParsedSelectorList.getArgumentMap();
        SimpleSelector simpleSelector =
selector.getSimpleSelector();
        TagComponent spanTagComponent =
XPathComponentCompilerService.compileSelector(argumentMap,
simpleSelector);

//        XPathComponent compiledCondition =
conditionalCssSelector.conditionToXPath(argumentMap,
selector.getSimpleSelector(),
selector.getCondition());
        CombinatorCondition combinatorCondition =
(CombinatorCondition) selector.getCondition();
        ConditionComponent compiledCondition =
andConditionalCssSelector.conditionToXPath(argumentMap,
simpleSelector, combinatorCondition);

        TagComponent cs =
spanTagComponent.cloneAndCombineTo(compiledCondition);
        assertThat(cs.toXPath(), is(".//*
[self::span and contains(concat(' ', normalize-
space(@class), ' '), ' a ') and contains(concat('
', normalize-space(@class), ' '), ' b ')])"));
        assertThat(cs.toXPathCondition(),
is("local-name() = 'span' and contains(concat('
', normalize-space(@class), ' '), ' a ') and
contains(concat(' ', normalize-space(@class), '
'), ' b ')"));
    }

@Override
public TagComponent toXPath(ArgumentMap
argumentMap, DescendantSelector
descendantSelector) {
        Selector ancestorCSSSelector =
```

```
descendantSelector.getAncestorSelector();
        TagComponent ancestorCompiled =
XPathComponentCompilerService.compileSelector(argumentMap,
ancestorCSSSelector);

        SimpleSelector descendantCSSSelector =
descendantSelector.getSimpleSelector();
        TagComponent childrenCompiled =
XPathComponentCompilerService.compileSelector(argumentMap,
descendantCSSSelector);

        return
DescendantGeneralComponent.combine(ancestorCompiled,
childrenCompiled);
}


/**
 * Returns a representation of the selector.
 */
public String toString() {
    SimpleSelector s = getSimpleSelector();
    if (s.getSelectorType() ==
SAC_PSEUDO_ELEMENT_SELECTOR) {
        return String.valueOf(
getAncestorSelector() ) + s;
    }
    return getAncestorSelector() + " > " + s;
}


/**
* @exception ParseException exception during the
parse
*/
 final public SimpleSelector element_name()
throws ParseException {
Token n;
   switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
   case IDENT:
     n = jj_consume_token(IDENT);
     {if (true) return
selectorFactory.createElementSelector(null,
```

```
convertIdent(n.image));}
     break;
   case ANY:
     jj_consume_token(ANY);
     {if (true) return
selectorFactory.createElementSelector(null,
null);}
     break;
   default:
     jj_la1[80] = jj_gen;
     jj_consume_token(-1);
     throw new ParseException();
   }
   throw new Error("Missing return statement in
function");
 }

/**
 * Creates a conditional selector.
 *
 * @param selector
 *            a selector.
 * @param condition
 *            a condition
 * @return the conditional selector.
 * @exception CSSException
 *                If this selector is not
supported.
 */
public ConditionalSelector
createConditionalSelector(
                SimpleSelector selector,
Condition condition) throws CSSException {
        return new
ConditionalSelectorImpl(selector, condition);
}

final public Selector selector() throws
ParseException {
        Selector sel;
        SimpleSelector pseudoElementSel = null;
```

```
        try {
                sel = simpleSelector(null, ' ');
                label_16: while (true) {
                        if (jj_2_1(2)) {
                                ;
                        } else {
                                break label_16;
                        }
                        jj_consume_token(S);
                        sel = simpleSelector(sel,
' ');
                }
                switch ((jj_ntk == -1) ? jj_ntk()
: jj_ntk) {
                case FIRST_LINE:
                case FIRST_LETTER:
                        pseudoElementSel =
pseudoElement();
                        break;
                default:
                        jj_la1[23] = jj_gen;
                        ;
                }
                if (pseudoElementSel != null) {
                        sel =
getSelectorFactory().createDescendantSelector(sel,
pseudoElementSel);
                }
                handleSelector(sel);
                {
                        return sel;
                }
        } catch (ParseException e) {
                {
                        throw
toCSSParseException("invalidSelector", e);
                }
        }

}
```

```
final public SimpleSelector elementName() throws
ParseException {
        Token t;
        SimpleSelector sel;
        try {
                t = jj_consume_token(IDENT);
                sel =
getSelectorFactory().createElementSelector(null,
unescape(t.image, false));
                if (sel instanceof Locatable) {
                        ((Locatable)
sel).setLocator(getLocator());
                }
                {
                        return sel;
                }
        } catch (ParseException e) {
                {
                        throw
toCSSParseException("invalidElementName", e);
                }
        }

}


/**
 * Parse CSS and create completion informations.
 *
 * @param css CSS
 */
private void processStylesheet(String css) {
  try {
    CSSOMParser parser = new CSSOMParser();
    InputSource is = new InputSource(new
StringReader(css));
    CSSStyleSheet stylesheet =
parser.parseStyleSheet(is);
    CSSRuleList list = stylesheet.getCssRules();
    //                  ArrayList assists = new
ArrayList();
    for (int i = 0; i < list.getLength(); i++) {
```

```
      CSSRule rule = list.item(i);
      if (rule instanceof CSSStyleRule) {
        CSSStyleRule styleRule = (CSSStyleRule)
rule;
        String selector =
styleRule.getSelectorText();
        SelectorList selectors =
parser.parseSelectors(new InputSource(new
StringReader(selector)));
        for (int j = 0; j <
selectors.getLength(); j++) {
          Selector sel = selectors.item(j);
          if (sel instanceof ConditionalSelector)
{
            Condition cond =
((ConditionalSelector) sel).getCondition();
            SimpleSelector simple =
((ConditionalSelector) sel).getSimpleSelector();

            if (simple instanceof
ElementSelector) {
              String tagName = ((ElementSelector)
simple).getLocalName();
              if (tagName == null) {
                tagName = "*";
              }
              else {
                tagName = tagName.toLowerCase();
              }
              if (cond instanceof
AttributeCondition) {
                AttributeCondition attrCond =
(AttributeCondition) cond;
                if (_rules.get(tagName) == null)
{
                  ArrayList<String> classes = new
ArrayList<String>();

//
classes.add(new AssistInfo(attrCond.getValue()));

classes.add(attrCond.getValue());
```

```
                  _rules.put(tagName, classes);
                }
                else {
                  ArrayList<String> classes =
_rules.get(tagName);

//
classes.add(new AssistInfo(attrCond.getValue()));

classes.add(attrCond.getValue());
                }
              }
            }
          }
        }
      }
    }
  }
  catch (Throwable ex) {
    // java.lang.Error: Missing return statement
in function
  }
}


/**
 * Parse CSS and create completion informations.
 *
 * @param css CSS
 */
private void processStylesheet(String css) {
  try {
    CSSOMParser parser = new CSSOMParser();
    InputSource is = new InputSource(new
StringReader(css));
    CSSStyleSheet stylesheet =
parser.parseStyleSheet(is);
    CSSRuleList list = stylesheet.getCssRules();
    //                  ArrayList assists = new
ArrayList();
    for (int i = 0; i < list.getLength(); i++) {
      CSSRule rule = list.item(i);
```

```
      if (rule instanceof CSSStyleRule) {
        CSSStyleRule styleRule = (CSSStyleRule)
rule;
        String selector =
styleRule.getSelectorText();
        SelectorList selectors =
parser.parseSelectors(new InputSource(new
StringReader(selector)));
        for (int j = 0; j <
selectors.getLength(); j++) {
          Selector sel = selectors.item(j);
          if (sel instanceof ConditionalSelector)
{
            Condition cond =
((ConditionalSelector) sel).getCondition();
            SimpleSelector simple =
((ConditionalSelector) sel).getSimpleSelector();

            if (simple instanceof
ElementSelector) {
              String tagName = ((ElementSelector)
simple).getLocalName();
              if (tagName == null) {
                tagName = "*";
              }
              else {
                tagName = tagName.toLowerCase();
              }
              if (cond instanceof
AttributeCondition) {
                AttributeCondition attrCond =
(AttributeCondition) cond;
                if (_rules.get(tagName) == null)
{
                  ArrayList<String> classes = new
ArrayList<String>();

//
classes.add(new AssistInfo(attrCond.getValue()));

classes.add(attrCond.getValue());
                  _rules.put(tagName, classes);
```

```
                }
                else {
                  ArrayList<String> classes =
_rules.get(tagName);

//
classes.add(new AssistInfo(attrCond.getValue()));

classes.add(attrCond.getValue());
                }
              }
            }
          }
        }
      }
    }
  }
  catch (Throwable ex) {
    // java.lang.Error: Missing return statement
in function
  }
}
```