A Really Early Look At An Extension API for Parsoid

S. Subramanya Sastry (Subbu), Parsing Team
EMWCon Spring 2020, April 2020
First things first!

Background context
Wikitext "parsers"

- Wikitext → M/W Core Parser → HTML
- Wikitext ← Parsoid ← HTML
M/W Core Parser: default parser (since 2003)

Clients: Desktop view, Mobile web, iOS app, Action API
Parsoid: alternate parser (since 2012)

**Clients:** VisualEditor, 2017 Wikitext Editor, Flow, Content Translation, Android App, Linter Extension, Kiwix Offline Reader, Google, REST API

Recently ported from JS to PHP, and being integrated into M/W Core
Use Parsoid everywhere

- Legacy parser cannot support Parsoid clients
- Parsoid’s annotated HTML provides more information
- Two parsers not tenable and hamstrings future work
- Parsing Team goal:
  - Make Parsoid the default MediaWiki wikitext engine
  - Initial focus: Start transitioning on the Wikimedia cluster in 15-18 months
Parsoid & Extensions
Today’s focus:
1. Tag extensions
2. Wikitext → HTML
High-level overview

- extension.json based registration
- Differences from current scenario / constraints:
  - Extension output is treated as a DOM fragment, not HTML / wikitext
  - No direct access to Parsoid itself - all interaction through an API object
  - Hooks will be transformation hooks, not event-centric hooks
  - Extensions won’t be able to maintain global ordering state (ex: counters)
- HTML → wikitext transformation optional
  - Parsoid provides default handling
Overview: Parser hooks

- Hooks named after legacy Parser internals won’t work
  - ParserBeforeStrip, ParserAfterStrip, ParserBeforeTidy, ...
- Parsoid’s transformation hooks don’t reference internals
  - sourceToDOM, domToSource, lintHandler, ...
- Current parser hooks will either become unnecessary OR will be mapped to equivalent functionality
Extension Registration

- Will use `extension.json` - details not yet worked out
- Parsoid will recognize extensions implementing the **Extension** interface
- **Extension interface** has one method: `getConfig()`
  - Returns array with config that declares ext tags & implementing classes
  - Implementation classes extend `ExtensionTag`
  - Implementations use `ParsoidExtensionAPI` to handle wikitext
Example config

'name' => 'cite',
'tags' => [
    [
        'name' => 'ref', 'class' => Ref::class, // extends ExtensionTag class
        'options' => [ 'wt2html' => [ ... ], 'html2wt' => [ ... ] ],
    ],
    [ ... one for <references> as well with html2wt options ... ]
],
'domProcessors' => [
    'wt2htmlPostProcessor' => RefProcessor::class, // may change to array
    // more types of DOM processors in the future
],
...
ExtensionTag

Declares transformation hooks with dummy impls

● sourceToDOM($api, string $src, $args): DOMDocument
● domToSource($api, DOMDocument $dom, ...): string
● lintHandler($api, $dom, $defaultLintHandler)
● .....  

Expect sourceToDOM will be implemented at the minimum
ParsoidExtensionAPI

- Categories of API methods:
  - Wikitext → DOM; DOM → wikitext (multiple methods)
  - HTML → DOM; DOM → HTML
  - Methods that deal with extension args
  - get* methods (title, page URI, config objects, etc.)
  - is* methods (query properties)
  - A few others (some transitional and will go away)

- Incomplete and will be updated based on need / feedback
  - Found some gaps based on analyzing extension source
Examples
RawHTML extension

class RawHTML extends ExtensionTag implements Extension {
    public function getConfig(): array {
        return [
            'tags' => [
                ['name' => 'rawhtml', 'class' => self::class],
            ],
        ];
    }
    public function sourceToDOM(ParsoidExtensionAPI $api, $src, $args) {
        return $api->htmlToDOM($src); // returns DOMDocument
    }
}
Observations

- Output DOM behaves like an independent document
  - Markup errors are handled locally and don’t affect top level page
  - Exceptions: HTML5 content model violations (e.g., links in links, fostered content in tables, etc.)

- Output DOM goes through untouched till the final stages
  - Content tunneled through the rest of the parser stages
  - Similar to strip-state mechanism that exts. explicitly manage currently
Examples continued
Extension tag types

- Don’t wrap wikitext: nowiki, pre, syntaxhighlight, rawhtml
  - $output = genDOM($input)
- Thin wrapper over wikitext: ref
  - $output = parseWT($input)
- Process content as more-or-less-wikitext: poem
  - $output = postProcessDOM(parseWT(mangle($input)))
- Content has wikitext snippets that are processed separately: gallery
  - $output = buildDOM(LOOP(parseWT(mangle($frag))))
function sourceToDOM(ParsoidExtensionAPI $api, $txt, $args): DOMDocument {
    $doc = $api->htmlToDOM(''); // Empty doc
    $pre = $doc->createElement('pre');
    $api->sanitizeArgs($pre, $args);
    $txt = decodeWtEntities(trimLeadingNL(stripNoWikis($txt)));
    $pre->appendChild($doc->createTextNode($txt));
    DOMCompat::getBody($doc)->appendChild($pre); // libxml fixes; T215000
    return $doc;
}
function sourceToDOM(ParsoidExtensionAPI $api, $txt, $args): DOMDocument {
    ... some checks to detect ref-in-ref scenarios ...
    return $api->extTagToDOM($args, $txt, [
        'wrapperTag' => 'sup', // DOM is wrapped in <sup> tag
        'parseOpts' => [
            'inlineContext' => true, // No paragraphs, No “indent-pre”
            'extTag' => 'ref', 'extTagOpts' => [ 'allowNestedRef' => ... ],
        ],
    ]);
Huh???

- That handler returned DOM of `<ref>`’s content
  - How does that content migrate to the references section?
  - What happened to numbered links?

- Handled by the `wt2htmlPostProcessor` we saw in the config earlier:
  - `<ref>` handler doesn’t have access to the final DOM yet
  - `<ref>` handler cannot reliably count in the right order to generate links
wt2htmlPostProcessor

- Invoked by Parsoid when the full page is constructed
  - Almost at the end when most (but not all) information is the DOM

- Cite’s processor does this:
  - Walks the tree depth-first in order to count and number refs
  - Migrates the `<sup>` content to the `<references>` section
Mapping legacy parser API and Parsoid API
Mapping: *parse*

Replacements for `parse`, `internalParse`, `startExternalParse`, `recursiveTagParse`, `recursiveTagParseFully`

- `extTagToDOM`: use when tag wraps wikitext (ex: `<ref>`)  
- `wikitextToDOM`: use when tag contains wikitext fragments, but not all of the tag’s content is standard wikitext (ex: `<gallery>`).
Extensions & wikitext

- No control over how much parsing happens
  - `recursiveTagParse`, `internalParse` in current Parser API return “half-parsed HTML” whereas other methods return “fully-parsed HTML”
- `wt2html` options provide some semantic control
  - Cannot turn on/off pipeline stages OR run stages selectively
- Dealing with special wikitext semantics
  - Mangle input as necessary (ex: `<poem>`, `<gallery>`)
Mapping: strip markers

- Strip markers: used to tunnel output through parser stages
- Not needed in Parsoid
  - Extension output always tunneled through \(\Rightarrow\) output doesn’t go through additional processing
  - Stripstate related hooks and methods don’t exist in Parsoid
  - If found necessary, will introduce equivalent functionality later on
Docs, next steps, ...
Learn more: Look at code

- Wikimedia\Parsoid\Ext:
  - `Extension`, `ExtensionTag`, `ParsoidExtensionAPI`
  - `Helpers`: `DOMDataUtils`, `WTUtils`, `Util`, etc.

- Wikimedia\Parsoid\Core:
  - `DOMSourceRange`, various exception classes

- Helper / util classes in other namespaces:
  - `DOMCompat`, `DOMUtils`
Learn more: docs, etc.

- [https://mediawiki.org/wiki/Parsoid/Extension_API](https://mediawiki.org/wiki/Parsoid/Extension_API)
- Look at Parsoid’s implementations for Poem, Pre, Cite, etc.
- Look at Parsoid docs for the Ext/ namespace @ [https://doc.wikimedia.org/Parsoid-PHP/master/](https://doc.wikimedia.org/Parsoid-PHP/master/)
- Parsoid HTML spec @ [https://www.mediawiki.org/wiki/Specs/HTML](https://www.mediawiki.org/wiki/Specs/HTML)
Next steps

- Will solicit feedback from multiple venues
  - EMWCon is the very first venue to get a detailed look at this draft
  - WMF-internal, wikitech-l, mediawiki-l, on-wiki, TechCom & RFC
- Will be refining this for a couple weeks more
  - Ongoing work to narrow Parser.php interface + deprecate lots of things
  - Now is a pretty good time to give us feedback
  - Hoping to have a usable first version (initially for WMF use) in 3 months

Will do our best to not break things unnecessarily
Thanks!
Questions?
Backup slides
Parsoid Sales Pitch

Parsoid deals with wikitext so you don’t have to
<poem>
function sourceToDOM(ParsoidExtensionAPI $api, $txt, $args): DOMDocument {
    $mTxt = $this->mangle($txt); // process :, newlines, ----, nowikis
    return $api->extTagToDOM($args, $mTxt, [
        'wrapperTag' => div, // DOM is wrapped in <div> tag
        'parseOpts' => [ 'extTag' => 'poem' ],
        'processInNewFrame' => true, // mangled $mTxt is different from $txt
        'clearDSROffsets' => true  // mangled $mTxt => DSRoffsets incorrect
    ]);}
}
More ...

- Poem extension treats `<nowiki>` blocks differently
  - Unlike normal wikitext, newlines inside becomes `<br>`s
  - Changing newlines to `<br>` in `mangle(..)` won’t work because `<nowiki>` will escape them!
  - Poem extension registers a DOM processor to deal with this
- Processor finds the `<nowiki>`s and fixes newlines
  - `typeof="mw:Extension/$extName"` attr. present on ext. wrappers
  - Processor looks for matching `typeof` to identify nowiki blocks
  - Replaces newlines inside them with `<br>` tags
$doc = ... ; // construct gallery scaffolding

foreach ($line in $txt) {
    $mLine = makeImageWikitext($line); // [[File:...|..|..]]

    $imgDOM = $api->wikitextToDOM ($mLine, [
        'parseOpts' => [ 'extTag' => 'poem', 'inlineContext' => true ],
        'processInNewFrame' => true,
        'shiftDSRFn' => function($dsr) { return updated $dsr; }
    ]);

    ... Process $imgDOM and add to $doc ...
}
