WIKIMEDIA
HACKATHON

Cool new things in MediaWiki code

Did you know MediaWiki is actually getting better over time? :o

Dependency injection

includes/specialpage/SpecialPageFactory.php,

```
'Listgrants' => [
  'class' => \SpecialListGrants::class,
  'services' => [ 'GrantsLocalization' ]
],
```

includes/specials/SpecialListGrants.php,
as of commit 5c9674df53,
GPL-2.0-or-later

```
public function __construct( GrantsLocalization $grantsLocalization ) {
  parent::__construct( 'Listgrants' );
  $this->grantsLocalization = $grantsLocalization;
}
$this->grantsLocalization->getGrantDescription( $grant, $lang )
```

Dependency injection

```
protected function newSpecialPage() {
  $dataTypeDefinitions = new DataTypeDefinitions( [
     'PT:wikibase-item' => [ 'value-type' => 'wikibase-entityid' ],
  ] );
  return new SpecialListDatatypes( $dataTypeDefinitions );
}
```

repo/tests/phpunit/includes/Specials/SpecialListDatatypesTest.php, as of Wikibase commit e03f5321da, GPL-2.0-or-later

Dependency injection

- declare which services your API module / special page / etc. needs
  - MediaWiki core: `ApiMain::MODULES`, `SpecialPageFactory::CORE_LIST`
  - extensions: `extension.json`
- use them instead of global state (`wfMessage()`, `wfGetDB()`, etc.)
- easily inject stubs or mocks in tests
  - if you can replace *all* global state, you can run your tests as unit tests instead of integration tests (without costly MediaWiki setup/teardown), which is ⚡ much faster ⚡

Service container

```
'BlockUtils' => static function ( MediaWikiServices $services ): BlockUtils {
  return new BlockUtils(
     new ServiceOptions(
        BlockUtils::CONSTRUCTOR_OPTIONS,
        $services->getMainConfig()
     ),
     $services->getUserIdentityLookup(),
     $services->getUserNameUtils()
  );
},
```

includes/ServiceWiring.php,
as of commit 5c9674df53,
GPL-2.0-or-later

Service container

```
'CognateCacheInvalidator' => static function ( MediaWikiServices $services ) {
  return new CacheInvalidator( $services->getJobQueueGroup() );
},
```

src/ServiceWiring.php,

```
public static function getCacheInvalidator(
  ContainerInterface $services = null
): CacheInvalidator {
  return ( $services ?: MediaWikiServices::getInstance() )
    ->get( 'CognateCacheInvalidator' );
}
```

src/CognateServices.php
as of Cognate commit f3468aed38,
GPL-2.0-or-later

Service container

- register your own services
  - MediaWiki core: `ServiceWiring.php` + `MediaWikiServices.php`
  - extensions: Dependency Injection § Service registration in extensions
- use them in your own API modules / special pages / etc., to create other services, or in other extensions
- the service container creates the services on demand and caches / reuses them (so there's only one instance)

Hook handler classes

```
"HookHandlers": {
  "Sidebar": {
     "class": "\\Wikibase\\Client\\Hooks\\SidebarHookHandler",
     "services": [
        "WikibaseClient.LanguageLinkBadgeDisplay",
        "WikibaseClient.NamespaceChecker"
     ]
  }
},
"Hooks": {
  "OutputPageParserOutput": "Sidebar",
  "SidebarBeforeOutput": "Sidebar"
},
```

extension-client.json (abridged),
as of Wikibase commit e03f5321da,
GPL-2.0-or-later

Hook handler classes

```
class SidebarHookHandler implements
  OutputPageParserOutputHook,
  SkinTemplateGetLanguageLinkHook,
  SidebarBeforeOutputHook
{

public function onOutputPageParserOutput( $outputPage, $parserOutput ): void {

public function onSkinTemplateGetLanguageLink(

public function onSidebarBeforeOutput( $skin, &$sidebar ): void {
```

client/includes/Hooks/SidebarHookHandler.php,
as of Wikibase commit e03f5321da,
GPL-2.0-or-later

Hook handler classes

- register hook handlers similarly to API modules and special pages (using an ObjectFactory specification)
- they implement interfaces with non-static methods (but same parameters and return value as old-style static method hook handlers)
- inject services into them with all the usual benefits
- one class can handle more than one hook, where it makes sense

SelectQueryBuilder

```
$expiryToDelete = $dbr->selectFieldValues(
  [ 'watchlist_expiry', 'watchlist' ],
  'we_item',
  $dbr->makeList(
    [ 'wl_id' => null, 'we_expiry' => null ],
    $dbr::LIST_OR
  ),
  __METHOD__,
  [],
  [ 'watchlist' => [ 'LEFT JOIN', 'wl_id = we_item' ] ]
);
```

includes/watcheditem/WatchedItemStore.php,
as of commit bc5ed0e350,
GPL-2.0-or-later

SelectQueryBuilder

```
$expiryToDelete = $dbr->newSelectQueryBuilder()
  ->select( 'we_item' )
  ->from( 'watchlist_expiry' )
  ->leftJoin( 'watchlist', null, 'wl_id = we_item' )
  ->where( $dbr->makeList(
      [ 'wl_id' => null, 'we_expiry' => null ],
      $dbr::LIST_OR
  ) )
  ->caller( __METHOD__ )
  ->fetchFieldValues();
```

includes/watcheditem/WatchedItemStore.php,
as of commit 5c9674df53,
GPL-2.0-or-later

SelectQueryBuilder

- “fluent” interface for building queries
- can build a whole query in one PHP expression, or save the query builder to a variable and construct it over multiple PHP statements (e.g. to modify it based on certain conditions)
  - you can even pass a SelectQueryBuilder instance around between different methods
- **so much more readable** than `select()` with its six parameters
- encourages SQL best practices: a lot of old code was just dumping all the tables into `$tables` and all the join conditions into `$conds` without any `$joinConds` (can cause production issues, see e.g. T246232)

SelectQueryBuilder subclasses

```
$this->newSelectQueryBuilder( $queryFlags ) // UserSelectQueryBuilder
  ->caller( __METHOD__ )
  ->whereUserNames( $normalizedName )
  ->fetchUserIdentity();
```

includes/user/ActorStore.php,

```
$titles = MediaWikiServices::getInstance()
  ->getPageStore()
  ->newSelectQueryBuilder() // PageSelectQueryBuilder
  ->wherePageIds( $lookups )
  ->caller( __METHOD__ )
  ->fetchPageRecords();
```

includes/Cache/TitleLocalCache.php,
as of commit 5c9674df53,
GPL-2.0-or-later

SelectQueryBuilder subclasses

- can be used to make certain querying patterns more accessible
- e.g. custom condition methods or custom result-returning methods
- MediaWiki core has `PageSelectQueryBuilder` and `UserSelectQueryBuilder`, there are a handful more in extensions

UpdateQueryBuilder

```
$dbw->newUpdateQueryBuilder()
  ->update( 'user' )
  ->set( [ 'user_touched' => $dbw->timestamp() ] )
  ->where( [ 'user_id' => $userId ] )
  ->caller( __METHOD__ )->execute();
```

includes/user/UserFactory.php,
as of commit bee13a2a6d,
GPL-2.0-or-later

UpdateQueryBuilder

- basically SelectQueryBuilder but for `update()` instead of `select()`

DeleteQueryBuilder

```
$dbw->newDeleteQueryBuilder()
  ->delete( 'actor' )
  ->where( [ 'actor_name' => $normalizedName ] )
  ->caller( __METHOD__ )->execute();
```

includes/user/ActorStore.php,
as of commit bee13a2a6d,
GPL-2.0-or-later

DeleteQueryBuilder

- basically SelectQueryBuilder but for `delete()` instead of `select()`

UnionQueryBuilder

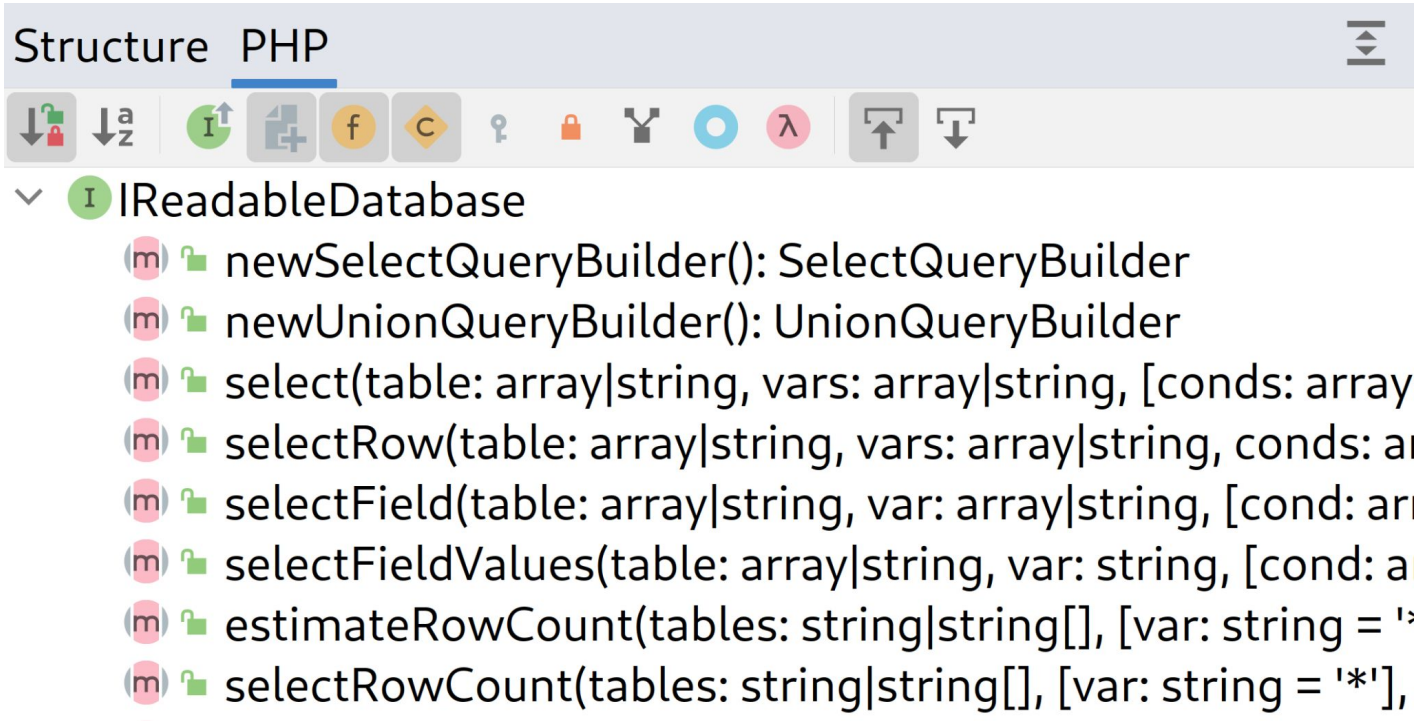
```php
$uqb = $db->newUnionQueryBuilder();
$uqb->add(
  $db->newSelectQueryBuilder()
     ->select( [ 'id' => 'rev_id', 'ts' => 'rev_timestamp' ] )
     ->from( 'revision' )
     ->where( [ 'rev_id' => $revids ] )
);
// ...
$res = $uqb->caller( __METHOD__ )->fetchResultSet();
```

includes/api/ApiQueryRevisions.php,
as of commit bee13a2a6d,
GPL-2.0-or-later

UnionQueryBuilder

- replaces `unionQueries()` + `query()` (there wasn't even a proper wrapper before – `unionQueries()` returned SQL as a string 😱)
- takes one or more SQBs
- very limited interface
- rarely used
- just so you know `^^`

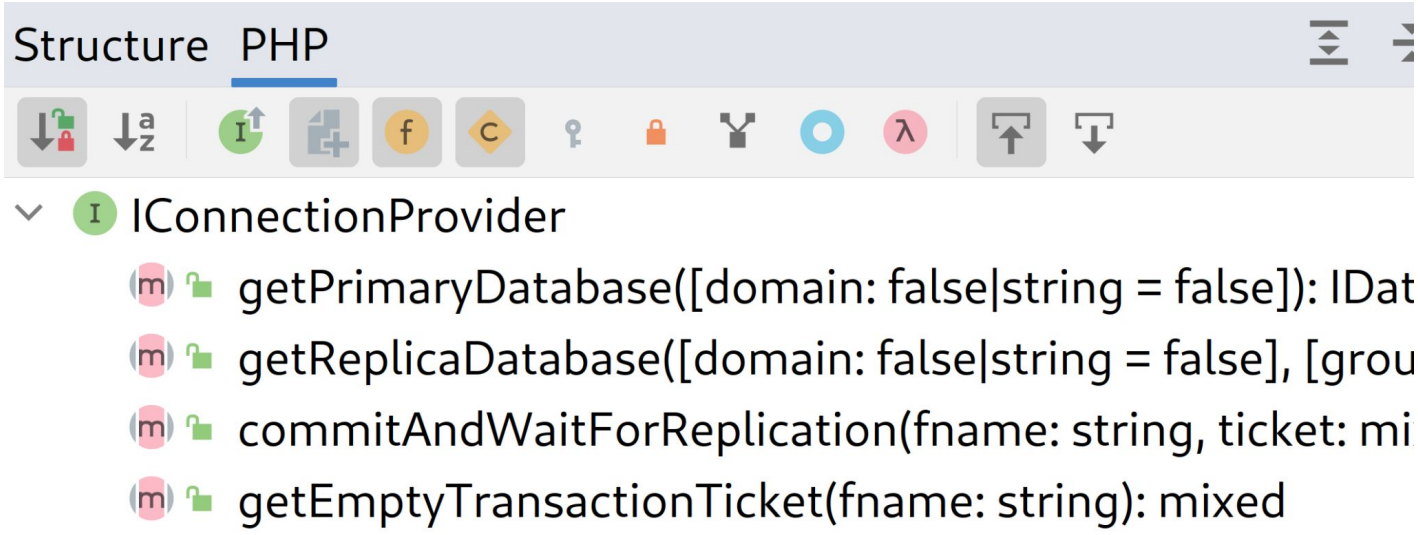
IReadableDatabase



IReadableDatabase

- exposes only the read-only methods of `IDatabase` (and you should primarily use `newSelectQueryBuilder()`)
- useful as a type hint that no write queries are expected or supported

IConnectionProvider



IConnectionProvider

- much simpler interface than `ILBFactory` / `ILoadBalancer`
- easier to use correctly
- implemented by `ILBFactory`, i.e. you inject the same service, just use a different interface for it

NormalizedException

```php
if ( $row->actor_name === '' ) {
  throw new InvalidArgumentException( "Actor name can not be empty for {$userId} and {$actorId}" );
}
```

includes/user/ActorStore.php,
as of commit c40084e898,
GPL-2.0-or-later

Top normalized_message

normalized_message	Count
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 3[redacted]5	48
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 3[redacted]239	27
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 1[redacted]9	3
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 1[redacted]	2
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 5[redacted]8	2
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 2[redacted]7	1
[{reqId}] {exception_url} InvalidArgumentException: Actor name can not be empty for 0 and 2[redacted]0	1

NormalizedException

```
throw new NormalizedException( 'Actor name can not be empty for {userId} and {actorId}', [
  'userId' => $userId,
  'actorId' => $actorId,
] );


// compare
$this->logger->error( 'Actor name can not be empty for {userId} and {actorId}', [
  'userId' => $userId,,
  'actorId' => $actorId,
] );
```

NormalizedException

- library (mw:NormalizedException, wikimedia/normalized-exception)
- you can use `NormalizedException` directly, or mix `NormalizedExceptionTrait` into any of your own exception types
- normalized exceptions are integrated with structured logging and log the normalized message along with the “context”, which can be interpolated into the message
- avoids seeing different copies of the “same” exception in logstash

Codex

- new design system
- dedicated session right afterwards, same room: An introduction to Codex: the design system for Wikimedia (T333611)

Codex design tokens

```
@import 'mediawiki.skin.variables.less';

  & + label::before {
     border-color: @border-color-input-binary;
     transition-property: @transition-property-base;
     transition-duration: @transition-duration-base;
  }

  &:hover {
     cursor: @cursor-base--hover;
  }
```

Codex design tokens

- see mw:Codex § Using Codex design tokens
- use a standardized set of tokens in design and implementation
- skins can change the values of these tokens where needed

default target changed

- ResourceLoader modules now target both desktop and mobile by default
- the target system is being deprecated in general, avoid relying on it

ES6

- IE11 support is dead 🎉🎊🥳 (T178356)
- *way* too many new features to list here
- update your eslint config (eslint-config-wikimedia v0.25.0)

That’s all!

Enjoy writing nicer MediaWiki code :)