Backend, Service, Listener

Die SOLR-Anbindung in VuFind 2.x

David Maus
Herzog August Bibliothek Wolfenbüttel

VuFind Anwendertreffen
16. bis 17. September 2013, Hamburg Harburg

Das bin ich!

- Mitarbeiter der HAB im Digital Humanities Forschungsverbund Niedersachen
- Weiterentwicklung von VuFind als PHP-basiertes Frontend für Daten aus Bibliothekskatalog, Digitaler Bibliothek, Forschungsprojekten

- Email: maus@hab.de
- Github: https://github.com/dmj

VuFind 2.x Paradigmen

- Basis Zend Framework 2

  Tipp: Schauen Sie im Zweifelsfall in den ZF2 Quelltext!

- PHP Namespaces und ClassLoader

  Suchstichwort: PSR-0

- Front Controller Pattern

  Vorher entscheiden, dann ausführen

- ServiceManager als zentrale Registry

  ...nicht so schön, aber besser als globale Variablen

Mängel SOLR-Anbindung VuFind 1

- „Organisch gewachsenes“ System (PHP4, PHP5)

  „works for me“ & monkey patching

- Enge Kopplung von Funktionalität („mit SOLR reden“), Indexschema und Anwendung
- Eine Klasse macht alles, unter anderem *auch* die Anwendung der Suchspezifikation
- Keine sinnvolle Erweiterung möglich

  ...außer „works for me“ & monkey patching

VuFind 2.x SOLR Anbindung

- Funktionale Differenzierung

  Verarbeitungsschritte als austauschbare Objekte modelliert

  Trennung von Funktionalität und Implementation

  - VuFindSearch – Suchfunktionalität („Framework“)
  - VuFind\Search – Spezifische Verwendung der Suchfunktionalität („Application“)

- Parameteraggregation

  Suchparameter aus verschiedenen Quellen aggregiert

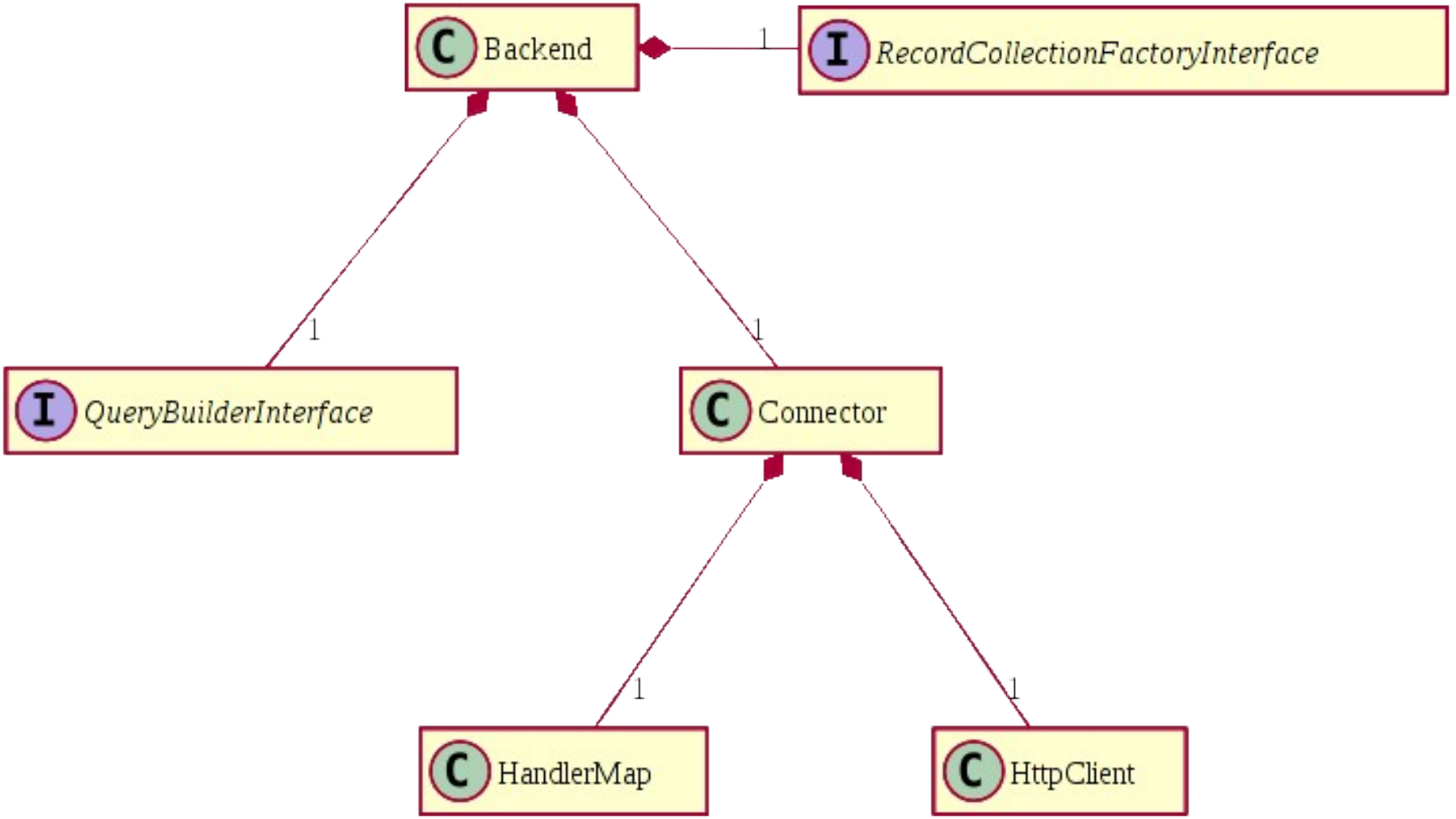
  Entsprechung von SOLR-Funktionalität und VuFind-Klassen

- Ereignisse als Erweiterungspunkte

  Flexibler EventManager des Zend Frameworks

  Beeinflussung von Suchfunktionen unabhängig von VuFind

VuFind 2.x SOLR Backend



VuFindSearch\Backend\Solr\Backend

- Übersetzen der Anfrage in SOLR Request Parameter
- Festlegen des SOLR Response Writer

  wt=json&json.nl=arrarr
- Verwendung eines QueryBuilder um Suche (Feld, Term) in SOLR Suchanfrage zu übersetzen

  auch spellcheck.q und hl.q
- Delegation des Abrufs an Connector
- Deserialisieren der Antwort
- Erstellen der Treffermenge durch separate Klasse

  VuFindSearch\Backend\Solr\Response\Json\RecordCollectionFactory

VuFindSearch\Backend\Solr\Backend::search()

```
public function search(AbstractQuery $query, $offset, $limit, ParamBag $params = null)
{
        $params = $params ?: new ParamBag();
        $this->injectResponseWriter($params);

        $params->set('rows', $limit);
        $params->set('start', $offset);
        $params->mergeWith($this->getQueryBuilder()->build($query));
        $response   = $this->connector->search($params);
        $collection = $this->createRecordCollection($response);
        $this->injectSourceIdentifier($collection);

        return $collection;
}
```

VuFindSearch\Backend\Solr\Connector

- Verbindung zu SOLR Index
- Verwendet HandlerMap um Suchfunktionen (search, retrieve, morelikethis, …) auf SOLR Request Handler zu mappen

  VuFindSearch\Backend\Solr\HandlerMap

  Unterstützung von Appends, Defaults, Invariants a la SOLR
- HTTP Transport
- Fehlerbehandlung: Auslösen einer Exception bei Abruffehlern

  VuFindSearch\Backend\Exception\BackendException

  VuFindSearch\Backend\Exception\RemoteErrorException (HTTP 5xx)

  VuFindSearch\Backend\Exception\RequestErrorException (HTTP 4xx)

VuFindSearch\Response\RecordCollectionFactoryInterface

- Produkt von Suchfunktionen ist eine Menge an Records
  - ! Unterstützung für weitere Antworttypen wünschenswert; vgl. alphabetic browse
- RecordCollectionFactory produziert RecordCollection basierend auf SOLR Response
  - ! Kopplung an Response Writer, nur JSON unterstützt
- Instanziieren der RecordDriver durch RecordCollectionFactory
- gleichzeitig Wrapper um SOLR Response (Facetten)

VuFindSearch\Response\RecordCollectionInterface

```
interface RecordCollectionInterface extends \Countable, \Iterator
{
    public function getTotal();
    public function getFacets();
    public function getRecords();
    public function getOffset();
    public function first();
    public function add(RecordInterface $record);
    public function setSourceIdentifier($identifier);
    public function getSourceIdentifier();
}
```

VuFindSearch\Response\RecordCollectionInterface

```
interface RecordCollectionInterface extends \Countable, \Iterator
{
    public function getTotal();
    public function getFacets();
    public function getRecords();
    public function getOffset();
    public function first();
    public function add(RecordInterface $record);
    public function setSourceIdentifier($identifier);
    public function getSourceIdentifier();
}
```

- Identifikation der RecordCollection und jedes Records durch `source identifier` == Name des Backends

VuFind\Search\BackendManager

- Konfiguration der Backends
- Verwendet einen lokalen („scoped“) ServiceManager um Instanzen der Backendklassen zu generieren
- Adressierung unterschiedlicher Backendinstanzen über anwendungsweit eindeutige Namen

  Verwendung von Aliasen möglich, mitunter hilfreich
- Factory Pattern um Backends zu instanziieren und konfigurieren

module/VuFind/config/module.conf.php

```
'vufind' => array(
    'plugin_managers' => array(
        'search_backend' => array(
            'factories' => array(
                'Solr' => 'VuFind\Search\Factory\SolrDefaultBackendFactory',
                'SolrAuth' => 'VuFind\Search\Factory\SolrAuthBackendFactory',
                'WorldCat' => 'VuFind\Search\Factory\WorldCatBackendFactory',
            ),
            'aliases' => array(
                // Allow Solr core names to be used as aliases for services:
                'authority' => 'SolrAuth',
                'biblio' => 'Solr',
                // Legacy:
                'VuFind' => 'Solr',
            )
        ),
    )
)
```

VuFindSearch\Service

- zentraler Integrationspunkt innerhalb der Webanwendung
- Backendinstanzen werden über ihren Namen angesprochen
- Service verwendet Ereignissystem um Namen zu Instanz aufzulösen
- Service löst Ereignisse vor (pre) und nach (post) dem Ausführen der Suchfunktion aus
- Service löst Ereignis im Fehlerfall aus (error)
- vordefinierte Objekte (Listener) können auf Ereignisse reagieren

Ereignis: VuFindSearch\Service::EVENT_RESOLVE

- „short-circuited“: Rückkehr zum Service sobald ein Listener eine Backendinstanz liefert

  Wer zuerst kommt mahlt zuerst

- Ereignis enthält Informationen über die Suchfunktion („context“) und alle formalen Parameter
- Internes Caching der aufgelösten Namen

  nur per Request!

VuFindSearch\Service::resolve()

```
protected function resolve($backend, $args)
{
    $response = $this->getEventManager()->trigger(
        self::EVENT_RESOLVE, $this, $args,
        function ($o) {
            return ($o instanceOf BackendInterface);
        }
    );
    if (!$response->stopped()) {
        throw new Exception\RuntimeException(
            sprintf(
                'Unable to resolve backend: %s, %s', $args['context'],
                $args['backend']
            )
        );
    }
    return $response->last()
}
```

Ereignis: VuFindSearch\Service::EVENT_PRE/EVENT_POST

- EVENT_PRE enthält das aufgelöste Backend (=Instanz), Name der Suchfunktion und die formalen Parameter
- EVENT_POST enthält zusätzlich die Antwort (RecordCollection)
- Antwort und sekundäre Suchparameter (ParamBag) können durch Listener manipuliert werden

Ereignis: VuFindSearch\Service::EVENT_ERROR

- Exceptions der Backends werden aufgefangen
- EVENT_ERROR enthält die Exception
- Exception wird erneut ausgelöst (rethrow)
- Anwendungsfall: Umgang mit verschiedenen SOLR-Versionen

  Response body eines Fehlers unterscheidet sich zwischen SOLR 3.x und 4.x

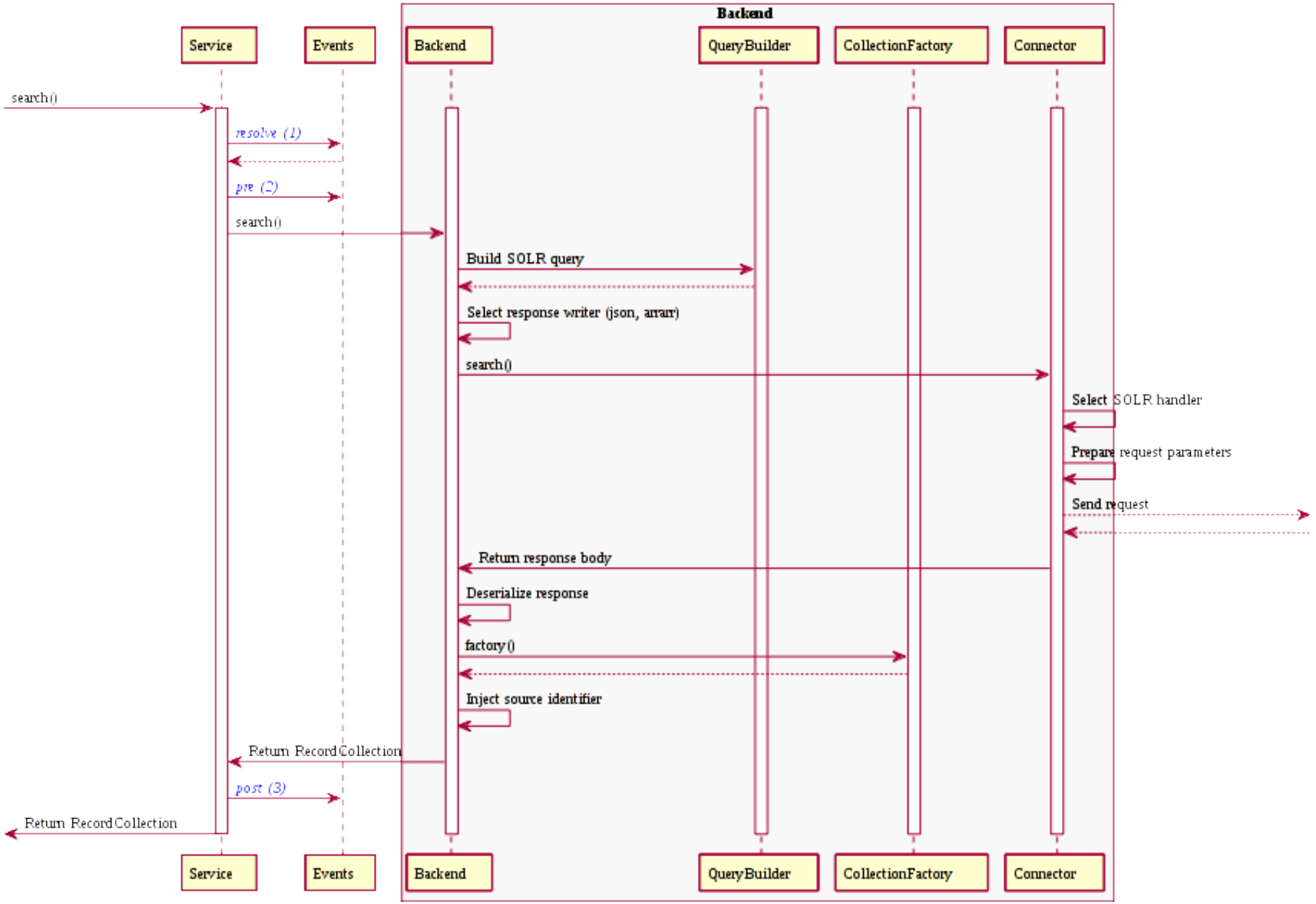
  Tagging der Exceptionklasse, Auswertung der Tags (Lucene Syntax Error et al.)

VuFindSearch\Service::retrieveBatch()

[siehe Quelltext]

VuFind\Search\Solr\InjectHighlightingListener

[siehe Quelltext]


Backend
Service
Events
Backend
QueryBuilder
CollectionFactory
Connector
search()
resolve (1)
pre (2)
search()
Build SOLR query
Select response writer (json, arrarr)
search()
Select SOLR handler
Prepare request parameters
Send request
Return response body
Deserialize response
factory()
Inject source identifier
Return RecordCollection
post (3)
Return RecordCollection
Service
Events
Backend
QueryBuilder
CollectionFactory
Connector

Ausblick VuFind 2.2 (Sommer 2014)

- Parameteraggregation auf den höheren Ebenen

  Options – Params – Results (aka SearchObject)

- Unterstützung beliebiger Facettierungen?

  Ihr Feedback ist gefragt!

- Vereinheitlichung weiterer Antworttypen

  Terms/AlphabeticBrowse => Systematischer Zugang ftw!