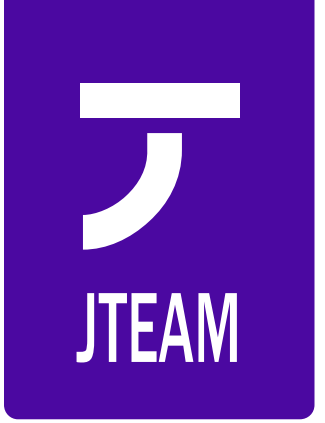


Integrating Solr in JEE Applications

Chris Male
Apache Solr / Lucene Committer
chris@jteam.nl / chrism@apache.org

Agenda

JTEAM

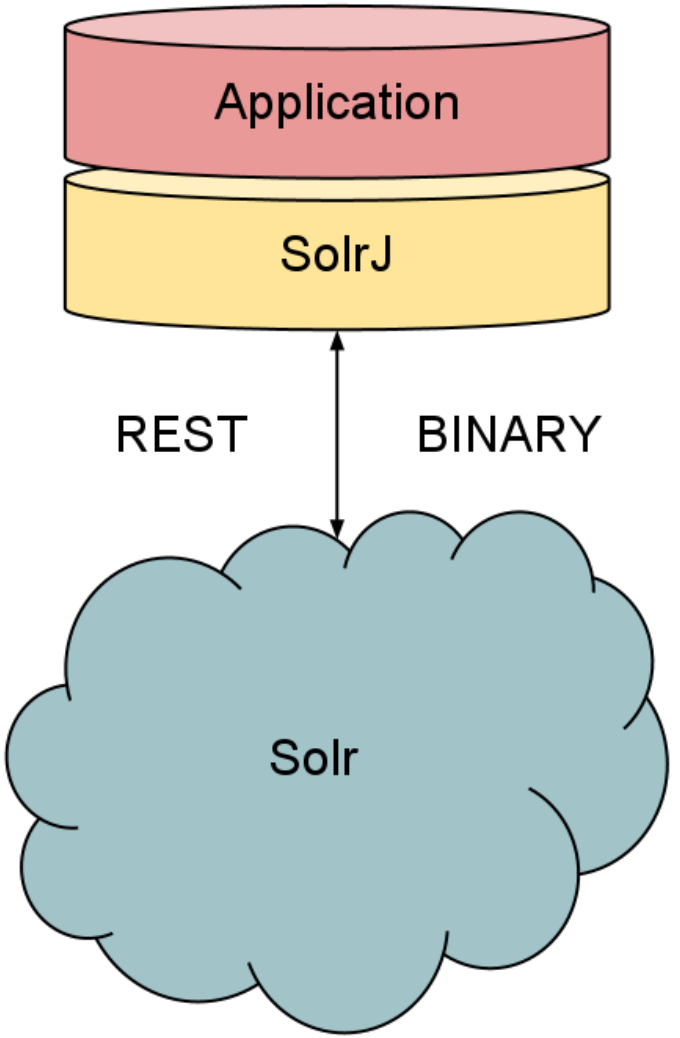
- SolrJ in Action
- Lessons from Databases
- Typed Schemas
- Typed Queries / Responses
- Document Binding

JEE Application Needs

- Clean interfaces
- Layering and abstractions
  - Dependency Injection
- Readable, Extensible, Maintainable
- Testable
- Limited client API exposure

SolrJ in Action

- Official Java client for Solr
- Uses Binary protocol
- HTTP vs Embedded
- General, all purpose API
- Some ORM-like binding
- Can handle custom requests, response
  - Roll your own if really custom



SolrJ in Action – Example

JTEAM

```
<bean id="solrServer"
class="org.apache.solr.client.solrj.impl.CommonsHttpSolrServer">

    <constructor-arg index="0" value="${solr.server}"/>

</bean>


<bean id="searchService"
class="de.berlinbuzzwords.example.DefaultSearchService">

    <property name="solrServer" ref="solrServer"/>

</bean>
```

SolrJ in Action – Example

JTEAM

```
SolrInputDocument document = new SolrInputDocument();
document.addField(“first_name”, “Chris”);
document.addField(“last_name”, “Male”);
document.addField(“age”, 26);
solrServer.add(document);
solrServer.commit();

SolrQuery query = new SolrQuery(“last_name:Male”);
QueryResponse response = solrServer.query(query);
assertEquals(1, response.getResults().numFound);
```

SolrJ in Action – Analysis

JTEAM

- Efficient, easy to use
- Standard Solutions are usually best
- Untyped, String crazy
- Doesn't speak domain's language
  - Searching for people vs Searching for cars
- Implementation detail
- Room for improvement

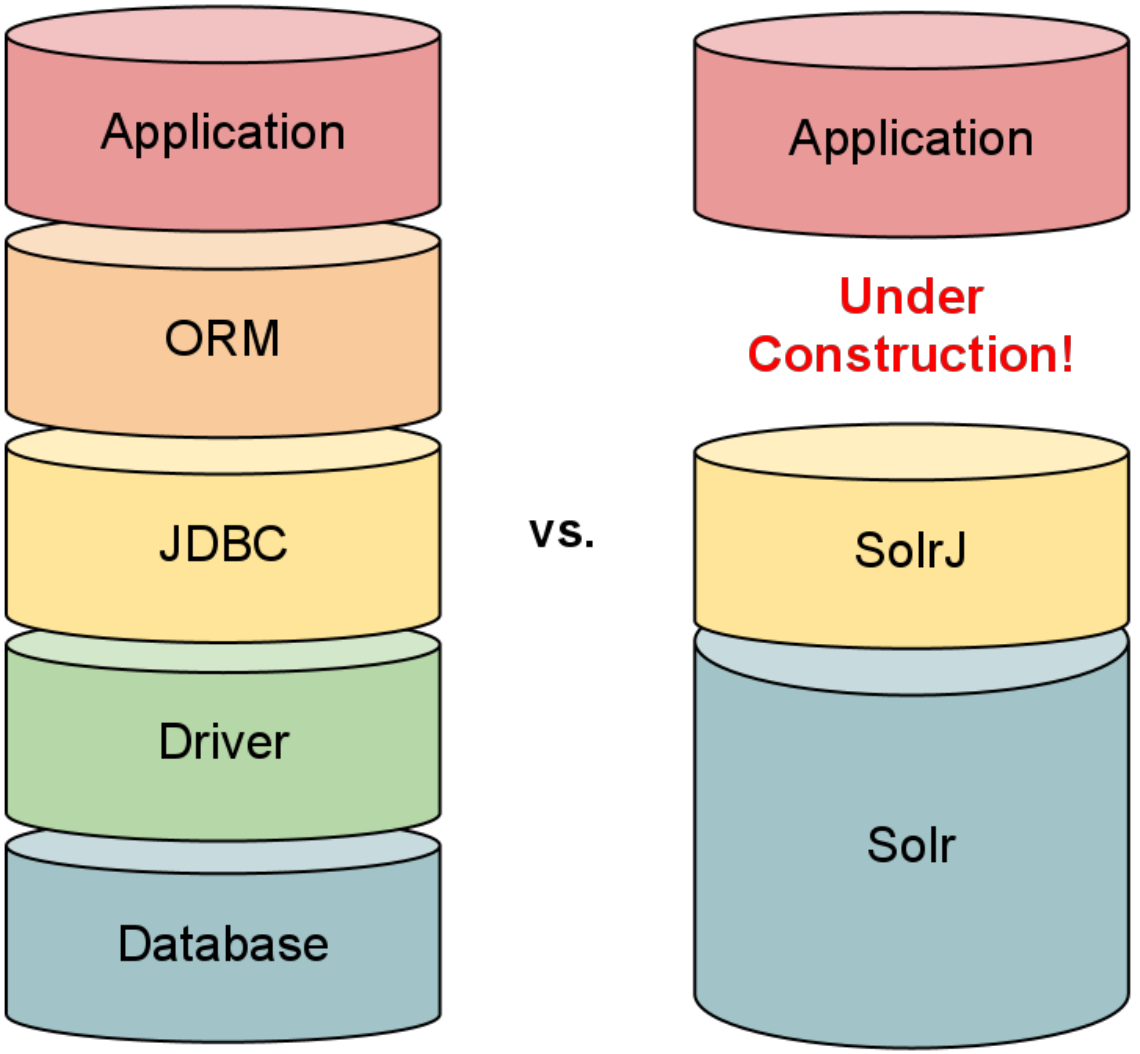
Lessons from Databases

JTEAM

- Database Drivers: Efficient protocols – like Solr's Binary Protocol
- JDBC
  - Standard API
  - General, easy to use, String crazy – Like SolrJ
- DAOs, Repositories
  - Domain model in driver seat
  - Typed queries, results
  - Refactor friendly
  - Less error-prone
  - Speaks domain language
  - Testable

Lessons from Databases

JTEAM



Typed Schemas

JTEAM

- Introduce Field construct
- Encode Solr Schema as Enum of Fields
- Field names stored as String once
- Application uses Enum
- Refactor friendly
- Uses domain language, not Solr language
  - FIRST_NAME vs *first_name_text*

Typed Schemas – Example

JTEAM

```
public interface Field {
    String getName();
    // other properties
}
public enum PeopleSchema implements Field {
    FIRST_NAME("first_name"),
    LAST_NAME("last_name"),
    AGE("age");
    String name;
    PeopleSchema(String name) { this.name = name; }
    String getName() { return name; }
}
```

Typed Queries

JTEAM

- Reduce SolrJ usage
- Domain specific Queries
  - *PeopleQuery* vs *CarQuery* vs *EventQuery*
- Use domain language
  - *withFirstName*, *withColour*, *startingAt*
- Use Typed Schema
- Convert to SolrQuery at execution time
  - execute(SolrServer)
- Hierarchy for common functionality
- Field based `Query`, `FilterQuery`, `Facet` etc

Typed Queries – Example

JTEAM

```
public class PeopleQuery extends AbstractSearchQuery {
    public PeopleQuery(String firstName) {
        setQuery(PeopleSchema.FIRST_NAME, firstName);
    }
    public PeopleQuery withLastName(String lastName) {
        addFilterQuery(PeopleSchema.LAST_NAME, lastName);
        return this;
    }
    public PeopleQuery betweenAge(int lowerAge, int upperAge) {
        addRangeQuery(PeopleSchema.AGE, lowerAge, upperAge);
        return this;
    }
    public ??? execute(SolrServer solrServer) {
        QueryResponse queryResponse = solrServer.execute(convert());
        return new ???(queryResponse);
    }}
```

Typed Queries – Example

JTEAM

```
PeopleQuery peopleQuery = new PeopleQuery(“Chris”)
                    .withLastName(“Male”)
                    .betweenAge(25, 27);
??? response = peopleQuery.execute(solrServer);
```

Typed Responses

JTEAM

- Responses from Queries should also be typed
  - `QueryResponse` is bad!
- Features based on Field
  - Facets per Field, Highlighting per Field
- `Page` of results
  - List of Results
  - Current page, page size, total results
- Parsed after execution
- Need to map field names back to Enum

```
public static Field findField(String fieldName) { … };
```

Typed Responses – Example

JTEAM

```
public class PeopleResult extends AbstractSearchResult<People> {
    public PeopleResult(QueryResponse response) {
        parse(response);
    }
    public FacetResult getAgeFacet() {
        return getFacetResult(PeopleSchema.AGE);
    }
    public String getSuggestedName() {
        return suggestion;
    }
    public Page<People> getResults() { // borrowed from superclass
        return page;
    }
}
```

Document Binding

JTEAM

- Marshalling / unmarshalling Domain model
- SolrJ provides simple support:
  - `DocumentObjectBinder`
  - `@Field("fieldName")`
  - `addBean(Object bean), List<T> getBeans(Class<T> beanType)`
- Cannot use Typed Schema – Java limitation
- String field names used again
- Cannot handle embedded objects
- Needs ALOT of work

Document Binding – Example

JTEAM

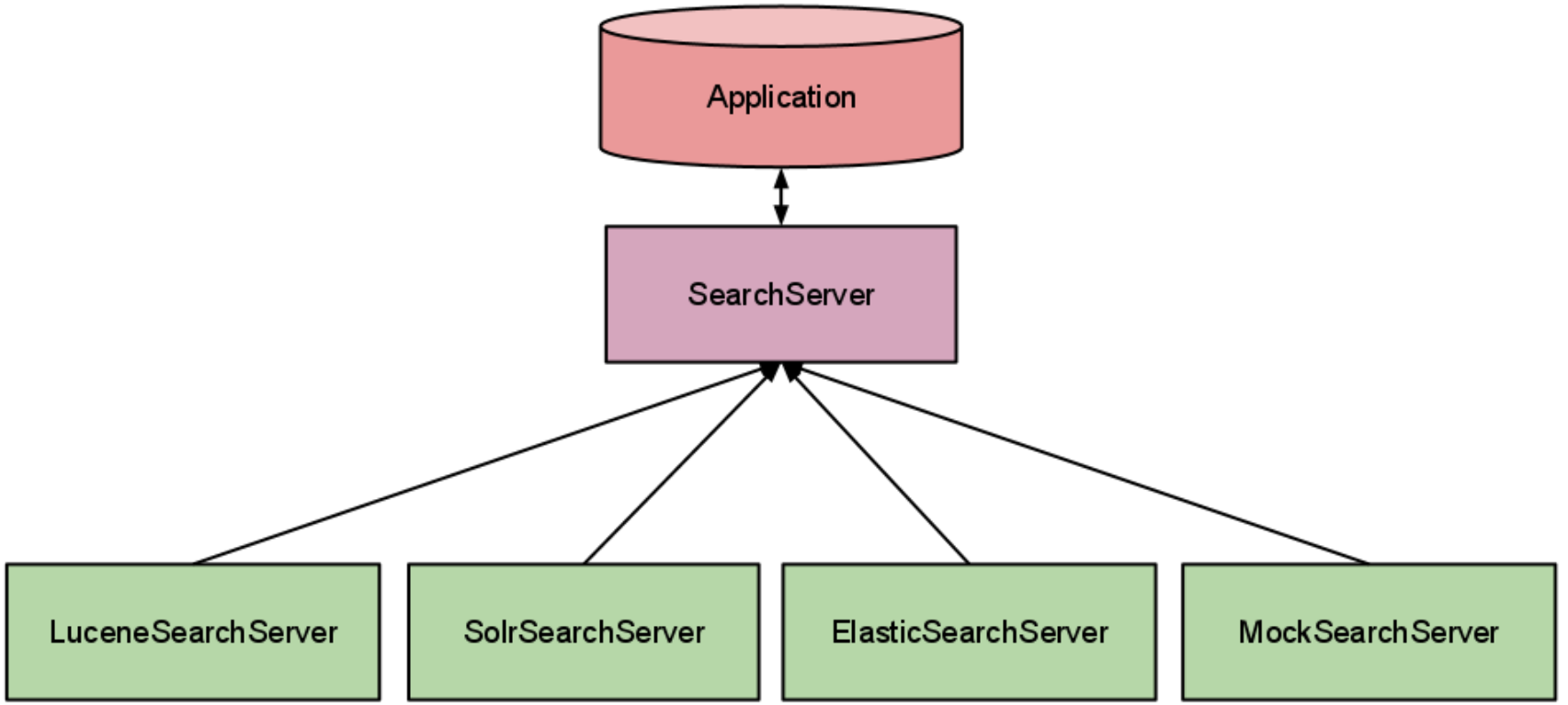
```
public class People {
    @Field(“first_name”)
    private String firstName;

    @Field(“last_name”)
    private String lastName;

    @Field(“age”)
    private int age;
    // getters & setters
}
```

Not Just Solr

- Typed searching not just for Solr
- Lucene, Elastic Search, NoSQL even
- Facets, highlighting, spell checking are common
- Reduce client library exposure
- Abstract `SearchServer` wrapper



Conclusion

- SolrJ standard client library
- Database experiences should be heeded
- Typed Queries & Results improve readbility, maintainability
- Document binding needs improvements
- Applicable to all search solutions

- **chris@jteam.nl**
- **http://www.jteam.nl**