A Brief History

- Complex Orchestration
- Complex Dependency from Service Explosion
- Incompatibilities
- Slow Rollout

Monolithic/Layered
Paradigm Shift

- Fine-grained Service Boundary
- Backward-compatible service evolution
- Standard Service Access Pattern
- Continuous Delivery

Microservices
Challenges

- Standard way to describe service in service-oriented architecture.
- Standard access pattern by multi-language diverse clients.
- Asynchronous and scalable to handle heavy service load.
- Fast and easy experience even for non-REST expert developers.
- Support long-term service evolution and growth.
Monolithic → Data Model Centric Microservices
Rest.li is an open source REST framework for building robust, scalable RESTful architectures using asynchronous, non-blocking IO. It fills a niche for applying RESTful principles at scale with an end-to-end developer workflow that promotes clean REST practices.
Clean REST Practice
Clean REST Practice

- End-to-End Developer Workflow
  - Consistent data modeling
  - Uniform interface design
  - Type-safe client bindings
- Online API Hub
  - API documentation
  - API service catalog
- Compatibility Checker Enabled API Evolution
Developer Flow

Data Schema
Consistent Data Modeling

- Data schema in PDSC (Pegasus Data Schema)
- PDSC is inspired by Apache Avro specification.
- JSON serialization format
- Rich type support:
  - **Named**: Record, Enum, Fixed, Typeref
  - **Unnamed**: Union, Array, Map
  - **Custom type**
- Custom validator and coercer
Data Schema Example

```json
{
    "type": "record",
    "name": "Photo",
    "namespace": "com.linkedin.restli.example",
    "doc": "A photo for rest.li",
    "fields": [
        {
            "name": "id",
            "type": "long"
        },
        {
            "name": "title",
            "type": "string"
        },
        {
            "name": "format",
            "type": "PhotoFormats"
        },
        {
            "name": "exif",
            "doc": "Exchangeable image file format",
            "type": "EXIF"
        }
    ]
}
```
Developer Flow

1. Data Schema
2. Generated Record Template
3. Resource Class

Resource Class

Resource Class

Resource Class
Uniform Interface Design

- Standardized RESTful resource pattern
- Annotated resource identifier
- Uniform service interface
  - CRUD
  - Finder
  - Action
  - BATCH
- REST specification IDL (Interface Description Language)
## Rest.li Resource Patterns

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Semantics</th>
<th>Interface</th>
<th>Base Template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection (@RestLiCollection)</td>
<td>Key/value map of entities</td>
<td>CollectionResource KeyValueResource</td>
<td>CollectionResourceTemplate</td>
</tr>
<tr>
<td>Simple (@RestLiSimple)</td>
<td>Singleton in scope</td>
<td>SimpleResource</td>
<td>SimpleResourceTemplate</td>
</tr>
<tr>
<td>Association (@RestLiAssociation)</td>
<td>Relationships between entities</td>
<td>AssociationResource</td>
<td>AssociationResourceTemplate</td>
</tr>
<tr>
<td>Sub-Resource</td>
<td>Child resources</td>
<td>All Above</td>
<td>All Above</td>
</tr>
<tr>
<td>Action (@RestLiActions)</td>
<td>Flexible for non-standard behavior</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Rest.li Resource Method (Basic)

<table>
<thead>
<tr>
<th>Rest.li Request</th>
<th>HTTP Method</th>
<th>URI Example</th>
<th>Entity Body (V - Data Model)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>POST</td>
<td>/statuses</td>
<td>V</td>
</tr>
<tr>
<td>GET</td>
<td>GET</td>
<td>/statuses/&lt;id&gt;</td>
<td></td>
</tr>
<tr>
<td>GET_ALL</td>
<td>GET</td>
<td>/statuses</td>
<td></td>
</tr>
<tr>
<td>UPDATE</td>
<td>PUT</td>
<td>/statuses/&lt;id&gt;</td>
<td>V</td>
</tr>
<tr>
<td>PARTIAL_UPDATE</td>
<td>POST</td>
<td>/statuses/&lt;id&gt;</td>
<td>Patch of V</td>
</tr>
<tr>
<td>DELETE</td>
<td>DELETE</td>
<td>/statuses/&lt;id&gt;</td>
<td></td>
</tr>
<tr>
<td>Finder</td>
<td>GET</td>
<td>/statuses?q=SEARCH&amp;keywords=linkedin</td>
<td></td>
</tr>
<tr>
<td>Action</td>
<td>POST</td>
<td>/statuses?action=purge</td>
<td>Dynamic Record</td>
</tr>
</tbody>
</table>

Rest.li: RESTful Service Architecture at Scale
## Rest.li Resource Method (Batch)

<table>
<thead>
<tr>
<th>Rest.li Request</th>
<th>HTTP Method</th>
<th>URI Example</th>
<th>Entity Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>BATCH_CREATE</td>
<td>POST</td>
<td>/statuses</td>
<td>{“elements”: [...]}</td>
</tr>
<tr>
<td>BATCH_GET</td>
<td>GET</td>
<td>/statuses? ids=List(1,2,3)</td>
<td></td>
</tr>
<tr>
<td>BATCH_UPDATE</td>
<td>PUT</td>
<td>/statuses? ids=List(1,2,3)</td>
<td>{“entities”: {“1”: {...}, “2”: {...}}</td>
</tr>
<tr>
<td>BATCH_PARTIAL_UPDATE</td>
<td>POST</td>
<td>/statuses? ids=List(1,2,3)</td>
<td>{“entities”: {“1”: {“patch”: {...}}, “2”: {“patch”: {...}}}</td>
</tr>
<tr>
<td>BATCH_DELETE</td>
<td>DELETE</td>
<td>/statuses? ids=List(1,2,3)</td>
<td></td>
</tr>
</tbody>
</table>
What Server Code I Need to Write

```java
CollectionResourceTemplate

// create(V): CreateResponse → CollectionResource
// batchGet(Set<K>): Map<K, V> → CollectionResource
// batchUpdate(BatchUpdateRequest<K, V>): BatchUpdateResult<K, V> → CollectionResource
// batchCreate(BatchCreateRequest<K, V>): BatchCreateResult<K, V> → CollectionResource
// batchDelete(BatchDeleteRequest<K, V>): BatchUpdateResult<K, V> → CollectionResource
// get(K): V → CollectionResource
// update(K, V): UpdateResponse → CollectionResource
// update(K, PatchRequest<V>): UpdateResponse → CollectionResource
// delete(K): UpdateResponse → CollectionResource
// getAll(PagingContext): List<V> → CollectionResource
```
Developer Flow

1. REST IDL
2. Data Schema
3. Resource Class
4. Generated

Record Template
```json
{
  "name": "photos",
  "namespace": "com.linkedin.restli.example.photos",
  "path": "/photos",
  "scheme": "com.linkedin.restli.example.Photo",
  "doc": "generated from: com.linkedin.restli.example.impl.PhotoResource",
  "collection": {
    "identifier": {
      "name": "photosId",
      "type": "Long"
    },
    "supports": ["batch_get", "create", "delete", "get", "partial_update", "update"],
    "methods": [{
      "method": "get"
    }, {
      "method": "create"
    }, {
      "method": "delete"
    }, {
      "method": "update"
    }, {
      "method": "partial_update"
    }, {
      "method": "batch_get"
    }],
  },
  "finders": [{
    "name": "titleAndOrFormat",
    "parameters": [{
      "name": "title",
      "type": "string",
      "optional": true
    }, {
      "name": "format",
      "type": "com.linkedin.restli.example.PhotoFormats",
      "optional": true
    }],
    "pagingSupported": true
  }],
  "actions": [{
    "name": "purge",
    "returns": "int"
  }],
  "entity": {
    "path": "/photos/{photosId}"
  }
}
```
What Client Code I Need to Write

```
POST /photos -H "Content-Type: application/json"
-d '{"title":"NewPhoto", "format": "PNG", "exif": {"location":...}}'
```

```java
Photo newPhoto = new Photo()
  .setTitle("NewPhoto")
  .setFormat(PhotoFormats.PNG)
  .setExif(new Exif().setLocation(newLocation));

Request<?> createReq = new PhotosRequestBuilder().create().input(newPhoto).build();
```

**Strong-typed data model (auto-generated)**

**Type-safe request builders (auto-generated)**
Developer Flow

Server

1. REST IDL
2. Data Schema
3. Record Template
4. Generated

Client

5. Resource Class
6. Type-Safe RequestBuilders
7. Generated

Human Readable Documentation
Rest.li API Hub

- Lucene Powered full-text search
- Data model and interface documentation

- Sample request generation
- Live interaction with real REST endpoint

Carnegie Mellon University
Software Engineering Institute
Backward Compatibility

- **Build-time Compatibility Checker**
  - **equivalent**: No changes to Resources or Data model.
  - **backwards**: Changes considered backwards.
  - **ignore**: Log the changes but allow changes to pass.
  - **off**: The compatibility checker will not be run at all.
Scale and Open
Rest.li Server Stack

**Rest.li Data Layer and RESTful Operations**

**D2 Dynamic Discovery and Load Balancing**

**R2 Request/Response and Network Communication**

- Inversion-of-Control
- Fault tolerance
- Async and Non-blocking
Request Response (R2)

Message Communication Abstraction over HTTP

Non-Streaming

Streaming
Dynamic Discovery (D2)

- Apache Zookeeper (*Announce and Discover*)
- Horizontal Scaling
  - Services, Servers (ephemeral nodes) and Clusters
- Client side load balancing
- Fault tolerance
- Backup request (*tail-latency*)
Rest.li Client

Implement and Interpret Rest.li Protocol
(Pluggable Transport Client)
Rest.li Client/Server Data Flow

Application Code

Rest.li Framework

Dynamic Discovery

Request/Response

HTTP

Request

Rest.li

Record Template

RequestBuilder

Record Template

RequestBuilder

Resource

Resource

Resource Invocation

D2

D2

Zookeeper

R2

R2

discover

announce

HTTP Request

HTTP Response/JSON
Open and Extensible

- Filter Chain pipeline
- Polyglot transport client
- Custom Codec (JSON, PSON, XML, etc)
Rich Eco-Systems

ParSeq

Multi-Language

C#  Android
Java  Java
Objective-C  Swift
Python
Describe WHAT data you want, let Deco worry about HOW to fetch them.
Deco Concept

Min Chen
Software Engineer at LinkedIn

This idea is innovative, may be generalized to other CS algorithms whose performance highly depends on the distribution of real data set.

The Case for Learned Index Structures

/arxiv-vanity.com

1 Like

Like Comment Share

/ profiles
/ companies
/ posts
Deco Concept

deco://posts/123?projection=(title, content, author~(firstName, lastName, company~(companyName)))

- Projection
- Decoration
- Resolution

Normalized Domain Modeling
ParSeq

Asynchronous Java made easier
Asynchronous Challenges

- Callback Hell
- Thread Safety
- Indeterminism
- Debugging
- How to achieve all-async?
### ParSeq Concept

<table>
<thead>
<tr>
<th>Promise(&lt;T&gt;)</th>
<th>Represents state of an asynchronous operation, also a container for an asynchronous operation result.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task(&lt;T&gt;)</td>
<td>Main abstraction unit representing an asynchronous operation.</td>
</tr>
<tr>
<td>Task Callable</td>
<td>Main ingredient of Task(&lt;T&gt;). It starts an asynchronous operation and returns a Promise(&lt;T&gt;) representing the operation state.</td>
</tr>
<tr>
<td>Engine</td>
<td>Harness to run a Task(&lt;T&gt;).</td>
</tr>
</tbody>
</table>
ParSeq Programming Model

- Declarative and Functional (What not How)
- Task composition, chaining, transformation and error handling
  - Task.async (task creation)
  - Task.par (parallel composition)
  - task.andThen (sequential composition)
  - task.map and task.flatMap (transformation)
  - task.recover and task.withTimeout (error handling)
- Ends up with one Task (execution plan) submitted to ParSeq engine to execute
ParSeq Execution Model

- All task Callables are executed sequentially, such that Callable of a previous Task in a sequence "happens before" (in Java Memory Model sense) Callable of a next Task.

```java
Task<Response<T>> task =
    Task.<Response<T>>async(name, () -> sendRequest(request, requestContext));
```

- No need to be concerned about:
  - Synchronized, locks, volatile, atomic, etc
  - Immutability
  - 3rd party libraries
  - Thinking about Java Memory Model
  - Thread safety does not mean atomicity
Seamless Integration with Rest.li

- Rest.li supports resource method returning Task.

```java
@RestMethod.Get
public Task<Greeting> get(final Long id) {
    // rest.li resource implementation
}
```

- ParSeqRestClient provides API to create Task from a rest.li request

```java
public <T> Task<Response<T>> createTask(final Request<T> request, final RequestContext requestContext) {
```

ParSeq Trace Visualization

seq (@0.000, 0.039, +147.602)

fetch[url=http://www.bing.com]
  @0.141  0.160  +12.067

fetch[url=http://www.google.com]
  @12.448  0.150  +135.108

Power Tool for Debugging
Multi-Language Support

- C#
- Java
- Objective-C
- Swift
- Python
## Comparison Matrix

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rest.li</th>
<th>Jersey</th>
<th>DropWizard</th>
<th>Olingo (Odata)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Schema</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Server Coding</td>
<td>😛</td>
<td>😛</td>
<td>😛</td>
<td>😛</td>
</tr>
<tr>
<td>URI Pattern</td>
<td>Uniform</td>
<td>Non-Uniform</td>
<td>Non-Uniform</td>
<td>Non-Uniform</td>
</tr>
<tr>
<td>Batch Support</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Service Metadata</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Dynamic Discovery/Load balancing</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Documentation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Service Catalog</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Graph Query</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Future of Rest.li

- Object Streaming in Rest.li Layer
- Unstructured data support
- Multi-language support enhancement
  - Go
  - Cross-language test suites
- Open source Deco
- ParSeq
  - Collection API
  - ParSeq Trace Analytics
Useful Links

- Rest.li ([https://github.com/linkedin/rest.li](https://github.com/linkedin/rest.li))
- Rest.li API Hub ([https://github.com/linkedin/restli-api-hub](https://github.com/linkedin/restli-api-hub))
- ParSeq ([https://github.com/linkedin/parseq](https://github.com/linkedin/parseq))
Data Schema (Custom)

```json
{
  "type": "typedef",
  "name": "IPAddress",
  "namespace": "com.linkedin.common",
  "ref": "bytes",
  "java": {
    "class": "java.net.InetAddress",
    "coercerClass": "com.linkedin.common.coercer.IPAddressCoercer"
  },
  "validate": {
    "com.linkedin.common.validator.IPAddressValidator": {}  
  }
}
```
ParSeq Execution Model (Visualized)

\[
\text{Task<?}> \ par3 = \text{Task.par(fetchBing, fetchGoogle, fetchYahoo)}
\]