Simplifying Salesforce REST in Java Using Annotations

The SPA library

David Buccola, Salesforce,
Principal Member of Technical Staff
@davidbuccola

SELL. SERVICE. MARKET. SUCCEED.

Safe harbor

Safe harbor statement under the Private Securities Litigation Reform Act of 1995:

This presentation may contain forward-looking statements that involve risks, uncertainties, and assumptions. If any such uncertainties materialize or if any of the assumptions proves incorrect, the results of salesforce.com, inc. could differ materially from the results expressed or implied by the forward-looking statements we make. All statements other than statements of historical fact could be deemed forward-looking, including any projections of product or service availability, subscriber growth, earnings, revenues, or other financial items and any statements regarding strategies or plans of management for future operations, statements of belief, any statements concerning new, planned, or upgraded services or technology developments and customer contracts or use of our services.

The risks and uncertainties referred to above include – but are not limited to – risks associated with developing and delivering new functionality for our service, new products and services, our new business model, our past operating losses, possible fluctuations in our operating results and rate of growth, interruptions or delays in our Web hosting, breach of our security measures, the outcome of any litigation, risks associated with completed and any possible mergers and acquisitions, the immature market in which we operate, our relatively limited operating history, our ability to expand, retain, and motivate our employees and manage our growth, new releases of our service and successful customer deployment, our limited history reselling non-salesforce.com products, and utilization and selling to larger enterprise customers. Further information on potential factors that could affect the financial results of salesforce.com, inc. is included in our annual report on Form 10-K for the most recent fiscal year and in our quarterly report on Form 10-Q for the most recent fiscal quarter. These documents and others containing important disclosures are available on the SEC Filings section of the Investor Information section of our Web site.

Any unreleased services or features referenced in this or other presentations, press releases or public statements are not currently available and may not be delivered on time or at all. Customers who purchase our services should make the purchase decisions based upon features that are currently available. Salesforce.com, inc. assumes no obligation and does not intend to update these forward-looking statements.

Introduction

Leveraging Salesforce from Java

You might choose Java when…

- Integrating an existing application written in java
  - Large body of existing code
- Integrating a completely new application
  - Developer expertise and environment

Network APIs for Java

Multiple styles of network API

- SOAP
- REST

You need a Java API to help put the bits on the wire

Different Levels of Java API

Low level APIs

- Protocol-specific
  - JAX-RS for REST
  - JAX-WS for SOAP
  - Salesforce Web Services Connector (WSC)

High level APIs

- More abstract, protocol independent, layered on low-level APIs
  - Java Persistence Annotations (JPA)

SPA is a High Level API

So…

- No religion about SOAP vs. REST
- No focus on network protocols
  - Implementation based on REST but that’s not necessarily important

Different Styles of Java API Access

Loosely typed bag of properties

- Property maps
- Jackson JsonNode
- Dynamic, flexible

Strongly typed beans

- Jackson bean bindings
- Java Persistence Annotations (JPA)
- Additional compiler and IDE assist to assure correctness

SPA is a Strongly Typed API

Not preaching loosely typed vs. strongly typed

- Different problems call for different approaches

If problem calls for strongly typed then…

- SPA can help you out
  - https://github.com/davidbuccola/force-spa

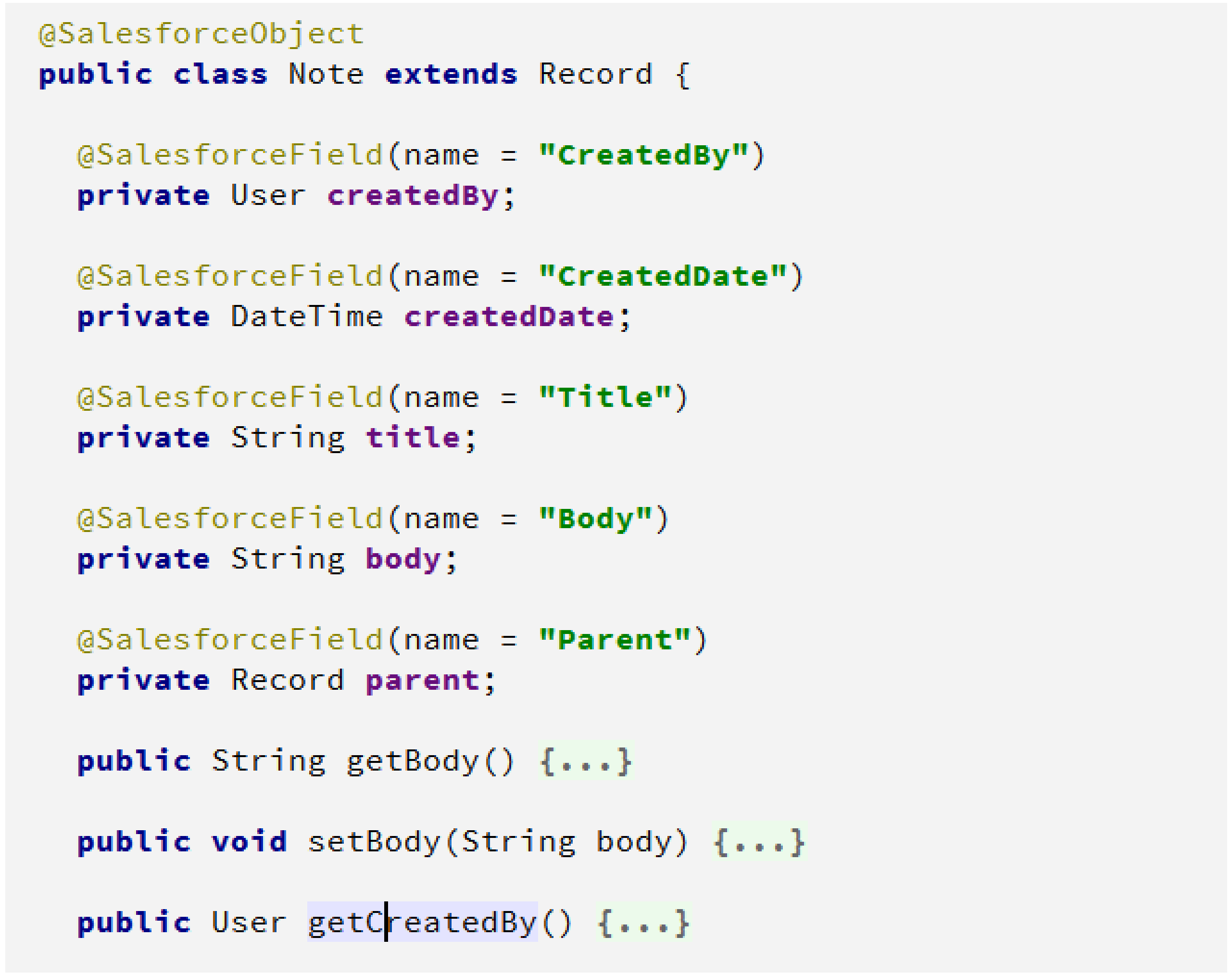
If problem calls for loosely typed then…

- Check out “Rich SObjects”
  - https://github.com/ryanbrainard/richsobjects

Quickstart Overview

Beans with Annotations

- Salesforce Persistence Annotations (SPA)
- Annotated beans that correspond to Salesforce objects
- In the style of Jackson or JPA



```
@SalesforceObject
public class Note extends Record {

  @SalesforceField(name = "CreatedBy")
  private User createdBy;

  @SalesforceField(name = "CreatedDate")
  private DateTime createdDate;

  @SalesforceField(name = "Title")
  private String title;

  @SalesforceField(name = "Body")
  private String body;

  @SalesforceField(name = "Parent")
  private Record parent;

  public String getBody() {...}

  public void setBody(String body) {...}

  public User getCreatedBy() {...}
```

RecordAccessor Interface

- CRUD+
  - create
  - get
  - delete
  - patch
  - update
  - query
  - more…

```
/**
 * A CRUD-based interface for interacting with persistent records in Salesforce through the use
 * of annotated Javabeans.
 */
public interface RecordAccessor {
  <T> String create(T record) throws ObjectNotFoundException, UnauthorizedException;

  <T> void delete(String id, Class<T> recordClass) throws RecordNotFoundException, UnauthorizedException;

  <T> T get(String id, Class<T> recordClass) throws RecordNotFoundException, UnauthorizedException;

  <T> void patch(String id, T recordChanges) throws RecordNotFoundException, UnauthorizedException;

  <T> void update(String id, T record) throws RecordNotFoundException, UnauthorizedException;

  <T> RecordQuery<T> createQuery(String soqlTemplate, Class<T> recordClass);
```

Simple Example

- Configure a RecordAccessorFactory
  - Jersey-based RecordAccessor factory
  - Configured to use username/password
    - In real life you'd use something better (OAuth)
- Create a RecordAccessor
  - Stateless, thread-safe, reusable
- Get your record

```java
public static void main(String[] args) {

  RecordAccessorFactory factory =
    new JerseyRecordAccessorFactory(
      new PasswordAuthorizationConnector("username", "password"));

  RecordAccessor accessor = factory.getRecordAccessor();

  Note note = accessor.get("002D000000CK5G6", Note.class);

  System.out.println(note.getTitle());
}
```

Like Jackson but Different

Built on top of Jackson

- Leverages Jackson annotation processing, serialization and deserialization

But…

- More semantic knowledge about Salesforce objects and relationships than raw Jackson
- Allows SPA to make your life easier, and more powerful

Like JPA but different

Some JPA-like capabilities

- Building complex queries for you automatically
- Pulling in trees of related objects in one shot

But…

- Not JPA, MUCH lighter weight
- Operates on simple passive beans
  - No complex runtime bean modification or instrumentation
- Stateless

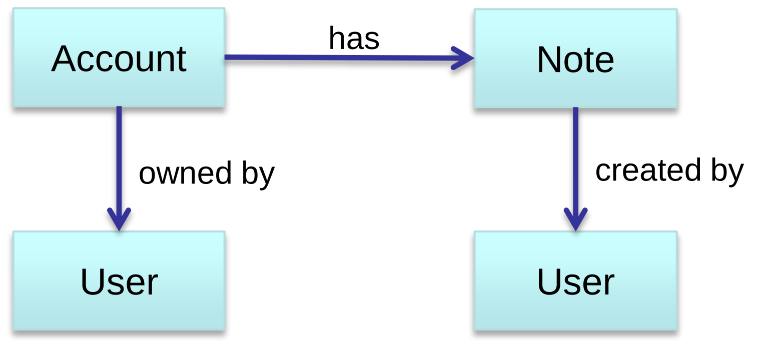
How SPA Helps

Subtleties of Salesforce REST

- Jackson Annotations can be used with Salesforce REST, but…
- There are various subtleties that require extra effort
- SPA Bridges the gap between raw Jackson and Salesforce REST

Help with Object Trees

- Reading multiple objects at once improves performance
- Supported by Salesforce REST but takes extra work
- SPA does the work for you



```
@SalesforceObject
class Account extends NamedRecord {
  @SalesforceField(name = "AnnualRevenue") Double annualRevenue;
  @SalesforceField(name = "Owner") User owner;
  @SalesforceField(name = "LastModifiedBy") User lastModifiedBy;
  @SalesforceField(name = "Notes") List<Note> notes;
}

@SalesforceObject
class Note extends Record {
  @SalesforceField(name = "CreatedBy") User createdBy;
  @SalesforceField(name = "CreatedDate") DateTime createdDate;
  @SalesforceField(name = "Title") String title;
  @SalesforceField(name = "Body") String body;
  @SalesforceField(name = "Parent") Record parent;
}

@SalesforceObject(name = "User")
class User extends Record {
  @SalesforceField(name = "Name") String name;
  @SalesforceField(name = "Email") String email;
  @SalesforceField(name = "SmallPhotoUrl") String smallPhotoUrl;
}
```

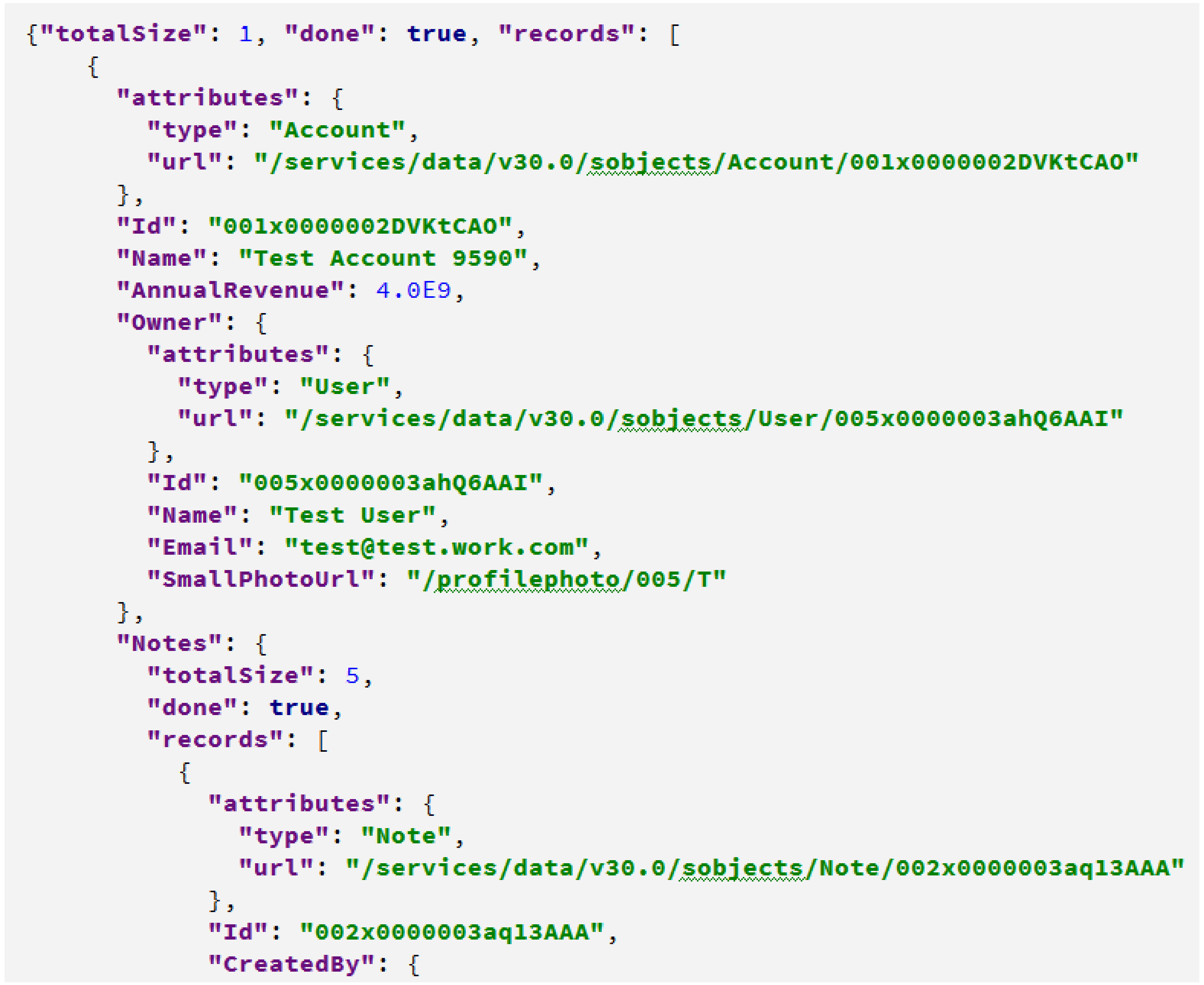
Builds the Tree Query

- Leverages the Annotations
- Helps with relationship complexities
- Maintenance simplified when leaf objects change

```
SELECT
  Id, Name, AnnualRevenue, LastActivityDate,
  Owner.Id,
  Owner.Name,
  Owner.Email,
  Owner.SmallPhotoUrl,
  LastModifiedBy.Id,
  LastModifiedBy.Name,
  LastModifiedBy.Email,
  LastModifiedBy.SmallPhotoUrl,
  (SELECT
    Id, CreatedDate, Title, Body,
    CreatedBy.Id,
    CreatedBy.Name,
    CreatedBy.Email,
    CreatedBy.SmallPhotoUrl,
    Parent.Id
  FROM Notes)
FROM Account
WHERE Id='001x000xxxERCyjAAH' LIMIT 1
```

Deserializes the Tree

- Deserialization is not automatic with vanilla Jackson
- Examples that take extra effort:
  - Parent-to-child relationships
  - Polymorphic relationships



```
{"totalSize": 1, "done": true, "records": [
    {
      "attributes": {
        "type": "Account",
        "url": "/services/data/v30.0/sobjects/Account/001x0000002DVKtCAO"
      },
      "Id": "001x0000002DVKtCAO",
      "Name": "Test Account 9590",
      "AnnualRevenue": 4.0E9,
      "Owner": {
        "attributes": {
          "type": "User",
          "url": "/services/data/v30.0/sobjects/User/005x0000003ahQ6AAI"
        },
        "Id": "005x0000003ahQ6AAI",
        "Name": "Test User",
        "Email": "test@test.work.com",
        "SmallPhotoUrl": "/profilephoto/005/T"
      },
      "Notes": {
        "totalSize": 5,
        "done": true,
        "records": [
          {
            "attributes": {
              "type": "Note",
              "url": "/services/data/v30.0/sobjects/Note/002x0000003aq13AAA"
            },
            "Id": "002x0000003aq13AAA",
            "CreatedBy": {
```

Polymorphic Relationships

Generates required SOQL syntax

```
SELECT
  Id,
  Body,
  TYPEOF Parent
    WHEN User THEN
      Id, Name, Email, SmallPhotoUrl
    WHEN Account THEN
      Id, Name, AnnualRevenue,
      Owner.Id, Owner.Name, Owner.Email, Owner.SmallPhotoUrl,
      LastModifiedBy.Id, LastModifiedBy.Name,
      LastModifiedBy.Email, LastModifiedBy.SmallPhotoUrl
    ELSE
      Id
  END
FROM FeedItem
```

Handles type information

```
{"totalSize": 182, "done": true, "records": [
  {
    "attributes": {
      "type": "FeedItem",
      "url": "/services/data/v30.0/sobjects/FeedItem/0D5x0000002DVKtCAO"
    },
    "Id": "0D5x0000002DVKtCAO",
    "Body": null,
    "Parent": {
      "attributes": {
        "type": "User",
        "url": "/services/data/v30.0/sobjects/User/005x0000003ahQ6AAI"
      },
      "Id": "005x0000003ahQ6AAI",
      "Name": "Test User",
      "Email": "test@test.work.com",
      "SmallPhotoUrl": "/profilephoto/005/T"
    }
  },
```

salesforce

Help with Read-Only Fields

- Some Salesforce fields can't be updated
  - CreatedBy, CreatedDate, etc…
- A hassle for certain programming patterns (read-modify-write)
- SPA leverages Jackson views to filter read-only fields on update
- Automatic for many common fields
- Configurable through annotations

Read-Modify-Write

```
User user = accessor.get(userId, User.class);

user.setSmallPhotoUrl(null);

accessor.update(user);
```

Annotated Read-Only Field

```
@SalesforceObject(name = "User")
public class User extends Record {

  @SalesforceField(name = "Name", updatable = false, insertable = false)
  private String name;

  @SalesforceField(name = "Email")
  private String email;

  @SalesforceField(name = "SmallPhotoUrl")
  private String smallPhotoUrl;
```

The Annotations

@SalesforceObject

- Identifies beans that correspond to a Salesforce Object
- Multiple beans can correspond to the same Salesforce Object
    - Allows you to define different “views” into the data

```java
/**
 * Identifies a type as representing a Salesforce object.
 */
@Documented
@Target(TYPE)
@Retention(RUNTIME)
public @interface SalesforceObject {

  /**
   * The name of the Salesforce object. Defaults to the Java
   * type name.
   */
  String name() default "";

  /**
   * Whether processing should leverage server-side metadata.
   */
  boolean metadataAware() default false;

  /**
   * Whether this is the primary bean for a Salesforce object
   * that has multiple bean definitions. This comes into play
   * during polymorphic parsing when no other hint exists to
   * help choose the right bean.
   */
  boolean primary() default false;
}
```

@SalesforceField

- Identifies bean members that correspond to a Salesforce field
- Only need annotated members for the fields you care about
- Can describe special behaviors:
  - ‘insertable’ – whether or not to serialize the value for ‘create’
  - ‘updatable’ - whether or not to serialize the value for ‘update’ or ‘patch’

```java
/**
 * Identifies a member (field or setter method) as representing
 * a Salesforce persistent field.
 */
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface SalesforceField {
  /**
   * The name of the Salesforce field. Defaults to the Java
   * property or field name.
   */
  String name() default "";

  /**
   * Whether the member's value should be persisted during
   * "create".
   */
  boolean insertable() default true;

  /**
   * Whether the member's value should be persisted during
   * "update" or "patch".
   */
  boolean updatable() default true;
}
```

@Polymorphic

- Identifies a relationship field as polymorphic
- Augments the @SalesforceField annotation
- Identifies valid choices for the related object type
    - Only need to identify the choices you care about

```
/**
 * Identifies a member (field or setter method) as being a
 * polymorphic field.
 */
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface Polymorphic {

  /**
   * A list of possible field types for a polymorphic field.
   */
  Class<?>[] value() default {};
}
```

The Record Accessor

Record Accessor Recap

- CRUD+
  - Create, get, update, patch, delete, query, and more
- Similarities to JPA but…
- More like Jackson internally
  - Operates on passive beans
  - No fancy runtime instrumentation or per-object state
  - Stateless, thread-safe, lightweight

```java
RecordAccessorFactory factory =
  new JerseyRecordAccessorFactory(
    new PasswordAuthorizationConnector(
      "username", "password"));

RecordAccessor accessor = factory.getRecordAccessor();

Account account = new Account();
account.setName("Test Account " + random.nextInt(10000));
account.setAnnualRevenue(4000000000.0);

String accountId = accessor.create(account);

for (int i = 1; i <= 5; i++) {
  Note note = new Note();
  note.setParent(Record.withId(accountId));
  note.setBody("Body text for note " + i);
  note.setTitle("Title for note " + i);

  accessor.create(note);
}

account = accessor.get(accountId, Account.class);
System.out.println(account);
```

Input Beans are not Modified

- If you’ve used JPA in the past…
  - JPA updates your bean state after write to do things like:
    - Clear modification state
    - Set a new ID field after create
- SPA will not
  - SPA leaves your input bean alone

Patch vs. Update

- SPA doesn't maintain state about modified fields in a bean
- You need to tell SPA how much you want to change
    - Patch – Change only selected things. "null" in bean field means don't change.
    - Update – Change everything. "null" in bean field means set to null in persistence

Update

```
    User user = accessor.get(userId, User.class);

    user.setSmallPhotoUrl(null);

    accessor.update(user);
```

Patch

```
    User userChanges = new User();

    userChanges.setSmallPhotoUrl("http://my.photos.com/me.jpg");

    accessor.patch(userId, userChanges);
```

Operation Lists

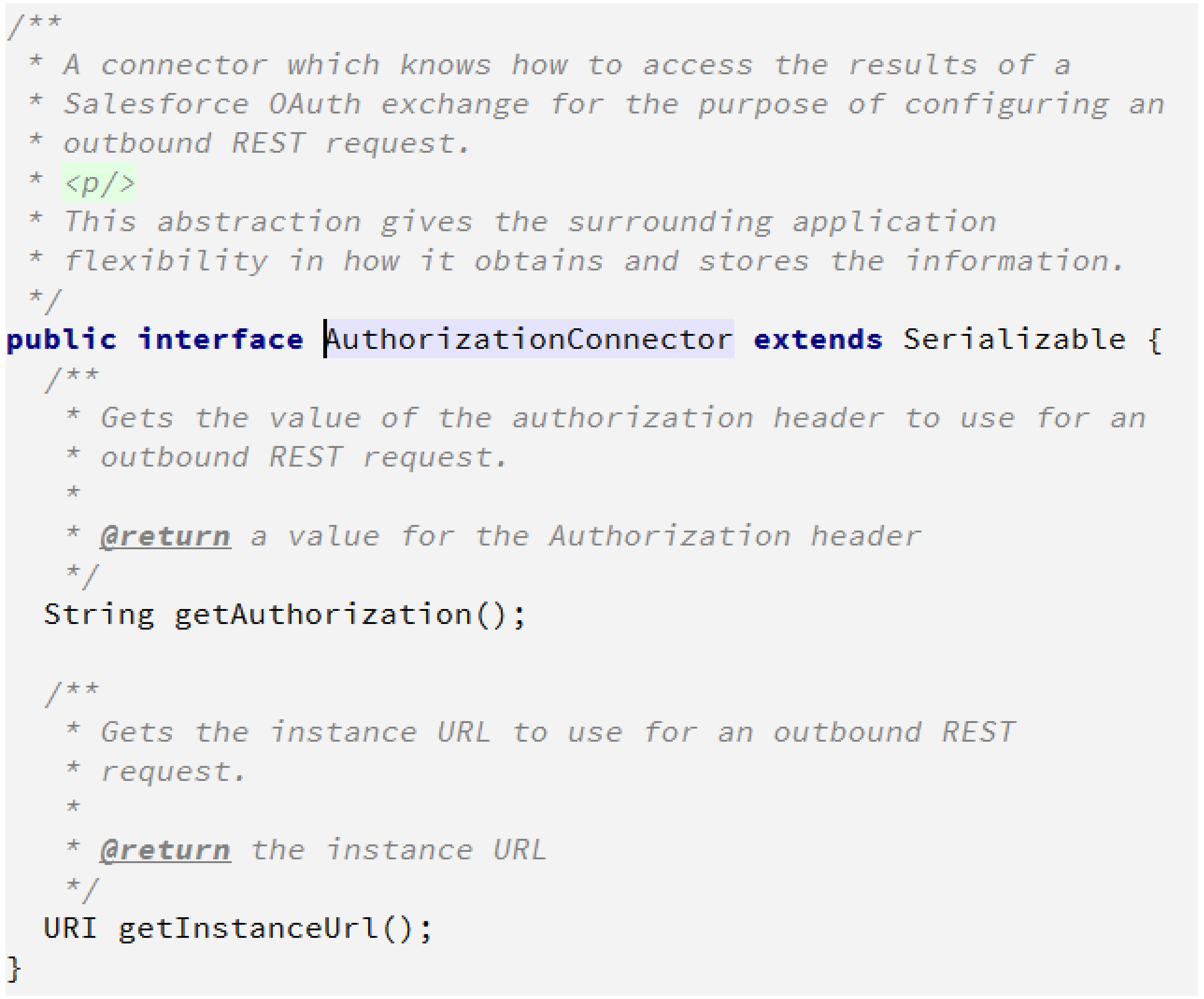
- SPA can execute lists of operations
- Powerful model that enables advanced capabilities
  - Access to statistics
    - Bytes sent and received
    - Rows processed
    - Elapsed time
  - Will soon execute as a batch
    - Reduces round trip latency
    - Leverages Connect batch resource
    - In pilot
  - Asynchronous execution planned

```
List<RecordOperation<?>> operations = new ArrayList<>();
for (int i = 1; i <= 5; i++) {
  Note note = new Note();
  note.setParent(Record.withId(accountId));
  note.setBody("Body text for note " + i);
  note.setTitle("Title for note " + i);

  operations.add(accessor.newCreateRecordOperation(note));
}

accessor.execute(operations);
```

Configuration

Authorization Connectors

- Many ways to do authorization
  - HTTP headers
  - OAuth
  - Username/Password
  - Etc…
- SPA delegates the choice
  - Standard authorization connectors
  - Custom authorization connectors



```
/**
 * A connector which knows how to access the results of a
 * Salesforce OAuth exchange for the purpose of configuring an
 * outbound REST request.
 * <p/>
 * This abstraction gives the surrounding application
 * flexibility in how it obtains and stores the information.
 */
public interface AuthorizationConnector extends Serializable {
  /**
   * Gets the value of the authorization header to use for an
   * outbound REST request.
   *
   * @return a value for the Authorization header
   */
  String getAuthorization();

  /**
   * Gets the instance URL to use for an outbound REST
   * request.
   *
   * @return the instance URL
   */
  URI getInstanceUrl();
}
```

Record Accessor Config

- Authorization Connector selection
- Configurable behaviors
- API Version
- Etc…

```
RecordAccessorConfig config =
  new RecordAccessorConfig()
    .withAuthorizationConnector(
      new PasswordAuthorizationConnector(
        "username", "password"))
    .withApiVersion(new ApiVersion(29, 0))
    .withAuditFieldWritingAllowed(true)
    .withFieldAnnotationRequired(false)
    .withObjectAnnotationRequired(false);

RecordAccessorFactory factory =
  new JerseyRecordAccessorFactory(config);

RecordAccessor accessor = factory.getRecordAccessor();
```

Simple Spring Configuration

- Designed for Spring configuration
  - Autowired or manually wired
  - Supports component scanning

Component scan

```
<context:annotation-config/>
<context:component-scan base-package="com.force.spa.core"/>
<context:component-scan base-package="com.force.spa.jersey"/>
```

Get injected with a little

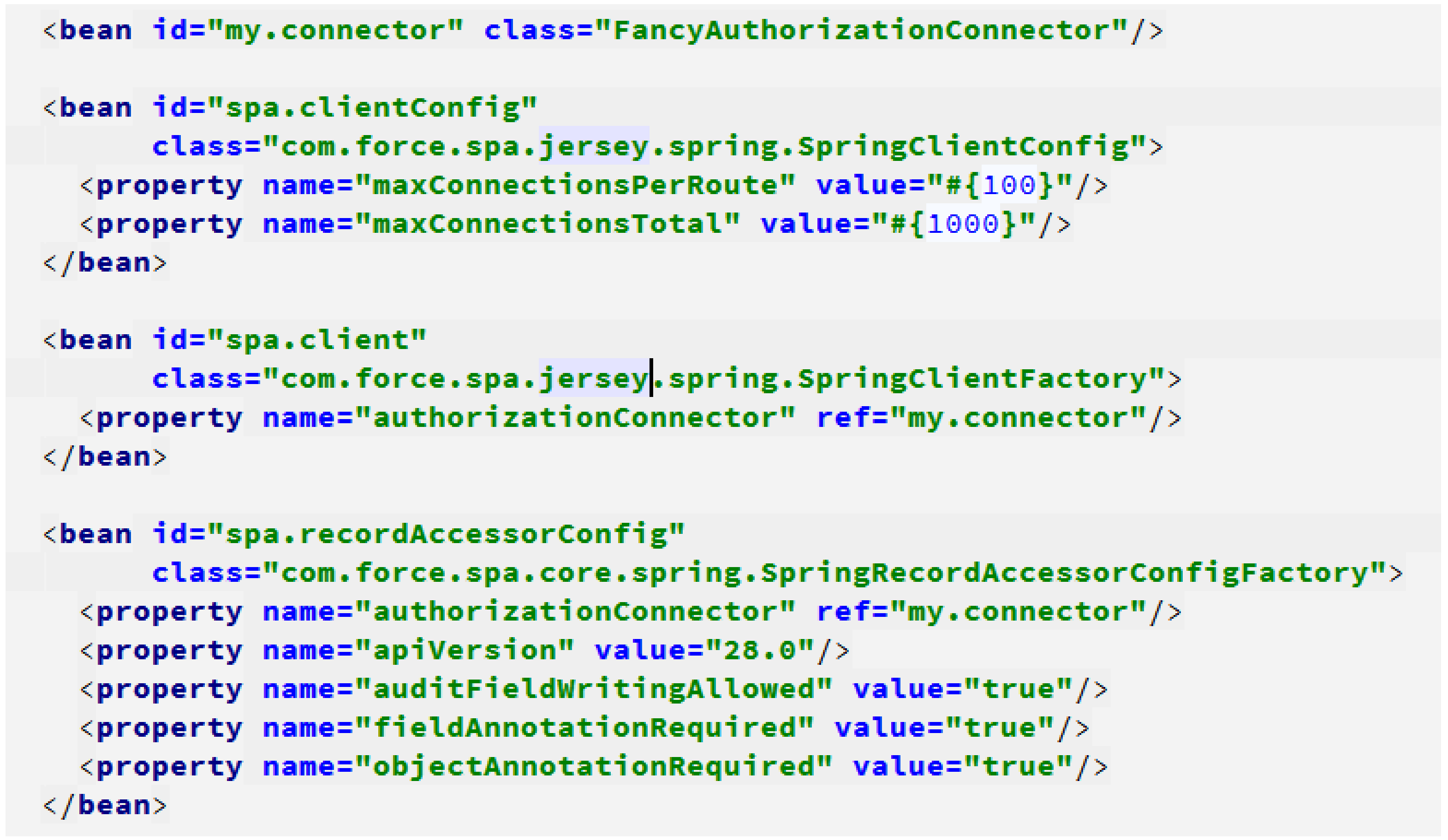
```
@Inject RecordAccessor recordAccessor;
```

Get injected with a lot

```
@Inject ClientConfig clientConfig;

@Inject AuthorizationConnector authorizationConnector;

@Inject RecordAccessorConfig recordAccessorConfig;

@Inject RecordAccessor recordAccessor;
```

Complex Spring Configuration

- Configure standard components by referencing their id (spa.xxx)
- Inject entirely new components



```xml
<bean id="my.connector" class="FancyAuthorizationConnector"/>

<bean id="spa.clientConfig"
      class="com.force.spa.jersey.spring.SpringClientConfig">
  <property name="maxConnectionsPerRoute" value="#{100}"/>
  <property name="maxConnectionsTotal" value="#{1000}"/>
</bean>

<bean id="spa.client"
      class="com.force.spa.jersey.spring.SpringClientFactory">
  <property name="authorizationConnector" ref="my.connector"/>
</bean>

<bean id="spa.recordAccessorConfig"
      class="com.force.spa.core.spring.SpringRecordAccessorConfigFactory">
  <property name="authorizationConnector" ref="my.connector"/>
  <property name="apiVersion" value="28.0"/>
  <property name="auditFieldWritingAllowed" value="true"/>
  <property name="fieldAnnotationRequired" value="true"/>
  <property name="objectAnnotationRequired" value="true"/>
</bean>
```

Where to Find It

SPA on the Internet

In Github:

- https://github.com/davidbuccola/force-spa

In Maven Central:

```
<dependency>
    <groupId>net.davidbuccola.force-spa</groupId>
    <artifactId>force-spa-jersey</artifactId>
    <version>1.1</version>
    <scope>compile</scope>
</dependency>
```

Q&A

David Buccola

Principal Member of Technical Staff, @davidbuccola

Thank you.