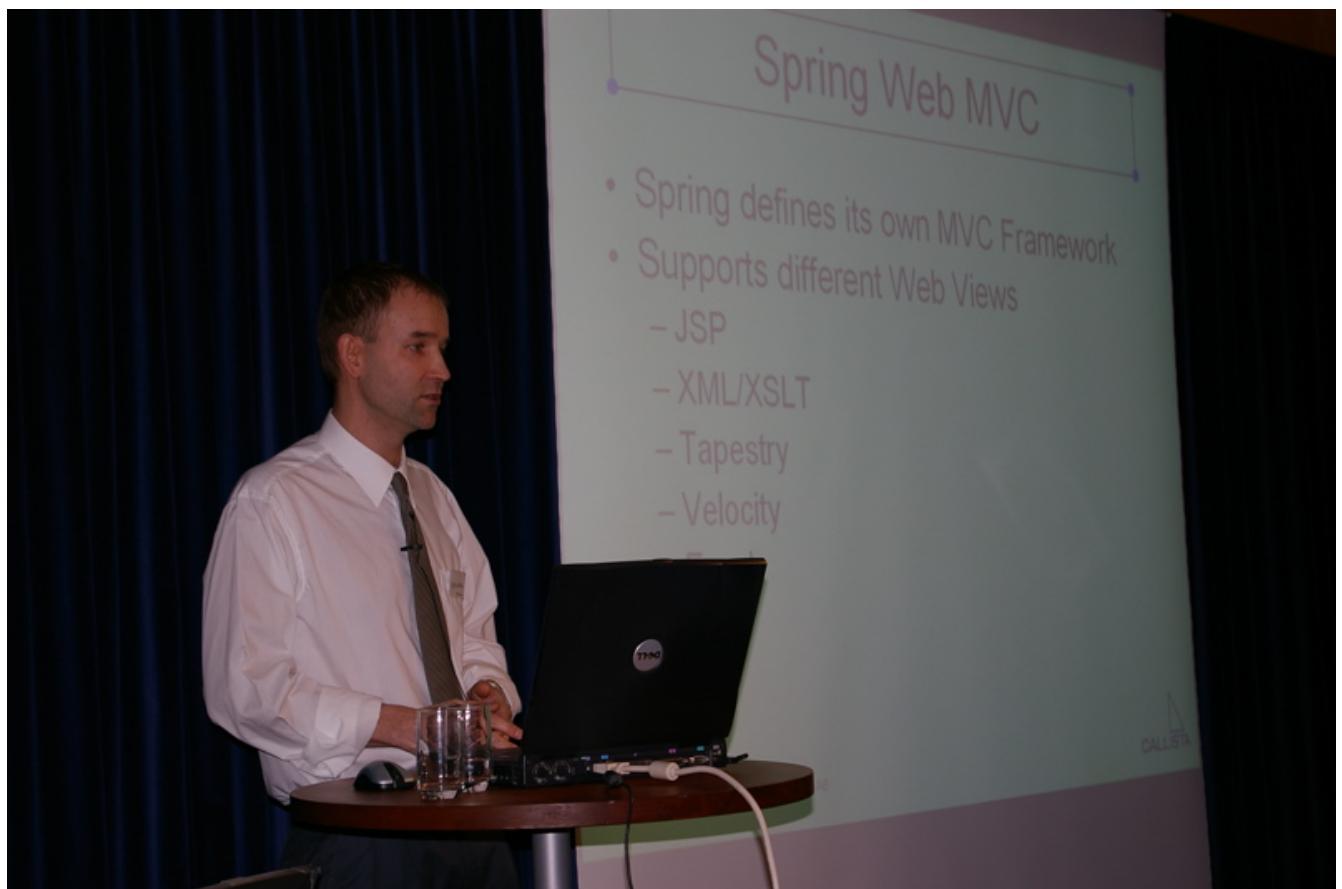


Spring Data

Björn Beskow | bjorn.beskow@callistaenterprise.se | 2014-01-29



Cadec 2005



2005: Spring = Innovation



Cadec 2010



2010: Spring = Legacy?



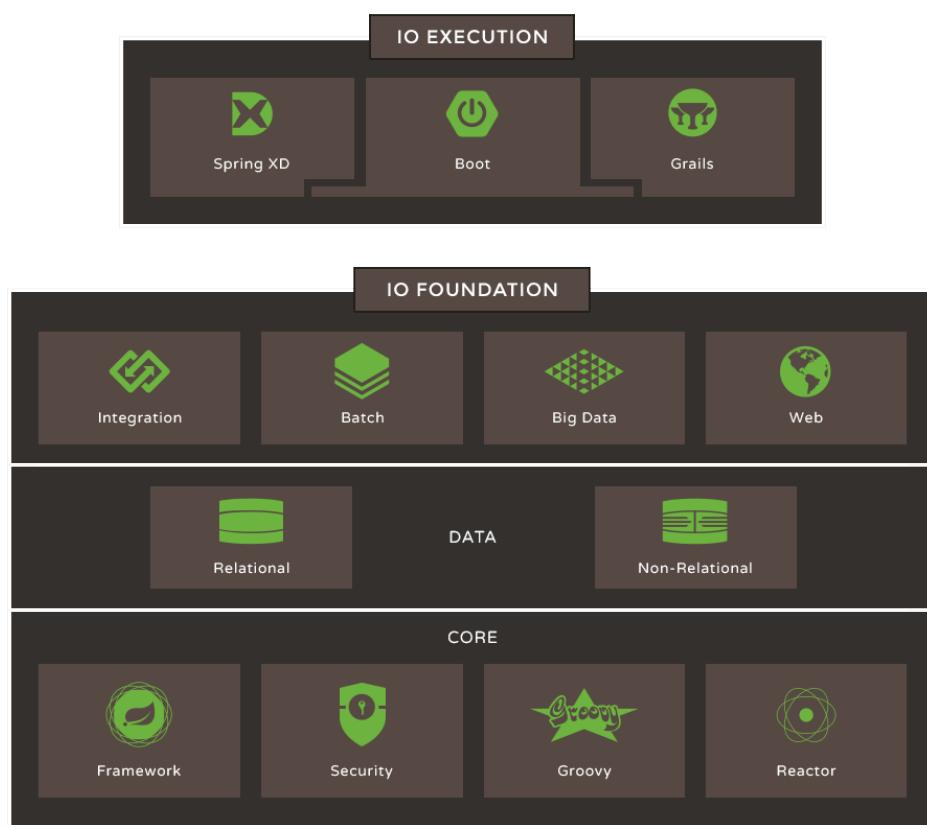
2014: Spring = Consolidation & Maturity



Consolidate.



Spring Data in the *spring.io* stack



The Spring Data project family

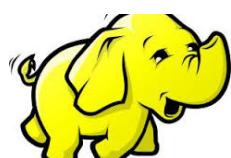
**Java
Persistence**



 mongoDB

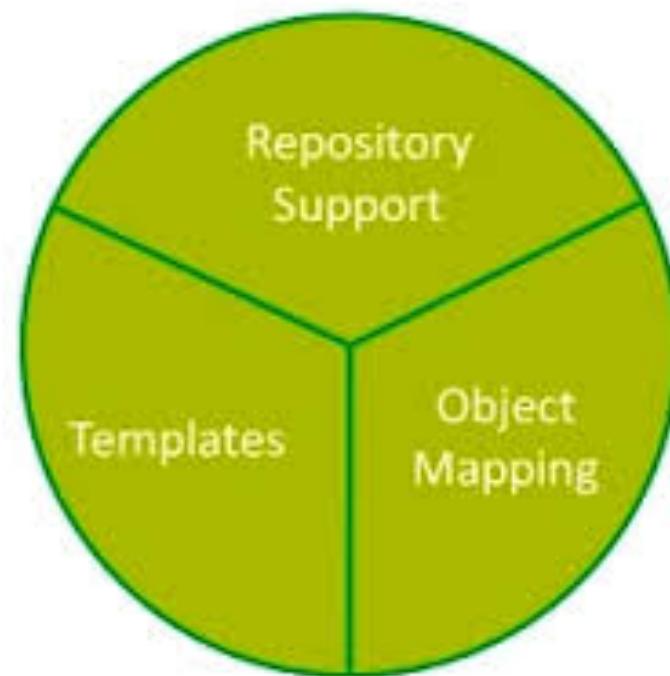
QUERY
DSL 

 redis



 springdata
Neo4j

Spring Data Main Abstractions



Spring Data JPA Repositories: Example

```
@Entity  
public class Invoice {  
  
    @Id  
    @GeneratedValue  
    private int id;  
  
    private String customerName;  
  
    private Date date;  
  
    ...  
}
```

Example: Repository Interface

```
public interface InvoiceRepository {  
  
    void create(Invoice invoice);  
    Invoice findById(Integer id);  
    Invoice update(Invoice invoice);  
    void delete(Invoice invoice);  
  
    List<Invoice> findByCustomerName(String customerName);  
    List<Invoice> findInvoicesSubmittedSince(Date date);  
    ...  
}
```

Repository Implementation

```
public class InvoiceRepositoryImpl implements InvoiceRepository {  
  
    @PersistenceContext  
    protected EntityManager entityManager;  
  
    public void create(Invoice invoice) {  
        entityManager.persist(invoice);  
    }  
  
    public Invoice findById(Integer id) {  
        return entityManager.find(Invoice.class, id);  
    }  
  
    public Invoice update(Invoice invoice) {  
        return entityManager.merge(invoice);  
    }  
  
    public void delete(Invoice invoice) {  
        entityManager.merge(invoice);  
        entityManager.remove(invoice);  
    }  
  
    ...  
}
```

Repository Implementation (contd.)

```
...  
  
@SuppressWarnings("unchecked")  
public List<Invoice> findByCustomerName(String customerName) {  
    Query q = entityManager  
        .createQuery("SELECT i FROM Invoice i WHERE customerName >= :customer");  
    q.setParameter("customer", customerName);  
    return (List<Invoice>) q.getResultList();  
}  
  
@SuppressWarnings("unchecked")  
public List<Invoice> findInvoicesSubmittedSince(Date date) {  
    Query q = entityManager  
        .createQuery("SELECT i FROM Invoice i WHERE date >= :date_since");  
    q.setParameter("date_since", date);  
    return (List<Invoice>) q.getResultList();  
}  
}
```

Generic DAO pattern

```
public interface GenericDao<T, PK extends Serializable> {  
  
    void create(T t);  
    T findById(PK pk);  
    T update(T t);  
    void delete(T t);  
  
}
```

Generic DAO pattern (contd.)

```
public interface InvoiceRepository extends GenericDao<Invoice, Integer> {  
    List<Invoice> findByCustomerName(String customerName);  
    List<Invoice> findInvoicesSubmittedSince(Date date);  
}
```

Generic DAO pattern (contd.)

```
public class GenericDaoImpl<T, PK extends Serializable> implements GenericDao<T, PK> {

    protected Class<T> entityClass;

    @PersistenceContext
    protected EntityManager entityManager;

    @SuppressWarnings("unchecked")
    public GenericDaoImpl() {
        ParameterizedType genericSuperclass = (ParameterizedType) getClass().getGenericSuperclass();
        this.entityClass = (Class<T>) genericSuperclass.getActualTypeArguments()[0];
    }

    public void create(T t) {
        entityManager.persist(t);
    }

    public T findById(PK pk) {
        return entityManager.find(entityClass, pk);
    }

    public T update(T t) {
        return entityManager.merge(t);
    }

    public void delete(T t) {
        entityManager.merge(t);
        entityManager.remove(t);
    }
}
```

Generic DAO pattern (contd.)

```
public class InvoiceRepositoryImpl extends GenericDaoImpl<Invoice, Integer>
    implements InvoiceRepository {

    @SuppressWarnings("unchecked")
    public List<Invoice> findByCustomerName(String customerName) {
        Query q = entityManager
.createQuery("SELECT i FROM Invoice i WHERE customerName >= :customer");
        q.setParameter("customer", customerName);
        return (List<Invoice>) q.getResultList();
    }

    @SuppressWarnings("unchecked")
    public List<Invoice> findInvoicesSubmittedSince(Date date) {
        Query q = entityManager
.createQuery("SELECT i FROM Invoice i WHERE date >= :date_since");
        q.setParameter("date_since", date);
        return (List<Invoice>) q.getResultList();
    }
}
```

Spring Data JPA Configuration

```
<beans xmlns="...">  
    ...  
    <bean id="entityManagerFactory" .../>  
    <jpa:repositories base-package="se.callista.cadec.repositories" />  
    ...  
</beans>
```

Spring Data JPA Repository

```
import org.springframework.data.repository.CrudRepository;  
  
public interface InvoiceRepository extends CrudRepository<Invoice, Integer> {  
    List<Invoice> findByCustomerName(String customerName);  
}
```

Spring Data JPA Repositories

- Interface-based programming model
- No implementation required
 - provided by Spring Data
- Queries derived from method names

Supported keywords for query methods

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	... where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	... where x.lastname = ?1 or x.firstname = ?2
Between	findByStartDateBetween	... where x.startDate between ?1 and ?2
LessThan	findByAgeLessThan	... where x.age < ?1
GreaterThan	findByAgeGreaterThan	... where x.age > ?1
After	findByStartDateAfter	... where x.startDate > ?1
Before	findByStartDateBefore	... where x.startDate < ?1
IsNull	findByAgeIsNull	... where x.age is null
IsNotNull, NotNull	findByAge(Is)NotNull	... where x.age not null
Like	findByFirstnameLike	... where x.firstname like ?1
NotLike	findByFirstnameNotLike	... where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	... where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	... where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	... where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	... where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	... where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	... where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	... where x.age not in ?1
True	findByActiveTrue()	... where x.active = true
False	findByActiveFalse()	... where x.active = false

Custom JPQL Queries

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;

public interface InvoiceRepository extends CrudRepository<Invoice, Integer> {

    List<Invoice> findByCustomerName(String customerName);

    @Query("SELECT i FROM Invoice i WHERE date >= ?1")
    List<Invoice> findInvoicesSubmittedSince(Date date);

}
```

Paging and Sorting

```
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;

public interface InvoiceRepository extends PagingAndSortingRepository<Invoice, Integer> {
    List<Invoice> findByCustomerName(String customerName);

    @Query("SELECT i FROM Invoice i WHERE date >= ?1")
    List<Invoice> findInvoicesSubmittedSince(Date date);
}
```

Redeclaring CRUD methods

```
import org.springframework.transaction.annotation.Transactional;

public interface InvoiceRepository extends PagingAndSortingRepository<Invoice, Integer> {

    /*
     * @see org.springframework.data.repository.CrudRepository#save(T)
     */
    @Transactional(timeout = 10)
    <S extends Invoice> S save(S entity);

    List<Invoice> findByCustomerName(String customerName);

    @Query("SELECT i FROM Invoice i WHERE date >= ?1")
    List<Invoice> findInvoicesSubmittedSince(Date date);

}
```

Spring Data MongoDB Example

```
public class Invoice {  
  
    @Id  
    private int id;  
  
    private String customerName;  
  
    private Date date;  
  
    ...  
}
```

Spring Data Mongo Configuration

```
<beans ...>
  ...
  <mongo:mongo id="mongo" />
  <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
    <constructor-arg ref="mongo" />
    <constructor-arg value="databaseName" />
  </bean>
  <mongo:repositories base-package="se.callista.cadec.repositories" />
  ...
</beans>
```

Spring Data Mongo Repository

```
import org.springframework.data.repository.CrudRepository;  
  
public interface InvoiceRepository extends CrudRepository<Invoice, Integer> {  
    List<Invoice> findByCustomerName(String customerName);  
}
```

Supported keywords for query methods

Keyword	Sample	Logical result
GreaterThan	findByAgeGreaterThan(int age)	{"age" : {"\$gt" : age}}
LessThan	findByAgeLessThan(int age)	{"age" : {"\$lt" : age}}
Between	findByAgeBetween(int from, int to)	{"age" : {"\$gt" : from, "\$lt" : to}}
IsNotNull, NotNull	findByFirstnameNotNull()	{"age" : {"\$ne" : null}}
IsNull, Null	findByFirstnameNull()	{"age" : null}
Like	findByFirstnameLike(String name)	{"age" : age} (age as regex)
Regex	findByFirstnameRegex(String firstname)	{"firstname" : {"\$regex" : firstname }}
(No keyword)	findByFirstname(String name)	{"age" : name}
Not	findByFirstnameNot(String name)	{"age" : {"\$ne" : name}}
Near	findByLocationNear(Point point)	{"location" : {"\$near" : [x,y]}}
Within	findByLocationWithin(Circle circle)	{"location" : {"\$within" : {"\$center" : [[x, y], distance]}}}
Within	findByLocationWithin(Box box)	{"location" : {"\$within" : {"\$box" : [[x1, y1], x2, y2]}}}True
IsTrue, True	findByActiveIsTrue()	{"active" : true}
IsFalse, False	findByActiveIsFalse()	{"active" : false}
Exists	findByLocationExists(boolean exists)	{"location" : {"\$exists" : exists }}

Mongo-specific Queries

```
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.data.mongodb.repository.Query;

public interface InvoiceRepository extends MongoRepository<Invoice, Integer> {

    List<Invoice> findByDealerNear(Point location, Distance distance);

    @Query("{'customerName': ?0}")
    List<Invoice> findInvoicesByCustomerName(String name);

}
```

Type-safe Queries

```
@Query("SELECT i FROM Invoice i " +  
       "WHERE customerName like ?1 AND date >= ?2 " +  
       "ORDER BY date DESC")
```

Criteria API Woe

```
// SELECT i FROM Invoice i WHERE customerName like ?1 AND date >= ?2 ORDER BY date DESC
public List<Invoice> findInvoicesByCustomerSince(String customer, Date date) {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Invoice> cq = cb.createQuery(Invoice.class);
    Root<Invoice> c = cq.from(Invoice.class);
    ParameterExpression<String> cust = cb.parameter(String.class);
    ParameterExpression<Date> since = cb.parameter(Date.class);
    cq.select(c).where(
        cb.and(cb.like(c.<String> get("customerName"), cust)),
        cb.greaterThanOrEqualTo(c.<Date> get("date")),
        since));
    cq.select(c).orderBy(cb.desc(c.get("date")));
    TypedQuery<Invoice> q = entityManager.createQuery(cq);
    q.setParameter(cust, customer);
    q.setParameter(since, date);
    return q.getResultList();
}
```

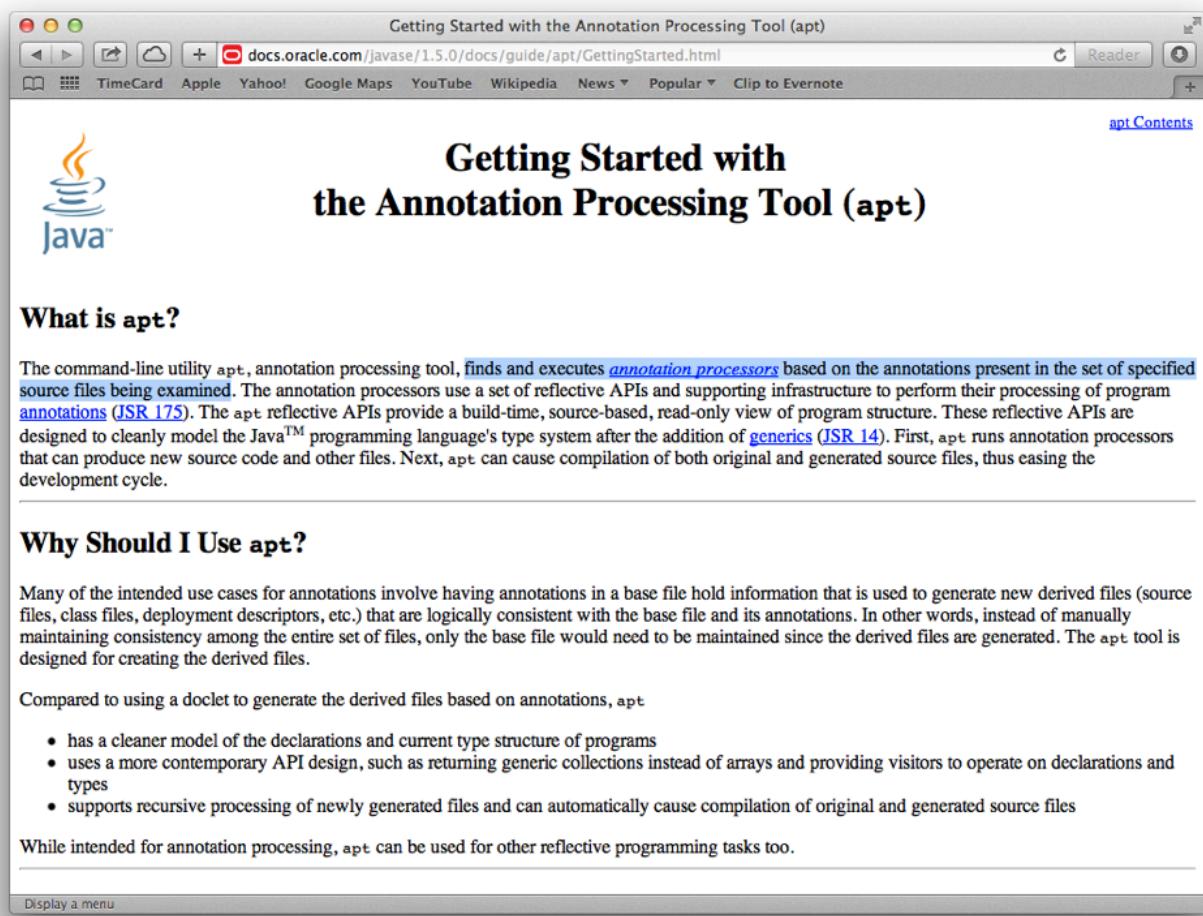
Open Source alternative: Querydsl

- Querydsl is a framework which enables the construction of type-safe SQL-like queries for multiple backends including JPA, Hibernate, MongoDB and SQL.
- Querydsl provides a type-safe, fluent API that is compact, elegant and easy to learn.

```
QInvoice invoice = QInvoice.invoice;  
List<Invoice> result = query.from(invoice)  
.where(invoice.customerName.like(customer).and(invoice.date.goe(date))  
.orderBy(invoice.date.desc());
```



QueryDSL meta-data generation: APT



The screenshot shows a web browser window with the title "Getting Started with the Annotation Processing Tool (apt)". The URL in the address bar is docs.oracle.com/javase/1.5.0/docs/guide/apt/GettingStarted.html. The page content includes the Java logo, the title "Getting Started with the Annotation Processing Tool (apt)", and sections titled "What is apt?", "Why Should I Use apt?", and a note about its intended use cases. The page is part of the Java documentation, as indicated by the Java logo and the URL.

What is apt?

The command-line utility `apt`, annotation processing tool, finds and executes [annotation processors](#) based on the annotations present in the set of specified source files being examined. The annotation processors use a set of reflective APIs and supporting infrastructure to perform their processing of program [annotations \(JSR 175\)](#). The `apt` reflective APIs provide a build-time, source-based, read-only view of program structure. These reflective APIs are designed to cleanly model the Java™ programming language's type system after the addition of [generics \(JSR 14\)](#). First, `apt` runs annotation processors that can produce new source code and other files. Next, `apt` can cause compilation of both original and generated source files, thus easing the development cycle.

Why Should I Use apt?

Many of the intended use cases for annotations involve having annotations in a base file hold information that is used to generate new derived files (source files, class files, deployment descriptors, etc.) that are logically consistent with the base file and its annotations. In other words, instead of manually maintaining consistency among the entire set of files, only the base file would need to be maintained since the derived files are generated. The `apt` tool is designed for creating the derived files.

Compared to using a doclet to generate the derived files based on annotations, `apt`

- has a cleaner model of the declarations and current type structure of programs
- uses a more contemporary API design, such as returning generic collections instead of arrays and providing visitors to operate on declarations and types
- supports recursive processing of newly generated files and can automatically cause compilation of original and generated source files

While intended for annotation processing, `apt` can be used for other reflective programming tasks too.

APT Config example: Maven

```
<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>1.1.0</version>
    <configuration>
        <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</processor>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>3.2.4</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>target/generated-
sources</outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```



Adding QueryDSL execution: JPA

```
import org.springframework.data.querydsl.QueryDslPredicateExecutor;

public interface InvoiceRepository extends PagingAndSortingRepository<Invoice, Integer>,
    QueryDslPredicateExecutor<Invoice> {

    . . .

}
```



Adding QueryDSL execution: MongoDB

```
import org.springframework.data.querydsl.QueryDslPredicateExecutor;

public interface InvoiceRepository extends MongoRepository<Invoice, Integer>,
    QueryDslPredicateExecutor<Invoice> {

    . . .

}
```



Criteria API Example

```
public class InvoiceRepositoryImpl implements InvoiceRepository {  
  
    // SELECT i FROM Invoice i WHERE customerName like ?1 AND date >= ?2 ORDER BY date DESC  
    public List<Invoice> findInvoicesByCustomerSince(String customer, Date date) {  
        CriteriaBuilder cb = entityManager.getCriteriaBuilder();  
        CriteriaQuery<Invoice> cq = cb.createQuery(Invoice.class);  
        Root<Invoice> c = cq.from(Invoice.class);  
        ParameterExpression<String> cust = cb.parameter(String.class);  
        ParameterExpression<Date> since = cb.parameter(Date.class);  
        cq.select(c).where(  
            cb.and(cb.like(c.<String> get("customerName"), cust)),  
            cb.greaterThanOrEqualTo(c.<Date> get("date"),  
since));  
        cq.select(c).orderBy(cb.desc(c.get("date")));  
        TypedQuery<Invoice> q = entityManager.createQuery(cq);  
        q.setParameter(cust, customer);  
        q.setParameter(since, date);  
        return q.getResultList();  
    }  
}
```

QueryDSL query example

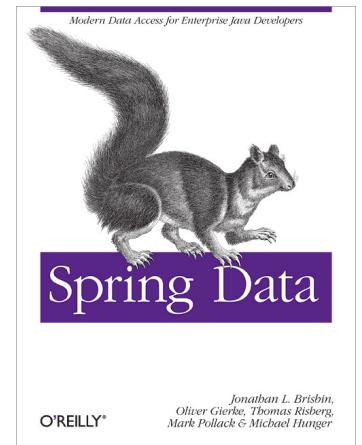
```
public class InvoiceService {  
  
    @Autowired  
    private InvoiceRepository invoiceRepository;  
  
    // SELECT i FROM Invoice i WHERE customerName >= ?1 AND date >= ?2 ORDER BY date  
DESC  
    public List<Invoice> findInvoicesByCustomerSince(String customer, Date date) {  
        QInvoice invoice = QInvoice.invoice;  
        Predicate where =  
invoice.customerName.like(customer).and(invoice.date.goe(date));  
        OrderSpecifier<Date> orderBy = invoice.date.desc();  
        return (List<Invoice>) invoiceRepository.findAll(where, orderBy);  
    }  
  
}
```



Spring Data: Consolidation & Maturity



References



- <http://projects.spring.io/spring-data/>
- <http://projects.spring.io/spring-data-jpa/>
- <http://projects.spring.io/spring-data-mongodb/>
- <http://www.querydsl.com/>
- <https://github.com/spring-projects/spring-data-jpa-examples>
- <http://www.amazon.com/Spring-Data-Mark-Pollack/dp/1449323952>

Questions

