31. Working with NoSQL technologies

Spring Data provides additional projects that help you access a variety of NoSQL technologies including MongoDB, Neo4J, Elasticsearch, Solr, Redis, Gemfire, Couchbase and Cassandra. Spring Boot provides auto-configuration for Redis, MongoDB, Elasticsearch, Solr and Cassandra; you can make use of the other projects, but you will need to configure them yourself. Refer to the appropriate reference documentation at projects.spring.io/spring-data.

31.1 Redis

Redis is a cache, message broker and richly-featured key-value store. Spring Boot offers basic auto-configuration for the Jedis client library and abstractions on top of it provided by Spring Data Redis. There is a `spring-boot-starter-redis` 'Starter POM' for collecting the dependencies in a convenient way.

31.1.1 Connecting to Redis

You can inject an auto-configured `RedisConnectionFactory`, `StringRedisTemplate` or vanilla `RedisTemplate` instance as you would any other Spring Bean. By default the instance will attempt to connect to a Redis server using `localhost:6379`:

```
@Component
public class MyBean {

    private StringRedisTemplate template;

    @Autowired
    public MyBean(StringRedisTemplate template) {
        this.template = template;
    }

    // ...

}
```

If you add a `@Bean` of your own of any of the auto-configured types it will replace the default (except in the case of `RedisTemplate` the exclusion is based on the bean name 'redisTemplate' not its type). If `commons-pool2` is on the classpath you will get a pooled connection factory by default.

31.2 MongoDB

MongoDB is an open-source NoSQL document database that uses a JSON-like schema instead of traditional table-based relational data. Spring Boot offers several conveniences for working with MongoDB, including the `spring-boot-starter-data-mongodb` 'Starter POM'.

31.2.1 Connecting to a MongoDB database

You can inject an auto-configured `org.springframework.data.mongodb.MongoDbFactory` to access Mongo databases. By default the instance will attempt to connect to a MongoDB server using the URL `mongodb://localhost/test`:

```
import org.springframework.data.mongodb.MongoDbFactory;
import com.mongodb.DB;

@Component
public class MyBean {

    private final MongoDbFactory mongo;

    @Autowired
    public MyBean(MongoDbFactory mongo) {
        this.mongo = mongo;
    }

    // ...

    public void example() {
        DB db = mongo.getDb();
        // ...
    }

}
```

You can set `spring.data.mongodb.uri` property to change the URL and configure additional settings such as the *replica set*:

```
spring.data.mongodb.uri=mongodb://user:secret@mongo1.example.com:12345,mongo2.exa
```

Alternatively, as long as you're using Mongo 2.x, specify a `host`/`port`. For example, you might declare the following in your `application.properties`:

```
spring.data.mongodb.host=mongoserver
spring.data.mongodb.port=27017
```

`spring.data.mongodb.host` and `spring.data.mongodb.port` are not supported if you're using the Mongo 3.0 Java driver. In such cases, `spring.data.mongodb.uri` should be used to provide all of the configuration.

If `spring.data.mongodb.port` is not specified the default of `27017` is used. You could simply delete this line from the sample above.

If you aren't using Spring Data Mongo you can inject `com.mongodb.Mongo` beans instead of using `MongoDbFactory`.

You can also declare your own `MongoDbFactory` or `Mongo` bean if you want to take complete control of establishing the MongoDB connection.

31.2.2 MongoTemplate

Spring Data Mongo provides a `MongoTemplate` class that is very similar in its design to Spring's `JdbcTemplate`. As with `JdbcTemplate` Spring Boot auto-configures a bean for you to simply inject:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Component;

@Component
public class MyBean {

    private final MongoTemplate mongoTemplate;
```

```
    @Autowired
    public MyBean(MongoTemplate mongoTemplate) {
        this.mongoTemplate = mongoTemplate;
    }

    // ...

}
```

See the `MongoOperations` Javadoc for complete details.

31.2.3 Spring Data MongoDB repositories

Spring Data includes repository support for MongoDB. As with the JPA repositories discussed earlier, the basic principle is that queries are constructed for you automatically based on method names.

In fact, both Spring Data JPA and Spring Data MongoDB share the same common infrastructure; so you could take the JPA example from earlier and, assuming that `City` is now a Mongo data class rather than a JPA `@Entity`, it will work in the same way.

```
package com.example.myapp.domain;

import org.springframework.data.domain.*;
import org.springframework.data.repository.*;

public interface CityRepository extends Repository<City, Long> {

    Page<City> findAll(Pageable pageable);

    City findByNameAndCountryAllIgnoringCase(String name, String country);

}
```

For complete details of Spring Data MongoDB, including its rich object mapping technologies, refer to their reference documentation.

31.2.4 Embedded Mongo

Spring Boot offers auto-configuration for Embedded Mongo. To use it in your Spring Boot

The port that Mongo will listen on can be configured using the `spring.data.mongodb.port` property. To use a randomly allocated free port use a value of zero. The `MongoClient` created by `MongoAutoConfiguration` will be automatically configured to use the randomly allocated port.

If you have SLF4J on the classpath, output produced by Mongo will be automatically routed to a logger named `org.springframework.boot.autoconfigure.mongo.embedded.EmbeddedMongo`.

You can declare your own `IMongodConfig` and `IRuntimeConfig` beans to take control of the Mongo instance's configuration and logging routing.

31.3 Gemfire

Spring Data Gemfire provides convenient Spring-friendly tools for accessing the Pivotal Gemfire data management platform. There is a `spring-boot-starter-data-gemfire` 'Starter POM' for collecting the dependencies in a convenient way. There is currently no auto-configuration support for Gemfire, but you can enable Spring Data Repositories with a single annotation (`@EnableGemfireRepositories`).

31.4 Solr

Apache Solr is a search engine. Spring Boot offers basic auto-configuration for the Solr 4 client library and abstractions on top of it provided by Spring Data Solr. There is a `spring-boot-starter-data-solr` 'Starter POM' for collecting the dependencies in a convenient way.

Solr 5 is currently not supported and the auto-configuration will not be enabled by a Solr 5 dependency.

31.4.1 Connecting to Solr

You can inject an auto-configured `SolrServer` instance as you would any other Spring bean. By default the instance will attempt to connect to a server using `localhost:8983/solr`:

```
@Component
```

```
    private SolrServer solr;

    @Autowired
    public MyBean(SolrServer solr) {
        this.solr = solr;
    }

    // ...

}
```

If you add a `@Bean` of your own of type `SolrServer` it will replace the default.

31.4.2 Spring Data Solr repositories

Spring Data includes repository support for Apache Solr. As with the JPA repositories discussed earlier, the basic principle is that queries are constructed for you automatically based on method names.

In fact, both Spring Data JPA and Spring Data Solr share the same common infrastructure; so you could take the JPA example from earlier and, assuming that `City` is now a `@SolrDocument` class rather than a JPA `@Entity`, it will work in the same way.

For complete details of Spring Data Solr, refer to their reference documentation.

31.5 Elasticsearch

Elasticsearch is an open source, distributed, real-time search and analytics engine. Spring Boot offers basic auto-configuration for the Elasticsearch and abstractions on top of it provided by Spring Data Elasticsearch. There is a `spring-boot-starter-data-elasticsearch` 'Starter POM' for collecting the dependencies in a convenient way.

31.5.1 Connecting to Elasticsearch

You can inject an auto-configured `ElasticsearchTemplate` or Elasticsearch `Client` instance as you would any other Spring Bean. By default the instance will attempt to connect to a local in-memory server (a `NodeClient` in Elasticsearch terms), but you can switch to a remote server

(i.e. a `TransportClient`) by setting `spring.data.elasticsearch.cluster-nodes` to a comma-separated 'host:port' list.

```
@Component
public class MyBean {

    private ElasticsearchTemplate template;

    @Autowired
    public MyBean(ElasticsearchTemplate template) {
        this.template = template;
    }

    // ...

}
```

If you add a `@Bean` of your own of type `ElasticsearchTemplate` it will replace the default.

31.5.2 Spring Data Elasticsearch repositories

Spring Data includes repository support for Elasticsearch. As with the JPA repositories discussed earlier, the basic principle is that queries are constructed for you automatically based on method names.

In fact, both Spring Data JPA and Spring Data Elasticsearch share the same common infrastructure; so you could take the JPA example from earlier and, assuming that `City` is now an Elasticsearch `@Document` class rather than a JPA `@Entity`, it will work in the same way.

For complete details of Spring Data Elasticsearch, refer to their reference documentation.

31.6 Cassandra

Cassandra is an open source, distributed database management system designed to handle large amounts of data across many commodity servers. Spring Boot offers auto-configuration for Cassandra and abstractions on top of it provided by Spring Data Cassandra. There is a `spring-boot-starter-data-cassandra` 'Starter POM' for collecting the dependencies in a convenient way.

31.6.1 Connecting to Cassandra

You can inject an auto-configured `CassandraTemplate` or a Cassandra `Session` instance as you would any other Spring Bean. The `spring.data.cassandra.*` properties can be used to customize the connection. Generally you will to provide `keyspace-name` and `contact-points` properties:

```
spring.data.cassandra.keyspace-name=mykeyspace
spring.data.cassandra.contact-points=cassandrahost1,cassandrahost2
```

```
@Component
public class MyBean {

    private CassandraTemplate template;

    @Autowired
    public MyBean(CassandraTemplate template) {
        this.template = template;
    }

    // ...

}
```

If you add a `@Bean` of your own of type `CassandraTemplate` it will replace the default.

31.6.2 Spring Data Cassandra repositories

Spring Data includes basic repository support for Cassandra. Currently this is more limited than the JPA repositories discussed earlier, and will need to annotate finder methods with `@Query`.

For complete details of Spring Data Cassandra, refer to their reference documentation.