Best Practice of Spring

Work flow:

1. > mkdir gs-rest-service/src/main/java
2. 在gs-rest-service项目下创建build.gradle

//设置脚本的运行环境, 借用maven依赖库管理, 依赖包的定义

//Spring Boot gradle plugin: all jars on the classpath -> “\*application-name.jar”; search public static void main() as runnable class；提供dependecies的默认版本（当然，依赖包可以指定版本覆盖）

buildscript {

repositories {

mavenCentral()

}

dependencies {

classpath("org.springframework.boot:spring-boot-gradle-plugin:1.3.1.RELEASE")

}

}

//声明构建项目类型

//指定项目为java项目，项目编译(在项目提示符下执行：gradle build)时生成项目的jar包。

apply plugin: 'java'

// java项目的eclipse开发环境构建.生成所需要的.project,.classpath等文件。

apply plugin: 'eclipse'

apply plugin: 'idea'

apply plugin: 'spring-boot'

//指定jar包名字:gs-rest-service-0.1.0.jar

jar {

baseName = 'gs-rest-service'

version = '0.1.0'

}

//指定仓库使用

repositories {

mavenCentral()

}

sourceCompatibility = 1.7

targetCompatibility = 1.7

//项目依赖定义，compile为编译级别依赖，还有testCompile为测试级别的依赖等， you don’t have to supply the version number if you are using the Spring Boot Gradle plugin

//”group:name:version”

dependencies {

compile("org.springframework.boot:spring-boot-starter-web")

testCompile("junit:junit")

}

task wrapper(type: Wrapper) {

gradleVersion = '2.3'

}

1. 导入项目到IDE STS：Import project directory as gradle project, set source path

Right click -> Properties -> Java Build Path -> Source -> Add Folder: src/main/java

1. **Spring boot**

@Controller: handle incoming web requests

@RestController: render the resulting string directly back to the caller besides @Controller’s function

@RequestMapping: provide “routing” information

@EnableAutoConfiguration: automatically configure spring application based on the jar dependencies that you have added. For example, if HSQLDB is on you classpath, and you have not manually configured and database connection beans, then in-memory databased will be auto-configured. If you add your DataSource bean, the default embedded database support will back away.

@ComponentScan: automatically pick up all application components, by default, application class is located in a root package, then all application components like @Configuration, @Component, @Service, @Repository, @Controller etc will be automatically registered as Spring Beans

@SpringBootApplication =

@Configuration +

@EnableAutoConfiguration +

@ComponentScan

@Autowired:

@ControllerAdvice: customize the JSON document to return for a particular controller and/or exception type.

@EnableGlobalMethodSecurity: add method-level security to a web application

@EnableWebSecurity: switch off the default web security configuration

Externalized configuration

properties files, YAML files, environment variables, command-line arguments

1. command line option arguments -> property (add it to the spring environment

>–-spring.profiles.active=dev //一般作为最终运行的profiles

b. SpringApplication.setAdditionalProfiles(“dev”) //一般作为默认profiles

c. application.yml

Spring:

profiles:

active: dev

从而application-dev.yml or application-prod.yml is used

至于外部配置值 -> object，请见Spring.doc

Spring MVC auto-configuration

Serve static resources: by defaults,

/static

/public

/resources

/META-INF/resources

src/main/webapp only works with war packaging

HttpMessageConverters: convert HTTP requests and responses. By defaults, objects can be converted to JSON (using the Jackson library) or XML (using the Jackon XML extension if available)

Template engines: by defaults, src/main/resources/templates/

Thymeleaf, Mustache, Groovy, FreeMaker, Velocity

Error Handling:

/error: map by default, handles all errors.

For machine clients it will produce a JSON response with details of the error, HTTP status and exception message.

For bowser clients there is a ‘whitelabel’ error view that renders the same data in HTML format (to customize it just add a view that resolves to ‘error’)

Controller advice to translate the server side exceptions to client-friendly json structures.

@ControllerAdvice

public class ExceptionTranslator {

@ExceptionHandler(ConcurrencyFailureException.class)

@ResponseStatus(HttpStatus.CONFLICT)

@ResponseBody

public ErrorDTO processConcurencyError(ConcurrencyFailureException ex) {

return new ErrorDTO(ErrorConstants.ERR\_CONCURRENCY\_FAILURE);

}

}

Spring security:

If spring security is on the classpath then web applications will be secure by default with ‘basic’ authentication on all HTTP endpoints, the default AuthenticationManager has a single user (‘user’ username and random password, printed at INFO level when the application starts up), to change username and password as follows:

application.yml:

security:

user:

name: admin

password: secret

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true, securedEnabled = true)

public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

@Inject

private UserDetailsService userDetailsService;

@Inject

private RememberMeServices rememberMeServices;

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

//

@Inject

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth

.userDetailsService(userDetailsService)

.passwordEncoder(passwordEncoder());

}

@Override

public void configure(WebSecurity web) throws Exception {

web.ignoring()

.antMatchers("/scripts/\*\*/\*.{js,html}") ;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

}

}

Spring data

Caching: EhCache

Messaging: RabbitMQ

AMQP (Advanced Message Queuing Protocol) is a platform-neutral, wire-level protocol for message-oriented middleware.

RabbitMQ is a lightweight, reliable, scalable and portable message broker based on the AMQP protocol.

application.yml:

spring:

rabbitmq:

host: localhost

port: 5672

username: admin

password: secret

spring session

Test: unit test, integration test

Sprint boot actuator:

dependencies { compile “spring-boot-starter-actuator”}

Endpoints: /actuator, /autoconfig, /health, /metrics, /auditing/, …

application.properties:

security.user.name= admin 访问endpoints的权限

security.user.password= secret

management.security.role= SUPERUSER

management.context-path= /manage /health -> /manage/health

1. **Spring Data – MongoDB (**[**http://projects.spring.io/spring-data/**](http://projects.spring.io/spring-data/) **)**

MongoDb[2] is an open-source NoSQL document database that uses a JSON-like schema instead of traditional table-based relational data.

Spring Data MongoDB[3] brings MongoDB to Spring applications in three ways:

* Annotations for object-to-document mapping
* Template-based database access with MongoTemplate
* Automatic runtime repository generation

In our back-server, spring data MongoDB is applied

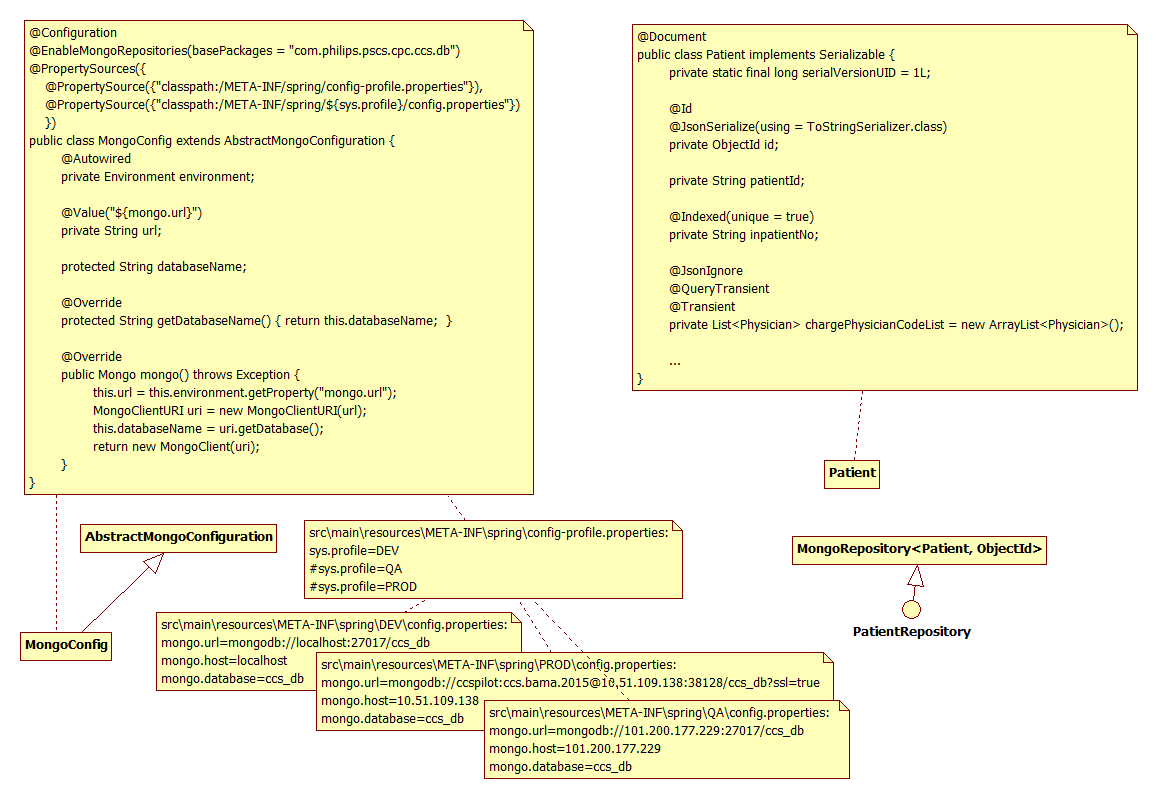


Figure 6. the diagram of spring data -mongodb

MongoConfig set configuration like MongoDB ip, port and database name. if database name does not exist, Spring MongoDB will automatically create one.

Patient is document like entity of JPA, will be mapped into collection of database ccs\_db.Patient

PatientRepository extends MongoRepository, it transitively extends the Repository marker interface. Any interface that extends Reposiotry will have an implementation automatically generated at runtime. PatientRepository will be implemented to read and write data to a MongoDB document PatientRepository interface has two parameters. The first is the type of @Document-annotated object that this repository deals with. The second is the type of the @Id-annotated property.

There are 3 common ways of defining query methods:

* Spring Data automatically generate 18 convenient methods [3]
* Spring Data supports a method-naming convention that helps Spring Data to automatically generate implementations for the methods that follow that convention.

For example, nothing else needs to be done to implement API: Patient findByPatientId(String patientId). When creating the repository implementation, Spring Data will examine any methods in the repository interface, parse the method name, and attempt to understand the method’s purpose in the context of the persisted object. In essence, Spring Data defines a sort of miniature domain-specific language (DSL) where persistence details are expressed in repository method signatures.

* Declare custom queries

You can use the @Query annotation to provide Spring Data with the query that should be performed. You still don’t write the implementation of the findPatientByPhilipsMail() method. You only give the query, hinting to Spring Data about how it should implement the method.

**public** **interface** PatientRepository **extends** MongoRepository<Patient, ObjectId>{

Patient findByInpatientNo(String inpatientNo);

@Query(“select p from Patient p where p.email like ‘%philips.com’ “)

List<Patient> findPatientsByPhilipsMail() ;

}

1. **Spring Data REST (**[**http://projects.spring.io/spring-data-rest/**](http://projects.spring.io/spring-data-rest/) **)**

Spring Data REST builds on top of **Spring Data repositories**, analyzes your application's **domain model** and **exposes hypermedia-driven HTTP resources** for aggregates contained in the model.

Via hypermedia-driven interface, there is no need to exchange a formal contract or interface document with your customers. **A hypermedia-driven site provides information to navigate the site's REST interfaces dynamically by including hypermedia links with the responses**. This capability differs from that of SOA-based systems and WSDL-driven interfaces. **With SOA, servers and clients usually must access a fixed specification** that might be staged somewhere else on the website, on another website, or perhaps distributed by email.

1. >mongod –dbpath ./data //启动mongo db
2. build.gradle

compile("org.springframework.boot:spring-boot-starter-data-rest")

compile("org.springframework.boot:spring-boot-starter-data-mongodb")

compile 'org.springframework.data:spring-data-rest-hal-browser'

1. create a domain object

src/main/java/com.philips.rest.domain.Person.java

public class Person {

@Id private String id ;

private String firstName ;

private String lastName ;

private String age ;

}

1. create repository

src/main/java/com.philips.rest.db.PersonRepository.java

public interface PersonRepository extends MongoRepository<Person, String>{

List<Person> findByLastName(@Param("name") String name) ;

}

1. run

e.1 expose RESTful API

|  |  |  |
| --- | --- | --- |
| Path | Method | Header |
| /persons{?page,size,sort}  (等于默认：?page=0&size=20) | GET |  |
| /persons/  {  “firstName”: “QiZhong”,  “lastName”: “Lin”,  “age”: 36  } | POST | Content-Type: application/json |
| /persons/{id} | GET |  |
| /persons/{id}  {  “firstName”: “QiZhong Jr.”,  “lastName”: “Lin”,  “age”: 3  } | PUT | Content-Type: application/json |
| /persons/{id}  {  “firstName”: “QiZhong Jr.”  } | PATCH | Content-Type: application/json |
| /persons/{id} | DELETE |  |
| /persons/search/findByLastName?name={name} | GET |  |

e.2 output HAL(hypertext application language) format

including resource link (from root URL to deep), **tools: HAL Browser**

localhost:8080/

1. **Spring Session (**[**http://projects.spring.io/spring-session/**](http://projects.spring.io/spring-session/) **)**

第一次login时，生成session id,同时在redis里以session id 为key,保存用户的信息；第二次就不需要login了，因为服务器与用户（浏览器）保存通话状态，服务端每次会发sessionid,浏览器会带上sessionid请求。

同浏览器，不论是同一tab,不同tab,或再启动一个浏览器，同一个session id有效，都会保存通知状态

//浏览器访问，sessionid采用默认的cookies传送，session信息存入redis数据库

src/main/java/session/config/HttpSessionConfig.java

@EnableRedisHttpSession

**public** **class** HttpSessionConfig {}

//restful 访问，sessionid采用httpheader传送，session信息存入redis数据库

第一次访问需要用户名和密码，服务端返回x-auth-token于httpheader

$curl –v localhost:8080 –u user:password

第二次用x-auth-token访问即可，不需要用户名和密码

$curl –v localhost:8080 –H “x-auth-token：…”

退出使token无效

$curl –v localhost:8080/logout –H “x-auth-token:…”

重复之前的过程

@EnableRedisHttpSession

**public** **class** HttpSessionConfig {

@Bean

**public** HttpSessionStrategy httpSessionStrategy() {

**return** **new** HeaderHttpSessionStrategy() ;

}

}

1. **Spring Data – JPA (relational database)**
2. build.gradle

compile("org.springframework.boot:spring-boot-starter-data-jpa")

1. create a domain object

src/main/java/com.philips.jpa.domain.Customer.java

**@Entity**

public class Customer {

**@Id**

**@GeneratedValue(strategy=GenerationType.AUTO)**

private long id ;

private String firstname ;

private String lastname ;

protected Customer() {}

public Customer(String firstname, String lastname){

this.firstname = firstname ;

this.lastname = lastname ;

}

}

1. create repository

src/main/java/com.philips.jpa.db.CustomerRepository.java

public interface CustomerRepository extends CrudRepository<Customer, Long>{

public List<Customer> findByFirstname(String firstname) ;

public List<Customer> findByLastname(String lastname) ;

}

1. run

src/main/java/com.philips.jpa.Application.java

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

private static final Logger log = LoggerFactory.getLogger(Application.class) ;

@Bean

public CommandLineRunner demo(CustomerRepository repository){

return (args) -> {

//save a couple of customers

repository.save(new Customer("Jack", "Bauer")) ;

repository.save(new Customer("Chloe", "O' Brain")) ;

// fetch all customers

repository.findAll().forEach((customer) -> log.info(customer.toString()));

// fetch an individual customer by ID

log.info(repository.findOne(1L).toString()) ;

// fetch customers by last name

repository.findByLastname("Bauer").forEach((item) -> log.info(item.toString()));

} ;

}

}

By default, Spring Boot will enable JPA repository support and look in the package (and its subpackages) where @SpringBootApplication is located

**in-memory database:** Spring Boot can auto-configure embedded H2, HSQL and Derby databases. You don’t need to provide any connection URLs, simply include a build dependency to the embedded database that you want to use. By default, JPA databases will be automatically created only if you use an embedded database (H2, HSQL or Derby).

Using H2’s web console (refer to Spring boot reference guide): /h2-console

**production database:** DataSource configuration is controlled by external configuration properties in spring.datasource.\*

application.yml:

spring:

datasource:

driver-class-name: org.postgresql.ds.PGSimpleDataSource

url: jdbc:postgresql://localhost:5432/bookstore

username: postgres

password: qzlin

You often won’t need to specify the driver-class-name since Spring boot can deduce it for most databases from the url.