

Spring[®] Framework

Notes for Professionals

Chapter 6: Bean scopes

Section 6.1: Additional scopes in web-aware context

- There are several scopes that are available only in a web-aware application context:
- request** - new bean instance is created per HTTP request
 - session** - new bean instance is created per ServletSession
 - application** - new bean instance is created per global session in Portlet environment
 - globalsession** - new bean instance is created per global session in environment
 - globalsession** - new bean instance is created per WebSocket session
 - websocket** - new bean instance is created per WebSocket session

XML Configuration

```
<bean id="myRequestBean" class="org.springframework.web.context.support.RequestScopedBean" scope="request"/>  
<bean id="mySessionBean" class="org.springframework.web.context.support.SessionScopedBean" scope="session"/>  
<bean id="myApplicationBean" class="org.springframework.web.context.support.ApplicationScopedBean" scope="application"/>  
<bean id="myGlobalSessionBean" class="org.springframework.web.context.support.GlobalSessionScopedBean" scope="globalsession"/>  
<bean id="myWebSocketBean" class="org.springframework.web.socket.support.WebSocketScopedBean" scope="websocket"/>
```

Java Configuration (prior to Spring 4.3)

```
@Configuration  
public class MyConfiguration {  
  
    @Bean  
    @Scope(value = WebApplicationContext.SCOPE_REQUEST, proxyMode = NoProxyMode.NO_PROXY_MODE_TARGET_CLASS)  
    public AnotherClass myRequestBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode = NoProxyMode.NO_PROXY_MODE_TARGET_CLASS)  
    public AnotherClass mySessionBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope(value = WebApplicationContext.SCOPE_APPLICATION, proxyMode = NoProxyMode.NO_PROXY_MODE_TARGET_CLASS)  
    public AnotherClass myApplicationBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope(value = WebApplicationContext.SCOPE_GLOBAL_SESSION, proxyMode = NoProxyMode.NO_PROXY_MODE_TARGET_CLASS)  
    public AnotherClass myGlobalSessionBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope(value = WebApplicationContext.SCOPE_WEBSOCKET, proxyMode = NoProxyMode.NO_PROXY_MODE_TARGET_CLASS)  
    public AnotherClass myWebSocketBean() {  
        return new AnotherClass();  
    }  
}
```

Java Configuration (after Spring 4.3)

```
@Configuration  
public class MyConfiguration {  
  
    @Bean  
    @Scope("request")  
    public AnotherClass myRequestBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope("session")  
    public AnotherClass mySessionBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope("application")  
    public AnotherClass myApplicationBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope("globalsession")  
    public AnotherClass myGlobalSessionBean() {  
        return new AnotherClass();  
    }  
  
    @Bean  
    @Scope("websocket")  
    public AnotherClass myWebSocketBean() {  
        return new AnotherClass();  
    }  
}
```

Chapter 10: RestTemplate

Section 10.1: Downloading a Large File

The `getForObject` and `getForEntity` methods of `RestTemplate` load the entire response in memory. This is not suitable for downloading large files since it can cause out of memory exceptions. This example shows how to stream the response of a GET request.

```
RestTemplate restTemplate = ...  
  
// Optional Accept header  
RequestCallback requestCallback = request -> request.getHeaders().  
    .getAccept(AcceptHeaderList(MediaType.APPLICATION_JSON_STREAM, MediaType.ALL));  
  
// Streams the response instead of loading it all in memory  
ResponseExtractor<Void> responseExtractor = response -> {  
    // Here I write the response to a file but do what you like  
    Path path = Paths.get("some/path");  
    Files.copy(response.getBody(), path);  
    return null;  
};  
  
restTemplate.execute(url, create("www.something.com"), HttpMethod.GET, requestCallback,  
    responseExtractor);
```

Note that you cannot simply return the `InputStream` from the extractor, because by the time the `execute` method returns, the underlying connection and stream are already closed.

Section 10.2: Setting headers on Spring RestTemplate request

The exchange methods of `RestTemplate` allow you to specify a `HttpEntity` that will be written to the request when executing the method. You can add headers (such as `User-Agent`, referer...) to this entity.

```
public void testHeader1(final RestTemplate restTemplate) {  
    //Set the headers you need some  
    final HttpHeaders headers = new HttpHeaders();  
    headers.set("User-Agent", "r11111");  
  
    //Create a new HttpEntity  
    final HttpEntity<String> entity = new HttpEntity<String>(headers);  
  
    //Execute the method writing your HttpEntity to the request  
    ResponseEntity<Map> response = restTemplate.exchange("https://httpbin.org/user-agent",  
        HttpMethod.GET, entity, Map.class);  
    System.out.println(response.getBody());  
}
```

Also you can add an interceptor to your `RestTemplate` if you need to add the same headers to multiple requests.

```
public void testHeader2(final RestTemplate restTemplate) {  
    //Add a ClientHttpRequestInterceptor to the RestTemplate  
    restTemplate.getInterceptors().add(new ClientHttpRequestInterceptor() {  
        @Override  
        public ClientHttpRequestInterceptor execute(ClientHttpRequest request, byte[] body,  
            ClientHttpRequestExecution execution) throws IOException {  
            request.getHeaders().set("User-Agent", "r11111"); //Set the header for each request  
            return execution.execute(request, body);  
        }  
    });  
}
```

Spring Framework Notes for Professionals

Chapter 15: JdbcTemplate

The `JdbcTemplate` class executes SQL queries, update statements and stored procedure calls, performs iteration over ResultSets and extraction of returned parameter values. It also catches JDBC exceptions and translates them to the generic, more informative, exception hierarchy defined in the `org.springframework.dao` package. Instances of the `JdbcTemplate` class are thread-safe once configured so it can be safely inject this shared reference into multiple DAOs.

Section 15.1: Basic Query methods

Some of the `queryFor*` methods available in `JdbcTemplate` are useful for simple SQL statements that perform CRUD operations.

Querying for Date

```
String sql = "SELECT create_date FROM customer WHERE customer_id = ?";  
int storeId = jdbcTemplate.queryForObject(sql, java.util.Date.class, customerId);
```

Querying for Integer

```
String sql = "SELECT store_id FROM customer WHERE customer_id = ?";  
int storeId = jdbcTemplate.queryForObject(sql, Integer.class, customerId);
```

OR

```
String sql = "SELECT store_id FROM customer WHERE customer_id = ?";  
int storeId = jdbcTemplate.queryForInt(sql, customerId);
```

Querying for String

```
String sql = "SELECT first_name FROM customer WHERE customer_id = ?";  
String firstName = jdbcTemplate.queryForObject(sql, String.class, customerId);
```

Querying for List

```
String sql = "SELECT first_name FROM customer WHERE store_id = ?";  
List<String> firstNameList = jdbcTemplate.queryForList(sql, String.class, storeId);
```

Section 15.2: Query for List of Maps

```
int storeId = ...  
DataSource dataSource = ...  
JdbcTemplate jdbcTemplate = new JdbcTemplate(dataSource);  
String sql = "SELECT * FROM customer WHERE store_id = ?";  
List<Map<String, Object>> mapList = jdbcTemplate.queryForList(sql, storeId);  
for (Map<String, Object> entryMap : mapList) {  
    for (Entry<String, Object> entry : entryMap.entrySet()) {  
        System.out.println(entry.getKey() + " : " + entry.getValue());  
    }  
}
```

Spring Framework Notes for Professionals

50+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Spring Framework	2
Section 1.1: Setup (XML Configuration)	2
Section 1.2: Showcasing Core Spring Features by example	3
Section 1.3: What is Spring Framework, why should we go for it?	6
Chapter 2: Spring Core	8
Section 2.1: Introduction to Spring Core	8
Section 2.2: Understanding How Spring Manage Dependency?	9
Chapter 3: Spring Expression Language (SpEL)	12
Section 3.1: Syntax Reference	12
Chapter 4: Obtaining a SqlRowSet from SimpleJdbcCall	13
Section 4.1: SimpleJdbcCall creation	13
Section 4.2: Oracle Databases	14
Chapter 5: Creating and using beans	16
Section 5.1: Autowiring all beans of a specific type	16
Section 5.2: Basic annotation autowiring	17
Section 5.3: Using FactoryBean for dynamic bean instantiation	18
Section 5.4: Declaring Bean	19
Section 5.5: Autowiring specific bean instances with @Qualifier	20
Section 5.6: Autowiring specific instances of classes using generic type parameters	21
Section 5.7: Inject prototype-scoped beans into singletons	22
Chapter 6: Bean scopes	25
Section 6.1: Additional scopes in web-aware contexts	25
Section 6.2: Prototype scope	26
Section 6.3: Singleton scope	28
Chapter 7: Conditional bean registration in Spring	30
Section 7.1: Register beans only when a property or value is specified	30
Section 7.2: Condition annotations	30
Chapter 8: Spring JSR 303 Bean Validation	32
Section 8.1: @Valid usage to validate nested POJOs	32
Section 8.2: Spring JSR 303 Validation - Customize error messages	32
Section 8.3: JSR303 Annotation based validations in Springs examples	34
Chapter 9: ApplicationContext Configuration	37
Section 9.1: Autowiring	37
Section 9.2: Bootstrapping the ApplicationContext	37
Section 9.3: Java Configuration	38
Section 9.4: Xml Configuration	40
Chapter 10: RestTemplate	43
Section 10.1: Downloading a Large File	43
Section 10.2: Setting headers on Spring RestTemplate request	43
Section 10.3: Generics results from Spring RestTemplate	44
Section 10.4: Using Preemptive Basic Authentication with RestTemplate and HttpClient	44
Section 10.5: Using Basic Authentication with HttpComponent's HttpClient	46
Chapter 11: Task Execution and Scheduling	47
Section 11.1: Enable Scheduling	47
Section 11.2: Cron expression	47

Section 11.3: Fixed delay	49
Section 11.4: Fixed Rate	49
Chapter 12: Spring Lazy Initialization	50
Section 12.1: Example of Lazy Init in Spring	50
Section 12.2: For component scanning and auto-wiring	51
Section 12.3: Lazy initialization in the configuration class	51
Chapter 13: Property Source	52
Section 13.1: Sample xml configuration using PropertyPlaceholderConfigurer	52
Section 13.2: Annotation	52
Chapter 14: Dependency Injection (DI) and Inversion of Control (IoC)	53
Section 14.1: Autowiring a dependency through Java configuration	53
Section 14.2: Autowiring a dependency through XML configuration	53
Section 14.3: Injecting a dependency manually through XML configuration	54
Section 14.4: Injecting a dependency manually through Java configuration	56
Chapter 15: JdbcTemplate	57
Section 15.1: Basic Query methods	57
Section 15.2: Query for List of Maps	57
Section 15.3: SQLRowSet	58
Section 15.4: Batch operations	58
Section 15.5: NamedParameterJdbcTemplate extension of JdbcTemplate	59
Chapter 16: SOAP WS Consumption	60
Section 16.1: Consuming a SOAP WS with Basic auth	60
Chapter 17: Spring profile	61
Section 17.1: Spring Profiles allows to configure parts available for certain environment	61
Chapter 18: Understanding the dispatcher-servlet.xml	62
Section 18.1: dispatcher-servlet.xml	62
Section 18.2: dispatcher servlet configuration in web.xml	62
Credits	64
You may also like	65

About

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:

<https://goalkicker.com/SpringFrameworkBook>

This *Spring® Framework Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow. Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official Spring® Framework group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

Chapter 1: Getting started with Spring Framework

Version Release Date

5.0.x	2017-10-24
4.3.x	2016-06-10
4.2.x	2015-07-31
4.1.x	2014-09-14
4.0.x	2013-12-12
3.2.x	2012-12-13
3.1.x	2011-12-13
3.0.x	2009-12-17
2.5.x	2007-12-25
2.0.x	2006-10-04
1.2.x	2005-05-13
1.1.x	2004-09-05
1.0.x	2003-03-24

Section 1.1: Setup (XML Configuration)

Steps to create Hello Spring:

1. Investigate Spring Boot to see if that would better suit your needs.
2. Have a project set up with the correct dependencies. It is recommended that you are using Maven or Gradle.
3. create a POJO class, e.g. `Employee.java`
4. create a XML file where you can define your class and variables. e.g `beans.xml`
5. create your main class e.g. `Customer.java`
6. Include [spring-beans](#) (and its transitive dependencies!) as a dependency.

`Employee.java`:

```
package com.test;

public class Employee {

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void displayName() {
        System.out.println(name);
    }
}
```

`beans.xml`:

[Click here to download full PDF material](#)