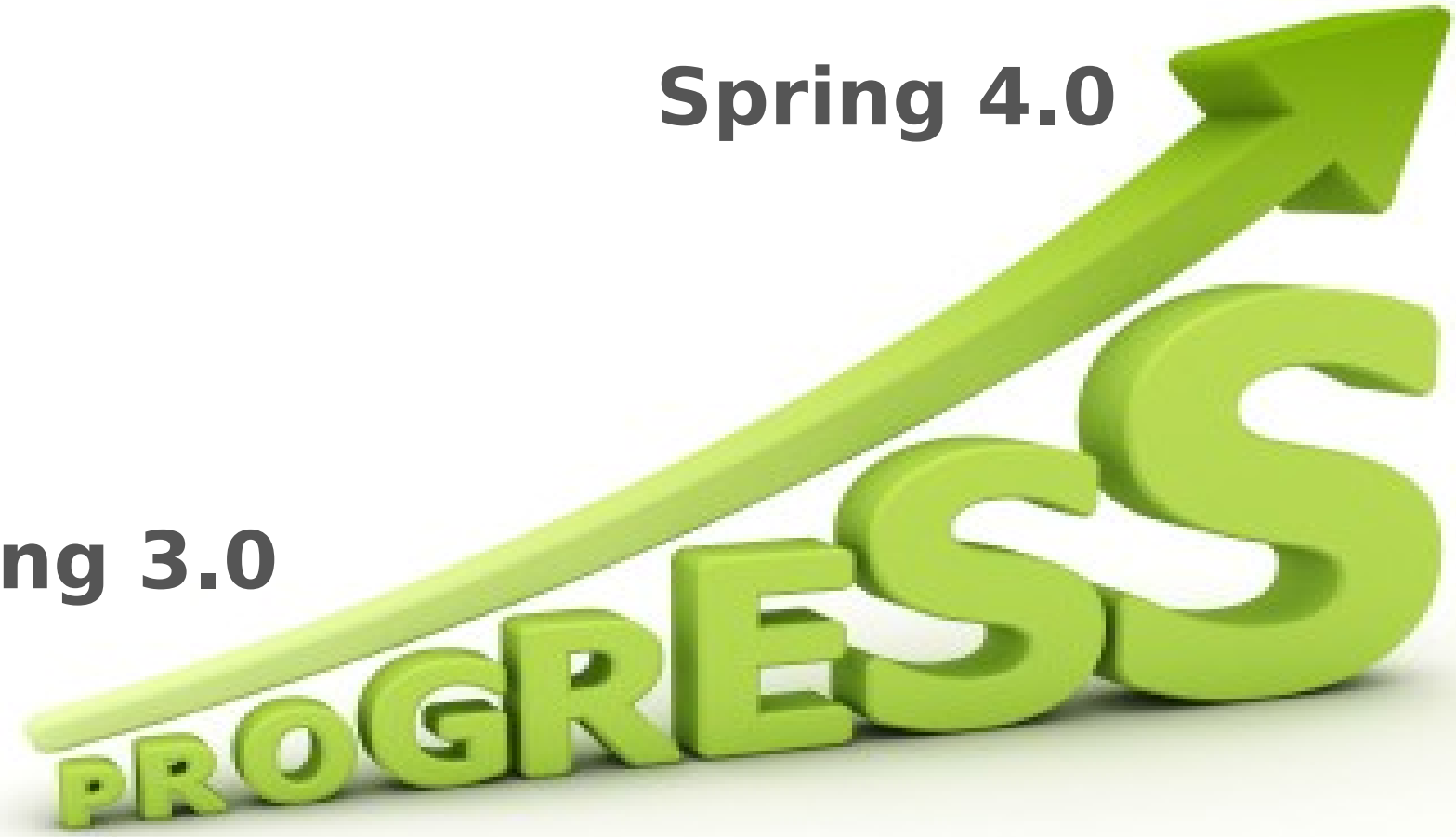
Spring Versions

Spring 4.2

Spring 4.0

Spring 3.0



Sumit Sinhmar Gole
Xebia Architects India pvt ltd

Spring 3.0
(past stop)

- Application configuration using Java.
- Servlet 3.0 API support
- Support for Java SE 7
- @MVC additions
- Declarative model validation
- Embedded database support
- Many more....

Spring 4.0
(Current Stop)

- Fully support for Java8 features.
  - Java 6 <= (Spring) = Java 8
- Removed depricated packages and methods.
- Groovy Bean Definition DSL.
- Core Container Improvements.
- General Web Improvements.
- WebSocket, SockJS, and STOMP Messaging.
- Testing Improvements.

Java 8.0 Support

- Use of lambda expressions.
  - In Java 8 a lambda expression can be used anywhere a functional interface is passed into a method.

    Example:

```
public interface RowMapper<T>
{
  T mapRow(ResultSet rs, int rowNum) throws SQLException;
}
```

Java 8.0 Support

- For example the Spring JdbcTemplate class contains a method.

```
public <T> List<T> query(String sql, RowMapper<T> rowMapper)
    throws DataAccessException
```

- Implementation:

```
jdbcTemplate.query("SELECT * from queries.products", (rs, rowNum) -> {    Integer id = rs.getInt("id");
String description = rs.getString("description");});
```

Java 8.0 Support

- Time and Date API
  - Spring 4 updated the date conversion framework (data to java object <--> Java object to data) to support JDK 8.0 Time and Date APIs.
  - @RequestMapping("/date/{localDate}")
  - public String get(@DateTimeFormat(iso = ISO.DATE) LocalDate localDate)
  - {
  - return localDate.toString();
  - }
  - Spring 4 support Java 8 but that does not imply the third party frameworks like Hibernate or Jakson would also support. So be carefull.....

Java 8.0 Support

- Repeating Annotations
  - Spring 4 respected the repeating annotatiuon convention intrduced by Java 8.
  - Example:

```
@PropertySource("classpath:/example1.properties")
@PropertySource("classpath:/example2.properties")
public class Application {
```

Java 8.0 Support

- Checking for null references in methods is always big pain....
  - Example:

```
public interface EmployeeRepository extends CrudRepository<Employee,
Long> {
    /**
     * returns the employee for the specified id or
     * null if the value is not found
    */
    public Employee findEmployeeById(String id);
}
```

Calling this method as:

```
Employee employee = employeeRepository.findEmployeeById("123");
employee.getName(); // get a null pointer exception
```

Java 8.0 Support

- Magic of java.util.Optional
  - public interface EmployeeRepository extends CrudRepository<Employee, Long> {
  - /**
  - * returns the employee for the specified id or
  - * null if the value is not found
  - */
  - public Optional<Employee> findEmployeeById(String id);
  - }

Java 8.0 Support

- Java.util.Optional force the programmer to put null check before extracting the information.
  - How?

```
Optional<Employee> optional =
employeeRepository.findEmployeeById("123");
if(optional.isPresent()) {
   Employee employee = optional.get();
   employee.getName();
}
```

Java 8.0 Support

- Spring 4.0 use java.util.Optional in following ways:
  - Option 1:
    - Old way :
      @Autowired(required=false)
      OtherService otherService;
    - New way:
      @Autowired
      - **Optional<OtherService>** otherService;
  - Option 2:
    @RequestMapping("/accounts/{accountId}",requestMethod=RequestMethod.POST)
    void update(**Optional<String> accountId**, @RequestBody Account account)

Java 8.0 Support

- Auto mapping of Parameter name
  - Java 8 support preserve method arguments name in compiled code so spring4 can extract the argument name from compiled code.
    - What I mean?

```
@RequestMapping("/accounts/{id}")
public Account getAccount(@PathVariable("id") String id)
```

    - can be written as

```
@RequestMapping("/accounts/{id}")
public Account getAccount(@PathVariable String id)
```

Java 8.0 support

Hands on!!!!

Groovy Bean Definition DSL

- @Configuration is empowered with Groovy Bean Builder.
  - Spring 4.0 provides Groovy DSL to configur Spring applications.
    - How it works?

```
<bean id="testBean" class="com.xebia.TestBeanImpl">
        <property name="someProperty" value="42"/>
        <property name="anotherProperty" value="blue"/>
    </bean>
```

```
import my.company.MyBeanImpl
beans = {
    myBean(MyBeanImpl) {
        someProperty = 42
        otherProperty = "blue"
    }
}
```

Groovy Bean Definition DSL

- Additionally it allows to configure Spring bean defination properties.
  - How?
    - sessionFactory(ConfigurableLocalSessionFactoryBean) { bean ->
    - // Sets the initialization method to 'init'. [init-method]
    - bean.initMethod = 'init'
    - // Sets the destruction method to 'destroy'. [destroy-method]
    - bean.destroyMethod = 'destroy'
  - Spring + Groovy ---> More Strong and clean implementation.

Groovy Bean Definition DSL

- How both work together?

```
beans {
  //
org.springframework.beans.factory.groovy.GroovyBeanDefinitionReader
    if (environment.activeProfiles.contains("prof1")) {
        foo String, 'hello'
    } else {
        foo String, 'world'
    }
    }
```

- In above code DSL uses GroovyBeanDefinitionReader to interpret Groovy code.

Groovy Bean Definition DSL

- If the bean defined in the previous code is placed in the file config/contextWithProfiles.groovy the following code can be used to get the value for String foo.

Example:

```
ApplicationContext context = new
GenericGroovyApplicationContext("file:config/contextWithPro
files.groovy");
String foo =  context.getBean("foo",String.class);
```

Groovy Bean Definition DSL

Hands on!!!!

Core Container Improvements.

Meta Annotations support

- Defining custom annotations that combines many spring annotations.
  - Example

```
import org.springframework.context.annotation.Scope;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional
@Repository
@Scope("prototype")
@Transactional(propagation = Propagation.REQUIRES_NEW,
timeout = 30, isolation=Isolation.SERIALIZABLE)
public class OrderDaoImpl implements OrderDao {
.....}
```

- Without writing any logic we have to repeat the above code in many classes!!!!!

Core Container Improvements.

- Spring 4 custom annotation
  - import org.springframework.context.annotation.Scope;
  - import org.springframework.stereotype.Repository;
  - import org.springframework.transaction.annotation.Isolation;
  - import org.springframework.transaction.annotation.Propagation;
  - import org.springframework.transaction.annotation.Transactional;
  -
  - @Repository
  - @Scope("prototype")
  - @Transactional(propagation = Propagation.REQUIRES_NEW, timeout = 30, isolation=Isolation.SERIALIZABLE)
  - public @interface MyDao {
  - }

Core Container Improvements.

- Spring 4 custom annotation

  How to use the custom annotation?

```
@MyDao
public class OrderDaoImpl implements OrderDao {
    ...
}
```

Core Container Improvements

Generic qualifiers

Java genric types can be used as implicit form of qualification.

How?

```
public class Dao<T> {
  ...
}
@Configuration
public class MyConfiguration {

  @Bean
  public Dao<Person> createPersonDao() {
    return new Dao<Person>();
  }

  @Bean
  public Dao<Organization> createOrganizationDao() {
    return new Dao<Organization>();
  }
}
```

Core Container Improvements

- How to call?
  @Autowired
  private Dao<Person> dao;

- Spring 4 container identify which bean to inject on the bases of Java generics.

Core Container Improvements

- Conditionally bean defination
  - Conditionally enable and disable bean defination or whole configuration.
  - Spring 3.x

```
@Configuration
   @Profile("Linux")
   public class LinuxConfiguration {

    @Bean
    public EmailService emailerService() {
      return new LinuxEmailService();
    }
   }
```

```
@Configuration
  @Profile(“Window”)
  public class indowConfiguration{

   @Bean
  public EmailService emailerService(){
   return new WindowEmailService();
  }
}
```

Core Container Improvements

@Conditional is a flexible annotation, is consulted by the container before @Bean is registered.

How?

The Conditional interface method

@Override public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) is overridden.

context : provides access to the environment, class loader and container.

metadata: metadata of the class being checked.

Core Container Improvements

- The previous example would be implemented as

```
public class LinuxCondition implements Condition{
@Override
  public boolean matches(ConditionContext context, AnnotatedTypeMetadata
metadata) {
    return context.getEnvironment().getProperty("os.name").contains("Linux");  }
}

public class WindowsCondition implements Condition{

  @Override
  public boolean matches(ConditionContext context, AnnotatedTypeMetadata
metadata) {
    return context.getEnvironment().getProperty("os.name").contains("Windows");
  }
}
```

Core Container Improvements

```
@Configuration
public class MyConfiguration {
  @Bean(name="emailerService")
  @Conditional(WindowsCondition.class)
  public EmailService windowsEmailerService(){
      return new WindowsEmailService();
  }
 @Bean(name="emailerService")
  @Conditional(LinuxCondition.class)
  public EmailService linuxEmailerService(){
    return new LinuxEmailService();
  }
}
```

Core Container Improvements

- @Order and Autowiring
  - Beans can be orderly wired by adding the @Order annotation in the @Bean implementations as follows

```
@Component
@Order(value=1)
public class Employee implements Person {
    ..
}
@Component
@Order(value=2)
public class Customer implements Person {
    ......
}
Using the
    @Autowired
    List<Person> people;
Results in
    [com.intertech.Employee@a52728a, com.intertech.Customer@2addc751]
```

Core Container Improvements

Hands on!!!!

General Web Improvments

- New @RestController annotation with Spring MVC application provide specific implementation for RestFull web services, removing requirement to add @ResponseBody to each of your @RequestMapping.
- Spring 3.x

```
@Controller public class SpringContentController
{
    @Autowired UserDetails userDetails;
    @RequestMapping(value="/springcontent",
    method=RequestMethod.GET,produces={"application/xml", "application/json"})
    @ResponseStatus(HttpStatus.OK)
    public @ResponseBody UserDetails getUser() {
        UserDetails userDetails = new UserDetails();
    userDetails.setUserName("Krishna"); userDetails.setEmailId("krishna@gmail.com");
    return userDetails;
    }
```

General Web Improvments

With @RestController annotation

- @Rest**Controller** public class SpringContentController

```
{
    @Autowired UserDetails userDetails;
    @RequestMapping(value="/springcontent",
    method=RequestMethod.GET,produces={"application/xml",
    "application/json"}) @ResponseStatus(HttpStatus.OK)
    public UserDetails getUser()
    {
        UserDetails userDetails = new UserDetails();
    userDetails.setUserName("Krishna");
    userDetails.setEmailId("krishna@gmail.com"); return
    userDetails;
    }
```

General Web Improvments

- Jakson integration improvements.
  - Filter the serialized object in the Http Response body with Jakson Serialization view.
  - @JsonView annotation is used to filter the fields depending on the requirement. e.g. When getting the Collection in Http response then pass only the summary and if single object is requested then return the full object.
  - Example

```
public class View {
    interface Summary {}
}
public class User {

    @JsonView(View.Summary.class)
    private Long id;

    @JsonView(View.Summary.class)
    private String firstname;

    @JsonView(View.Summary.class)
    private String lastname;
    private String email;
   private String address;
}
```

General Web Improvments

```
public class Message {

    @JsonView(View.Summary.class)
    private Long id;
    @JsonView(View.Summary.class)
    private LocalDate created;
    @JsonView(View.Summary.class)
    private String title;
    @JsonView(View.Summary.class)
    private User author;

    private List<User> recipients;

    private String body;
}
```

General Web Improvments

- In the controller

```
@Autowired
    private MessageService messageService;
    @JsonView(View.Summary.class)
    @RequestMapping("/")
    public List<Message> getAllMessages() {
        return messageService.getAll();
    }
```

- Output:

```
[ {
  "id" : 1,
  "created" : "2014-11-14",
  "title" : "Info",
  "author" : {
    "id" : 1,
    "firstname" : "Brian",
    "lastname" : "Clozel"
  }
}, {
  "id" : 2,
  "created" : "2014-11-14",
  "title" : "Warning",
  "author" : {
    "id" : 2,
    "firstname" : "Stéphane",
    "lastname" : "Nicoll"
  }
}
```

General Web Improvments

- @ControllerAdvice improvements
  - Can be configured to support defined subset of controllers through annotations, basePackageClasses, basePackages.
    - Example:

```
@Controller
@RequestMapping("/api/article")
class ArticleApiController {

    @RequestMapping(value = "{articleId}", produces = "application/json")
    @ResponseStatus(value = HttpStatus.OK)
    @ResponseBody
    Article getArticle(@PathVariable Long articleId) {
        throw new IllegalArgumentException("[API] Getting article problem.");
    }
}
```

General Web Improvments

- Exception handler handling the api request can be restricted as follows:

```
@ControllerAdvice(annotations = RestController.class)
class ApiExceptionHandlerAdvice {

    /**
     * Handle exceptions thrown by handlers.
     */
    @ExceptionHandler(value = Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    @ResponseBody
    public ApiError exception(Exception exception, WebRequest request) {
        return new ApiError(Throwables.getRootCause(exception).getMessage());
    }
}
```

General Web Improvments

Hands on!!!!

Testing Improvements

- Active @Bean on the bases of profile can be resolved programmatically by ActiveProfilesResolver and registering it via resolver attribute of @ActiveProfiles.

Example:

Scenario : If we want to do testing across the environments (dev, stagging etc.) for integration testing.

Step 1 : Set the active profile

```
<container>
                    <containerId>tomcat7x</containerId>
                    <systemProperties>

<spring.profiles.active>development</spring.profiles.active>
                    </systemProperties>
</container>
```

Testing improvements

```
public class SpringActiveProfileResolver implements
ActiveProfilesResolver {

    @Override
    public String[] resolve(final Class<?> aClass) {
        final String activeProfile System.getProperty("spring.profiles.active");
        return new String[] { activeProfile == null ? "integration" :
activeProfile };
    }
```

- And in the test class we use the annotation.

```
@ActiveProfiles(resolver = SpringActiveProfileResolver.class)
public class IT1DataSetup {
......
}
```

Testing improvements

- SocketUtils class introduced in the spring-core module allow to start an in memory SMTP server, FTP server, servlet container etc to enable integration testing that require use of sockets.
  - Example:

```
@Test public void testErrorFlow() throws Exception {
final int port=SocketUtils.findAvailableServerSocket();
  AbstractServerConnectionFactory scf=new
TcpNetServerConnectionFactory(port);
  scf.setSingleUse(true);
  TcpInboundGateway gateway=new TcpInboundGateway();
  gateway.setConnectionFactory(scf);
```

Testing improvemments

Hands on!!!!