Modern Java Component Design with Spring Framework 4.3

Juergen Hoeller
Spring Framework Lead
Pivotal

The State of the Art: Component Classes

```
@Service
@Lazy
public class MyBookAdminService implements BookAdminService {

    // @Autowired
    public MyBookAdminService(AccountRepository repo) {
        ...
    }

    @Transactional
    public BookUpdate updateBook(Addendum addendum) {
        ...
    }
}
```

Composable Annotations

```
@Service
@Scope("session")
@Primary
@Transactional(rollbackFor=Exception.class)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyService {}


@MyService
public class MyBookAdminService {
    ...
}
```

Composable Annotations with Overridable Attributes

```
@Scope(scopeName="session")
@Retention(RetentionPolicy.RUNTIME)
public @interface MySessionScoped {

    @AliasFor(annotation=Scope.class, attribute="proxyMode")
    ScopedProxyMode mode() default ScopedProxyMode.NO;
}
```

```
@Transactional(rollbackFor=Exception.class)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyTransactional {

    @AliasFor(annotation=Transactional.class)
    boolean readOnly();
}
```

Convenient Scoping Annotations (out of the box)

- **Web scoping annotations coming with Spring Framework 4.3**
  - @RequestScope
  - @SessionScope
  - @ApplicationScope

- **Special-purpose scoping annotations across the Spring portfolio**
  - @RefreshScope
  - @JobScope
  - @StepScope

spring by Pivotal

The State of the Art: Configuration Classes

```
@Configuration
@Profile("standalone")
@EnableTransactionManagement
public class MyBookAdminConfig {

    @Bean
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }

    @Bean
    public BookAdminService bookAdminDataSource() {
        ...
    }
}
```

Configuration Classes with Autowired Bean Methods

```
@Configuration
@Profile("standalone")
@EnableTransactionManagement
public class MyBookAdminConfig {

    @Bean
    public BookAdminService myBookAdminService(
            DataSource bookAdminDataSource) {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource);
        return service;
    }
}
```

Configuration Classes with Autowired Constructors

```
@Configuration
public class MyBookAdminConfig {

    private final DataSource bookAdminDataSource;

    // @Autowired
    public MyBookAdminService(DataSource bookAdminDataSource) {
        this.bookAdminDataSource = bookAdminDataSource;
    }

    @Bean
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(this.bookAdminDataSource);
        return service;
    }
}
```

Configuration Classes with Base Classes

```
@Configuration
public class MyApplicationConfig extends MyBookAdminConfig {

    ...
}


public class MyBookAdminConfig {

    @Bean
    public BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }
}
```

Configuration Classes with Java 8 Default Methods

```
@Configuration
public class MyApplicationConfig implements MyBookAdminConfig {

    ...
}


public interface MyBookAdminConfig {

    @Bean
    default BookAdminService myBookAdminService() {
        MyBookAdminService service = new MyBookAdminService();
        service.setDataSource(bookAdminDataSource());
        return service;
    }
}
```

Generics-based Injection Matching

```
@Bean
public MyRepository<Account> myAccountRepository() { ... }

@Bean
public MyRepository<Product> myProductRepository() { ... }


@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(MyRepository<Account> repo) {
        // specific match, even with other MyRepository beans around
    }
}
```

Ordered Collection Injection

```
@Bean @Order(2)
public MyRepository<Account> myAccountRepositoryX() { ... }

@Bean @Order(1)
public MyRepository<Account> myAccountRepositoryY() { ... }


@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(List<MyRepository<Account>> repos) {
        // 'repos' List with two entries: repository Y first, then X
    }
}
```

Injection of Collection Beans

```
@Bean
public List<Account> myAccountList() { ... }


@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(List<Account> repos) {
        // if no raw Account beans found, looking for a
        // bean which is a List of Account itself
    }
}
```

Lazy Injection Points

```
@Bean @Lazy
public MyRepository<Account> myAccountRepository() {
    return new MyAccountRepositoryImpl();
}
```

```
@Service
public class MyBookAdminService implements BookAdminService {

    @Autowired
    public MyBookAdminService(@Lazy MyRepository<Account> repo) {
        // 'repo' will be a lazy-initializing proxy
    }
}
```

Component Declarations with JSR-250 & JSR-330

```
import javax.annotation.*;
import javax.inject.*;

@ManagedBean
public class MyBookAdminService implements BookAdminService {

    @Inject
    public MyBookAdminService(Provider<MyRepository<Account>> repo) {
        // 'repo' will be a lazy handle, allowing for .get() access
    }

    @PreDestroy
    public void shutdown() {
        ...
    }
}
```

Optional Injection Points on Java 8

```
import java.util.*;
import javax.annotation.*;
import javax.inject.*;

@ManagedBean
public class MyBookAdminService implements BookAdminService {

    @Inject
    public MyBookAdminService(Optional<MyRepository<Account>> repo) {
        if (repo.isPresent()) { ... }
    }

    @PreDestroy
    public void shutdown() {
        ...
    }
}
```

Declarative Formatting with Java 8 Date-Time

```
import java.time.*;
import javax.validation.constraints.*;
import org.springframework.format.annotation.*;

public class Customer {

    // @DateTimeFormat(iso=ISO.DATE)
    private LocalDate birthDate;

    @DateTimeFormat(pattern="M/d/yy h:mm")
    @NotNull @Past
    private LocalDateTime lastContact;

    ...
}
```

Annotated MVC Controllers

```
@Controller
@CrossOrigin
public class MyRestController {

    @RequestMapping(path="/books/{id}", method=GET)
    public Book findBook(@PathVariable long id) {
        return this.bookAdminService.findBook(id);
    }

    @RequestMapping("/books/new")
    public void newBook(@Valid Book book) {
        this.bookAdminService.storeBook(book);
    }
}
```

Precomposed Annotations for MVC Controllers

```
@RestController
@CrossOrigin
public class MyRestController {

    @GetMapping("/books/{id}")
    public Book findBook(@PathVariable long id) {
        return this.bookAdminService.findBook(id);
    }

    @PostMapping("/books/new")
    public void newBook(@Valid Book book) {
        this.bookAdminService.storeBook(book);
    }
}
```

STOMP on WebSocket

```
@Controller
public class MyStompController {

    @SubscribeMapping("/positions")
    public List<PortfolioPosition> getPortfolios(Principal user) {
        ...
    }

    @MessageMapping("/trade")
    public void executeTrade(Trade trade, Principal user) {
        ...
    }
}
```

Annotated JMS Endpoints

```
@JmsListener(destination="order")
public OrderStatus processOrder(Order order) {
    ...
}

@JmsListener(id="orderListener", containerFactory="myJmsFactory",
    destination="order", selector="type='sell'", concurrency="2-10")
@SendTo("orderStatus")
public OrderStatus processOrder(Order order, @Header String type) {
    ...
}
```

Annotated Event Listeners

```
@EventListener
public void processEvent(MyApplicationEvent event) {
    ...
}

@EventListener
public void processEvent(String payload) {
    ...
}

@EventListener(condition="#payload.startsWith('OK')")
public void processEvent(String payload) {
    ...
}
```

Spring Framework 4.3 Roadmap

- **Last 4.x feature release!**
- **4.3 RC1: March 2016**
- **4.3 GA: May 2016**

- **Extended support life until 2020**
  - on JDK 6, 7, 8 and 9
  - on Tomcat 6, 7, 8 and 9
  - on WebSphere 7, 8.0, 8.5 and 9

- **Spring Framework 5.0 coming in 2017**
  - on JDK 8+, Tomcat 7+, WebSphere 8.5+

Learn More. Stay Connected.

- Current: Spring Framework 4.2.5 (February 2016)
- Coming up: Spring Framework 4.3 (May 2016)
- Check out Spring Boot! http://projects.spring.io/spring-boot/

Twitter: twitter.com/springcentral

YouTube: spring.io/video

LinkedIn: spring.io/linkedin

Google Plus: spring.io/gplus