

What's New in Spring 3.0

Introducing the new features added in the
Spring Framework 3.0 release

- **Review: Spring 2.5**
- Java and annotations
- Spring EL
- JavaConfig
- REST support (client and server)
- Threading and Scheduling
- Embedded database support
- Everything else!

- Comprehensive support for **annotation-based configuration**
 - @Autowired
 - with optional @Qualifier or custom qualifier
 - @Transactional
 - @Service, @Repository, @Controller
- Common **Java EE 5** annotations supported too
 - @PostConstruct, @PreDestroy
 - @PersistenceContext, @PersistenceUnit
 - @Resource, @EJB, @WebServiceRef
 - @TransactionAttribute

Spring 2.5: Annotated Bean Component



```
@Service
public class RewardNetworkService
    implements RewardNetwork {

    @Autowired
    public RewardNetworkService(AccountRepository ar) {
        ...
    }

    @Transactional
    public RewardConfirmation rewardAccountFor(Dining d) {
        ...
    }
}
```

Spring 2.5: Minimal XML Bean Definitions



```
<!-- Activating annotation-based configuration -->
```

```
<context:annotation-config/>
```

```
<bean class="com.myapp.rewards.RewardNetworkImpl"/>
```

```
<bean class="com.myapp.rewards.JdbcAccountRepository"/>
```

```
<!-- Plus shared infrastructure configuration beans:  
PlatformTransactionManager, DataSource, etc -->
```

Spring 2.5: Test Context Framework



```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration
public class RewardSystemIntegrationTests {

    @Autowired
    private RewardNetwork rewardNetwork;

    @Test
    @Transactional
    public void testRewardAccountForDining() {
        // test in transaction with auto-rollback
    }
}
```

Spring 2.5: @MVC Controller Style



```
@Controller
```

```
public class MyController {
```

```
    private final MyService myService;
```

```
@Autowired
```

```
public MyController(MyService myService) {
```

```
    this.myService = myService;
```

```
}
```

```
@RequestMapping("/removeBook")
```

```
public String removeBook(@RequestParam("book")
```

```
    String bookId) {
```

```
    this.myService.deleteBook(bookId);
```

```
    return "redirect:myBooks";
```

```
}
```

```
}
```

Topics in this session



- Review: Spring 2.5
- **Java and annotations**
- Spring EL
- JavaConfig
- REST support (client and server)
- Threading and Scheduling
- Embedded database support
- Everything else!

- Spring 3.0 **requires Java 5+**
 - Increased focus on annotations
 - APIs modified to support generics and varargs
- Support for **JUnit3 deprecated**
- Better integrated support for threading and scheduling
 - Eg. Spring's TaskExecutor interface now extends `java.util.concurrent.Executor`

- BeanFactory interface returns typed bean instances as far as possible
 - T getBean(String name, Class<T> requiredType)
 - Map<String, T> getBeansOfType(Class<T> type)
- Typed ApplicationListener<E>
 - ApplicationEventMulticaster detects declared event type and filters accordingly
- Templates modified
 - Eg. JmsTemplate: T execute(SessionCallback<T> sc)

Java: New configuration annotations



- Autowiring scalar values with @Value

```
public class MyClassWithValues {
```

```
    @Value("#{myprops.dbaseUrl}")
```

```
    private String databaseUrl;
```

```
    @Value("#{myprops.dbaseUser}")
```

```
    private String databaseUser;
```

```
}
```

```
<context:property-placeholder
```

```
    location = "dbase.properties"/>
```

Java: New configuration annotations



- More powerful options for custom annotations
 - combining meta-annotations e.g. on stereotype
 - automatically detected (no configuration necessary!)

```
@Service
```

```
@Scope("request")
```

```
@Transactional(rollbackFor=Exception.class)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface MyService { }
```

```
@MyService
```

```
public class RewardsService {
```

```
    ...
```

```
}
```

Topics in this session



- Review: Spring 2.5
- Java and annotations
- **Spring EL**
- JavaConfig
- REST support (client and server)
- Threading and Scheduling
- Embedded database support
- Everything else!

- **Custom expression parser** implementation shipped as part of Spring 3.0
 - package `org.springframework.expression`
 - next-generation expression engine
 - inspired by Spring Web Flow 2.0's expression support
 - pluggable and customizable
- **Compatible with Unified EL but significantly more powerful**
 - navigating bean properties, maps, etc
 - method invocations
 - construction of value objects

```
<bean class="mycompany.RewardsTestDatabase">  
  <property name="databaseName"  
    value="#{systemProperties.databaseName}"/>  
  <property name="keyGenerator"  
    value="#{strategyBean.databaseKeyGenerator}"/>  
</bean>
```

@Repository

public class RewardsTestDatabase {

@Value("#{systemProperties.databaseName}")

public void setDatabaseName(String dbName) {...}

@Value("#{strategyBean.databaseKeyGenerator}")

public void setKeyGenerator(KeyGenerator kg) {...}

}

- Example showed access to **EL attributes**
 - "systemProperties", "strategyBean"
 - implicit references in expressions
- **Implicit attributes** to be exposed by default, depending on runtime context
 - e.g. "systemProperties", "systemEnvironment"
 - global platform context
 - access to all Spring-defined beans by name
 - similar to managed beans in JSF expressions
 - extensible through Scope SPI
 - e.g. for step scope in Spring Batch

- Implicit **web-specific attributes** to be exposed by default as well
 - "contextParameters": web.xml init-params
 - "contextAttributes": ServletContext attributes
 - "request": current Servlet/PortletRequest
 - "session": current Http/PortletSession
- Exposure of all implicit JSF objects when running within a JSF request context
 - "param", "initParam", "facesContext", etc
 - full compatibility with JSF managed bean facility

Topics in this session



- Review: Spring 2.5
- Java and annotations
- Spring EL
- **JavaConfig**
- REST support (client and server)
- Threading and Scheduling
- Embedded database support
- Everything else!

- Spring 3.0 includes the core functionality of the Spring JavaConfig project
 - **configuration classes** defining managed beans
 - common handling of **annotated factory methods**
- Supported annotations:
 - @Configuration
 - @Bean
 - @Lazy
 - @DependsOn
 - @Value
 - @Primary

Annotated Factory Methods



@Configuration

```
public class AppConfig {
```

```
    @Value("#{jdbcProperties.url}") String jdbcUrl;
```

```
    @Bean @Primary @Lazy
```

```
    public RewardsService rewardsService() {
```

```
        RewardsServiceImpl service =
```

```
            new RewardsServiceImpl();
```

```
        service.setDataSource(...);
```

```
        return service;
```

```
    }
```

```
}
```

Topics in this session



- Review: Spring 2.5
- Java and annotations
- Spring EL
- JavaConfig
- **REST support (client and server)**
- Threading and Scheduling
- Embedded database support
- Everything else!

- Spring MVC to provide first-class support for **REST-style mappings**
 - extraction of URI template parameters
 - content negotiation in view resolver
- Goal: **native REST support** within Spring MVC, for UI as well as non-UI usage
 - in natural MVC style
- Alternative: **using JAX-RS** through integrated JAX-RS provider (e.g. Jersey)
 - using the JAX-RS component model to build programmatic resource endpoints

REST on the server: MVC @PathVariable



```
http://host.com/bookingsapp/hotels/1/bookings/2
```

```
@RequestMapping(
    value="/hotels/{hotel}/bookings/{booking}",
    method=GET)
public String getBooking(
    @PathVariable("hotel") long hotelId,
    @PathVariable("booking") long bookingId, Model model)
{
    Hotel hotel = hotelService.getHotel(hotelId);
    Booking booking = hotel.getBooking(bookingId);
    model.addAttribute("booking", booking);
    return "booking";
}
```

REST on the server: Data binding



```
http://host.com/bookingsapp/hotels/1/dates/2008-12-18
```

```
@RequestMapping(value="/hotels/{hotel}/dates/{date}")
```

```
public void date(@PathVariable("hotel") long hotelId,  
                 @PathVariable("date") Date date)
```

```
{
```

```
    ...
```

```
}
```

```
@InitBinder
```

```
public void initBinder(WebDataBinder binder) {
```

```
    SimpleDateFormat df =
```

```
        new SimpleDateFormat("yyyy-mm-dd");
```

```
    binder.registerCustomEditor(Date.class,
```

```
        new CustomDateEditor(df, false);
```

```
}
```

REST on the server: Additional Features



- HTTP Method Conversion
 - Browser POST becomes PUT or DELETE
 - Via HiddenHttpMethodFilter
 - Transparent to Java controller
- Shallow ETags support
 - Bandwidth savings
 - But same processing requirements

REST on the client: RestTemplate



```
<bean class="org.springframework.web.client.RestTemplate">
  <property name="messageConverters">
    <list>
      <bean class="org.springframework.http.converter.xml.
        SourceHttpMessageConverter"/>
    </list>
  </property>
</bean>
```

```
Source source = restTemplate.getForObject(
  Constants.PHOTO_SEARCH_URL, Source.class, key, var);
```

Topics in this session



- Review: Spring 2.5
- Java and annotations
- Spring EL
- JavaConfig
- REST support (client and server)
- **Threading and Scheduling**
- Embedded database support
- Everything else!

- Spring 3.0 introduces a **major overhaul of the scheduling package**
 - extended `java.util.concurrent` support
 - prepared for JSR-236 "Concurrency Utilities for Java EE"
 - TaskScheduler interface with Triggers
 - including **cron expression triggers**
 - `@Async` annotation for asynchronous user methods
 - `@Scheduled` for cron-triggered methods
- **XML scheduling namespace**
 - cron expressions with method references
 - convenient executor service setup

Threading and scheduling: @Async



```
public class AsyncStore {  
    @Async  
    public Future<Object> getValue(String key) {  
        Object result;  
        try {  
            Thread.sleep(1000);  
            result = theMap.get(key);  
        } catch (InterruptedException e) {...}  
        return new AsyncResult<Object>(result);  
    }  
}
```

```
Future<Object> asyncVal = new AsyncStore().getValue("foo");
```

Threading and scheduling: @Scheduled



```
public class ScheduledTask {
```

```
    @Scheduled(fixedRate=1000)
```

```
    public void bleepEverySecond() {
```

```
        bleepService.bleep();
```

```
    }
```

```
}
```

```
<bean class="org.mycompany.ScheduledTask">
```

```
<task:annotation-driven/>
```

Topics in this session



- Review: Spring 2.5
- Java and annotations
- Spring EL
- JavaConfig
- REST support (client and server)
- Threading and Scheduling
- **Embedded database support**
- Everything else!

- Support for HSQL, H2 and Apache Derby
- Lightweight DataSource for development

```
<jdbc:embedded-database id = "dataSource">  
  <jdbc:script location = "classpath:schema.sql">  
  <jdbc:script location = "classpath:test-data.sql">  
</jdbc:embedded-database>
```

- Use EmbeddedDatabaseBuilder to create an embedded database programmatically

EmbeddedDatabaseBuilder builder =

```
    new EmbeddedDatabaseBuilder();
```

EmbeddedDatabase db = builder.type(*H2*)

```
    .script("schema.sql").script("test-data.sql").build();
```

Topics in this session



- Review: Spring 2.5
- Java and annotations
- Spring EL
- JavaConfig
- REST support (client and server)
- Threading and Scheduling
- Embedded database support
- **Everything else!**

Everything else (1)

- New type conversion
 - Better than stateful PropertyEditors
- Revised OXM
 - Moved from Spring Web Services
 - support for REST-style payload conversions
- Support for serializing scoped proxies
- Spring MVC improvements
 - Eg. @RequestHeader and @Cookie annotations
- Portlet 2.0 support

Early Java EE 6 support:

- JSR-330 (@Inject)
 - Joint JSR from SpringSource and Google
 - Integrated with JSR-299
- JSR-303 (Declarative validation)
 - Based on Hibernate Validator 4
 - Validation Constraints using annotations
- JSF 2.0
- JPA 2.0