

# What's new in Spring 3.1?

Arjen Poutsma

@poutsma

SpringSource - a division of VMware

# Overview

- Spring 3.0
- Spring 3.1
- Release Schedule

# Spring 3.0

# Spring 3.0 Themes

- Java 5+
- Spring Expression Language
- REST support
- Declarative model validation
- Early support for Java EE 6

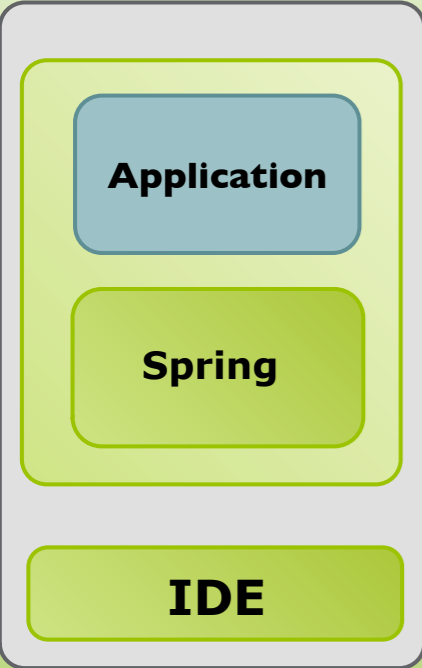
# Spring 3.1

# Spring 3.1 Themes

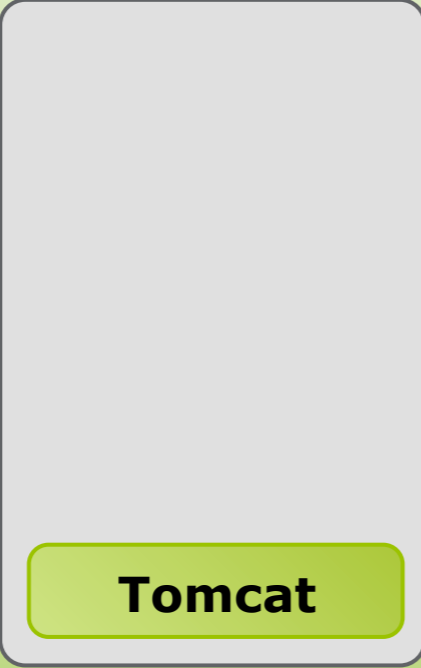
- Environment abstraction
- Java application configuration
- Cache abstraction
- @MVC improvements

# Environment Abstraction

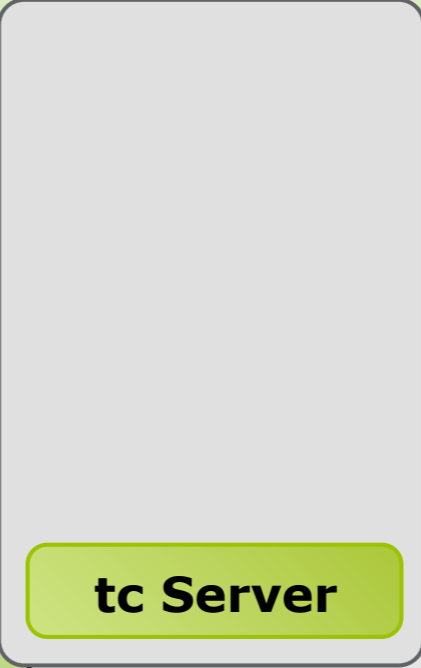
# Development



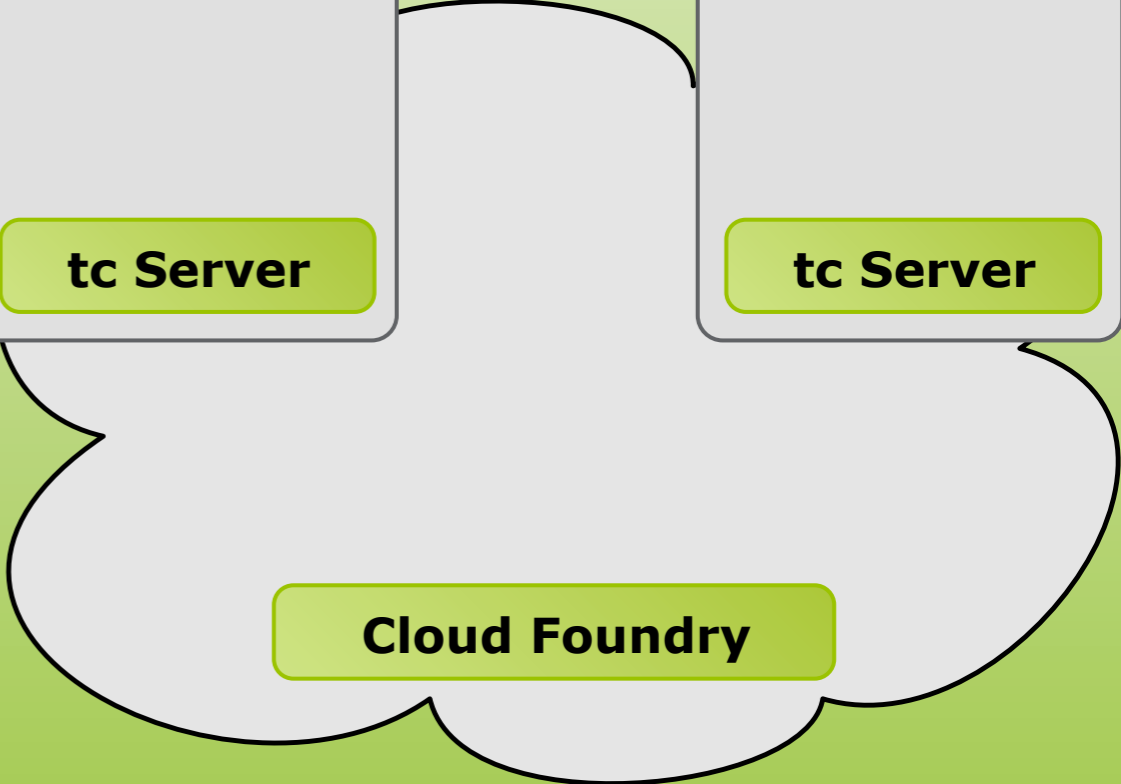
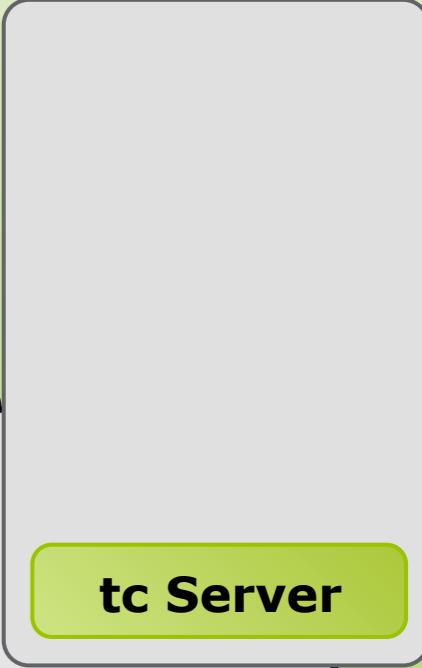
# Integration Testing



# Pre-Production



# Production

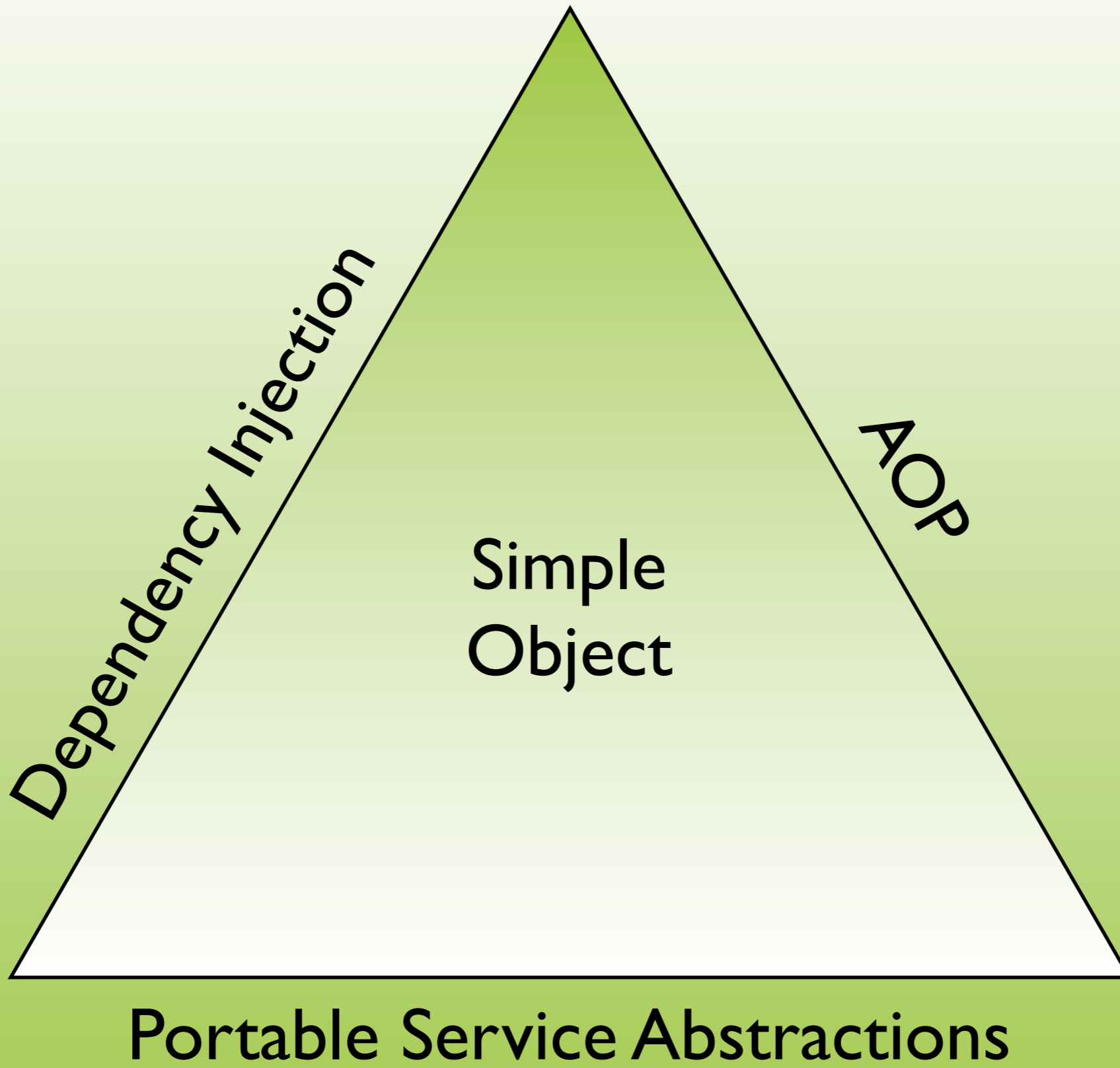


# The Environment Challenge

- Various environments
  - Development in IDE
  - Integration testing in container
  - Pre-production in container
  - Production
  - New: the Cloud
- Bonus challenge: immutable artefacts

# Current solutions

- JNDI
  - Only works in container
- System Properties
  - Requires access to environment
- Separate application contexts
  - Separate artefacts
  - Requires access to environment
  - Relatively confusing




# Environment Abstraction

- Group bean definitions by profile
  - e.g. development, testing, production
- Specify which environment to use

# Group Beans

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="..."
  profile="dev">
  </beans>
```



```
<beans xmlns="http://www.springframework.org/schema/beans"
  xsi:schemaLocation="...">
  <beans profile="dev"> <!-- ... --> </beans>
  <beans profile="prod"> <!-- ... --> </beans>
</beans>
```



# Grouping Beans (2)


```
@Profile("dev")
@PropertySource("classpath:/com/company/app/db.properties")
@Configuration
public class DbConfig {

    @Inject Environment env;

    @Bean
    public DataSource dataSource() {
        SimpleDriverDataSource dataSource = new SimpleDriverDataSource();
        dataSource.setDriverClass(env.getPropertyAsClass("db.driverClass"));
        dataSource.setUrl(env.getProperty("db.url"));
        dataSource.setUsername(env.getProperty("db.username"));
        dataSource.setPassword(env.getProperty("db.password"));
    }
}
```

# Activate Profile

```
GenericXmlApplicationContext context = new GenericXmlApplicationContext();  
context.getEnvironment().setActiveProfiles("production");  
context.refresh();
```



```
<servlet>  
  <servlet-name>dispatcher</servlet-name>  
  <servlet-class>org.springframework...DispatcherServlet</servlet-class>  
  <init-param>  
    <param-name>spring.profiles.active</param-name>  
    <param-value>production</param-value>  
  </init-param>  
</servlet>
```



# Activate Profile (2)

- System properties
  - -Dspring.profiles.active="dev"
  - -Dspring.profiles.default="common"
- TestContext Framework

```
@RunWith(SpringJUnit4ClassRunner.class)
@Configuration
@ActiveProfiles("dev")
public class TransferServiceTest {

    @Autowired
    TransferService transferService;

    @Test
    public void testTransferService() {
        // test the transferService
    }
}
```

# Java Application Configuration

# Java-Based Application Configuration

XML	JavaConfig
Namespaces	@Enable*
FactoryBean	Builders
GenericXmlContextLoader	AnnotationConfigContextLoader

# @Enable

```
@Configuration
@EnableTransactionManagement(proxyTargetClass=true)
public class DataConfig {

    @Bean
    public PlatformTransactionManager txManager() {
        return new HibernateTransactionManager(sessionFactory());
    }

    @Bean
    public SessionFactory sessionFactory() throws Exception {
        return new AnnotationSessionFactoryBuilder(dataSource())
            .setAnnotatedClasses(Order.class, Account.class)
            .setSchemaUpdate(true)
            .buildSessionFactory();
    }

    @Bean
    public DataSource dataSource() {
        // ... configure and return JDBC DataSource ...
    }
}
```

```
<tx:annotation-driven transaction-manager="txManager"
    proxy-target-class="true"/>
```

# @Enable

- @EnableTransactionManagement
- @EnableAsync
- @EnableScheduling
- @EnableLoadTimeWeaving
- @EnableWebMvc
- @EnableAspectJAutoProxy

# c Namespace

```
<bean class="..." c:age="10" c:name="myName"/>
```

```
<bean class="..." c:name-ref="nameBean"  
    c:spouse-ref="spouseBean"/>
```

# Cache Abstraction

# Caching Abstraction

- `@Cacheable` and `@CacheEvict`
- Pluggable!

# @Cacheable

```
@Cacheable("books")
```

```
@Cacheable(value="books", key="#isbn")
```

```
@Cacheable(value="books", key="#root.methodName")
```

```
@Cacheable(value="books", key="#isbn", condition="#isbn.group == 1")
```

# Cache Configuration

```
<cache:annotation-driven proxy-target-class="false" order="0"/>  
  
<bean id="cacheManager" class="org.springframework.cache.support.MapCacheManager">  
  <property name="caches">  
    <bean class="org.springframework.cache.concurrent.ConcurrentCacheFactoryBean">  
      <property name="name" value="books"/>  
    </bean>  
  </property>  
</bean>
```

# Cache Providers

- Two available implementations
  - EhCache
  - ConcurrentMap
  - JSR 107
  
- Pluggable!

# @MVC

## Improvements

# @MVC

- Introduced in Spring 2.5
- Major improvements in 3.0
  - REST
- Further improvements in 3.1
  - Complete refactoring

# @MVC arguments and return values

- Arguments
  - ModelAndView
  - ServletRequest and ServletResponse
  - @RequestBody, @RequestHeader
  - HttpEntity,
  - ...
- Return Types
  - ModelAndView
  - @ResponseBody
  - ResponseEntity
  - ...

# Behind the Curtains

```
for (Annotation paramAnn : paramAnns) {
    if (RequestParam.class.isInstance(paramAnn)) {
        RequestParam requestParam = (RequestParam) paramAnn;
        paramName = requestParam.value();
        required = requestParam.required();
        defaultValue = parseDefaultValueAttribute(requestParam.defaultValue());
        annotationsFound++;
    }
    else if (RequestHeader.class.isInstance(paramAnn)) {
        RequestHeader requestHeader = (RequestHeader) paramAnn;
        headerName = requestHeader.value();
        required = requestHeader.required();
        defaultValue = parseDefaultValueAttribute(requestHeader.defaultValue());
        annotationsFound++;
    }
    else if (RequestBody.class.isInstance(paramAnn)) {
        requestBodyFound = true;
        annotationsFound++;
    }
    else if (CookieValue.class.isInstance(paramAnn)) {
        CookieValue cookieValue = (CookieValue) paramAnn;
        cookieName = cookieValue.value();
        required = cookieValue.required();
        defaultValue = parseDefaultValueAttribute(cookieValue.defaultValue());
        annotationsFound++;
    }
    else if (PathVariable.class.isInstance(paramAnn)) {
        PathVariable pathVar = (PathVariable) paramAnn;
        pathVarName = pathVar.value();
        annotationsFound++;
    }
    else if (ModelAttribute.class.isInstance(paramAnn)) {
        ModelAttribute attr = (ModelAttribute) paramAnn;
        attrName = attr.value();
        annotationsFound++;
    }
}
```

# Custom argument and return value handlers

```
public interface HandlerMethodArgumentResolver {  
  
    boolean supportsParameter(MethodParameter parameter);  
  
    Object resolveArgument(MethodParameter parameter,  
                           ModelAndViewContainer mavContainer,  
                           NativeWebRequest webRequest,  
                           WebDataBinderFactory binderFactory) throws Exception;  
}
```

```
public interface HandlerMethodReturnValueHandler {  
  
    boolean supportsReturnType(MethodParameter returnType);  
  
    void handleReturnValue(Object returnValue,  
                           MethodParameter returnType,  
                           ModelAndViewContainer mavContainer,  
                           NativeWebRequest webRequest) throws Exception;  
}
```

# @MVC 2.0

- Enabled by default
  - `<mvc:annotation-driven/>`
  - `@EnableWebMvc`

# Consumes and Produces

- Two new elements on the RequestMapping annotation
  - Consumes: consumable media types
    - Content-Type header
  - Produces: producible media types
    - Accept header

# Consumes

```
@RequestMapping(value = "/something", consumes = "application/pdf")
public void handlePdf(Writer writer) throws IOException {
    ...
}

@RequestMapping(value = "/something", consumes = "text/*")
public void handleHtml(Writer writer) throws IOException {
    ...
}
```

# Produces

```
@RequestMapping(value = "/something", produces = "text/html")
public void handleHtml(Writer writer) throws IOException {
    ...
}

@RequestMapping(value = "/something", produces = "application/xml")
public void handleXml(Writer writer) throws IOException {
    ...
}
```

# Other @MVC tid-bits

- @EnableWebMvc
  - WebMvcConfiguration
- @PathVariable added to model
- Uri templates in “redirect:” locations
- Servlet 3.0 support
  - no web.xml
  - async requests
  - file upload support

# Summary

# Spring 3.1

- Environment and Profiles
- Java-based Configuration improvements
- Testing with `@Configuration` and Profiles
- Cache Abstraction
- MVC and REST Improvements
- Servlet 3.0
- `c:` Namespace

# Release Schedule

- RCI out yesterday!
- GA “Soon”
- Spring 3.2
  - Java SE 7
  - JDBC 4.1, JSF 2.2, JPA 2.1
  - Fork-join

# Q & A

If time permits...