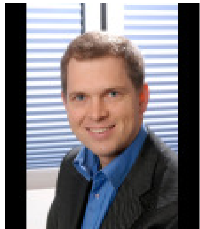
spring source

Spring in the Java EE ecosystem

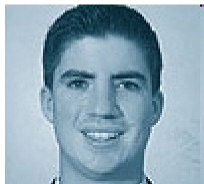
Alexis Moussine-Pouchkine
Michael Isvy

spring source

Speakers



- Alexis Moussine-Pouchkine
  - Java EE technology architect at Sun
  - GlassFish ambassador



- Michael ISVY
  - Trainer and Senior Consultant at SpringSource
  - Opensource contributor on Spring projects

spring source

Agenda

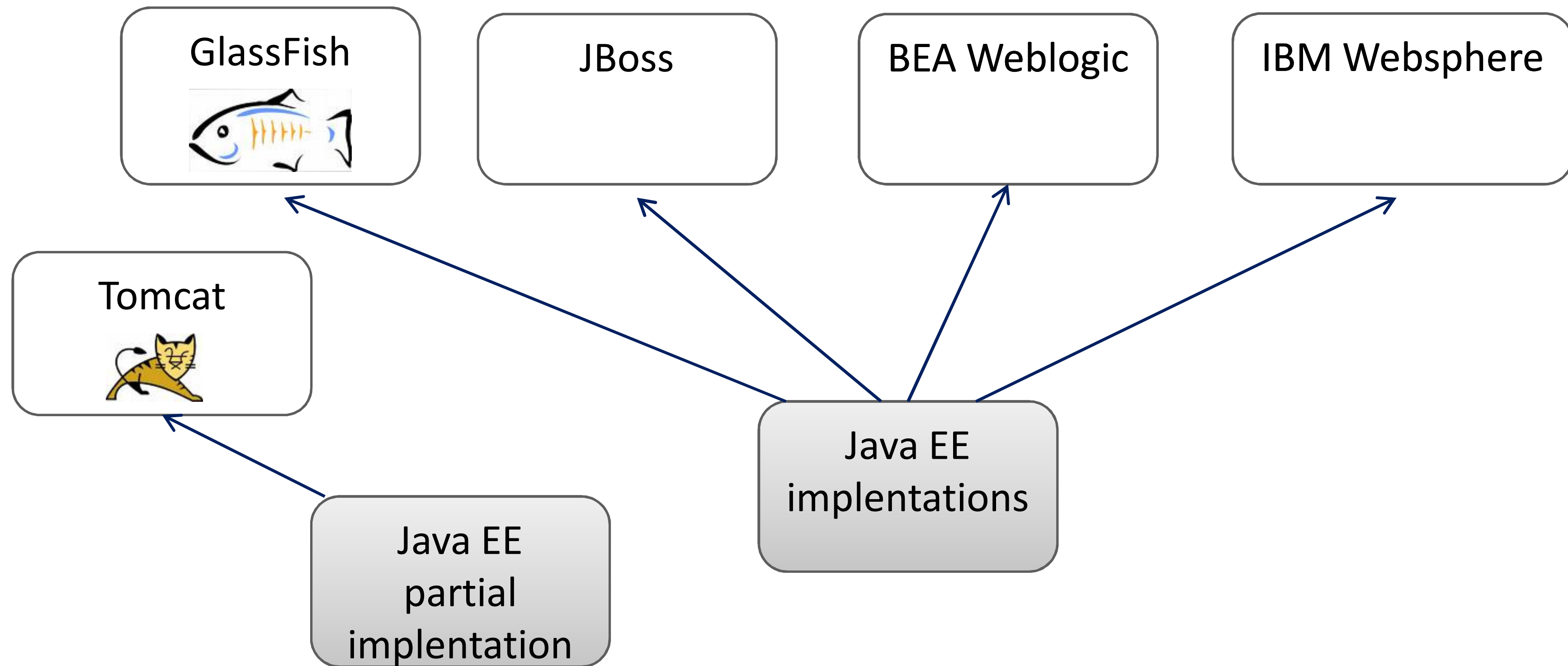
- Spring versus Java EE
- Technical use-cases
  - Dependency Injection
  - Transactions
  - Security
  - AOP
- Conclusion

spring source

Spring versus Java EE

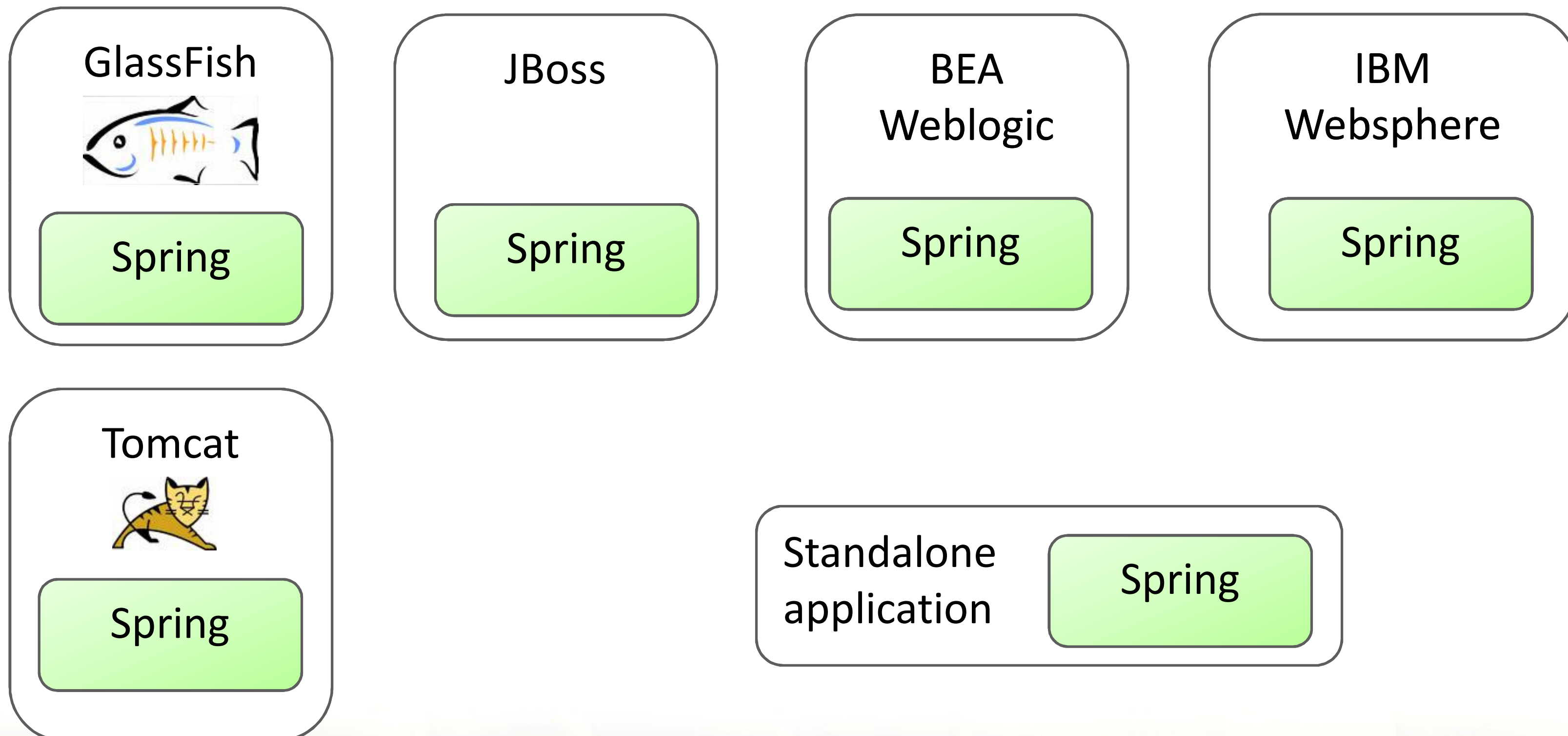
- Java EE has a strong default configuration
  - Provides an architecture for your classes, transaction policy etc
- Spring is a toolbox
  - Everything is disabled by default
  - You only enable what you need
    - Transactions, Security, Remoting etc
  - Widely used framework

Java EE and implementations

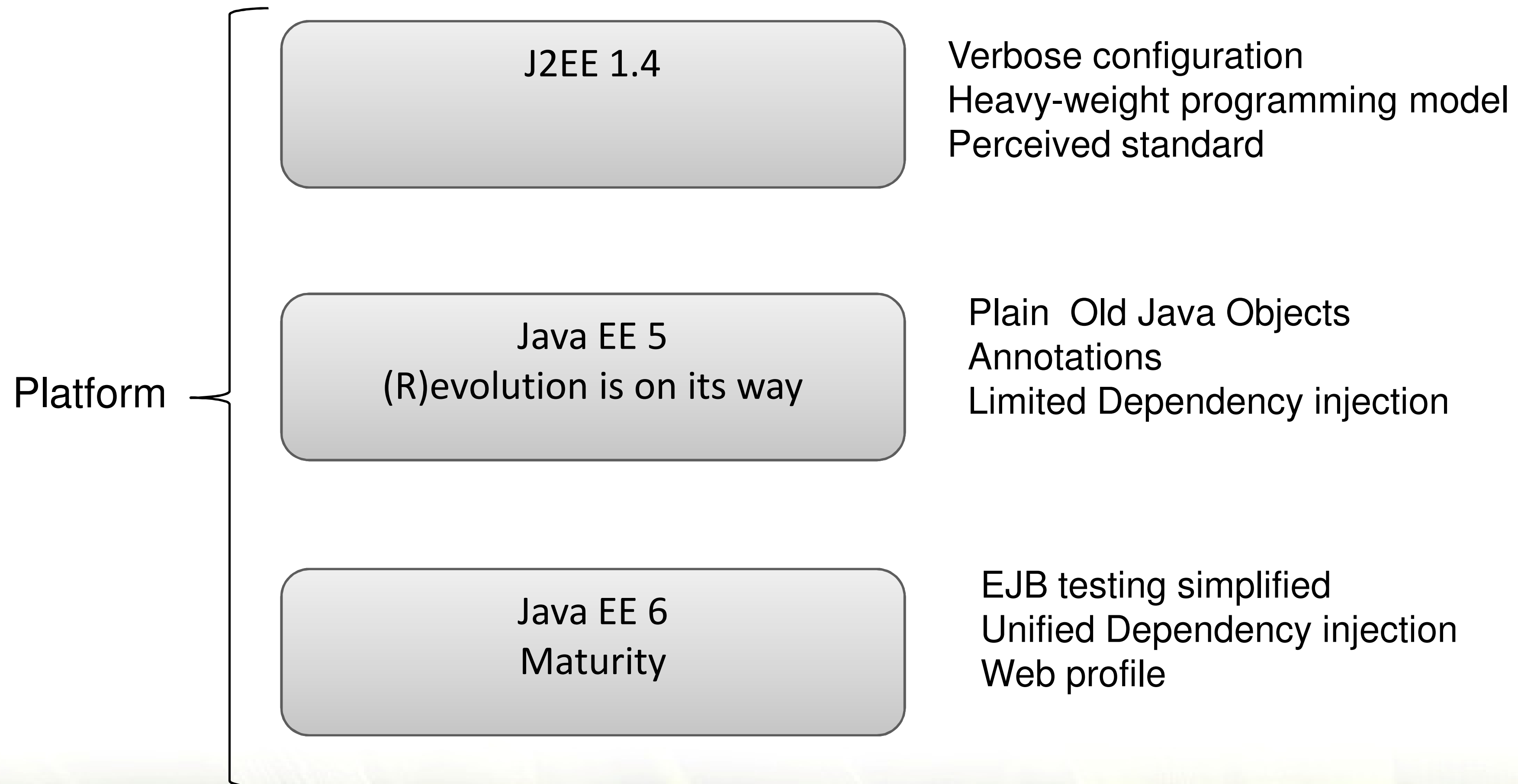


Spring and Java environments

- Spring can be used in any kind of production environment



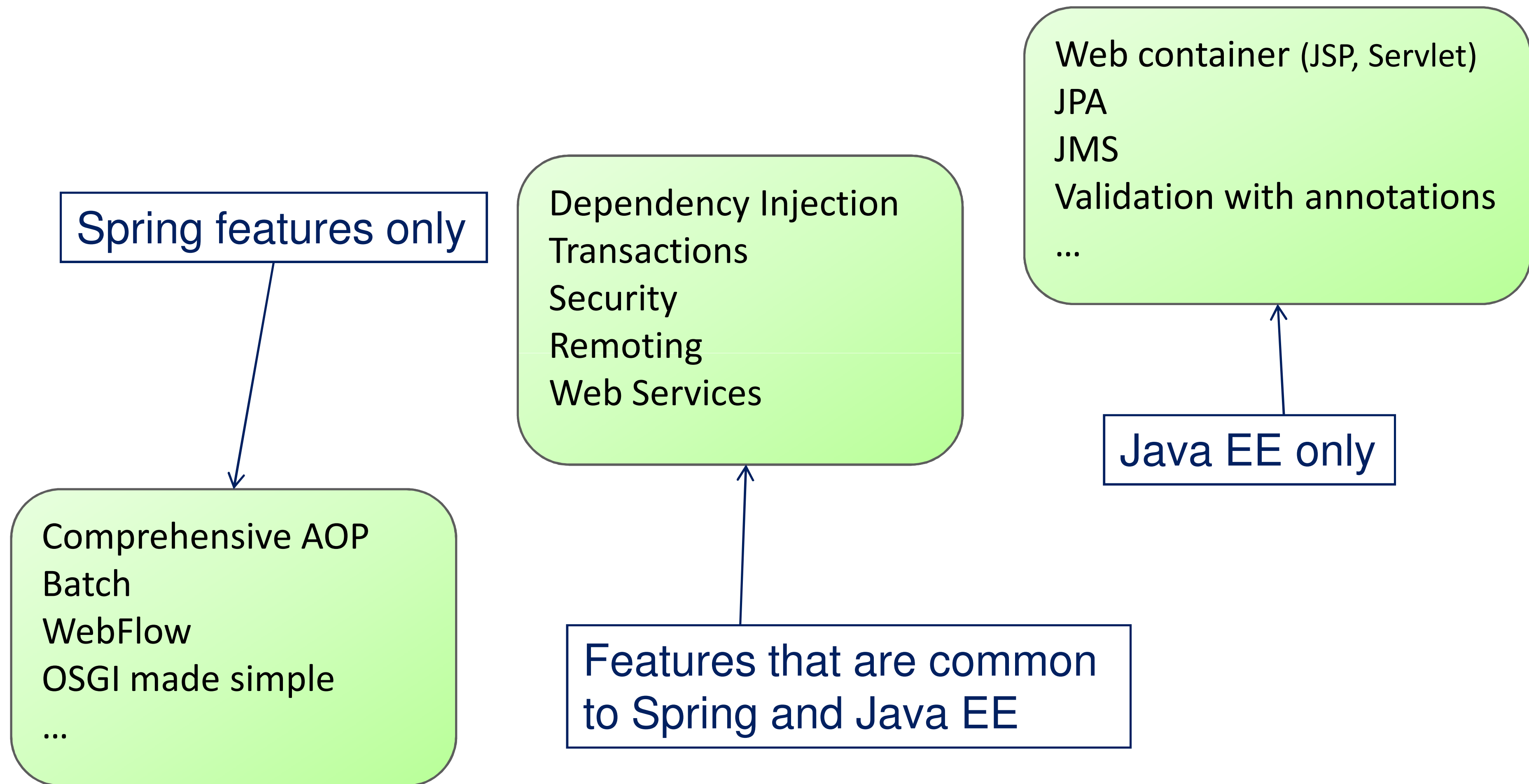
From J2EE to Java EE



Notable in Java EE 6

- EJB 3.1
  - EJBContainer.createEJBContainer()
  - EJB Lite (local, injection, intercept., security)
  - Optional Local Interfaces + WAR packaging
  - Singleton - Asynchronous calls
  - Timer Service - Portable JNDI name
- Web Profile
  - Subset with JPA, EJB 3.1,injection and more
- Extensibility
- Almost cooked:
  - December (2009! ;-), GlassFish v3 as Ref. Implementation

Features



Dependency Injection – Java EE

- Common Annotation
  - `@Resource`
- Specialized cases
  - `@EJB`, `@WebServicesRef`, `@DataSource`
- Requires managed objects
  - EJB, Servlet and JSF Managed Bean in EE 5
  - Any Managed Beans in Java EE 6

```
@Resource(name="jms/myQueue")
private Destination waitingQueue;
```

```
@ManagedBean
public class Client {
    @EJB
    private MyService service;
    @WebServicesRef
    private BidService bidWS;
}
```

Dep. Inj. : the 4 Spring Ways

- XML Configuration

```
<bean id="clientService“ class="app.impl.ClientServiceImpl">
      <property name="clientDAO" ref="clientDAO" />
   </bean>

   <bean id="clientDAO" class=“app.impl.AccountImpl”>
      <!-- … -->
   </bean>
```

- XML with the p namespace

```
<bean id="clientService" class="app.impl.ClientServiceImpl"
                              p:clientDAO-ref=”clientDAO” />

   <bean id=“clientDAO”  class=“app.impl.AccountImpl”>
      <!-- … -->
   </bean>
```

Dep. Inj. : the 4 Spring Ways

- Annotations

```
public class ClientServiceImpl implements ClientService
    private ClientDAO clientDAO;
    @Autowired
    public void setClientDAO(ClientDAO clientDAO) {
        this.clientDAO = clientDAO;
    }
}
```

Can be replace by @Inject starting from Spring 3.0

- JavaConfig

```
@Configuration
public class AppConfig {
   @Bean @Lazy
   public RewardsService rewardsService() {
      RewardsServiceImpl service =  new RewardsServiceImpl();
      service.setDataSource(…);
      return service;
   }
}
```

Transactions with Java EE

```
@Stateless
public class TransferService {
    @PersistenceContext
    private EntityManager em;
  // transactional by default
    public void transfer(Amount a) {
        em.persist(a);
    }
}
```

- The Bean is thread-safe and can use non-thread-safe resources
- All business methods are transactional by default

Transactions with Spring (1/2)

- Nothing is transactional by default. Two approaches:
  - Class-specific declaration

```
public class ClientServiceImpl implements ClientService
    @Transactional
    public void updateClient(Client) { //… }
}
```

  - Generic declaration

```
<aop:pointcut id="txMethods"
            expression="execution(public * com.springsource..*Service.*(..))"/>  …

<tx:advice id="txAdvice">
  <tx:attributes>
            <tx:method name="find*" read-only="true" timeout="10"/>
            <tx:method name="update*" timeout="30"/>
  </tx:attributes>
</tx:advice>
```

Transactions with Spring (2/2)

- Any class can be transactional
  - Not necessarily an EJB
  - Not necessarily a Spring bean (if you use AspectJ)
- Compile Time Weaving may be used
  - So the application starts up quicker
- Same syntax for all kinds of Transactional policies
  - JTA or not
  - Hibernate and JDBC used in the same transaction
  - …

Security: the Java EE way

- Servlet 2.5

```
boolean HttpServletRequest.isUserInRole("javaee");
Principal HttpServletRequest.getUserPrincipal();
```

- Servlet 3.0

```
@WebServlet(name="SecureServlet", urlPatterns={"/SecureServlet"})
@ServletSecurity(httpMethodConstraints={
    @HttpMethodConstraint(value="GET"),
    @HttpMethodConstraint(value="POST", rolesAllowed={"javaee"}),
    @HttpMethodConstraint(value="TRACE",
        emptyRoleSemantic=ServletSecurity.EmptyRoleSemantic.DENY)
})
public class SecureServletMethods extends HttpServlet { … }
```

spring source

Security: Java EE Servlets

- More Servlet 3.0 (container-defined security)

```
// Direct auth once login configured
boolean HttpServletRequest.authenticate(HttpServletResponse resp);

// Direct auth with specified credentials (Auth type = LOGIN)
void HttpServletRequest.login("username", "password");

// Removes the current principal from the session
void HttpServletRequest.logout();
```

Security: the Spring way

- Nothing is secured by default. Two approaches:
  - Class-specific declaration

```
public class ClientServiceImpl implements ClientService
    @RolesAllowed("ROLE_MEMBER")
    public void updateClient(Client) { //… }
}
```

  - Generic declaration

```
<security:global-method-security>
  <security:protect-pointcut
    expression="execution(* com.springsource..*Service.*(..))"
                access="ROLE_USER" />
</security:global-method-security>
```

Security: the Spring way

- Dedicated project: Spring Security
  - Emulates Java EE Security (syntax is very close)
- Plenty of features out of the box
  - Single Sign On
  - Voters for Security rules
  - Captchas
  - Cookie authentication
  - ...

Java EE Interceptors

- Defining the interceptor :

```
public class LogInterceptor {
    @AroundInvoke
    public Object logCall(InvocationContext context) {
        log.info("Method="+context.getMethod());
        return context.proceed();
    }
}
```

- Applying it :

```
@Interceptors(LogInterceptor.class)
@Stateless
public class ClientServiceBean {
    public void findClient() { ... }
}
```

Works on Java EE 6 Managed Beans too...

Aspect Oriented Programming with Spring

- Java EE6 proposes limited use of AOP
- Key point of the Spring framework
  - Example: monitoring the time spent in all my DAO classes

```
@Around("execution(* com.springsource..*DAO.*(..))")
public Object monitor(ProceedingJoinPoint point) {
    stopWatch.start();
    Object returnedObject repositoryMethod.proceed();
    stopWatch.stop();
    // do the logging
    return returnedObject;
}
```

Conclusion

- Spring and Java EE cannot be integrated together in a transparent fashion
  - Features and Behaviors are still different
- Spring supports many standard annotations
  - @javax.annotations.PostConstruct
  - @javax.annotations.Resource
  - By using them, your code is less Spring-dependent
- If you use Java EJBs, you can use Spring and Aspect Oriented Programming on top of them
  - With AspectJ

Questions/Answers

- Alexis Moussine-Pouchkine

  Alexis.Moussine-Pouchkine@sun.com

  Twitter: alexismp

- Michael ISVY

  michael.isvy@springsource.com

  Twitter: michaelisvy