



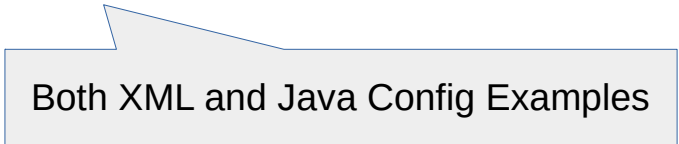
CS544 EA

Applications

SH Web Apps: Transactions

Spring and Hibernate Transactions

- We'll add **@Transactional** annotations
 - Configure Spring to find them
 - Configure the Hibernate TX manager to use them



Both XML and Java Config Examples

Springconfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans">
```

...

```
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
```

```
<tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

Create a txManager bean using the EntityManagerFactory

Tell Spring to look for @Transactional annotations and use the txManager

Needs tx namespace

Config.java

```
@Configuration
@ComponentScan("cs544")
@EnableTransactionManagement
```

```
public class Config {
```

```
  @Bean
```

```
  public PlatformTransactionManager transactionManager(EntityManagerFactory emf) {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(emf);
    return transactionManager;
  }
```

...

```
}
```

Tell Spring to look for @Transactional annotations

Needs a transactionManager bean in order to function

Minimal @Transactional

- Adding @Transactional to @Service classes will give **reasonable** transactional boundaries

```
@Service
@Transactional
public class CustomerService {
    @Resource
    private CustomerDao customerDao;

    public List<Customer> getCustomers() {
        return customerDao.getAll();
    }
}
```

More Serious

```
@Service
@Transactional(propagation = Propagation.REQUIRES_NEW)
public class CustomerService {
    @Resource
    private CustomerDao customerDao;

    public List<Customer> getCustomers() {
        return customerDao.getAll();
    }
}
```

Each service level method
should have own TX

```
@Repository
@Transactional(propagation = Propagation.MANDATORY)
public class CustomerDao {
    @PersistenceContext
    private EntityManager em;

    public List<Customer> getAll() {
        return em.createQuery("from Customer", Customer.class).getResultList();
    }
}
```

DAO methods should never
be called without a TX

Full XML Transaction Config

```
<beans ...>
```

```
...
```

```
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager">  
  <property name="sessionFactory" ref="sessionFactory" />  
</bean>
```

```
<aop:config>  
  <aop:pointcut expression="execution(* example.service.*(..))" id="serviceTx"/>  
  <aop:advisor advice-ref="serviceTxAdvice" pointcut-ref="serviceTx"/>  
</aop:config>
```

```
<tx:advice id="serviceTxAdvice" transaction-manager="txManager">  
  <tx:attributes>  
    <tx:method name="get*" propagation="REQUIRED"/>  
    <tx:method name="add*" propagation="REQUIRED"/>  
    <tx:method name="update*" propagation="REQUIRED"/>  
  </tx:attributes>  
</tx:advice>
```

Example of how you can configure
spring transactions with all XML
(no @Transactional annotations)

```
<aop:config>  
  <aop:pointcut expression="execution(* example.dao.*(..))" id="daoTx"/>  
  <aop:advisor advice-ref="daoTxAdvice" pointcut-ref="daoTx"/>  
</aop:config>
```

```
<tx:advice id="daoTxAdvice" transaction-manager="txManager">  
  <tx:attributes>  
    <tx:method name="set*" propagation="SUPPORTS"/>  
    <tx:method name="*" propagation="REQUIRED"/>  
  </tx:attributes>  
</tx:advice>  
</beans>
```