Spring Application Framework

Harezmi Bilişim Çözümleri

Spring Nedir?

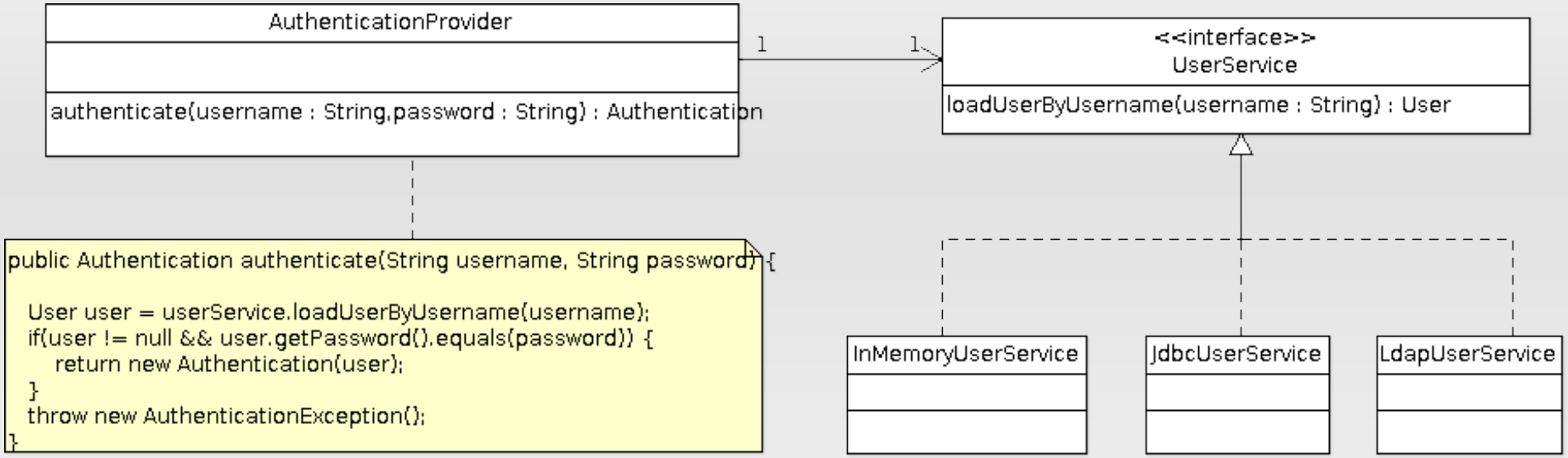
- **Enterprise Java** uygulamalarını
  - **kolay**,
  - **hızlı** ,
  - **test edilebilir**

  biçimde geliştirmek ve **monolitik uygulama sunucuları**nın dışında ortamlarda da çalıştırabilmek için ortaya çıkmış bir “**framework**”tür

Spring'i Öne Çıkaran Özellikler

- POJO tabanlı bir programlama modeli sunar
- "**Program to interface**" yaklaşımını temel ilke kabul etmiştir
- Uygulama kodunun framework'e bağımlı olmasına gerek duyulmaz
- **Test edilebilirlik** her noktada **ön planda**dır
- **Modüler bir framework**tür
- Sadece ihtiyaç duyulan modüller istenilen kapsamda kullanılabilir

“Program To Interface” Yaklaşımı

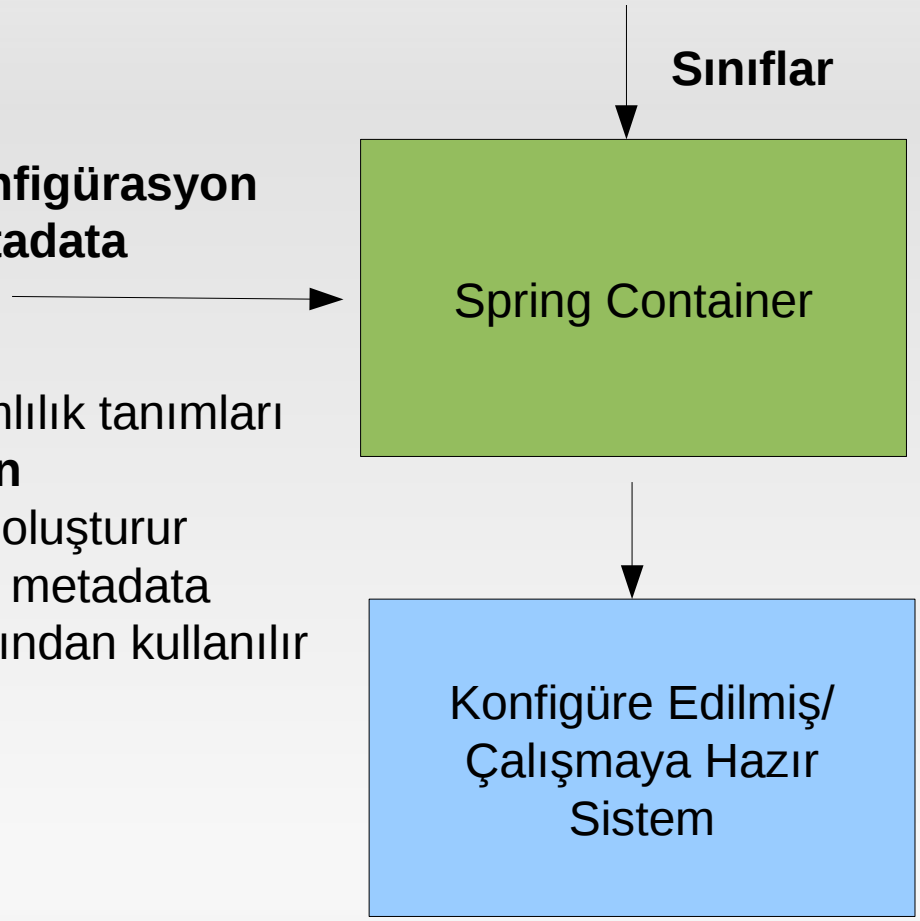


- Sınıflar arasındaki bağımlılıklar **arayüz ve soyut sınıfla**ra doğrudur
- Değişiklikler daha kolay biçimde ele alınabilir, sistem **esnek** bir yapıdadır
- Bu sayede sınıfların **test edilebilir**liği de artmaktadır
- Testlerde **mock ve stub nesneler** daha rahat kullanılabilir

IoC Container ve Dependency Injection

- Spring **nesneleri oluşturma ve bir araya getirme işi**ne sistematik bir yol sunmaktadır
- Nesneler bağımlılıkların hangi **concrete sınıflar**la sağlandıklarını bilmezler
- Bağımlı olunan nesnelerin nereden geldiği de bilinmez
- **Bağımlılıkların yönetimi sınıfların kendilerinden container'a geçmiştir**

IoC Container ve Konfigürasyon Metadata



Spring IoC container tarafından yönetilen nesnelere "**bean**" adı verilir
Diğer Java nesnelerinden hiç bir farkı yoktur

Bean ve bağımlılık tanımları **konfigürasyon metadata**'sını oluşturur
Konfigürasyon metadata container tarafından kullanılır

Spring Container, konfigürasyon metadata formatından bağımsızdır
XML, annotasyon veya java Tabanlı olabilir

ApplicationContext Oluşturulması

- Spring Container'ın diğer adı **ApplicationContext**'dir
- Oluşturmak için programatik veya dekleratif yöntemler mevcuttur
- Standalone uygulamalarda **programatik** yöntem kullanılır
- Web uygulamalarında çoğunlukla **dekleratif** yöntem kullanılır
- Standalone ortamdan web ortamına geçiş web.xml ayarlarından ibarettir

Spring ve Entegrasyon Birim Testleri

Spring ve Test Güdümlü Programlama

- Entegrasyon testlerinin **standalone ortamda yazımı ve çalıştırılması**nı mümkün kılar
- ApplicationContext'i oluşturan **bean tanımlarının doğru yapılıp yapılmadığı**nın kontrolü sağlanır
- JDBC ve ORM araçları ile **veri erişiminin testi** yapılır
  - SQL, HQL sorgularının kontrolü yapılır
  - ORM mapping'lerin düzgün yapılıp yapılmadığı test edilmiş olur

Spring TestContext Framework

- Otomatik **ApplicationContext yönetimi** yapar
- Spring container'ı test sınıfları arasında yeniden oluşturmamak için **cache desteği** sağlar
- TestCase içerisinden **ApplicationContext'e erişmek** de mümkündür
- Test sınıflarına **dependency injection** yapılabilir
- Test metodlarının **transactional context** içerisinde çalıştırılması mümkündür

Spring ile Veri Erişimi

Spring ile Veri Erişimi

JAVA Eğitimleri

- Değişik teknolojiler için kullanımı **kolay ve standart bir veri erişim** desteği sağlar
  - JDBC, JPA, Hibernate, JDO vb desteklenir
- **Kapsamlı ve transparan** bir transaction yönetim altyapısına sahiptir
- Farklı veri erişim teknolojileri **aynı anda** kullanılabilir
- Veri erişim teknolojilerinin exception hiyerarşilerini **standart bir exception hiyerarşisi**ne çevirir

JDBC ile Veri Erişimi

- Veritabanı bağlantı parametrelerinin belirtilmesi ve bağlantının kurulması
- *SQL sorgusunun oluşturulması*
- Oluşturulan Statement'ın derlenip çalıştırılması
- Dönen ResultSet üzerinde işlem yapan bir döngünün kurulması
- *Bu döngü içerisinde her bir kaydın işlenmesi*
- Meydana gelebilecek hataların ele alınması
- Transaction'ın sonlandırılması ve veritabanı bağlantısının kapatılması

JDBC ile Veri Erişimi

- **Template Method örüntüsü tabanlı** genel bir yaklaşım sunar
  - JdbcTemplate
- Utility veya helper sınıflarına benzetilebilir
- Template tarafından dikte edilen **standart bir kullanım şekli** kod geneline hakim olur
- **Callback**'ler yardımı ile template nesneye çalıştıracağı kod belirtilir

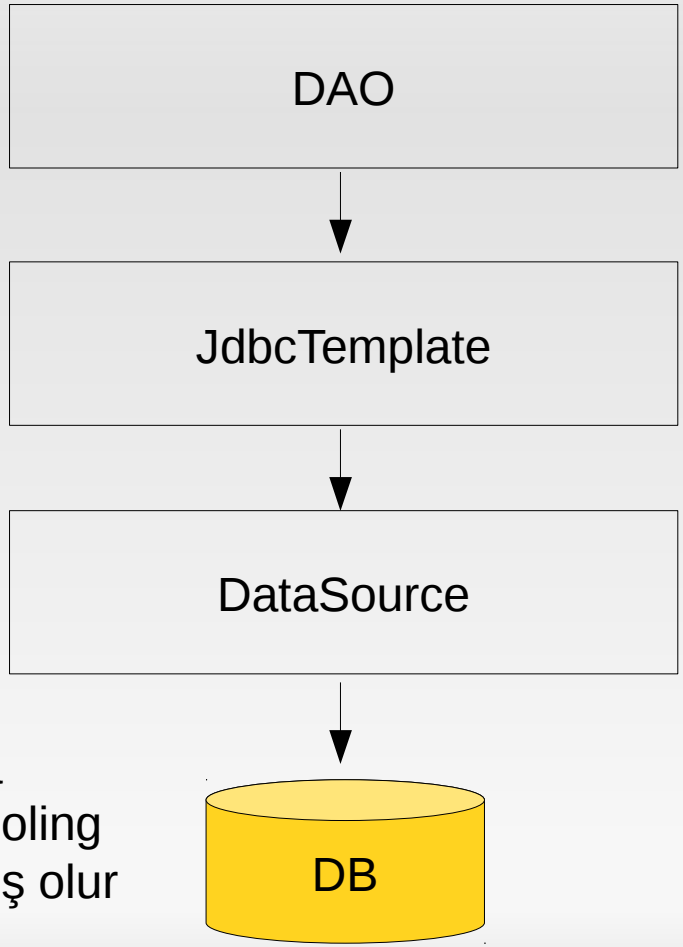
JDBC ile Veri Erişimi

Uygulama tarafında JdbcTemplate ile Çalışırken sadece **Callback** yazmak yeterlidir

Spring veritabanı bağlantılarını **DataSource** nesnesinden alır

DataSource **connection factory**'dir

DataSource'un kendi başına yönetilmesi ile connection pooling vs. uygulamadan izole edilmiş olur



JDBC ile veri erişimi **JdbcTemplate** üzerine kurulmuştur

Çalışması için **DataSource** nesnesine ihtiyaç vardır

Thread safe'dir, birden Fazla bean tarafından erişilebilir

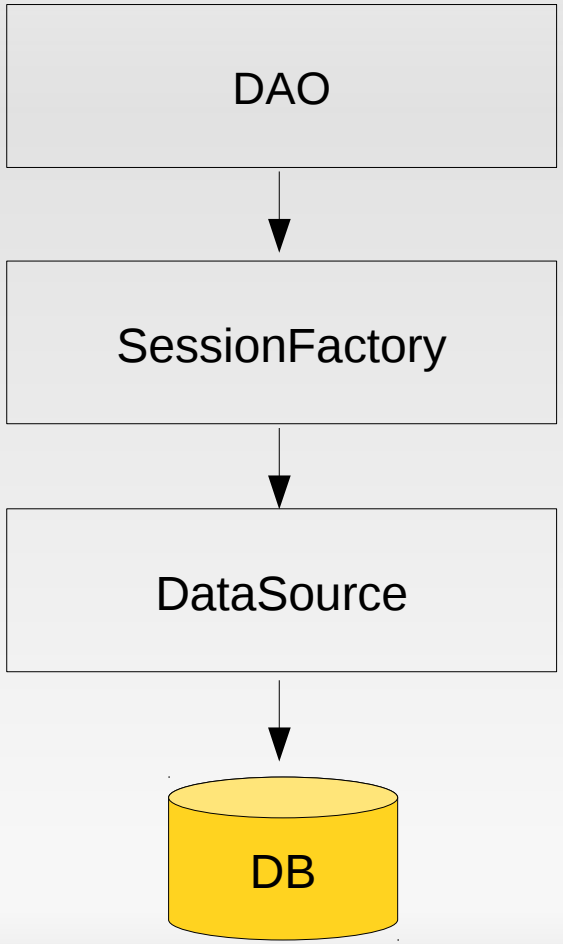
ORM ile Veri Erişimi

- **Entegre** bir transaction yönetim altyapısı üzerinden çalışır
- JDBC işlemleri ile ORM işlemlerini **aynı TX** içerisinde beraber kullanabilirsiniz
- **Ortak** bir veri erişim **exception hiyerarşisi** sunar
- **Entegrasyon** birim testlerinin yazılmasını ve çalıştırılmasını kolaylaştırır

Hibernate ile Veri Erişimi

DataSource ve SessionFactory **Spring managed bean**'lar olarak tanımlanır

SessionFactory nesnesini ApplicationContext içerisinde yönetmek için **LocalSessionFactoryBean** ile bean tanımı yapılır



Spring ve Transaction Yönetimi

Spring ve Transaction Yönetimi

- Değişik veri erişim yöntemlerini aynı anda kullanmayı sağlayan **ortak bir transaction altyapısı** vardır
- Adı **PlatformTransactionManager**'dır
- **Global** ve **lokal** transactionları destekler
- **Dekleratif** ve **programatik** TX yönetimi mümkündür

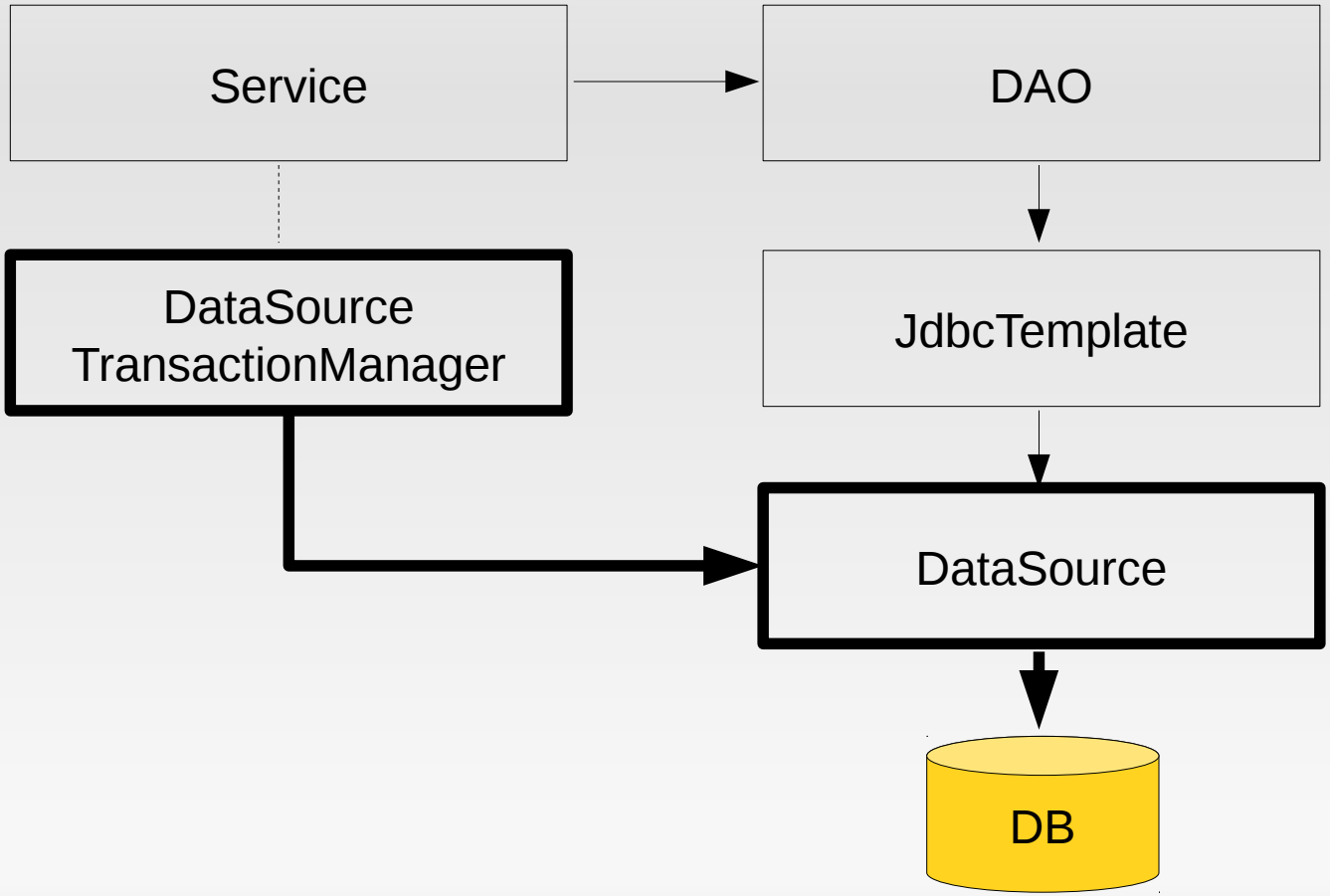
Spring ve Transaction Yönetimi

- Spring TX altyapısında **en önemli nokta** veri erişim teknolojisine **uygun PlatformTransactionManager tanımlanması**dır
  - JDBC: DataSourceTransactionManager
  - Hibernate: HibernateTransactionManager
  - JPA: JpaTransactionManager
  - JTA: JtaTransactionManager

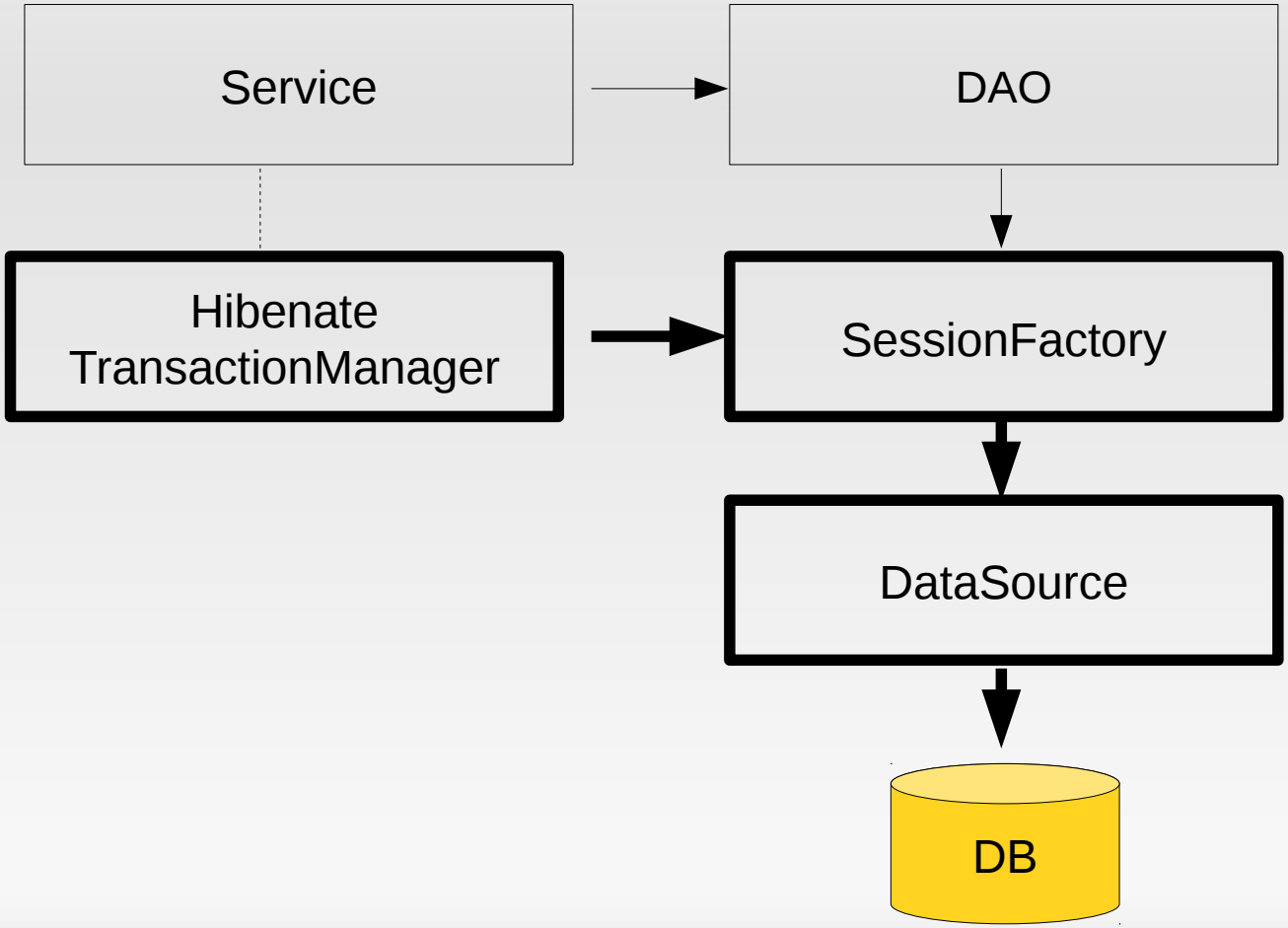
Spring ve Transaction Yönetimi

- **PlatformTransactionManager** soyutlaması sayesinde uygulama TX altyapısından bağımsızdır
- Bu sayede uygulamalar **lokal TX'den global TX'e transparan biçimde geçiş** yapabilir
- Ya da **veri erişim teknolojisi** rahatlıkla değiştirilebilir
- Örneğin, HibernateTransactionManager'dan JtaTransactionManager'a geçiş için **sadece bean tanımlarında değişiklik** yeterlidir

TransactionManager Konfigürasyonu - JDBC



TransactionManager Konfigürasyonu - Hibernate



Dekleratif Transaction Yönetimi

- Çoğunlukla **tercih edilen yöntem**dir
- Uygulama içinde TX yönetimi ile ilgili iş yapılmaz
- **Spring AOP** ile implement edilmiştir, fakat kullanmak için AOP bilmeye gerek yoktur!
- **EJB CMT**'ye çok benzer
- **Metot düzeyinde** TX yönetimi yapılabilir

Dekleratif Transaction Yönetimi

- **Dekleratif biçimde kendinize özgü rollback kuralları** tanımlanabilir
- Transaction öncesinde ve sonrasında **size özgü kod** çalıştırabilirsiniz

Dekleratif Transaction Yönetimi

- Sistem exception'ları (**runtime**) TX **rollback nedenidir**
- Application exception'larında(**checked**) TX **rollback edilmez**
- Ancak çoğunlukla bu davranış uygulamaya göre değiştirilir
- Gerektiğinde uygulama içerisinden exception fırlatmadan da **rollback** yaptırılabilir

Spring MVC, Web Programlama ve REST Kabiliyetleri

Web Uygulamalarında ApplicationContext Oluşturma

- Spring **herhangi bir Web UI teknolojisi** ile birlikte kullanılabilir
- ApplicationContext, **ContextLoaderListener** ile bootstrap edilip **ServletContext** üzerinden erişilebilir
- Örneğin bir Servlet içerisinden ApplicationContext **bean lookup** ile istenilen bean alınıp kullanılabilir

Web Uygulamalarında ApplicationContext Oluşturma

- **WebApplicationContext** ApplicationContext'i **extend** eder
- **İlave kabiliyetler** sunar
- ApplicationContext nesnelerini dekleratif oluşturmak için **web.xml** içinde **ContextLoaderListener** kullanılır
- WebApplicationContext nesnesi ServletContext'e **bind** edilir

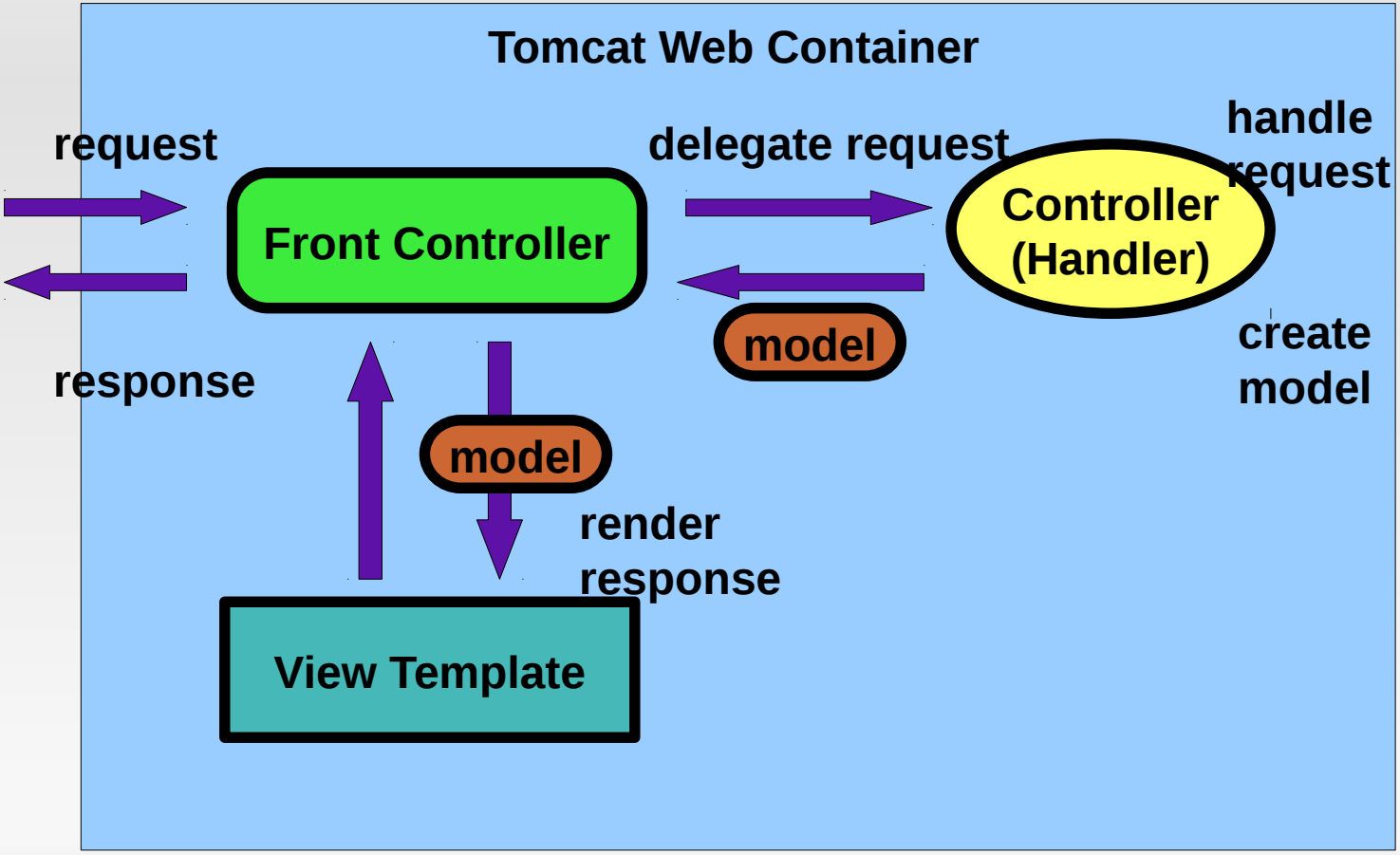
Spring MVC Web Framework

- Spring MVC de diğer pek çok MVC framework gibi “**request-driven**”dır
- **DispatcherServlet etrafında kurulu** bir frameworktür
- DispatcherServlet bir **HttpServlet** implementasyonudur
- **Front Controller** pattern’ına iyi bir örnektir

DispatcherServlet

- **web.xml** içerisinde tanımlanıp, burada hangi requestleri ele alacağı belirtilmelidir
- Spring IoC Container ile **tam bir entegrasyon**a sahiptir
- Ayağa kalkarken kendine ait bir **WebApplicationContext** yaratır

Front Controller J2EE Pattern



Controller Bean Tanımları

- Web katmanından **servis katmanına erişim** sağlayan bean'lardır
- HTTP **request**'lerini **handle** ederler
- Kullanıcı **input'unu alır ve modele dönüştürür**
- Spring 3 ile birlikte Controller beanları için herhangi bir **arayüz implement edilmesi**, **sınıftan türetilmesi** gerekmez

Controller Bean Tanımları

- **@Controller** annotasyonu bir bean'ın MVC controller olduğunu anlatır
- Controller bean'ları da **ApplicationContext içerisinde tanımlanmalı**dırlar
- **Component scan** kabiliyeti ile **@Controller** annotasyonuna sahip sınıfların bean olarak tanımlanması sağlanabilir

Spring ve REST

- RESTful uygulamalar geliştirmek için Java EE6 standartı **JAX-RS**'dir
  - Jersey, CXF, Restlet gibi impl mevcuttur
- Spring MVC 3.1 ile birlikte **RESTful** Web servisleri ve uygulamalar geliştirmek mümkün hale gelmiştir
- Fakat Spring MVC bir REST impl. değildir

Spring ve REST

JAVA Eğitimleri

- REST servisleri **@Controller** ve **@RequestMapping** ile tanımlanan bean'lar üzerinde hayata geçirilir
  - @PathVariable
  - @RequestBody
  - @ResponseBody
  - @ResponseStatus annotasyonları ile REST servisleri yazılabilmektedir

Response Content Tipinin Belirlenmesi


```
@RequestMapping(value="/vets",produces={"application/json","application/xml"})
@ResponseBody
public Collection<Vet> getVets() {
    return petClinicService.getVets();
}

@RequestMapping("/vets")
@ResponseBody
public String getVetsAsHtml() {
    StringBuilder builder = new StringBuilder();
    Collection<Vet> vets = petClinicService.getVets();
    builder.append("<html><body>");
    for(Vet vet:vets) {
        builder.append(vet.getFirstName() + " " + vet.getLastName() + "<br>");
    }
    builder.append("</body></html>");
    return builder.toString();
}
```

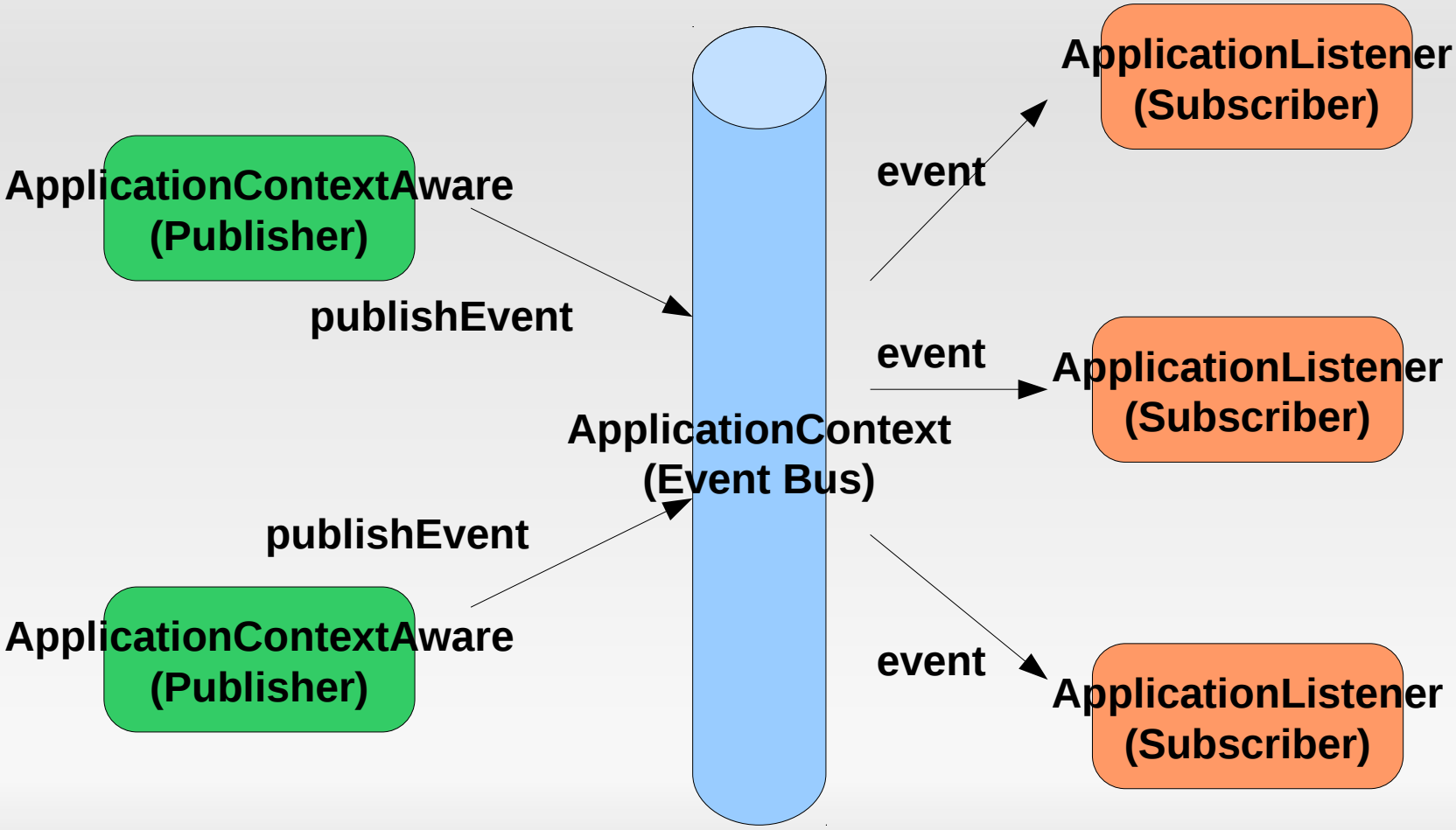
http://localhost:8080/petclinic/mvc/vets.html -> text/html
http://localhost:8080/petclinic/mvc/vets.json -> application/json
http://localhost:8080/petclinic/mvc/vets.xml -> application/xml

Spring ve Event Yönetimi

Event Yönetimi

- ApplicationContext belirli durumlarda **event'ler fırlatır**
- Bunlar **ApplicationEvent** tipinde nesnelerdir
- Bean'larımız bu **event'leri yakalayıp** değişik işler yapabilirler
- Bunun için **ApplicationListener** arayüzü implement edilir
- **Observer** örüntüsüdür

Spring Container ve Observer,Mediator Örüntüleri



Built-in Event Tipleri

- ContextRefreshedEvent
- ContextClosedEvent
- RequestHandledEvent
- Ayrıca Spring Security gibi diğer framework'lerin de **built-in event'leri** mevcuttur
  - AuthenticationSuccessEvent
  - AbstractAuthenticationFailureEvent

Uygulamaya Özel Event Tipleri

- Uygulamaya özel event'ler de tanımlanabilir
- Daha sonra bu event'ler fırlatılabilir
- **ApplicationContext.publishEvent()**
- ApplicationListener nesnelerine event'ler **senkron** biçimde verilir
- Bu durumda publishEvent() bütün listener'lar çalışana değin **süreci bloklar**

Metot Düzeyinde Cache ve Validasyon Kabiliyeti

Spring ve Metot Düzeyinde Caching

- **Cache ihtiyacı** olan servis bean'larına bu özelliği sağlamak amacıyla geliştirilmiştir
- ORM 2nd level cache **domain nesneleri**nin cache'lenmesi içindir
- Spring cache ise daha çok **web veya remote metot çağrıları**ndan dönen sonuçların cache'lenmesi için düşünülmüştür

Spring ve Metot Düzeyinde Caching


```
public class FooService {

    @Cacheable("fooCache")
    public Foo findFoo(String name) {
        //...
    }

    @Cacheable(
        value="default",key="#date.time")
    public Foo findFoo(Date date) {
        //...
    }


    @Cacheable(
        value="default",condition="#i>10")
    public Foo findFoo(int i) {
        //...
    }
}
```

Cache ismidir,
cache **key** default durumda **metot parametreleri**nden elde edilir

Cache key **SpEL expression** yardımı ile de elde edilebilir

Cache işleminin ne zaman devreye gireceğini belirlemek için SpEL yardımı ile "**condition**" da tanımlanabilir

Spring ve Metot Düzeyinde Caching

JAVA Eğitimleri

```
public class FooService {

   @CacheEvict(value="fooCache")
   public void updateFoo(String name, int age) {
      //...
   }

   @CachePut("fooCache")
   public Foo insertFoo(String name, int age) {
      //...
   }

}
```

Key değerine karşılık gelen entry cache'den çıkarılır

Metot return değeri cache'e entry olarak eklenir
Default durumda metot input parametreleri ile key değeri belirlenir

Spring ve Metot Düzeyinde Validasyon

- **Validasyon ihtiyacı** olan servis bean'larına bu özelliği sağlamak amacıyla geliştirilmiştir
- **JSR-303** Bean Validation API'si kullanılır
- Metot input parametreleri veya return değerleri validasyona tabi tutulabilir

Spring ve Metot Düzeyinde Validasyon

JAVA Eğitimleri

```
public class User {
    @NotEmpty
    private String username;
    @Email
    private String email;
    @Min(18) @Max(64)
    private int age;
}

@Validated
public class UserService {
    public User findUser(@NotEmpty String    nationalityId) {
    //...
    }

    public void createUser(@Valid User   user) {
    //...
    }

}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com