Texto íntegro referente à apresentação

Texto em letras menores são os que não estão na apresentação. A parte de consulta (final do documento) é importante, pois explica os slides sem estar no meio deles, pois eram grandes além de precisarem estar vinculados.

Slide 1: Capa - Título Spring, nomes e nUSPs
Slide 2: Spring
• Framework open-source baseado na obra de Rod Johson
• Suas principais idéias estão descritas no livro:
Johson, R. "Expert One-on-One: J2EE Design e Development", Peer Information; 1st edition, 2002
• Permite a construção de aplicações Java mais rapidamente por permitir que o desenvolvimento se concentre no problema de negócio ao invés da conexão de componentes e sistemas

- Surgiu como proposta de simplificação do desenvolvimento de J2EE
- Evitar a complexidade desnecessária com:
  – Startup rápido
  – Poucas dependências
  – Independente de ambiente
  – Infra-estrutura para testes

Slide 3: Beneficios
• Modularidade
  • "Plain old" Objetos Java mantém o código conciso, simples e modular
  • capaz de montar um sistema complexo com componentes de baixo acoplamento de uma forma consistente e transparente
• Produtividade
  • Em torno de 70% dos desenvolvedores relatam ganhos de produtividade e redução no tempo de implantação com o uso de Spring
• Portabilidade
  • Aplicações rodam em Tomcat, todos os servidores Java EE e em plataformas na nuvem.
• Testabilidade
  • Dependências claramente especificadas tornam os testes unitários e integrados mais fáceis

- É um framework leve com inversão de controle e orientado à aspectos

(**Leve**, pois elimina a necessidade de código repetitivo,como lookups;
Com **inversão de controle** que resolve automaticamente as dependências entre beans;
Código comum a várias seções, como gerenciamento de transações, são implementados como **aspectos**.)

- Não restrito ao Java - Spring.NET
- Diponibiliza:
  - Container Java completo e leve (Envolve todas as camadas de uma aplicação J2EE típica)

- Camada de abstração comum para o gerenciamento de transações
- Camada de abstração para tecnologias populares, como Hibernate e JDBC

(Simplifica o tratamento de erros, oferecendo uma hierarquia de exceção significativa)

Slide 4: Valores do Spring

• Não invasivos

- Minimiza a dependência da aplicação com o framework.(O código da aplicação não deve depender da API Spring)

Vantagens

• A aplicação pode ser executada sem o framework
• Migração para futuras versões do Spring é mais facil
• Aplicação do framework em código legado.

• Facilidade do uso de boas práticas

– Faz a implementação de interfaces ao invés de classes
– Inversão de Controle (IoC): Código responsável pela chamada de objetos está protegido se a aplicação for desenvolvida com interfaces

• Promove a "plugabilidade"

– permite o uso de serviços
– Dependência entre serviços é expressa em termos de interfaces
– Troca de serviços sem causar impacto no resto da aplicação

• Configuração

– Uso de arquivos de configuração ao invés de configuração em código java. (Hardcoded)
– Arquivos XML ou .properties

• Escolha de arquitetura (a troca de frameworks não influencia outra camadas)

– Exemplo: Troca entre frameworks de mapeamento O/R sem impacto na camada de lógica de negócio ou Troca entre frameworks MVC sem impacto nas camadas de persistência

• Modelo de programação consistente

– é capaz de montar um sistema complexo a partir de um conjunto de componentes de baixo acoplamento(POJOs) de uma forma consistente e transparente
– Abordagem consistente em diversas partes do framework: Uma vez aprendida um parte é mais fácil compreender o resto
– Utilizável em qualquer ambiente
– Separa aplicação do ambiente de execução: Independência de contexto

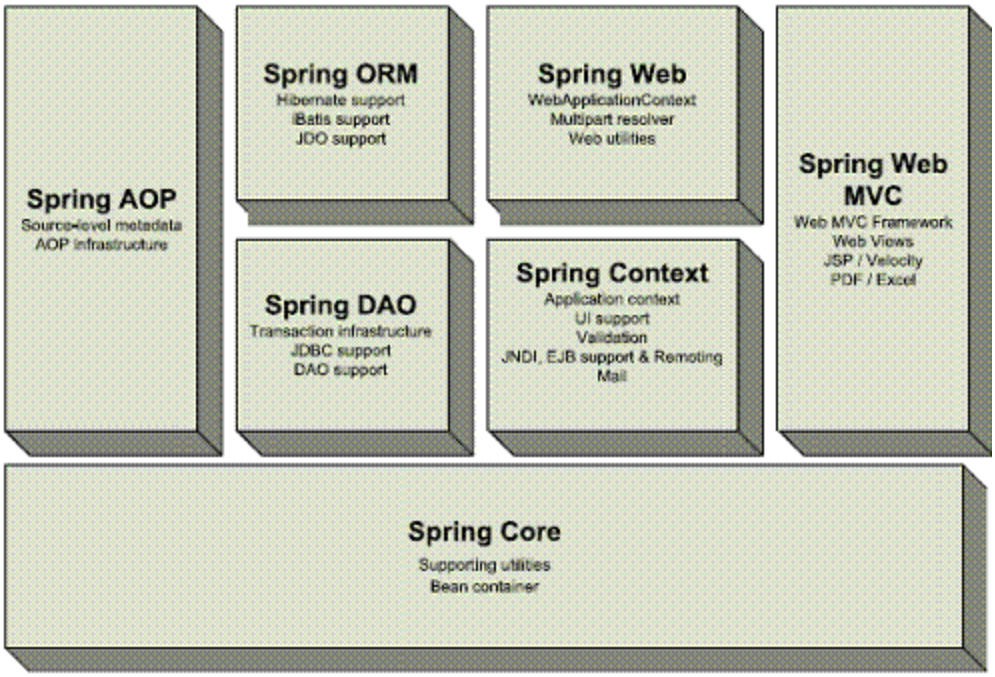
• Facilita um Design Orientado a Objeto
• Escolha da Arquitetura - Repetiu
• Não reinventa a roda

– Embora possua um escopo abrangente, o Spring não introduz soluções de:
  • Mapeamento O/R
  • Abstrações de Log
  • Pool de Conexões
  • Coordenador de transações distribuidas
  • Protocolos

• Facilita o teste - Faltou

– Desenvolvimento com POJOs - Fáceis de testar
– Teste de código é uma tarefa crítica e o container não deve interferir neste objetivo

Slide 5: Estrutura do Spring



Principais Módulos:
Spring Core
O módulo Spring Core representa as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory. Trata-se de uma implementação do padrão *Factory*, responsável em remover a programação de *Singletons* e permitindo o baixo acoplamento entre a configuração e a especificação de dependências, de sua lógica de programação.
Spring DAO
O módulo Spring DAO provê uma camada de abstração para JDBC, eliminando grande parte da codificação necessária para interagir com um banco de dados.
Spring ORM
O módulo ORM provê integração do Spring com outros frameworks para persistência de objetos, como Hibernate e iBatis.
Spring AOP
Para prover uma implementação de Orientação a Aspectos que permite a definição de *pointcuts* e *methods interceptors*, existe o módulo Spring AOP.
Spring Web
Para prover funcionalidades específicas para projetos Web, tem-se o módulo Spring Web. São funcionalidades como componentes para *upload* de arquivos e suporte para utilização de Inversão de Controle neste tipo de aplicação.
Na inicialização do container IoC é usado o servetlistener e o application context orientado para web. Quando usamos Spring junto com Struts e WebWork é este pacote que fará a interação. Esta camada implementa a camada MVC.
Spring MVC
O módulo Spring MVC fornece uma implementação de framework Web, similar ao Struts.
Provê a implementação do Model-View-Controller (MVC) para aplicações web.Provê uma clara separação entre o modelo de código de domínio e os formulário web e permite que você use todos os outro recursos do Spring Framework.

Slide 6: Spring é montado por 3 alicerces

• Inversão de controle (IoC), Programação orientada à aspectos (AOP) e Portable Services Abstraction (ORM, DAO, Web MVC, Web e outros).

Slide 7: Container de IoC

Slide 8: Container de IoC

• Dependências entre objetos (beans) são tratadas por meio de interfaces java ou classes abstratas feitas por uma Entidade Externa – Container IoC

• A interface org.springframework.beans.factory.BeanFactory representa o Container IoC do Spring. Existem diversas implementações de BeanFactory, sendo a XmlBeanFactory (toda configuração de dependência entre os objetos é definida em um arquivo XML) a implementação mais comum.

• O Container IoC é o responsável pelo gerenciamento destes beans.

Slide 9: Container de IoC - Inversão de controle

• padrão é muito usado em projetos orientados a objeto, responsável pelo gerenciamento de beans (Objeto)

• técnica para diminuir o acoplamento entre as classes

• Permite a “cola” de aplicações sem que as mesmas estejam preparadas para isso

- permite utilizar conceitos como interface, herança e polimorfismo
- objetivo de reduzir o acoplamento, facilitar o reuso e os testes no projeto de software.

Slide 10: Container de IoC - Injeção de dependência

Visa remover dependências desnecessárias entre as classes ou torná-las mais suaves e ter um design de software que seja fácil de manter e evoluir.

• Injeção por construtor

A dependência é resolvida entre dois objetos passando para objeto1 uma implementação concreta de objeto2 através do seu construtor.

• Injeção por setter

- Utiliza-se do método de set para injetar a dependência

• Injeção por métodos- pouco usado

-Container implementa método em tempo de execução

Slide 11: Container de IoC - Injeção de construtor

```
• public class objeto1 {
        public objeto1 (objeto2 ob2) {
                this.ob2 = ob2;
        }
}
```

Slide 12: Container de IoC - Injeção setters

```
class objeto1 {
        private objeto2 ob2;
        public voidsetOb2(objeto2 ob2) {
                this.ob2 = ob2;
        }
}
```

Slide 13: Programação Orientada a Aspectos

Slide 14: OAP

• Permite separar e organizar o código de acordo com a sua importância para a aplicação (separation of concerns).

A OAP busca resolver problemas como complexidade do sistema permitindo que o programador implemente questões de segurança, log, transações, e etc através de aspectos.

• O aspecto é constituído de uma ou mais peças de advices (fragmentos de código, como métodos) e uma lista de join points (pontos no programa principal na qual os advices são
inseridos).

• OAP tenta se separar os problemas, aumentar a modularidade e reduzir as redundâncias.

Slide 15: Portable Services Abstraction

Slide 16: PSA (ORM, DAO, Web MVC, Web e outros)

- Tudo que não é IoC ou AOP
- Rodam em qualquer sistema: Ex.: Hibernate, JDBC, LDAP, Spring Beans.

Slide 17: PSA (ORM, DAO, Web MVC, Web e outros)

Spring Bean Container

Tipos:

Bean Factory

• Provê suporte básico para a injeção de dependência

• Gerencia configs e ciclo de vida de beans

Application Context

• Cria um Bean Factory e adiciona serviços:

• Trata mensagens para internacionalização

• Carrega recursos genéricos

• Dispara e trata eventos

Slide 18: PSA (ORM, DAO, Web MVC, Web e outros)
Container cria beans em ordem de acordo com grafo de dependencias.
• Beans são criados usando contrutores (geralmente sem argumentos) ou métodos de factory
• Dependência não injetadas via contrutores são resolvidas por setters
• Sem chamadas a Contexts e lookups
• Ambos beans iniciados apenas na chamada factory.getBean()
• Programação orientada a interfaces: implementações utilizadas podem mudar com alteração apenas no XML

Slide19: Estrutura do Spring - Core
- representa as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory.
-Trata-se de uma implementação do padrão Factory, responsável em remover a programação de Singletons e permitindo o baixo acoplamento entre a configuração e a especificação de dependências, de sua lógica de programação.

Slide 20: Estrutura Spring - DAO
• Visa tornar mais fácil a atividade de trabalhar com tecnologias de acesso de dados como JDBC, Hibernate ou JDO de uma forma consistente.
•Permite alternar entre as tecnologias de persistência com bastante facilidade e também permite o código sem se preocupar com captura de exceções que são específicos para
cada tecnologia.
•Trabalha com uma variedade de tecnologias de acesso dados;
Para tornar mais fácil trabalhar com uma variedade de tecnologias de acesso dados como JDBC, JDO e Hibernate de uma forma consistente, Spring fornece um conjunto declasses abstratas DAO que se pode estender. Essas classes abstratas possuem métodos para fornecer a fonte de dados e quaisquer outras definições de configuração que são específicos para a tecnologia de acesso a dados relevantes.
•Fornece um conjunto declasses abstratas DAO:

- JdbcDaoSupport
  superclasse para objetos de acesso a dados JDBC. Requer umDataSource a ser fornecida; por sua vez, essa classe fornece uma instânciaJdbcTemplate inicializada a partir do DataSource fornecida para subclasses.
- HibernateDaoSupport
  superclasse para objetos de acesso a dados Hibernate. Requer um SessionFactory a ser fornecida; por sua vez, essa classe fornece uma instânciaHibernateTemplate inicializada a partir do SessionFactory fornecido para subclasses.Alternativamente, pode ser inicializado diretamente através de um HibernateTemplate,reutilize as configurações destes como SessionFactory, o modo de flush, tradutorexceção, e assim por diante.
- JdoDaoSupport

super classe para objetos de acesso a dados JDO. Requer umPersistenceManagerFactory a ser fornecida; por sua vez, essa classe fornece uma instância JdoTemplate inicializada a partir do PersistenceManagerFactory fornecido parasubclasses.

• JpaDaoSupport

super classe para objetos de acesso a dados JPA. Requer umEntityManagerFactory a ser fornecida; por sua vez, essa classe fornece uma instânciaJpaTemplate inicializada a partir do EntityManagerFactory fornecido para subclasses.

Slide 21: Estrutura Spring - Hierarquia de exceções

• Spring fornece uma tradução conveniente sobre exceções de tecnologias específicas;

•As exceções do Spring envolvem a exceção original, convertendo-as em um conjunto de exceções de tempo de execução;

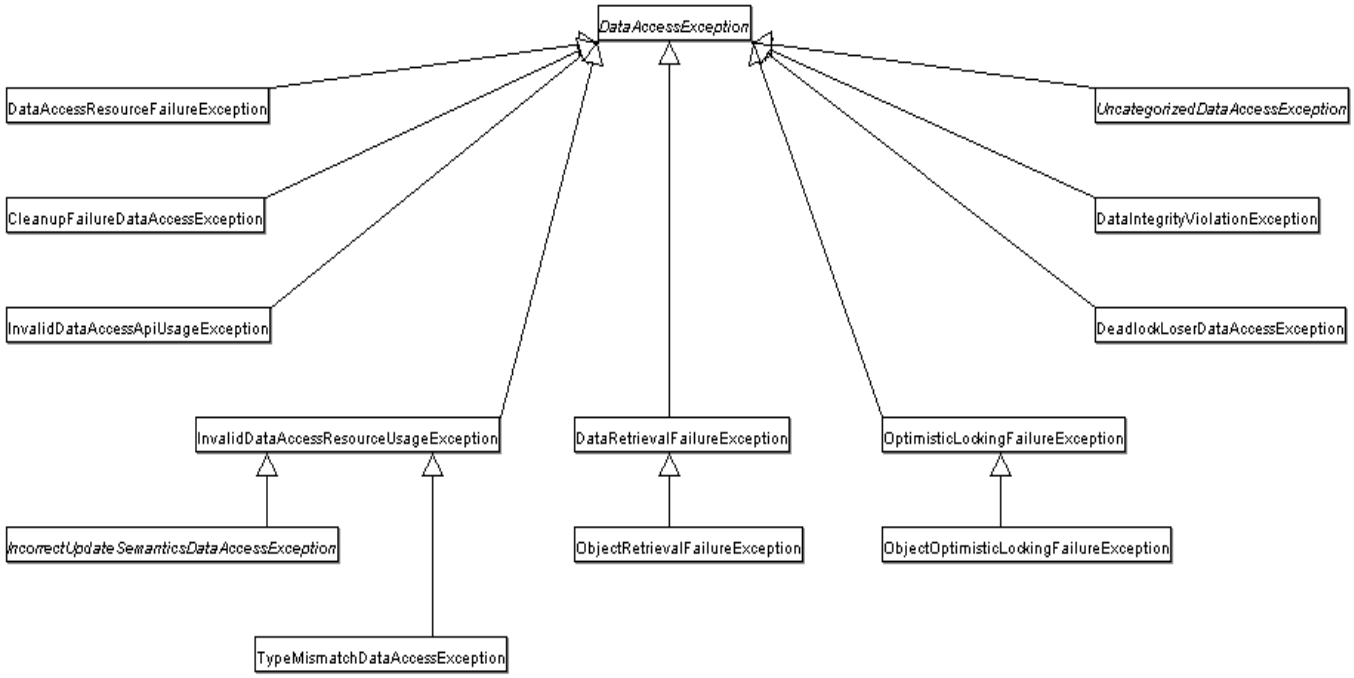
•Permite lidar com exceções de persistência mais facilmente;

•A raíz é a exceção DataAccessException;

Spring fornece uma tradução conveniente sobre exceções de tecnologias específicas como SQLException para a sua própria hierarquia de classe de exceções, sendo a DataAccessException a exceção raiz. Essas exceções envolvem a exceção original de modo que não é qualquer risco que se pode perder uma informação quanto ao que poderia ter dado errado.
Além de exceções JDBC, Spring também pode envolver exceções específicas do Hibernate , convertendo-as de proprietárias a um conjunto de exceções de tempo de execução focadas (o mesmo é verdadeiro para JDO e exceções JPA). Isso permite ao Spring lidar com exceções mais persistência, as quais são não-recuperáveis, a não ser nas camadas adequadas, sem ter blocos catch-and-throw ou declarações de exceção nos DAOs.

Slide 22: Estrutura Spring - Hierarquia de exceções

Hierarquia de exceção que Spring dispõe (um subconjunto):



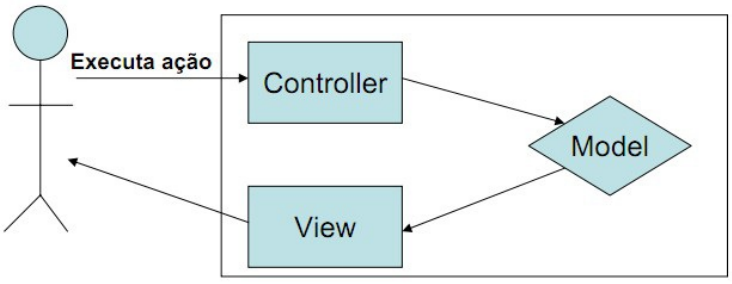
Slide 23: Estrutura do Spring - ORM
•O Spring ORM fornece integração com os seguintes frameworks de persistência: Hibernate, JDO, Oracle TopLink em termos de gestão de recursos, suporte à implementação da DAO e estratégias de transação.
•O seu fundamental mecanismo é o mapeamento de objetos para as tabelas do banco de dados (Object/Relational Mapping).

Slide 24: Estrutura Spring - WEB
• Spring Web
Para prover funcionalidades específicas para projetos Web, tem-se o módulo Spring Web. São funcionalidades como componentes para upload de arquivos e suporte para utilização de Inversão de Controle neste tipo de aplicação.

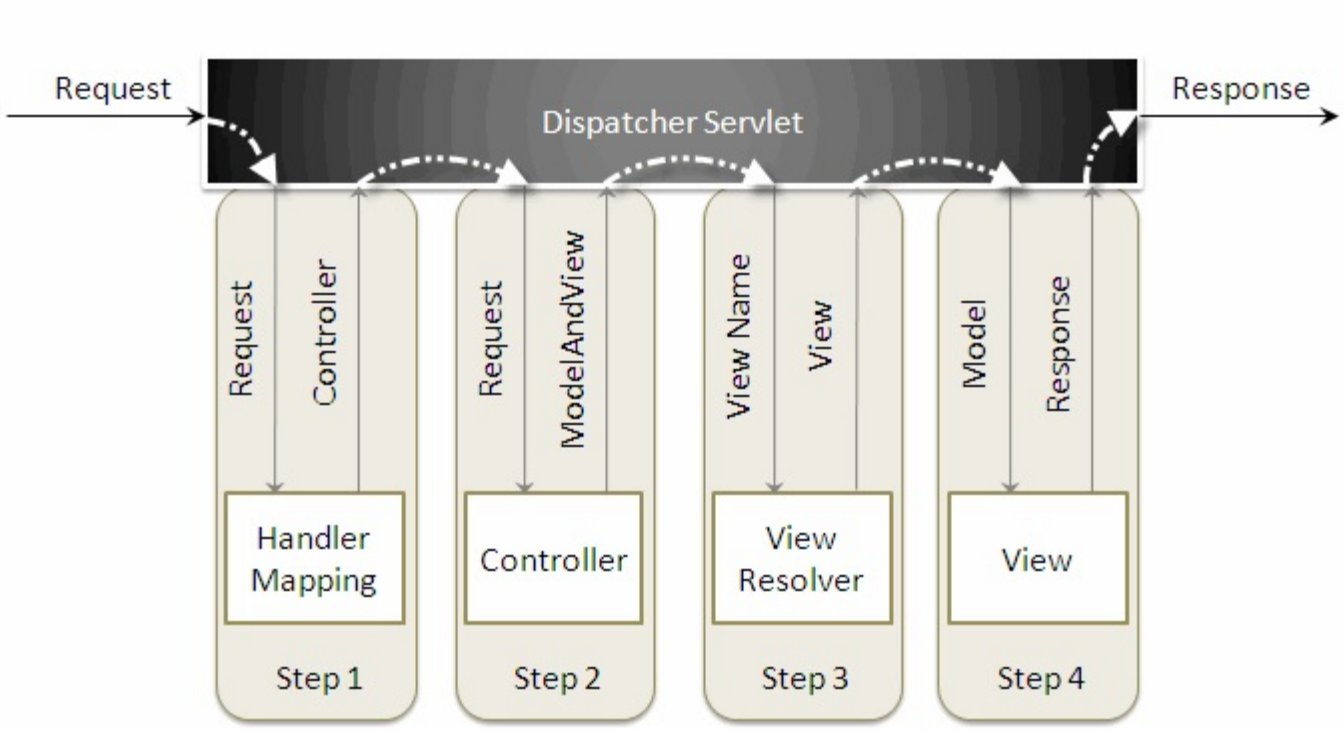
Slide 25: Estrutura do Spring- MVC
• Separa a lógica de negócio, lógica de apresentação e lógica de navegação.
– Model: encapsular dados da aplicação
– View: Interface.
– Controller: Intermediário entre as camadas view e model.

Slide 26: Spring MVC
Maioria dos controladores Spring derivam da interface: “org.springframework.web.servlet.mvc.Controller”

Slide 27: Estrutura do Spring- MVC



Slide 28: Estrutura do Spring- MVC



Slide 29: Estrutura do Spring- MVC

• Quando uma solicitação é enviada para o Spring MVC, a seguinte seqüência de eventos acontece:

• Step 1

- • O DispatcherServlet recebe o pedido.
- • O DispatcherServlet consulta o HandlerMapping e chama o Controller associado à solicitação.

• Step 2

- • O Controller processa o pedido chamando os métodos de serviço adequados e retorna um objeto ModeAndView ao DispatcherServlet.
- • O objeto ModeAndView contém os dados do Model e o nome da View

Slide 30: Spring MVC

• Step 3

- • O DispatcherServlet envia o nome da View ao viewResolver para encontrar a View atual para invocar.

• Step 4

- • Agora, o DispatcherServlet vai passar o objeto Model para a View para realizar o resultado.
- • A View com a ajuda dos dados do Model retornará o resultado para o usuário.

Slide 31: Bibliografia

Slide 32: Bibliografia

----------Consulta

Inversão de dependência
A inversão de dependência é o que torna uma simples biblioteca de classes diferente de um framework. Uma biblioteca consiste em um conjunto de classes que um usuário instancia e utiliza seus métodos. Após a chamada ao método, o controle do fluxo da aplicação retorna para o usuário. Entretanto, em um framework este fluxo é diferente. Para utilizar um framework, código próprio da aplicação deve ser criado e mantido acessível ao framework, podendo ser através de classes que estendem classes do próprio framework. O framework, então, realiza a chamada deste código da aplicação. Após a utilização do código da aplicação, o fluxo retorna para ele.

Injeção de Dependência
O padrão *Dependency Injection*, idealizado por Martin Fowler, trata-se de uma especialização do padrão Inversion of Control. Aplicações como Spring e PicoContainer, denominados de *lightweight containers,* adotam a inversão de controle, entretanto, todo framework utiliza-se de inversão de controle. A pergunta é, então, que tipo de inversão de controle o Spring, por exemplo, realiza? Afirmar que o Spring é um bom framework porque aplica a inversão de controle é um erro, já que qualquer framework deve aplicar este padrão.
Forma de se resolver as dependências entre objetos automaticamente 'injetando' referências sob demanda. O framework IoC, geralmente via arquivos de config. XML, determina como os objetos devem ser injetados.
Com isso, a responsabilidade de objetos dependentes vai pra config., reduz o acoplamento e encoraja desenv. baseado em interfaces, permite a aplicação ser reconfigurada sem mexer no código fonte (somente alterando o XML, por exemplo).

Constructor Injection e Setter Injection
Basicamente, existem dois tipos de injeção de dependência: Constructor Injection e Setter Injection. No primeiro tipo, Constructor Injection, a dependência é resolvida através de um construtor do objeto a receber o objeto dependente. A dependência é resolvida entre dois objetos passando para objeto1 uma implementação concreta de objeto2 através do seu construtor.

```
public class objeto1 {
   publicobjeto1 (objeto2  ob2) {
      this.ob2 = ob2;
         }
}
```

O tipo Setter Injection, a dependência entre os dois objetos é resolvida através de um método *Setter* no objeto2.

```
class objeto1 {
   private objeto2 ob2;

   public voidsetOb2(objeto2 ob2) {
```

```
        this.ob2 = ob2;
            }
}
```

A adoção destes dois padrões visa permitir ao desenvolvedor focalizar-se na implementação das características específicas da aplicação, delegando para um framework, como o Spring, a tarefa de especificar a dependência entre alguns objetos.

Spring é montado sobre três alicerces:
Inversão de controle (IoC), Programação orientada à aspectos (AOP) e Portable Services Abstraction (ORM, DAO, Web MVC, Web e outros).

Container de Inversão de Controle (IoC)
Conceito adotado pelo Spring de beans: qualquer objeto que forma sua aplicação e que está sob controle do Spring, é considerado um bean. Um bean trata-se apenas de um objeto de sua aplicação e nada mais.
O Container IoC é o responsável pelo gerenciamento destes beans.
Estes beans, entretanto, provavelmente possuem dependências entre si. Estas dependências são definidas através de metadados. O Container IoC obtém essas configurações e, partindo destas configurações, gerencia a dependência entre os beans.
A interface org.springframework.beans.factory.**BeanFactory** representa o Container IoC do Spring. Existem diversas implementações de BeanFactory, sendo a XmlBeanFactory (toda configuração de dependência entre os objetos é definida em um arquivo XML) a implementação mais comum.

Programação orientada à aspectos (AOP - Aspects Oriented Programming)
Permite separar e organizar o código de acordo com a sua importância para a aplicação (*separation of concerns*). A OAP permite que o código utilizado para implementar funcionalidades secundárias e que encontra-se espalhado por toda a aplicação (*crosscutting concern*) seja encapsulado e modularizado. Quando o código não está devidamente encapsulado aumenta a complexidade do sistema e torna a manutenção do sistema muito mais difícil.
A OAP busca resolver esse problema permitindo que o programador implemente essas questões (segurança, log, transações, e etc) através de aspectos. O aspecto é constituído de uma ou mais peças de *advices* (fragmentos de código, como métodos) e uma lista de *join points* (pontos no programa principal na qual os *advices* são inseridos). Por exemplo, um módulo de segurança pode incluir um *advice* que faz uma verificação de segurança, com instruções para inserir este fragmento de código no início dos métodos a(), b() e c() de algumas classes. Os "pontos de inserção" são conhecidos como *pointcuts*.
OAP tenta se separar os problemas, aumentar a modularidade e reduzir as redundâncias.

Portable Services Abstraction - PSA (ORM, DAO, Web MVC, Web e outros)
PSA no Spring é basicamente tudo que não é IoC ou AOP.
Rodam em qualquer sistema
• Ex.: Hibernate, JDBC, LDAP, Spring Beans. ...

Spring Bean Container
Tipos:
Bean Factory
• Provê suporte básico para a injeção de dependência

• Gerencia configs e ciclo de vida de beans

Application Context

• Cria um Bean Factory e adiciona serviços:
• Trata mensagens para internacionalização
• Carrega recursos genéricos
• Dispara e trata eventos

Container cria beans em ordem de acordo com grafo de deps.

• Beans são criados usando contrutores (geralmente sem argumentos) ou métodos de factory
• Dependência não injetadas via contrutores são resolvidas por setters
• Sem chamadas a Contexts e lookups
• Ambos beans iniciados apenas na chamada factory.getBean()
• Programação orientada a interfaces: implementações utilizadas podem mudar com alteração apenas no XML

ORM

O Spring Framework fornece integração com os seguintes frameworks de persistência: Hibernate, JDO, Oracle TopLink em termos de gestão de recursos, suporte à implementação da DAO e estratégias de transação. O seu fundamental mecanismo é o mapeamento de objetos para as tabelas do banco de dados (Object/Relational Mapping). Ao invés de fazer o acesso à banco de dados relacionais utilizando o tradicional JDBC/SQL, ele usa a persistência de objetos mapeando-os para as tabelas de dados.

DAO

O Data Access Object (DAO) do Spring visa tornar mais fácil a atividade de trabalhar com tecnologias de acesso de dados como JDBC, Hibernate ou JDO de uma forma consistente. Permite alternar entre as tecnologias de persistência com bastante facilidade e também permite o código sem se preocupar com captura de exceções que são específicos para cada tecnologia.

(The Data Access Object (DAO) support in Spring is aimed at making it easy to work with data access technologies like JDBC, Hibernate or JDO in a consistent way. This allows one to switch between the aforementioned persistence technologies fairly easily and it **also allows one to code** without worrying about catching exceptions that are specific to each technology.)

DAO

Interface independente do tipo de acesso de dados utilizado para busca ou persistência.

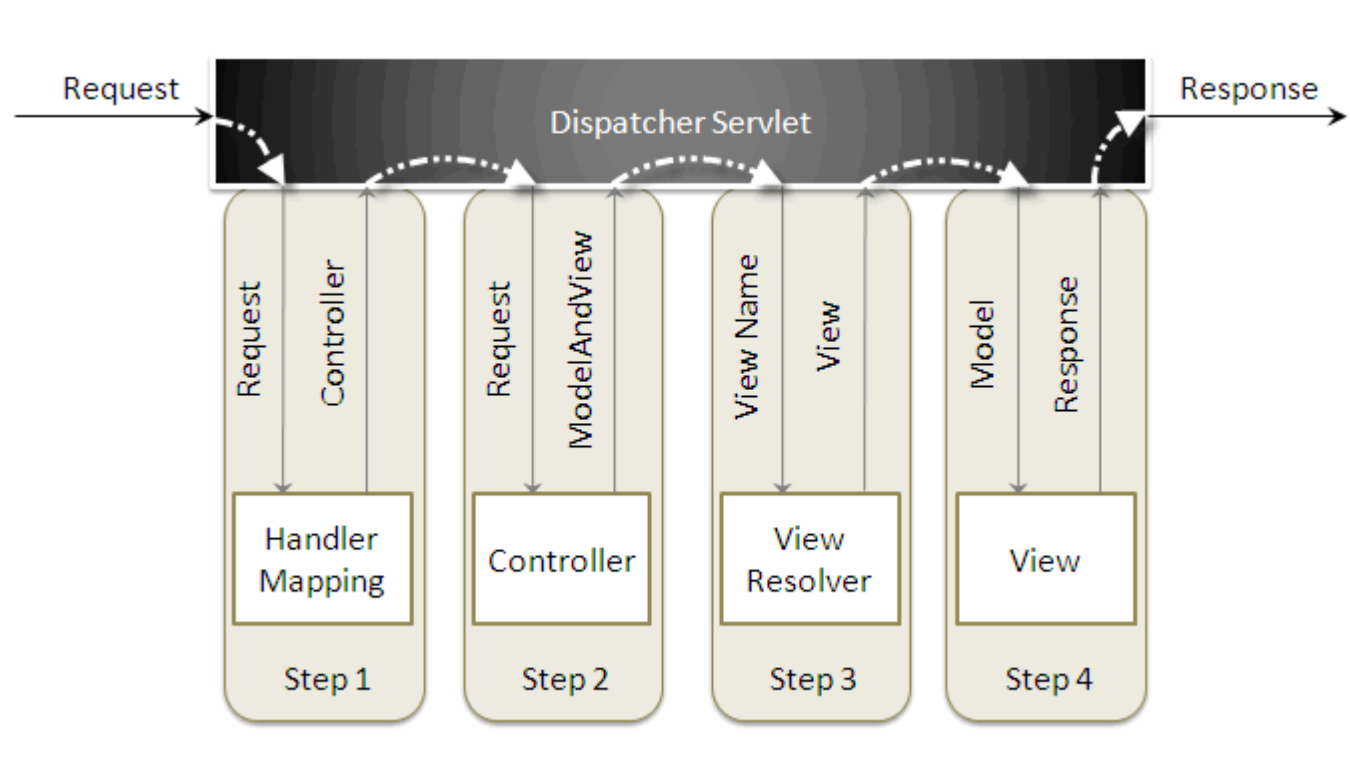
Encapsula o acesso aos objetos persistentes do domínio

– Persiste objetos transientes
– Atualiza objetos persistidos
– Busca objetos persistidos

• Desacoplamento entre objetos de serviço e objetos de persistência
• Implementação por interfaces
– Permite a troca da camada de persistência sem impactar a camada de serviço
– Container de IoC usa injeção de dependência dos objetos DAOs aos

objetos de serviço

Web MVC
Spring MVC ajuda na construção de aplicações web flexível e de baixo acoplamento. O padrão de projeto Model-View-Controller ajuda a separar a lógica de negócio, lógica de apresentação e lógica de navegação. Os models são responsáveis por encapsular os dados do aplicativo. As Views retornam resposta para o usuário com a ajuda do objeto model. Os controllers são responsáveis por receber a solicitação do usuário e chamar os serviços de back-end.
Quando uma solicitação é enviada para o Spring MVC, a seguinte seqüência de eventos acontece:
O DispatcherServlet recebe o pedido.
O DispatcherServlet consulta o HandlerMapping e chama o Controller associado à solicitação.
O Controller processa o pedido chamando os métodos de serviço adequados e retorna um objeto ModeAndView ao DispatcherServlet. O objeto ModeAndView contém os dados do Model e o nome da View
O DispatcherServlet envia o nome da View ao viewResolver para encontrar a View atual para invocar.
Agora, o DispatcherServlet vai passar o objeto Model para a View para realizar o resultado.
A View com a ajuda dos dados do Model retornará o resultado para o usuário.



http://www.vaannila.com/spring/spring-mvc-tutorial-1.html

Web

Spring includes:

- **The most complete lightweight container**, providing centralized, automated configuration and wiring of your application objects. The container is *non-invasive*, capable of assembling a complex system from a set of loosely-coupled components (POJOs) in a consistent and transparent fashion. The container brings agility and leverage, and improves application testability and scalability by allowing software components to be first developed and tested in isolation, then scaled up for deployment in any environment (J2SE or J2EE).
- **A common abstraction layer for transaction management**, allowing for pluggable transaction managers, and making it easy to demarcate transactions without dealing with low-level issues. Generic strategies for JTA and a single JDBC DataSource are included. In contrast to plain JTA or EJB CMT, Spring's transaction support is not tied to J2EE environments.
- **A JDBC abstraction layer** that offers a meaningful exception hierarchy (no more pulling vendor codes out of SQLException), simplifies error handling, and greatly reduces the amount of code you'll need to write. You'll never need to write another finally block to use JDBC again. The JDBC-oriented exceptions comply to Spring's generic DAO exception hierarchy.
- **Integration with Toplink, Hibernate, JDO, and iBATIS SQL Maps**: in terms of resource holders, DAO implementation support, and transaction strategies. First-class Hibernate support with lots of IoC convenience features, addressing many typical Hibernate integration issues. All of these comply to Spring's generic transaction and DAO exception hierarchies.
- **AOP functionality**, fully integrated into Spring configuration management. You can AOP-enable any object managed by Spring, adding aspects such as declarative transaction management. With Spring, you can have declarative transaction management without EJB... even without JTA, if you're using a single database in Tomcat or another web container without JTA support.
- **A flexible MVC web application framework**, built on core Spring functionality. This framework is highly configurable via strategy interfaces, and accommodates multiple view technologies like JSP, Velocity, Tiles, iText, and POI. Note that a Spring middle tier can easily be combined with a web tier based on any other web MVC framework, like Struts, WebWork, or Tapestry.

Integração com Toplink, Hibernate, JDO e iBATIS SQL Maps

- Funcionalidade AOP

permite o gerenciamento de transações declarativas sem utilizar o EJB e JPA

- Framework para aplicações web MVC flexível

pode ser altamente configurável através de interfaces de estratégia, e acomoda

tecnologias de visão múltipla, como JSP, Velocity, Tiles, iText, ePOI. Uma parte da camada intermediária pode ser facilmente combinada com uma camada de web com base em qualquer outro web framework MVC, como Struts,WebWork ou Tapestry