

INVERSION OF CONTROL ИНВЕРСИЯ УПРАВЛЕНИЯ На примере Spring

Черных Сергей /schernykh@naumen.ru

ИНТЕРФЕЙСЫ VS. КЛАССЫ

ИНТЕРФЕЙСЫ VS. КЛАССЫ

- Неявный интерфейс класса - что доступно извне
- Интерфейс - это способ взаимодействия
- Класс реализует интерфейс
- Что необходимо и достаточно реализовать

ПОЧЕМУ ЭТО ВАЖНО?

- Я пишу один, без команды!
- Моим кодом не пользуются другие программисты!



- Инкапсуляция - основа хорошего дизайна
- Явный контракт легче соблюдать чем неявный
- Легче читать код
- Легче тестировать
- Легче рефакторить
- Вы через год - это другой человек

ФРЕЙМВОРКИ И БИБЛИОТЕКИ

ФРЕЙМВОРКИ И БИБЛИОТЕКИ

- Всё уже написано
- И протестировано (как тестами, так и другими программистами)
- Новые технологии (на которые у вас нет времени)
- Улучшения (на которые у вас нет времени)
- Простое вхождение в проект

ФРЕЙМВОРКИ И БИБЛИОТЕКИ. ПРОБЛЕМЫ

- Отступления - могут быть дорогими / невозможными
- Развитие может уйти в ненужную для компании сторону
- Переход с версии на версию требует затрат
- Окончание поддержки и развития

ФРЕЙМВОРК/БИБЛИОТЕКА

- Что означают эти термины?
- Чем они отличаются?

БИБЛИОТЕКА

БИБЛИОТЕКА

- Набор функций, которые можно вызвать
- Библиотека делает работу и возвращает результат
- Примеры: прочитать файл; сделать запись в лог

ФРЕЙМВОРК

ФРЕЙМВОРК

- Воплощает какой-то дизайн, архитектуру
- Поведение уже встроено в фреймворк
- Фреймворк - это каркас

ЧТОБЫ ИСПОЛЬЗОВАТЬ ФРЕЙМВОРК

- Встроить своё поведение в различные места фреймворка
- Собственные классы, или реализации интерфейсов (наследники)
- Код фреймворка вызывает ваш код в конкретных точках

НЕМНОГО ПРО УПРАВЛЕНИЕ

ПРИМЕР 1: КОМАНДНАЯ СТРОКА

```
public static void main(String args[]) {  
    Scanner scanner = new Scanner(System.in);  
    String str = "";  
    while (!str.equals("exit")) {  
        str = scanner.nextLine();  
        ...  
    }  
}
```

- Полное управление
- Ждём команды пользователя - пишем результат

ПРИМЕР 2: HTTP-ЗАПРОС

```
@RequestMapping(path = "/")
public ModelAndView index() {
    List clients = influxDAO.getDbList();
    ...
    return new ModelAndView("clients", model, HttpStatus.OK);
}
```

- Мы просто возвращаем ответ
- Что происходило до вызова метода?
- Теряем управление

INVERSION OF CONTROL (ИНВЕРСИЯ УПРАВЛЕНИЯ)

- Ключевая характеристика фреймворка
- Контроль остаётся в коде фреймворка
- Пользователь теряет контроль

КАК ВСТРОИТЬ СВОЙ КОД

- Фреймворк генерирует события, клиентский код - подписывается
- Интерфейс, который должен быть реализован клиентом
- Шаблонный метод (Template method) (см. JUnit)

JUNIT

- Мы не управляем порядком вызовов
- Только встраиваем свой код

```
public abstract class TestCase extends Assert implements Test {  
    ....  
    protected void runTest() throws Throwable {  
    }  
    protected void setUp() throws Exception {  
    }  
    protected void tearDown() throws Exception {  
    }  
}
```

КАК ВСТРОИТЬ СВОЙ КОД

- Шаблон “Фабрика” (Factory)
- Service locator (JNDI)
- Внедрение зависимостей

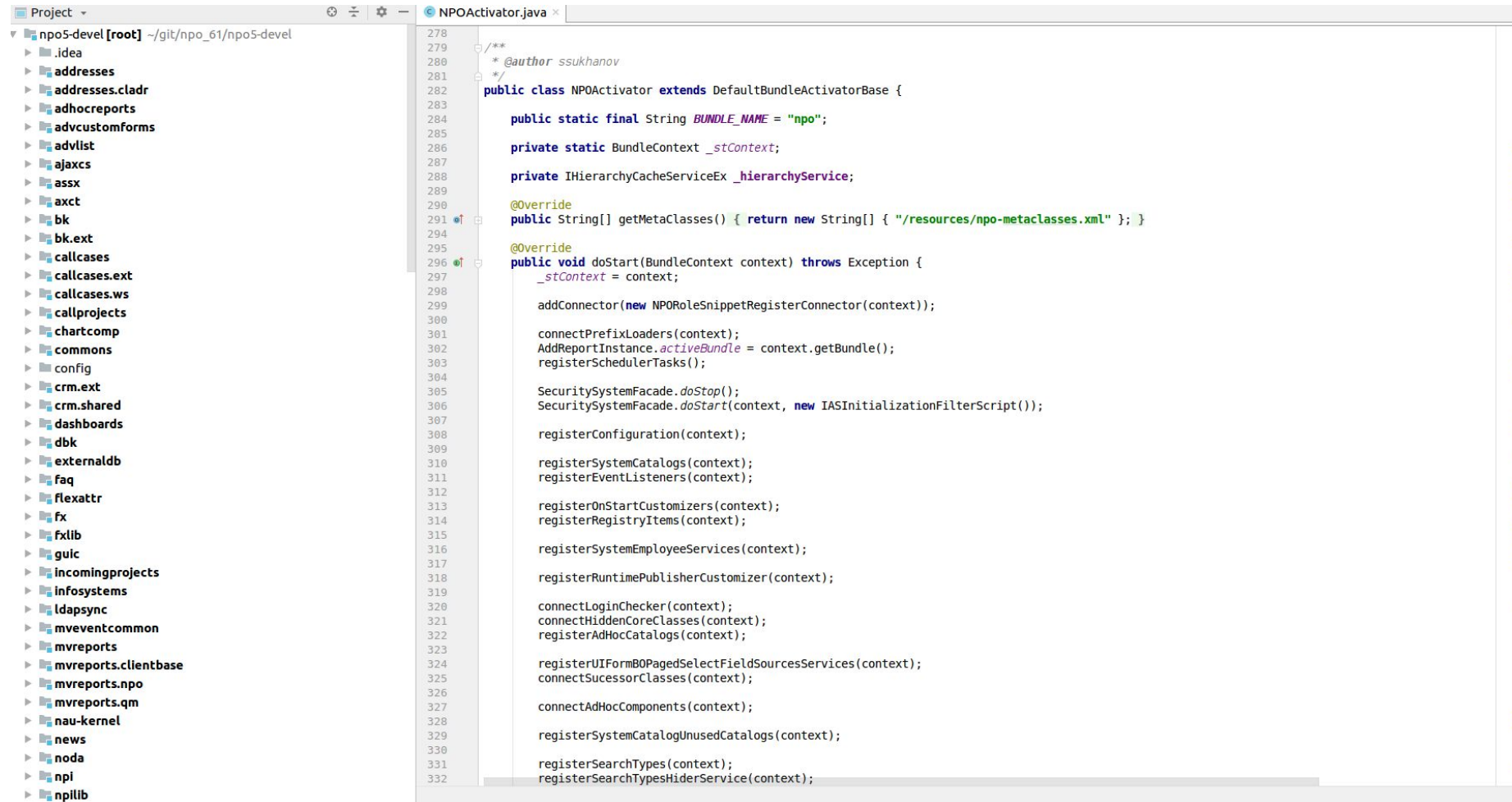
SERVICE LOCATOR

```
public Collection<Movie> find() {  
    MovieFinder finder = (MovieFinder)  
        ServiceLocator.getService("MovieFinder");  
    ...  
}
```

SERVICE LOCATOR

- Считается антипаттерном - скрывает зависимости

SERVICE LOCATOR



The image shows a screenshot of an IDE with a project structure on the left and the source code for `NPOActivator.java` on the right. The project structure includes various sub-projects like `addresses`, `advcustomforms`, `advlist`, `ajaxcs`, `assx`, `axct`, `bk`, `bk.ext`, `callcases`, `callcases.ext`, `callcases.ws`, `callprojects`, `chartcomp`, `commons`, `config`, `crm.ext`, `crm.shared`, `dashboards`, `dbk`, `externaldb`, `faq`, `flexattr`, `fx`, `fxlib`, `guic`, `incomingprojects`, `infosystems`, `ldapsync`, `mveventcommon`, `mvreports`, `mvreports.clientbase`, `mvreports.npo`, `mvreports.qm`, `nau-kernel`, `news`, `noda`, `npi`, and `npilib`.

```
278
279
280  /**
281   * @author ssukhanov
282   */
283  public class NPOActivator extends DefaultBundleActivatorBase {
284
285      public static final String BUNDLE_NAME = "npo";
286
287      private static BundleContext _stContext;
288
289      private IHierarchyCacheServiceEx _hierarchyService;
290
291      @Override
292      public String[] getMetaClasses() { return new String[] { "/resources/npo-metaclasses.xml" }; }
293
294      @Override
295      public void doStart(BundleContext context) throws Exception {
296          _stContext = context;
297
298          addConnector(new NPORoleSnippetRegisterConnector(context));
299
300          connectPrefixLoaders(context);
301          AddReportInstance.activeBundle = context.getBundle();
302          registerSchedulerTasks();
303
304          SecuritySystemFacade.doStop();
305          SecuritySystemFacade.doStart(context, new IASInitializationFilterScript());
306
307          registerConfiguration(context);
308
309          registerSystemCatalogs(context);
310          registerEventListeners(context);
311
312          registerOnStartCustomizers(context);
313          registerRegistryItems(context);
314
315          registerSystemEmployeeServices(context);
316
317          registerRuntimePublisherCustomizer(context);
318
319          connectLoginChecker(context);
320          connectHiddenCoreClasses(context);
321          registerAdHocCatalogs(context);
322
323          registerUIFormBOPagedSelectFieldSourcesServices(context);
324          connectSucessorClasses(context);
325
326          connectAdHocComponents(context);
327
328          registerSystemCatalogUnusedCatalogs(context);
329
330          registerSearchTypes(context);
331          registerSearchTypesHiderService(context);
332
```

SERVICE LOCATOR

```
1022 private void registerWebPageManagerConfiguration(BundleContext context) {
1023
1024     List<IWebPageContentManagerConfigurator> resources = new ArrayList<>();
1025
1026     resources.add(new WebPageResourcesSourceRegistrator(WebPageContentManagerFacade.PAGE,
1027         (IWebPageResourceSource) (component, resources) -> {
1031         resources.addAll(SPI.getReference(IFaviconProvider.class).getResources());
1032         return CollectStatus.Static;
1033     }));
1036
1037     addConnector(new WebPageContentManagerServiceConnector(context, resources));
1038 }
1039
1040 private void registerChangesInterceptors(BundleContext context) {
1041     List<ObjectChangesInterceptorResource> resources = Arrays.asList( //
1042         new ObjectChangesInterceptorResource(new ProjectRoleChangesInterceptor(), order: 10), //
1043         new ObjectChangesInterceptorResource(new PartnerStateChangesInterceptor(), order: 10), //
1044         new ObjectChangesInterceptorResource(new ScriptChangesInterceptor(), order: 10) //
1045     );
1046
1047     addConnector(new ObjectChangesInterceptorsServiceConnector(context, resources));
1048
1049     addConnector(new ObjectChangesClassesServiceConnector(context,
1050         new ObjectChangesServiceClassesResource(new BusinessObjectServiceInterceptor(),
1051             new ClassListAcceptor(AdvCustomFormGroupView.class, IAdvCustomFormScriptOwner.class))));
1052 }
1053
1054 private void registerProjectRoles(BundleContext context) {
1055     IProjectRolesService service = SPI.getReference(IProjectRolesService.class);
1056
1057     service.addRole(new ProjectRole( roleId: "scripting",
1058         Predicates.or(byScriptType(ScriptLicenseType.BASIC), byScriptType(ScriptLicenseType.ADVANCED))));
1059     service.addRole(new ProjectRole( roleId: "advanced-scripting", byScriptType(ScriptLicenseType.ADVANCED)));
1060     service.addPropertyFiller(new AdvCustomFormProjectPropertiesFiller());
1061 }
```

DEPENDENCY INJECTION ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ

- Пример от Мартина Фаулера

<http://www.martinfowler.com/articles/injection.html>

- Определим интерфейс

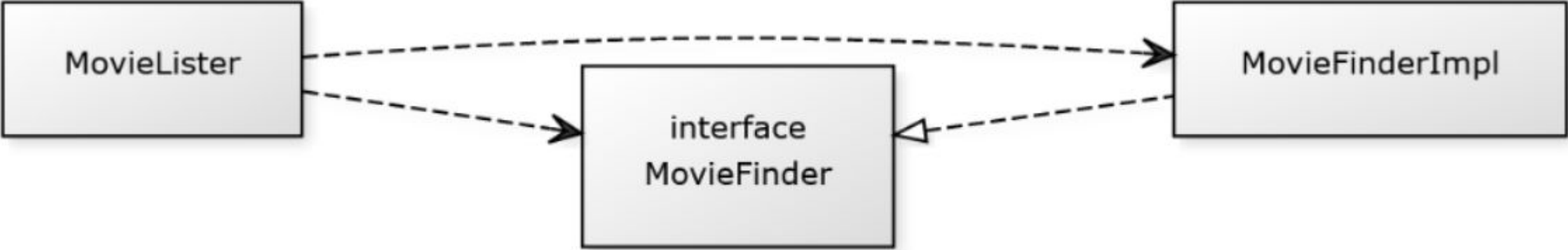
```
public interface MovieFinder {  
    List<Movie> findAll();  
}
```

```
public class MovieLister {...  
  
    public Collection<Movie> moviesDirectedBy(String director) {  
  
        List allMovies = finder.findAll();  
  
        for(Iterator it = allMovies.iterator(); it.hasNext();) {  
            Movie movie = it.next();  
            if(!movie.getDirector().equals(director)) it.remove();  
        }  
  
        return allMovies;  
    }  
}
```

- Откуда мы возьмём конкретную реализацию?

```
public class MovieLister {  
  
    private MovieFinder finder;  
  
    public MovieLister() {  
        finder = new ColonDelimitedMovieFinder("movies1.txt");  
    }  
}
```

Схема зависимостей



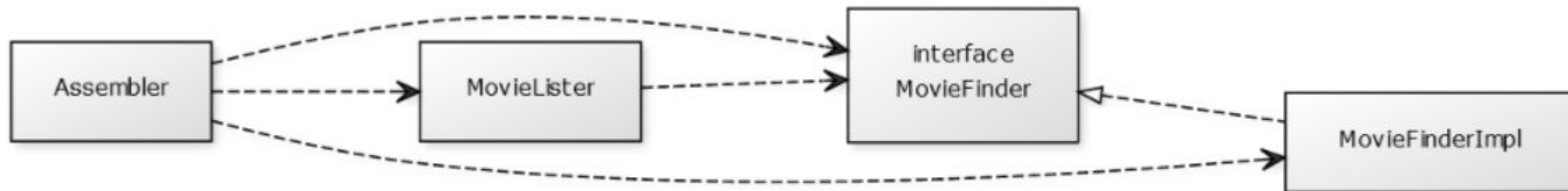
В ЧЕМ ПРОБЛЕМА?

- Lister зависит и от интерфейса, и от реализации Finder-a
- Если убрать зависимость - откуда брать реализацию?
- Как использовать разные реализации в разных условиях?

DEPENDENCY INJECTION

Решение

- Assembler, управляющий зависимостями
- Предоставляет конкретную реализацию для интерфейса



DEPENDENCY INJECTION

- Spring: внедрение через поле

```
@Component
public class ColonDelimitedMovieFinder implements MovieFinder {
}
```

```
@Component
public class MovieLister {
    @Autowired
    private MovieFinder finder;
}
```

DEPENDENCY INJECTION

Было/Стало

```
public class MovieLister {  
    private MovieFinder finder;  
  
    public MovieLister() {  
        finder = new ColonDelimitedMovieFinder("movies1.txt");  
    }  
}
```

```
@Component  
public class MovieLister {  
    @Autowired  
    private MovieFinder finder;  
}
```

•Spring: внедрение через конструктор

```
@Autowired  
public class ColonDelimitedMovieFinder implements MovieFinder {  
}
```













```
@Component  
public class MovieLISTER {  
  
    private MovieFinder finder;  
  
    @Autowired  
    public MovieLISTER(MovieFinder fnd) {  
        this.finder = fnd;  
    }  
...  
}
```

ВНЕДРЕНИЕ ЧЕРЕЗ КОНСТРУКТОР

- Удобно для тестирования
- Лучше видны зависимости
- Не удобно делать слишком много зависимостей

SPRING FRAMEWORK

SPRING ВСЕ ПРОЕКТЫ

| | | | | | |
|--|---|--|---|---|---|
|  <p>SPRING IO PLATFORM Provides a cohesive, versioned platform for building modern applications. It is a modular, enterprise-grade distribution that delivers a curated set of dependencies.</p> |  <p>SPRING BOOT Takes an opinionated view of building Spring applications and gets you up and running as quickly as possible.</p> |  <p>SPRING FRAMEWORK Provides core support for dependency injection, transaction management, web apps, data access, messaging and more.</p> |  <p>SPRING CLOUD DATA FLOW An orchestration service for composable data microservice applications on modern runtimes.</p> |  <p>SPRING CLOUD Provides a set of tools for common patterns in distributed systems. Useful for building and deploying microservices.</p> |  <p>SPRING DATA Provides a consistent approach to data access – relational, non-relational, map-reduce, and beyond.</p> |
|  <p>SPRING INTEGRATION Supports the well-known Enterprise Integration Patterns via lightweight messaging and declarative adapters.</p> |  <p>SPRING BATCH Simplifies and optimizes the work of processing high-volume batch operations.</p> |  <p>SPRING SECURITY Protects your application with comprehensive and extensible authentication and authorization support.</p> |  <p>SPRING HATEOAS Simplifies creating REST representations that follow the HATEOAS principle.</p> |  <p>SPRING SOCIAL Easily connects your applications with third-party APIs such as Facebook, Twitter, LinkedIn, and more.</p> |  <p>SPRING AMQP Adopts core Spring concepts to the development of AMQP-based messaging solutions.</p> |
|  <p>SPRING MOBILE Simplifies the development of mobile web apps through device detection and progressive rendering options.</p> |  <p>SPRING FOR ANDROID Provides key Spring components for use in developing Android applications.</p> |  <p>SPRING WEB FLOW Supports building web applications with controlled navigation such as checking in for a flight or applying for a loan.</p> |  <p>SPRING WEB SERVICES Facilitates the development of contract-first SOAP web services.</p> |  <p>SPRING LDAP Simplifies the development of applications using LDAP using Spring's familiar controller-based approach.</p> |  <p>SPRING SESSION Spring Session provides an API and implementations for managing a user's session information.</p> |
|  <p>SPRING SHELL Provides a powerful foundation for building command-line apps using a Spring-based programming model.</p> |  <p>SPRING XD Simplifies the development of big data applications by addressing ingestion, analytics, batch jobs and data export.</p> |  <p>SPRING FLD A JavaScript library that offers a basic extensible HTML5 visual builder for pipelines and simple graphs.</p> |  <p>SPRING KAFKA Provides Familiar Spring abstractions for Apache Kafka.</p> | | |

SPRING FRAMEWORK

- **Dependency Injection**
- Aspect-Oriented Programming including Spring's declarative transaction management
- Spring MVC web application and RESTful web service framework
- Foundational support for JDBC, JPA, JMS
- Much more...

ВНЕДРЕНИЕ ЗАВИСИМОСТЕЙ В SPRING

- IoC-контейнер - любая реализация DI, например Spring
- BeanFactory (Фабрика бинов) - Ассемблер, сборщик
- Bean (Бин) - объект системы, содержащий логику

БИН (BEAN)

Это объект системы, который

- создан
- управляется

Spring-ом, т.е. IoC-контейнером

КАК СОЗДАТЬ BEAN

Может быть помечен аннотацией

- `@Named` (JSR-330)
- `@Component` (Spring)
- `@Service` (Spring)

и много других в Spring

КАК СОЗДАТЬ *BEAN*

```
@Component
public class CsvMovieFinder implements MovieFinder {
    private String fileName;
        ...
}
```

КАК СОЗДАТЬ *BEAN*

Может быть создан конфигурацией

```
@Configuration
public class MovieFinderConfiguration {

    @Bean
    public MovieFinder movieFinder() {
        ...
        Collection movies =
            moviesDao.getAllMoviesFromDataBase();
        ...
        return new CollectionMovieFinder(movies);
    }
}
```

Например, если нужно выполнить дополнительные действия, которые не хочется делать частью логики бина

КАК СОЗДАТЬ *BEAN*

Bean может быть даже строкой

```
@Configuration
public class DatabaseConfiguration {

    @Bean
    public String databaseVendor() {
        ...
        String vendor = getVendor();
        ...
        return vendor;
    }
}
```

КАК ВНЕДРИТЬ *BEAN*

- *@Inject* (JSR-330)
- *@Autowired* (Spring)

КАК ВНЕДРИТЬ *BEAN*

Внедрение через поле

```
@Component  
public class MovieLister {  
  
    @Autowired  
    private MovieFinder finder;  
}
```


КАК ВНЕДРИТЬ *BEAN*

- Внедрение через конструктор

```
@Component
public class MovieLister {

    private MovieFinder finder;

    @Autowired
    public MovieLister(MovieFinder finder) {
        this.finder = finder;
    }
}
```

ВНЕДРЕНИЕ BEAN

Если внедряется один бин:

- Должен определяться однозначно
- Или не должно быть других бинов с таким интерфейсом
- Или внедрение должно быть по имени
- Или должны быть заданы приоритеты

КАК ВНЕДРИТЬ BEAN

Внедрение списка бинов

```
@Component
public class MartinScorsese implements Director {
@Component
public class JamesCameron implements Director {
```

```
@Component
public class DirectorsService {
    @Autowired
    private List<Director> allDirectors;
```

Если есть несколько реализаций одного интерфейса

КАК ВНЕДРИТЬ BEAN

Внедрение бина по имени

```
@Component("csvFinder")
public class CsvMovieFinder implements MovieFinder {

@Component("oracleFinder")
public class OracleMovieFinder implements MovieFinder {
```

```
@Component
public class MovieLISTER {
    @Autowired @Qualifier("csvFinder")
    private MovieFinder finder;
```

Если есть несколько реализаций одного интерфейса

ИМЯ BEAN

- Указано явно в аннотации `@Named/@Component`
- Имя класса со строчной буквы, если создан через аннотацию
- Имя метода со строчной буквы, если создан в конфигурации

КАК ПОЛУЧИТЬ BEAN

- Получение бина из фабрики

```
@Component
public class MovieLister {...

    @Autowired
    private BeanFactory factory;

    private String beanName;

    public Collection moviesDirectedBy(String director) {
        MovieFinder finder = factory.getBean(beanName);
        List allMovies = finder.findAll();
    }
}
```

ЖИЗНЕННЫЙ ЦИКЛ БИНА

- `@PostConstruct`
- `@PreDestroy`

ЖИЗНЕННЫЙ ЦИКЛ БИНА

@PostConstruct

- Вызван конструктор
- Внедрены все зависимости
- Сразу после этого - вызов метода

ЖИЗНЕННЫЙ ЦИКЛ БИНА

- `@PostConstruct`

```
@Component
public class CachingMovieLister {

    @PostConstruct
    public void populateMovieCache() {
        // Загружаем данные из БД в кэш
        cache.addAll(readFromDataBase());
    }
}
```

ЖИЗНЕННЫЙ ЦИКЛ БИНА

@PreDestroy

- Бин удаляется из фабрики

ЖИЗНЕННЫЙ ЦИКЛ БИНА

@PreDestroy

- Зависит от Scope

```
@Component
public class CachingMovieLister {

    @PreDestroy
    public void clearMovieCache() {
        // Освобождаем кэш
        cache.clear();
    }
}
```

ОБЛАСТЬ ВИДИМОСТИ

(SCOPE)

- Singleton - по умолчанию
- Prototype - новый экземпляр при каждом вызове
- Request - на один HTTP-запрос
- Session - на одну HTTP-сессию

ОБЛАСТЬ ВИДИМОСТИ

```
@Configuration
public class MovieFinderConfiguration {

    @Bean
    @Scope("prototype")
    public CsvMovieFinder csvMovieFinder() {
        return new CsvMovieFinder("movies.csv");
    }
}
```

ОБЛАСТЬ ВИДИМОСТИ

**ВОПРОС: ЧТО ПРОИСХОДИТ ЕСЛИ ВНЕДРИТЬ
PROTOTYPE-БИН В SINGLETON-БИН?**

```
@Component
public class SingletonBean {

    @Autowired
    private PrototypeBean prototypeBean;

}
```

ВНЕДРЕНИЕ PROTOTYPE БИНОВ

- **@LOOKUP**

```
@Component
public class SingletonLookupBean {

    @Lookup
    public PrototypeBean getPrototypeBean() {
        return null;
    }
}
```

ВНЕДРЕНИЕ СИСТЕМНЫХ СВОЙСТВ

```
@Component
public class SomeBean {

    @Value("#{systemProperties['influxHost']}")
    private String influxHost;
}
```

Системное свойство можно задать через

Аргумент VM:

- -DinfluxHost="localhost"
- Переменную окружения

Аналог: `System.getProperty("influxHost")`

ВНЕДРЕНИЕ КОНФИГУРАЦИИ

```
@Component
public class InfluxDAO {
    public InfluxDAO(
        @Value("${influx.host}") String influxHost...
    )
}
```

- Конфигурация задаётся в property файлах
- Внимание! \$ вместо #

АННОТАЦИИ В SPRING МОГУТ "НАСЛЕДОВАТЬСЯ"

```
@Target({ElementType.TYPE})  
@Retention(RetentionPolicy.RUNTIME)  
@Documented  
@Component  
public @interface Controller {
```

- Controller - это тоже Component, т.е. Bean

АННОТАЦИИ В SPRING МОГУТ "НАСЛЕДОВАТЬСЯ"

```
@Documented  
@Controller  
@ResponseBody  
public @interface RestController {
```

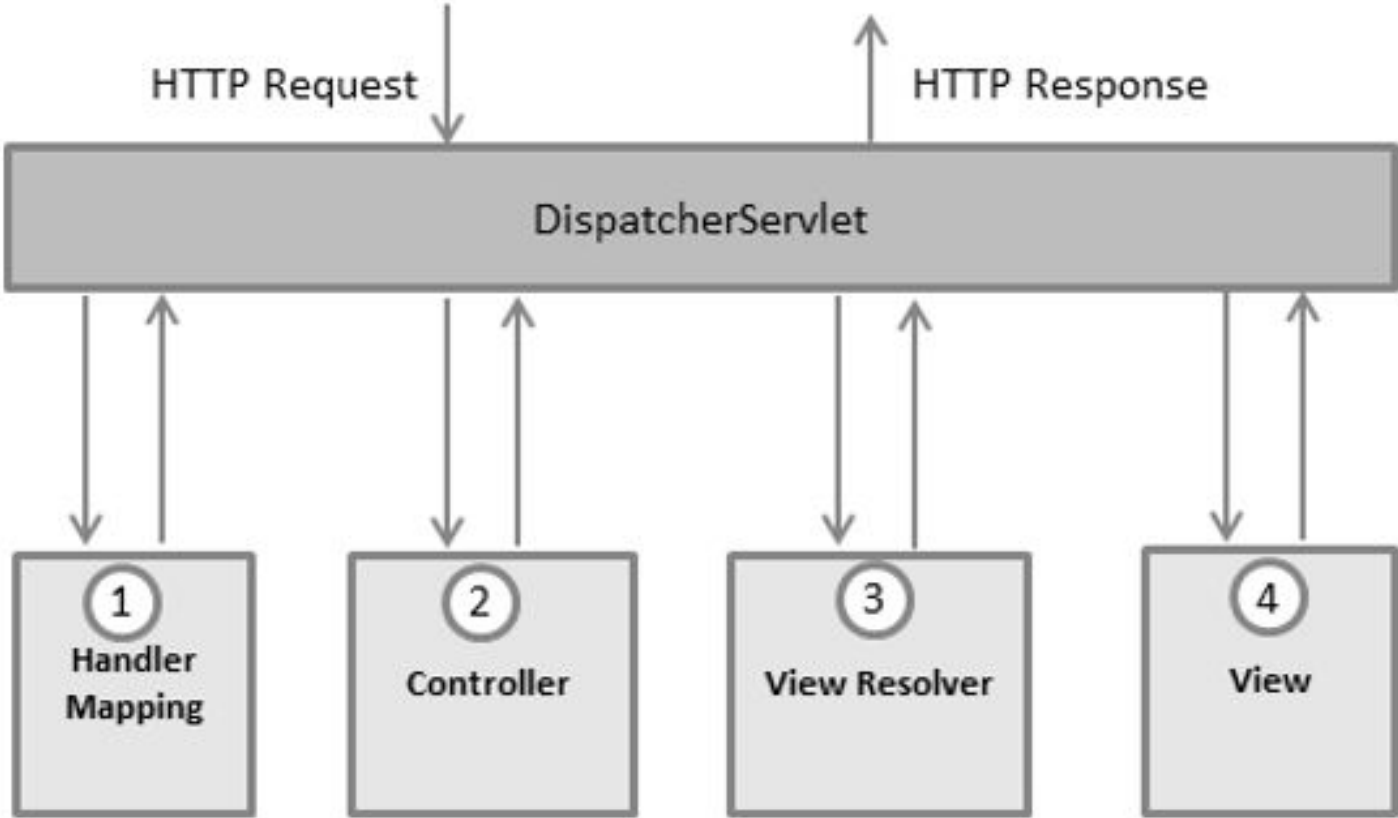
- RestController - это тоже Controller, т.е. Component...

SPRING MVC

SPRING MVC

- Model - инкапсуляция данных приложения
- View - отображение данных (сделает HTML на выходе)
- Controller - обработка запроса, создание Модели и передача в Вид

DISPATCHER SERVLET



КОНТРОЛЛЕР

```
@Controller
public class HistoryController {

    @RequestMapping(path = "/history/{client}")
    public ModelAndView indexLast864(
        @PathVariable("client") String client,
        @RequestParam(name = "count") int count)

        ...
        Map model = new HashMap<>(d.asModel());
        model.put("client", client);

        return new ModelAndView("history", model, HttpStatus.OK);
}
```

ВИД - HISTORY.JSP

```
...  
<div class="container">  
    <br>  
    <h1>Performance data for "${client}"</h1>  
    <h3>  
        <a class="btn btn-success btn-lg" href="/">Client list</a>  
    </h3>  
    <h4 id="date_range"></h4>
```

- `${client}` - подстановка значения из модели

КОНТРОЛЛЕР - 2

```
@Controller
public class HelloController {
    @RequestMapping(value = "/hello", method = RequestMethod.GET)
    public String printHello(ModelMap model) {
        model.addAttribute("message", "Hello Spring MVC Framework!");
        return "hello";
    }
}
```

Конец