Введение в Spring

Spring

- **Spring Framework** (или коротко **Spring**) — универсальный фреймворк с открытым исходным кодом для Java EE платформы.

Spring

Spring Framework обеспечивает решения многих задач, с которыми сталкиваются Java разработчики и организации, которые хотят создать информационную систему, основанную на платформе Java.

Spring

Spring Framework

не полностью связан с платформой Java EE/Jakarta, но имеет масштабную интеграцию с ней, что является важной причиной его популярности.

Spring история

Первая версия была написана Родом Джонсоном, который впервые опубликовал её вместе с изданием своей книги «*Expert One-on-One Java EE Design and Development*» (Wrox Press, октябрь 2002 года).

Spring история версий

Spring был впервые выпущен под лицензией Apache 2.0 license в июне 2003 года.

- Первый стабильный релиз 1.0 был выпущен в марте 2004.
- Spring 2.0 был выпущен в октябре 2006

Spring история версий

Spring 2.5 в ноябре 2007,
Spring 3.0 в декабре 2009,
Spring 4.0 в декабре 2013,
Spring 5.0 в сентябре 2017,
Spring 6.0 в ноябре 2022

Текущая версия — 6.0.8

Spring

https://youtu.be/BmBr5diz8WA

https://youtu.be/cou_qomYLNU

https://youtu.be/rd6wxPzXQvo

https://youtu.be/61duchvKI6o

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Spring

https://youtu.be/BmBr5diz8WA

https://youtu.be/cou_qomYLNU

https://youtu.be/rd6wxPzXQvo

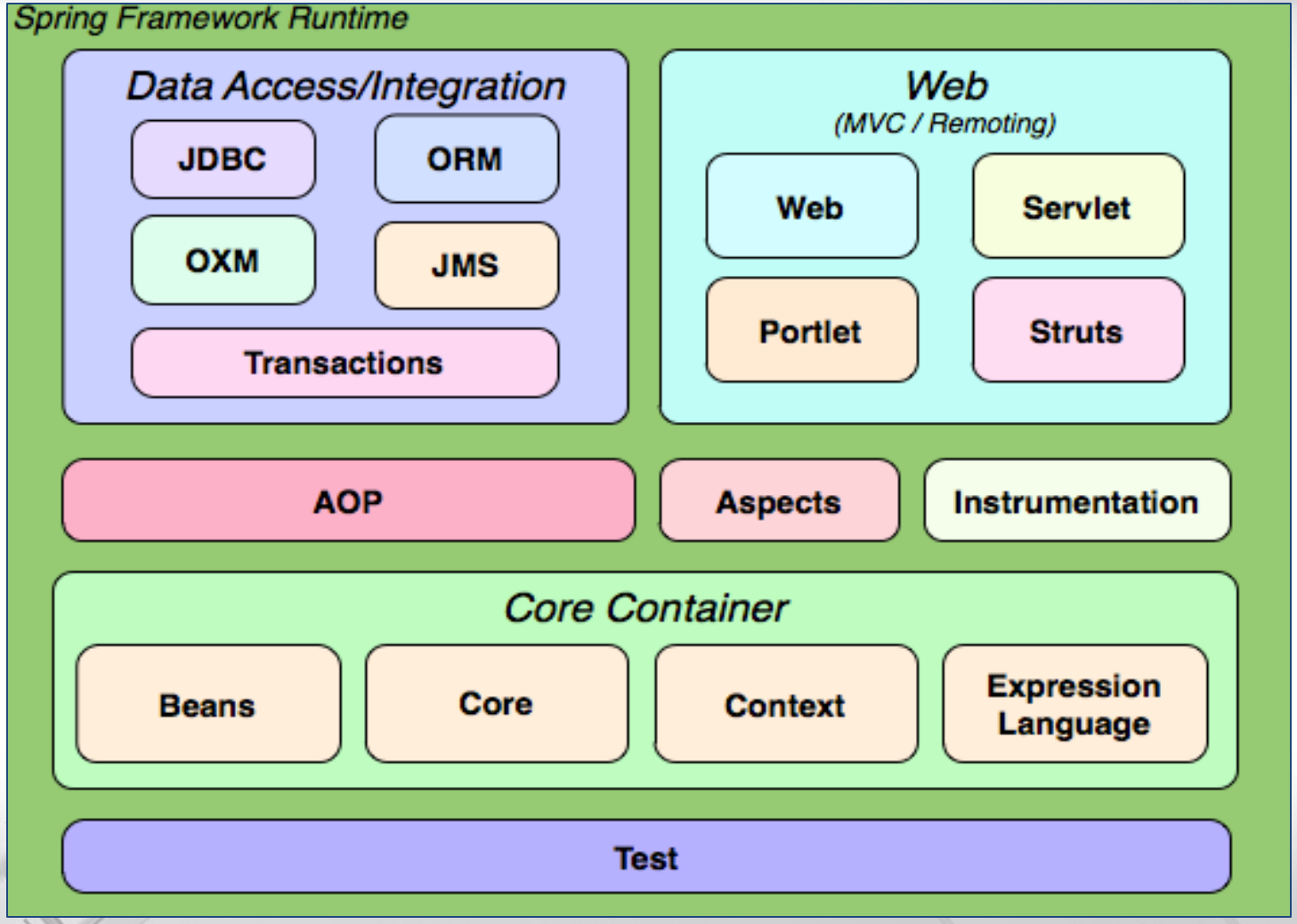
https://youtu.be/61duchvKI6o

Spring

- **Spring** Framework

можно рассматривать как коллекцию меньших фреймворков или фреймворков во фреймворке.

Spring


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
(MVC / Remoting)
Web
Servlet
Portlet
Struts
AOP
Aspects
Instrumentation
Core Container
Beans
Core
Context
Expression Language
Test

Spring

Inversion of Control:

это конфигурирование компонентов приложений и управление жизненным циклом Java-объектов.

Spring

Аспектно-ориентированное программирование (AOP):

работает с функциональностью, которая не может быть реализована возможностями объектно-ориентированного программирования на Java без потерь.

Spring

Доступ к данным (Data Access)

работает с системами управления реляционными базами данных на Java-платформе, используя JDBC- и ORM-средства и обеспечивая решения задач, которые повторяются в большом числе Java-based environments.

Spring

Фреймворк управления транзакциями (Trasactions)

координация различных API управления транзакциями и инструментарий настраиваемого управления транзакциями для объектов Java.

Spring

Фреймворк удалённого доступа

конфигурируемая передача Java-объектов через сеть в стиле RPC, поддерживающая RMI, CORBA, HTTP-based протоколы, включая web-сервисы (SOAP).

Spring

Аутентификация и авторизация

конфигурируемый инструментарий процессов аутентификации и авторизации, поддерживающий много популярных и ставших индустриальными стандартами протоколов, инструментов, практик через дочерний проект Spring Security.

Spring

Удалённое управление

конфигурируемое представление и управление Java-объектами для локальной или удалённой конфигурации с помощью JMX.

Работа с сообщениями

конфигурируемая регистрация объектов-слушателей сообщений для прозрачной обработки сообщений из очереди сообщений с помощью JMS, улучшенная отправка сообщений по стандарту JMS API.

Тестирование

каркас, поддерживающий классы для написания модульных и интеграционных тестов.

Spring

Фреймворк MVC

каркас, основанный на HTTP и сервлетах, предоставляющий множество возможностей для расширения и настройки (customization).

Spring

Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании.

Spring Web

Spring Web Flow (SWF)

субпроект SpringFramework, целью которого является предоставление инфраструктуры для разработки веб-приложений со сложной структурой.

Spring Web

Цели:

- описать правила навигации по страницам
- управлять состояниями навигации
- облегчить повторное использование кода

Inversion of Control (IoC)

Инверсия управления

это принцип объектно-ориентированного программирования, который служит для уменьшения связанности слоёв сложных приложений.

Inversion of Control (IoC)

Принцип заключается в том что каждый слой приложения работает с нижестоящим слоем не на прямую, а опосредованно (через слой абстракций).

Inversion of Control (IoC)

Как следствие, это позволяет легко и безболезненно заменить реализацию каждого из слоёв на новую, при этом не затронув работу вышестоящих слоёв.

IoC vs DI

В 2004 году **Мартин Фаулер** предложил отказаться от термина
Inversion of Control (**IoC**)
в пользу **Dependency Injection** (**DI**),
мотивируя это тем, что термин **IoC** слишком общий и вызывает проблемы в понимании концепции происходящего.

Dependency Injection (DI)

это шаблон проектирования позволяющий разработчикам «развязать» (decouple) компоненты их приложения, тем самым ослабив связи между ними.

Dependency Injection (DI)

Constructor Injection

внедрение через конструктор

Setter Injection

внедрение через сеттеры

Method Injection

внедрение через вызов геттеров

Spring – "контекст" (context)

Контекст

это среда, в которой существует объект.

Объект

это разрабатываемая компонента (javaBean), а среда – все остальные компоненты (beans), необходимые для жизненного цикла вашей компоненты.

Spring – "контекст" (context)

Например, контроллер, позволяющий отправить клиенту сообщение по почте, в данном случае является объектом, а сам мэйлер и логгер – являются средой окружения.

Spring – "контекст" (context)

Описание набора компонент (beans) и взаимосвязей между ними и является контекстом приложения (application context).

Для разработчика, использующего Spring Framework - Application Context представляет собой набор xml – файлов и аннотаций, описывающий компоненты (beans) и взаимосвязи между ними.

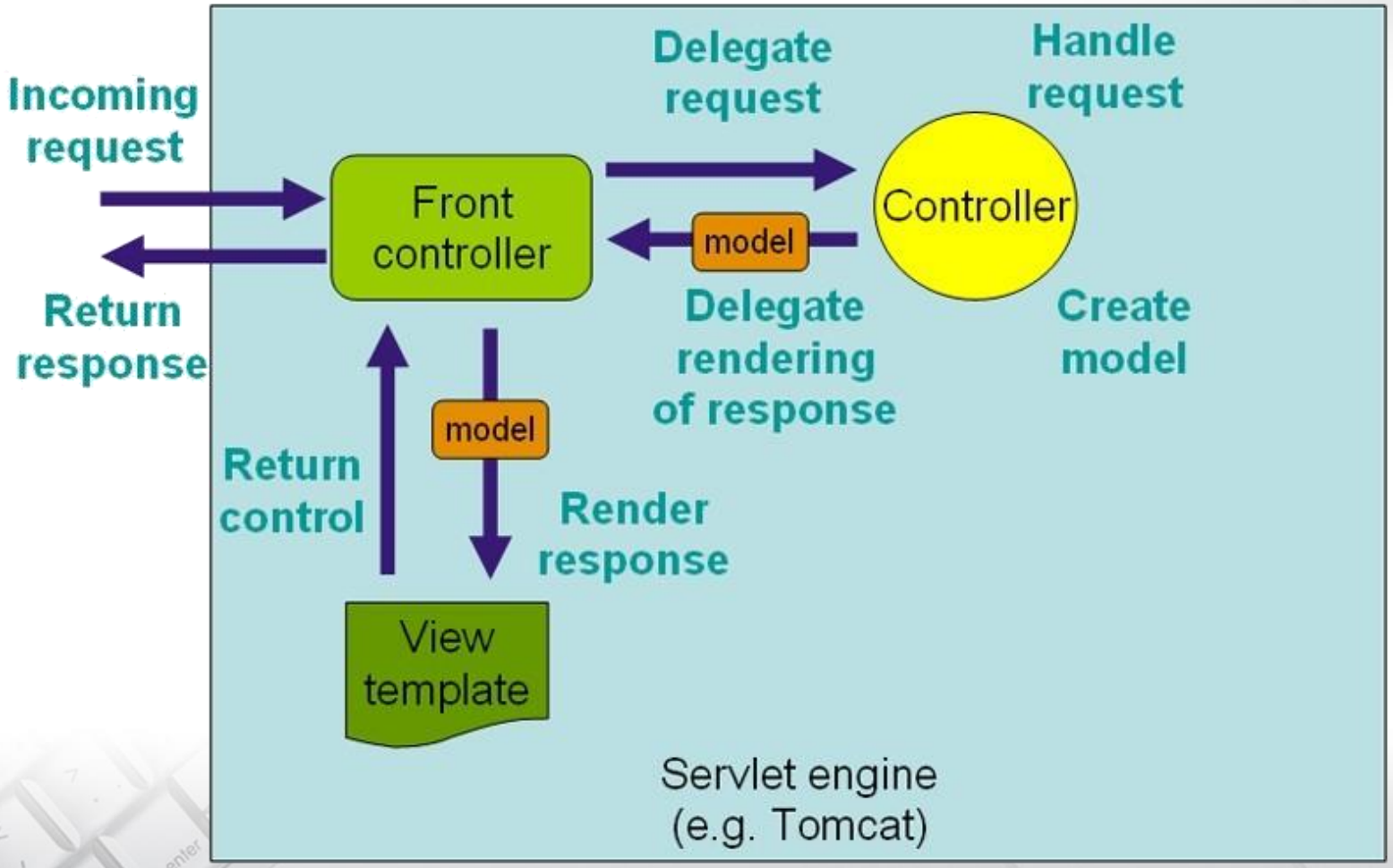
Spring MVC

Spring MVC построен вокруг центрального сервлета **(DispatcherServlet)**, который распределяет запросы по контроллерам, а также предоставляет другие возможности при разработке веб приложений.

DispatcherServlet

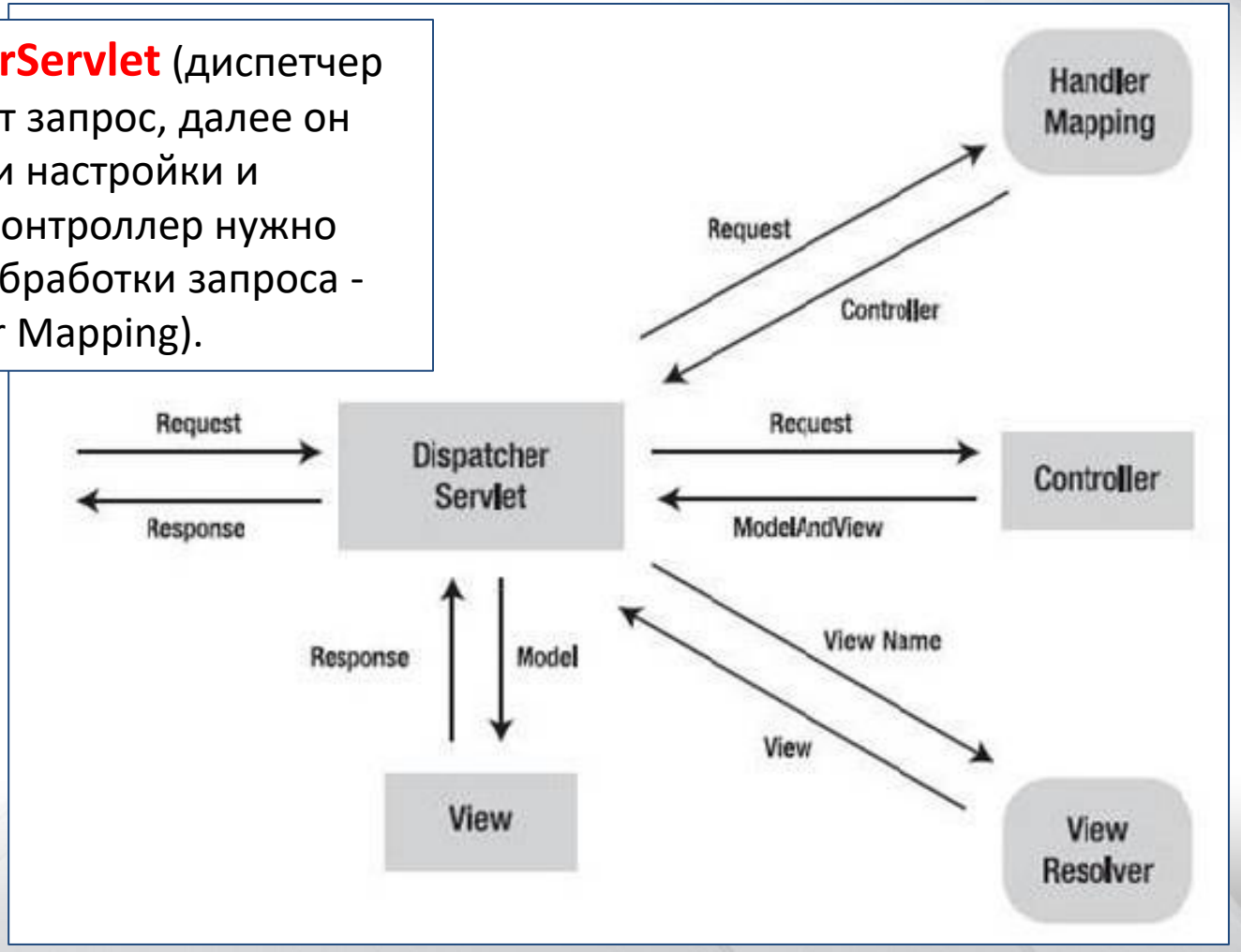
- **DispatcherServlet** — это обычный сервлет (наследуется от базового класса HttpServlet).
- Этот сервлет необходимо описывать в дескрипторе развертывания web.xml вашего веб приложения.

Spring MVC


Incoming request
Delegate request
Handle request
Front controller
Controller
model
Return response
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)

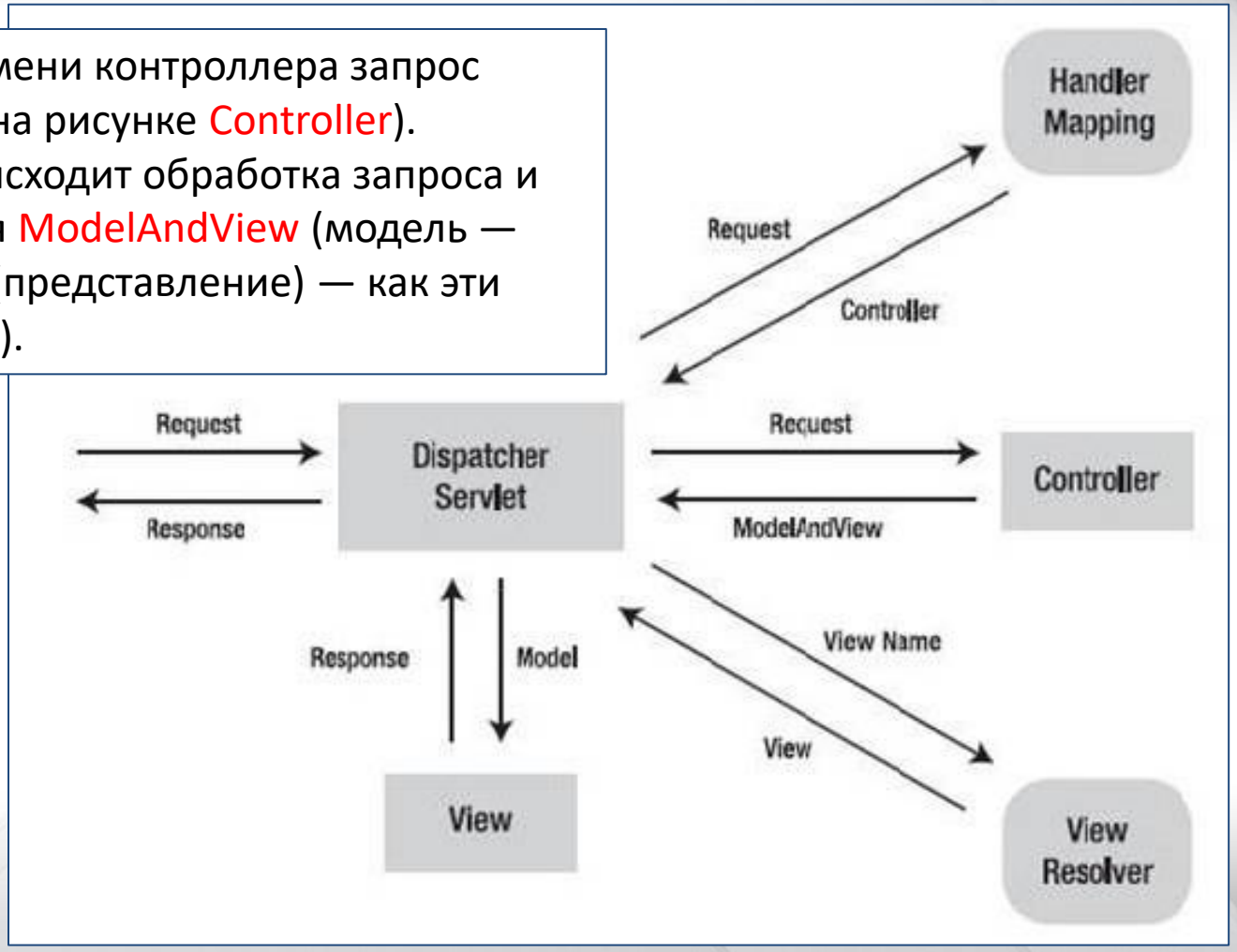
Spring MVC

Сначала **DispatcherServlet** (диспетчер сервлетов) получает запрос, далее он просматривает свои настройки и определяет какой контроллер нужно использовать для обработки запроса - (на рисунке Handler Mapping).



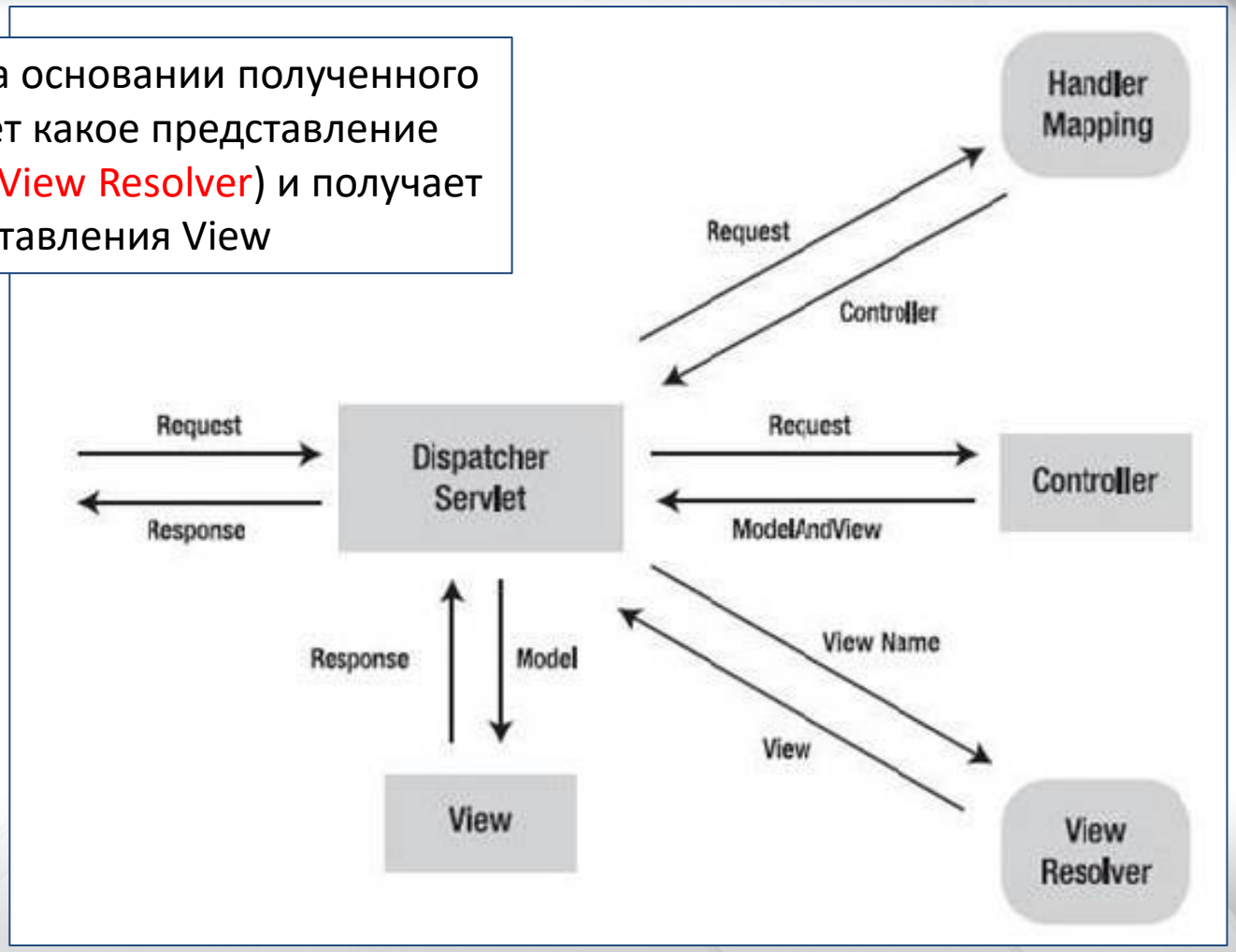
Spring MVC

После получения имени контроллера запрос передается в него (на рисунке Controller).
В контроллере происходит обработка запроса и обратно посылается ModelAndView (модель — сами данные; view (представление) — как эти данные отображать).



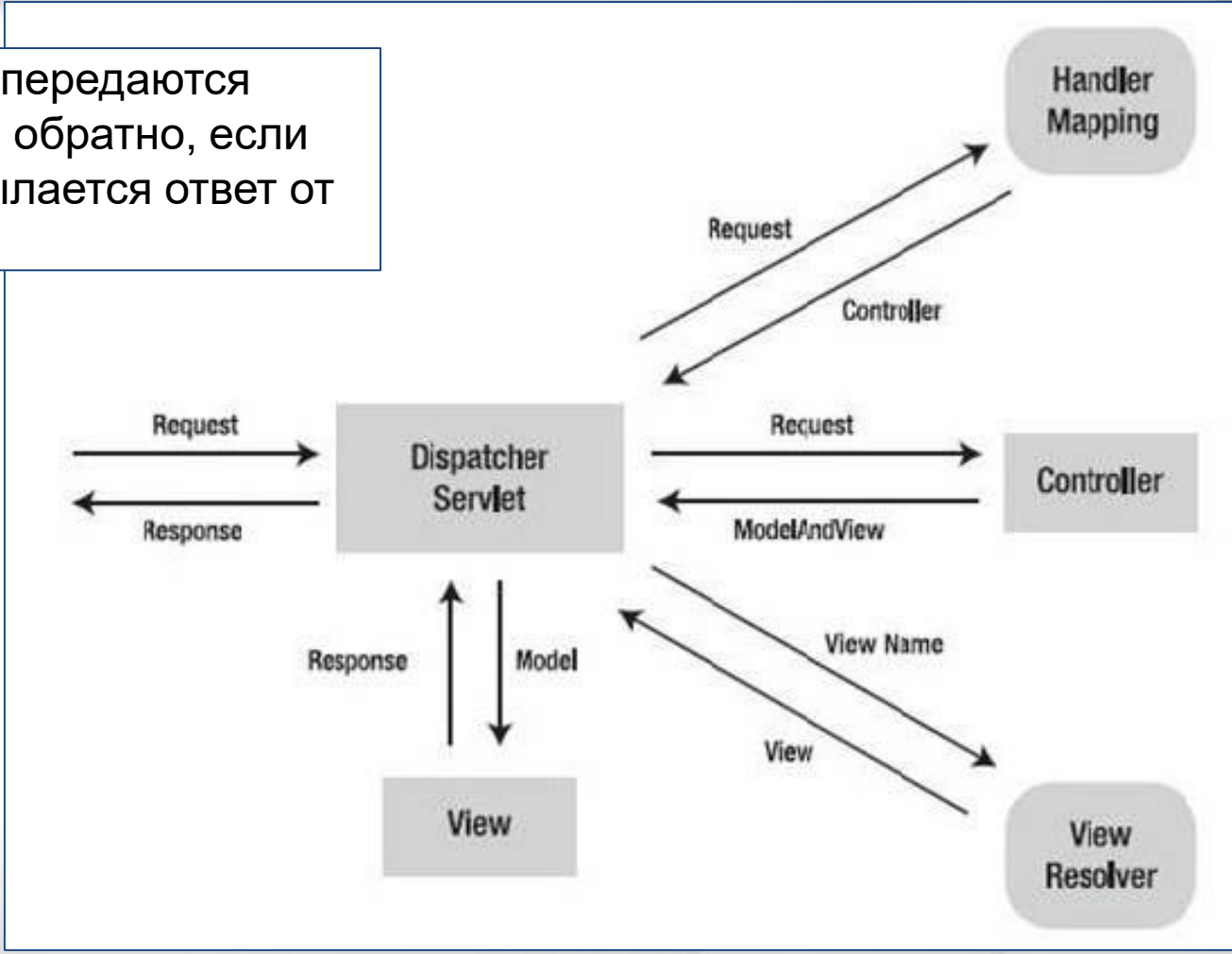
Spring MVC

DispatcherServlet на основании полученного ModelAndView ищет какое представление ему использовать (View Resolver) и получает в ответе имя представления View



Spring MVC

В представление передаются данные (model) и обратно, если необходимо, посылается ответ от представления.



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Основы шаблона Spring MVC

- **Model**

представление данных, сами данные

- **View**

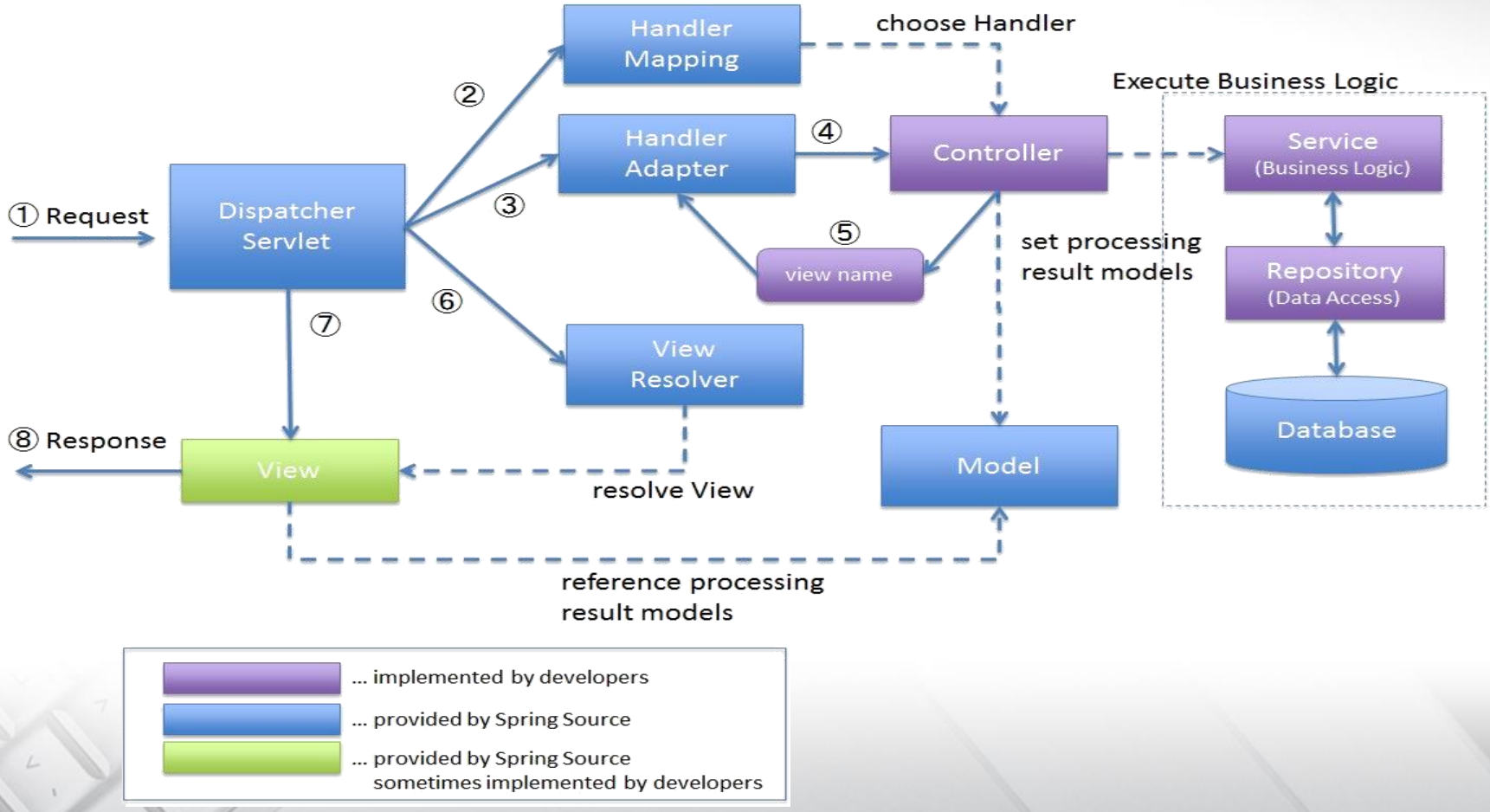
представление, вид, отображение

- **Controller**

управление, связь между моделью и видом

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Жизненный цикл запроса



Controller

Аннотация @Controller служит для сообщения Spring'у о том, что данный класс является bean'ом и его необходимо подгрузить при старте приложения.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Controller

```
@RequestMapping(value="/simple1")
```

служит для указания, что данный контроллер будет обрабатывать запрос, URI которого `"/simple1"`

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Controller

```
@Controller
public class SimpleController {
  @RequestMapping(value = "/simple1")
  public String simple1() {
    return "simple";
   }
}
```

simple — сообщение фронт контролеру

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Controller

```
@Controller
public class SimpleController {
    @RequestMapping(value = "/simple2")
    public ModelAndView simple2() {
      ModelAndView mav = new ModelAndView();
      mav.setViewName("simple");
      return mav;
    }
}
```

ModelAndView - агрегирует все параметры модели и имя отображения

Controller

Аннотация @RequestMapping в классе

Все методы будут получать запросы с URI, который будет начинатся строкой, указанной в аннотации @RequestMapping перед объявлением класса контроллера.

Controller

@PathVariable - автозаполнение переменной из пути (url) запроса

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Controller

```
@RequestMapping(value = "/test",
method = RequestMethod.GET)

@GetMapping("/test")


@RequestMapping(value = "/test",
method = RequestMethod.POST)

@PostMapping("/test")
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

WebRequest

```
@RequestMapping(value = "/webrequest")
public String webRequest(
        WebRequest webRequest,
        Model model) {

   model.addAttribute(
        "content",
        "Session id  (WebRequest): " +
        webRequest.getSessionId() );
   return "test";
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

@RequestParam

```
@RequestMapping(value = "/requestparam")
public String requestParam(
        @RequestParam("foo") int foo,
        Model model) {
    model.addAttribute("content","foo=" + foo);
    return OTHER_VIEW_NAME;
}
```

@RequestParam

```
@RequestMapping(value = "/responsebody")
@ResponseBody
public String responseBody() {
    return "Hello World";
}
```

Этой аннотацией мы отдаем ответ непосредственно браузеру, минуя слой представлений.
То есть, то, что отдаем в методе, то и получит браузер.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

@CookieValue

```
@GetMapping(value = "/cookie")
public String cookie(
    @CookieValue("JSESSIONID") String jsessionid,
    Model model) {
    model.addAttribute("msg",
          "JSESSIONID: " + jsessionid );
   return "info";
}
```

Аннотация @CookieValue позволяет привязать параметр метода контролера к HTTP-cookie

@Autowired

@Autowired

аннотация позволяет автоматически установить значение поля и связывать бины.

@Autowired

- Используя аннотацию **@Autowired**, не нужно заботиться о том, как лучше всего передать классу или bean'у экземпляр другого bean'а.
- Фреймворк Spring сам найдет нужный bean и подставит его значение в свойство, которое отмечено данной аннотацией.

@Autowired

Атрибут **`required=false`** сообщает фреймворку о том, что наличие соответствующего bean'а не является обязательным при компиляции программы

@Qualifier

@Qualifier указывает конкретного кандидата для автозаполнения если есть несколько кандидатов

@Qualifier

@Qualifier указывае конкретного кандидата для автозаполнения если кандидатов несколько.

```
@Autowired
@Qualifier("fooService2")  private
FooService fooService;
```

@Component

@Component

аннотация для любого компонента фреймворка.

@Service

@Service - (Сервис-слой приложения) аннотация объявляющая, что этот класс представляет собой сервис – компонент сервис-слоя.

Сервис является подтипом класса @Component. Использование данной аннотации позволит искать бины-сервисы автоматически.

@Repository

@Repository - (Доменный слой) Аннотация показывает, что класс функционирует как репозиторий и требует наличия прозрачной трансляции исключений.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

@RestController

@RestController
Аннотация объединяет поведение двух аннотаций
@Controller и @ResponseBody

@Transactional

@Transactional

Перед исполнением метода помеченного данной аннотацией начинается транзакция, после выполнения метода транзакция коммитится, при выбрасывании любого RuntimeException откатывается.

@Scope

@Scope — служит для указания области видимости бина.

Пример:

```
@Service
@Scope("prototype")  public
class UserService
```

Области видимости бинов

Singleton

Возвращает один и тот же экземпляр бина на каждый запрос контейнера Spring IoC (по умолчанию).

Prototype

Создает и возвращает новый экземпляр бина на каждый запрос.

Области видимости бинов

Request

создает и возвращает экземпляр бина на каждый HTTP запрос.

Session

создает и возвращает экземпляр бина для каждой HTTP сессии.

Области видимости бинов

global-session

Создает и возвращает экземпляр бина для глобальной HTTP сессии.

application

Жизненный цикл экземпляра ограничен в пределах ServletContext. Действует, только если вы используете web-aware ApplicationContext

Жизненный цикл бина

- Жизненный цикл Spring бина — время существование класса.
- Spring бины инициализируются при инициализации Spring контейнера.
- Когда контейнер уничтожается, уничтожаются и все бины.