**Spring Framework. MVC и Security.**

Introduction to Spring Framework <https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html>

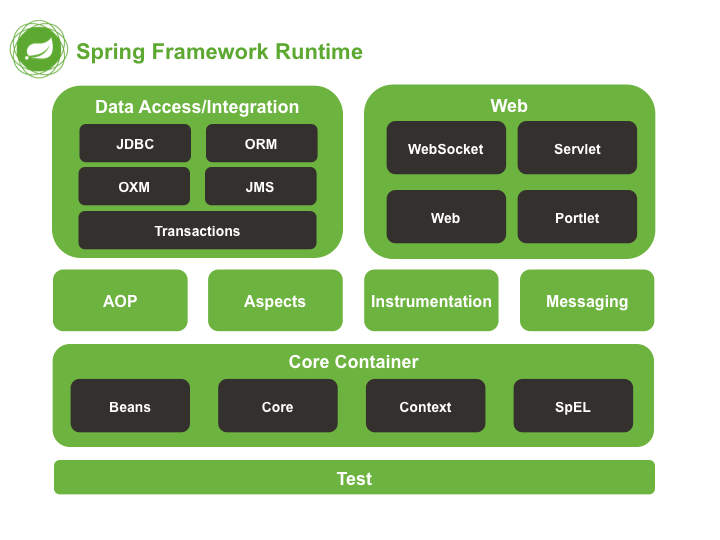
Как писать на Spring в 2017 <https://habr.com/ru/post/333756/>

Spring MVC — основные принципы <https://habr.com/ru/post/336816/>

Spring Security <https://spring.io/projects/spring-security>

Краткий обзор Spring Security <https://habr.com/ru/post/203318/>

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения.



*Основные компоненты Spring Framework*

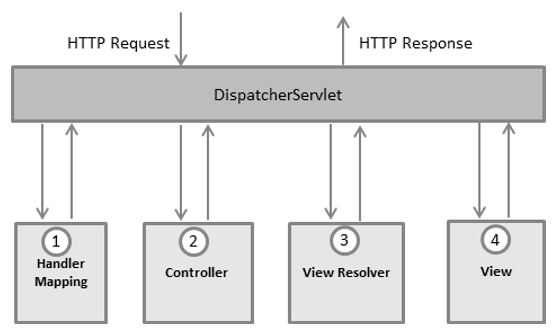
В настоящее время, под термином "Spring" часто подразумевают целое семейство проектов. В большинстве своем, они развиваются и курируются компанией Pivotal и силами сообщества. Ключевые проекты семейства Spring это:

* IoC (Inversion of Control) контейнер;
* AOP-фреймворк (включая интеграцию с AspectJ);
* Data Access фреймворк;
* Transaction management;
* MVC-фреймворк;
* Remote Access фреймворк;
* Batch processing;
* Фреймворк аутентификации и авторизации;
* Remote Management;
* Messaging-фреймворк;
* Testing-фреймворк.
* **Spring MVC**

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.

* *Model (Модель)* инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* *View (Отображение, Вид)* отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* *Controller (Контроллер)* обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Вся логика работы Spring MVC построена вокруг DispatcherServlet, который принимает и обрабатывает все HTTP-запросы (из UI) и ответы на них. Рабочий процесс обработки запроса DispatcherServlet'ом проиллюстрирован на следующей диаграмме:



Последовательность событий, соответствующая входящему HTTP-запросу:

* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

* **Spring Security**

Spring Security это Java/JavaEE framework, предоставляющий механизмы построения систем аутентификации и авторизации, а также другие возможности обеспечения безопасности для корпоративных приложений, созданных с помощью Spring Framework. Проект был начат Беном Алексом (Ben Alex) в конце 2003 года под именем «Acegi Security», первый релиз вышел в 2004 году. Впоследствии проект был поглощён Spring'ом и стал его официальным дочерним проектом. Впервые публично представлен под новым именем Spring Security 2.0.0 в апреле 2008 года.

Ключевые объекты контекста Spring Security:

* SecurityContextHolder, в нем содержится информация о текущем контексте безопасности приложения, который включает в себя подробную информацию о пользователе(Principal) работающем в настоящее время с приложением. По умолчанию SecurityContextHolder используетThreadLocal для хранения такой информации, что означает, что контекст безопасности всегда доступен для методов исполняющихся в том же самом потоке. Для того что бы изменить стратегию хранения этой информации можно воспользоваться статическим методом класса SecurityContextHolder.setStrategyName(String strategy). Более подробно SecurityContextHolder
* SecurityContext, содержит объект Authentication и в случае необходимости информацию системы безопасности, связанную с запросом от пользователя.
* Authentication представляет пользователя (Principal) с точки зрения Spring Security.
* GrantedAuthority отражает разрешения выданные пользователю в масштабе всего приложения, такие разрешения (как правило называются «роли»), например ROLE\_ANONYMOUS, ROLE\_USER, ROLE\_ADMIN.
* UserDetails предоставляет необходимую информацию для построения объекта Authentication из DAO объектов приложения или других источников данных системы безопасности. Объект UserDetailsсодержит имя пользователя, пароль, флаги: isAccountNonExpired, isAccountNonLocked, isCredentialsNonExpired, isEnabled и Collection — прав (ролей) пользователя.
* UserDetailsService, используется чтобы создать UserDetails объект путем реализации единственного метода этого интерфейса

Процесс аутентификации :

1. Пользователю будет предложено войти в систему предоставив имя (логин или email) и пароль. Имя пользователя и пароль объединяются в экземпляр класса UsernamePasswordAuthenticationToken(экземпляр интерфейса Authentication) после чего он передается экземпляру AuthenticationManager для проверки.
2. В случае если пароль не соответствует имени пользователя будет выброшено исключение BadCredentialsException с сообщением “Bad Credentials”.
3. Если аутентификация прошла успешно возвращает полностью заполненный экземпляр Authentication.
4. Для пользователя устанавливается контекст безопасности путем вызова метода SecurityContextHolder.getContext().setAuthentication(…), куда передается объект который вернул AuthenticationManager.