Раздел 2 (Применение фреймворка SpringBoot для создания веб-приложений)

Что такое Spring Framework?
Spring Framework, или просто **Spring** — один из самых популярных фреймворков для создания веб-приложений на Java. **Фреймворк** — это что-то похожее на библиотеку (возможно этот термин вам более знаком), но есть один момент. Грубо говоря, используя библиотеку, вы просто создаете объекты классов, которые в ней есть, вызываете нужные вам методы, и таким образом получаете нужный вам результат. То есть, тут более императивный подход: вы четко указываете в своей программе в какой конкретный момент надо создать какой объект, в какой момент вызвать конкретный метод, итд. С фреймворками дела обстоят слегка иначе. Вы просто пишете какие-то свои классы, прописываете там какую-то часть логики, а создает объекты ваших классов и вызывает методы за вас уже сам фреймворк. Чаще всего, ваши классы имплементируют какие-то интерфейсы из фреймворка или наследуют какие-то классы из него, таким образом получая часть уже написанной за вас функциональности. Но не обязательно именно так. В спринге например стараются по максимуму отойти от такой жесткой связности (когда ваши классы напрямую зависят от каких-то классов/интерфейсов из этого фреймворка), и используют для этой цели аннотации. Дальше мы еще к этому моменту вернемся. Но важно понять, что ***спринг*** — это просто набор каких-то классов и интерфейсов, которые уже написаны за вас :) Еще хочу сразу отметить, что спринг можно использовать не только для веб-приложений, но и для так знакомых всем нам самых обычных консольных программок. И сегодня мы что-то такое даже напишем.

Структура
Но спринг — это не один какой-то конкретный фреймворк. Это скорее общее названия для целого ряда небольших фреймворков, каждый из которых выполняет какую-то свою работу.

Как видно, у спринга модульная структура. Это позволяет подключать только те модули, что нам нужны для нашего приложения и не подключать те, которыми мы заведомо не будем пользоваться. Насколько мне известно, то именно этот подход и помог спрингу обойти своего конкурента в то время (EJB) и захватить лидерство. Потому что приложения, использующие ***EJB*** тянули очень много зависимостей за

собой, да и вообще получались медленные и неповоротливые. ***На изображении видно, что спринг фреймворк состоит как-бы из нескольких модулей:***

- data access;
- web;
- core;
- и других.

Сегодня мы познакомимся с некоторыми концепциями основного модуля, такими как: бины, контекст и другими. Как можно было догадаться, модуль *data access* содержит в себе средства для работы с данными (в основном, с базами данных), *web* — для работы в сети (в том числе и для создания веб-приложений, о которых будет позже). Кроме того, есть еще так-называемая целая спринг-инфраструктура: множество других проектов, которые не входят в сам фреймворк официально, но при этом бесшовно интегрируются в ваш проект на спринге (например, тот же *spring security* для работы с авторизацией пользователей на сайте, который, я надеюсь, мы тоже как-нибудь пощупаем).

Почему Spring в Java?

Ну кроме того, что это модно-стильно-молодежно, могу сразу сказать, что как только вы им хоть немного овладеете — вы поймете сколько всякой разной работы вам теперь не приходится делать, и сколько всего берет на себя Spring. Можно написать пару десятков строк конфигов, написать парочку классов — и получится работающий проект. Но как только начинаешь задумываться сколько там всего находится "под капотом", сколько работы выполняется, и сколько пришлось бы писать кода, если делать такой же проект на голых сервлетах или на сокетах и чистой Java — волосы встают дыбом :) Есть даже такое выражение, как "магия" Spring. Это когда ты видишь, что все работает, но ты примерно прикидываешь сколько там всего должно происходить чтобы все работало и как оно там все работает — то кажется, что происходит это все благодаря действительно какой-то магии)) Проще назвать это все магией, чем попытаться объяснить как оно там все взаимосвязано. :) Ну и второй аргумент "за" изучение Spring — это то, что в примерно 90% вакансий на джуна (по моим личным наблюдениям) — требуется либо знание, либо хотя бы общее представление о джентельменском наборе спринга из data, web-mvc и security :) Но сегодня только об основах.

DI/IoC

Если вы пытались что-то читать по спрингу, то первое с чем вы сталкивались — это наверное вот эти вот буковки: *DI/IoC*. Сейчас я вам

очень рекомендую отвлечься от этой статьи и почитать вот эту статью на хабре! ***IoC (Inversion of Control)*** — инверсия управления. Об этом я уже вскользь упоминал, когда писал, что при использовании библиотеки вы сами прописываете в своем коде какой метод какого объекта вызвать, а в случает с фреймворками — чаще всего уже фреймворк будет вызывать в нужный ему момент тот код, который вы написали. То есть, тут уже не вы управляете процессом выполнения кода/программы, а фреймворк это делает за вас. Вы передали ему управление (инверсия управления). Под ***DI*** понимают то ***Dependency Inversion*** (инверсию зависимостей, то есть попытки не делать жестких связей между вашими модулями/классами, где один класс напрямую завязан на другой), то ***Dependency Injection*** (внедрение зависимостей, это когда объекты котиков создаете не вы в main-е и потом передаете их в свои методы, а за вас их создает спринг, а вы ему просто говорите что-то типа "хочу сюда получить котика" и он вам его передает в ваш метод). Мы чаще будем сталкиваться в дальнейших статьях со вторым.

Бины и контекст

Одно из ключевых понятий в спринге — это бин. По сути, это просто объект какого-то класса. Допустим, для нашей программы надо использовать 3 объекта: котика, собачку и попугайчика. И у нас есть куча классов с кучей методов, где иногда нам нужен для метода котик, а для другого метода — собачка, а иногда у нас будут методы, где нужен котик и попугайчик (например метод для кормежки котика, хе-хе), а в каких-то методах — все три объекта понадобятся. Да, мы можем в main-е сначала создать эти три объекта, а потом их передавать в наши классы, а уже изнутри классов — в нужные нам методы... И так по всей программе. А если еще и представить, что периодически мы захотим менять список принимаемых параметров для наших методов (ну решили переписать что-то или добавить функциональности) — то нам придется делать довольно много правок по коду если надо будет что-то поменять. А теперь если представить, что таких объектов у нас не 3, а 300? Как вариант, это собрать все наши такие объекты в какой-то один общий список объектов (*List<Object>*) и во все методы передавать его, а изнутри методов уже доставать тот или иной объект, который нам нужен. Но что если представить, что по ходу программы у нас в этот список может добавиться какой-то объект, или (что хуже) удалиться? Тогда во всех методах, где мы достаем объекты из списка по их индексу — все может поломаться. Тогда мы решаем хранить не список, а мапу, где ключом будет имя нужного нам объекта, а значением — сам объект, и тогда мы сможем из него доставать нужные нам объекты просто по их имени: *get("попугайчик")* и в ответ получили объект попугайчика. Или например ключ — это класс объекта, а

значение — сам объект, тогда мы сможем указать уже не имя объекта, а просто класс нужного нам объекта, тоже удобно. Или даже написать какую-то обертку над мапой, где сделать методы, чтобы в каких-то случаях доставать объекты по их имени, а в других случаях — по классу. Вот это и получится у нас ***application context*** из спринга. Контекст — это набор бинов (объектов). Обращаясь к контексту — мы можем получить нужный нам бин (объект) по его имени например, или по его типу, или еще как-то. Кроме того, мы можем попросить спринг самого сходить поискать в своем контексте нужный нам бин и передать его в наш метод. Например, если у нас был такой метод: **public void** doSomething(Cat cat) {
    ...
} нам спринг когда вызывал этот метод — передавал в него объект нашего котика из своего контекста. Теперь мы решаем, что нашему методу кроме котика нужен еще и попугайчик. Используя спринг — для нас нет ничего проще! Мы просто пишем: **public void** doSomething(Cat cat, Parrot parrot) {
    ...
} и спринг, когда будет вызывать этот наш метод — сам поймет, что сюда надо передать котика и попугайчика, сходит к себе в контекст, достанет эти два объекта и передаст их в наш метод. Передав спрингу бразды правления нашей программой — мы так же переложили на него ответственность за создание объектов и передачу их в наши методы, которые он будет вызывать. Возникает вопрос: ***а как спринг будет знать какие объекты (бины) создавать?***

Способы конфигурации приложения
Существует ***три основных способа конфигурации приложения*** (то-есть, указания спрингу какие именно объекты нам нужны для работы):

1. при помощи xml файлов/конфигов;
2. при помощи java-конфигов;
3. автоматическая конфигурация.

Разработчики спринга выстраивают их в таком порядке приоритетности:

- наиболее приоритетный способ, которому стоит отдавать предпочтение — это автоматическая конфигурация;
- если при помощи автоматической конфигурации нет возможности правильно настроить все возможные бины — использовать джава-конфигурацию (создание объектов используя джава код);
- ну и самый низкоприоритетный способ — это по-старинке, используя xml конфиги.

Кроме того, спринг позволяет комбинировать эти способы. Например, все то, что может быть настроено автоматически — пусть спринг сделает сам,

там где надо указать какие-то особые параметры — сделать при помощи джава-конфигов, и кроме того, можно подключить какие-то легаси конфиги в xml формате. В общем, достаточно гибко это все можно сделать. Но все же, если все можно сделать при помощи автоматической настройки — используйте ее. Я буду рассматривать только автоматическую настройку и джава-конфиги; xml конфиги и так почти в каждом примере по спрингу в интернете используются, да и поняв как работает джава-конфигурация — не должно возникнуть проблем с тем, чтобы "прочитать" xml файл, который делает то же. Автоматическая конфигурация используется тогда, когда нужные нам для работы объекты — это объекты написанных нами классов. Если для создания объекта нашего класса нужна какая-то очень специфическая логика, или если у нас нет возможность отметить какой-то класс нужной нам аннотацией, которую подхватила бы автоматическая конфигурация — это можно сделать в джава-конфигах. В следующей части мы создадим мавен-проект, подключим в него парочку центральных модулей спринга и создадим наши первые бины.