サンプルプログラムについて

本書のサンプルプログラムは、サポートページからダウンロードできます。

http://www.kohgakusha.co.jp/support.html

また、最新のプログラムは、以下からダウンロードできます。

https://github.com/making/hajiboot-samples

■ 本書の対象読者

本書は「Java」の経験者を対象にしています。「Spring Framework」の利用経験がなくても大丈夫です。

「Java」で「Web アプリケーション」を作ったことがあるけれど、「簡単で新しい」フレームワークを、「知りたい」「試したい」、という人にぴったりな内容になっています※。

本書では「ビルド・ツール」として「Maven」を使います。「Maven」の知識があると、より深く理解できますが、「Maven」を使ったことがなくても読み進めることができるようになっています。

■ 本書の読み進め方

本書では「Spring Framework」の簡単な説明から始まり、「Spring Boot」に関する説明を少しずつ加えていきます。内容が連続的になっているため、最初から順番に読み進めていくことをお勧めします。

すでに「Spring Framework」の利用経験がある場合は、「第２章」は不要だと思うかもしれません。

しかし、基本的な解説の中にも「Spring Boot」に関するエッセンスも含めているので、省略せずに読んでいただければ、と思います。

※ ただし、「Spring Boot」を中心とした内容であるため、本書では「Spring Framework」に関する詳しい説明はしていません。

第1章 「Spring Boot」とは

「Spring Boot」とは「Spring Framework」という Java のフレームワークで、アプリケーションを簡単に作るための仕組みです。

本章では「Spring Boot」が生まれた背景を説明し、簡単なアプリケーションを作ることで「Spring Boot」に対するイメージを深めます。

1.1 「Spring Framework」の歴史

「Spring Framework」は、2003 年に登場した、歴史のある Java フレームワークです。

当時、重厚だった「J2EE」のアンチテーゼとして発表され、「DI」(Dependency Injection) と「AOP」(Aspect Oriented Programming) で注目されました。

登場から10 年以上経った現在も活発に開発が行なわれており、現在のバージョンは「4.3.2.RELEASE」です (2016 年 8 月時点)。「http://spring.io」で公開されています。

http://spring.io

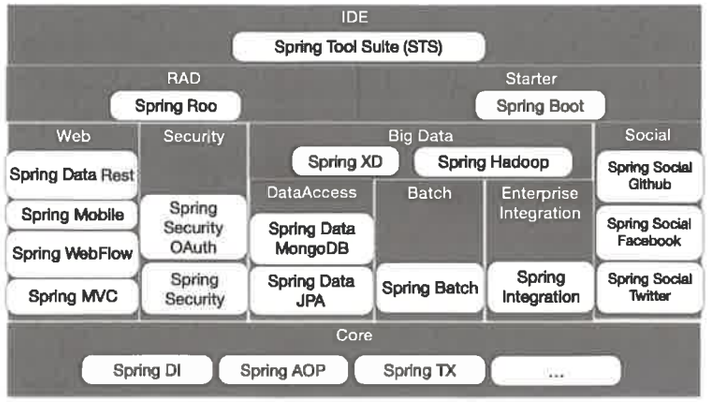
「Spring Framework」は「DI」「AOP」だけでなく、

- Spring MVC (Web アプリケーション向け MVC フレームワーク)
- Spring Batch (バッチ処理用フレームワーク)
- Spring Security (認証・認可フレームワーク)

・Spring Integration（システム連携フレームワーク）
・Spring Data（データアクセス抽象化フレームワーク）

など、さまざまなプロダクトが用意されています[1]。

これらをカテゴライズすると、次の図のようになります。



「Spring Framework」の構造

広い範囲をカバーしている反面、それぞれのプロダクトのことを知らない人にとっては、多すぎて取っ付きにくいと思えるかもしれません。

1.2 「Spring Boot」とは

「Spring Framework」は時代とともにサブプロジェクトがどんどん増え肥大化し、どれを使えばいいのか分かりづらいと言われてきました。実際に、各プロダクトを組み合わせて使うには、多くの初期設定が必要でした。組み合わせ方法にもノウハウを要してきました。

そこで現われたのが、「Spring Boot」です。
「Spring Boot」は、

・あらかじめオススメのプロジェクトの組み合わせが含まれている
・自動で設定が有効になる
・組み込みサーバが同梱される

という特徴をもち、アプリのコードを少量書いてすぐに実行（起動）可能にします。
そのため、いま非常に注目を集めています[2]。

同様な思想をもつフレームワークとしては、「Dropwizard」[3]が有名でした。

[1] http://spring.io/projects
[2] http://assets.thoughtworks.com/assets/technology-apr-2016-en.pdf
[3] https://dropwizard.github.io/dropwizard/

Dropwizard

「Spring Boot」は「Dropwizard」の影響を受けて開発されました。

「Spring Boot」も「Dropwizard」も、これまでのJavaのWebアプリケーションのように、『アプリケーションを「war」にパッケージングして、常駐のアプリケーション・サーバにデプロイする』というモデルではなく、『ビルドしたものが即実行可能になる』というモデルを採っています。

「Dropwizard」も良い選択肢ですが、「Dropwizard」に比べて、「Spring Boot」は、

・実績の多い「Spring Framework」のノウハウが活かせる
・構成要素である「Spring Framework」の開発元が開発している
・たくさんのチュートリアル[4]とサンプル[5]が用意されている

といった点がメリットです。
また、「Spring Boot」はWebアプリケーションに特化しているわけではありません。コマンドライン・アプリケーションやバッチ・アプリケーションにも使えます。

特に、チュートリアルやサンプルといった学習コンテンツが豊富なことは大きな魅力です。本書を読み終えた後も、これらのコンテンツを参照することで、より深い知識を得ることができるでしょう。

なお、本書で扱う「Spring Boot」のバージョンは執筆時点での最新版である「1.4.0.RELEASE」です。

「Spring Boot」は開発サイクルが短く、かつ下位互換性のない変更も含まれるので、別のバージョンを利用する場合は気を付けてください。
「Spring Boot」のバージョンアップ情報は、「https://github.com/making/hajiboot-samples」にて公開するので、バージョンアップの際はご確認ください。

[4] http://spring.io/guides
[5] https://github.com/spring-projects/spring-boot/tree/master/spring-boot-samples

1.3 はじめての「Spring Boot」

まずは最も簡単な「Spring Boot」アプリケーションを作ってみましょう。

「Java SE 8」のインストールが終わっていない場合は、**附録**の「**[附録 A]「Java SE 8」のインストール**」を参照して、インストールしてください。

[1.3.1] 「Spring Initializr」による雛形プロジェクト

「Spring Boot」には、「Spring Initializr」という雛形(ひながた)生成 Web サービスが用意されています。
これを利用して、はじめての「Spring Boot アプリケーション」を作りましょう。

ブラウザで、「http://start.spring.io」にアクセスしてください。
画面左の「Artifact」と書かれた入力項目に「hajiboot」と入力してください。

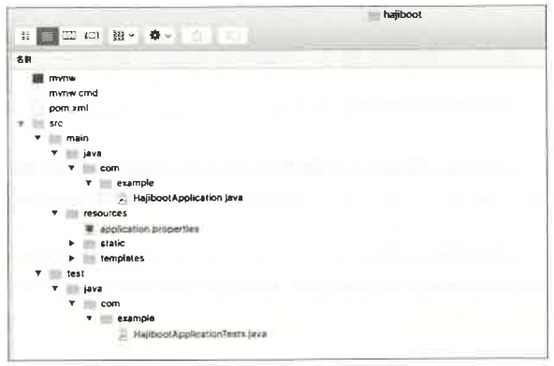
次に、入力項目の画面右の「Search for dependencies」と書かれた入力項目に「Web」と入力して、エンターキーを押下してください。
これは「Spring Boot」における「Web」の機能を使うことを示します。

Spring Initializr

*

その他の入力項目はそのままにして、「Generate Project」ボタンをクリックしてください。「hajiboot.zip」がダウンロードされます。

「hajiboot.zip」を展開すると、以下のようなフォルダ構成になっていることが分かります。これは「Maven」のプロジェクトです。



プロジェクト構成

以降では、ここで生成された「hajiboot」フォルダで、コマンドをいくつか実行します。

作業ディレクトリを「hajiboot」に移動してください。

【ターミナル】作業ディレクトリの移動

```
$ cd hajiboot
```

[1.3.2] 「pom.xml」の設定

「Maven」の「依存関係定義ファイル」である「pom.xml」を確認してください。
「Maven」に詳しくない方は、中身が分からなくても気にしないでください。

「Maven」の「依存関係定義ファイル」(pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example</groupId>
  <artifactId>hajiboot</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>hajiboot</name>
  <description>Demo project for Spring Boot</description>

    <!-- (1) -->
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.4.0.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <!-- (2) -->
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <!-- (3) -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <!-- (4) -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</art
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
```

```
    <plugins>
      <!-- (5) -->
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

プログラム解説

項番	説明
(1)	「Spring Boot」の設定情報群を継承。 ここで指定したバージョンが「Spring Boot」のバージョンになる。 「Spring Boot」をバージョンアップする際には、この <version> タグ内の設定値を変更する。
(2)	「Java SE 8」が使われるように設定。
(3)	「Spring Boot」で Web アプリケーションを作るための基本的なライブラリへの依存情報。 この記述だけで、Web アプリケーションを作るために必要な「Spring Framework」のライブラリや「サードパーティ・ライブラリ」が利用できる。 個々のバージョンは (1) で設定した「spring-boot-starter-parent」内に定義されているため、指定する必要はない。
(4)	「Spring Boot」で「ユニット・テスト」をするための基本的なライブラリへの依存情報。
(5)	「Spring Boot」のアプリケーションを、簡単に、実行し、ビルドするための、「Maven プラグイン」。

ノート この設定だけで、どのくらいのライブラリを使えるか、次のコマンドを実行して、確認してみましょう。

【ターミナル】使えるライブラリの確認

```
$ ./mvnw dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
[INFO] Building hajiboot 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.10:tree (default-cli) @ hajiboot ---
[INFO] com.example:hajiboot:jar:0.0.1-SNAPSHOT
[INFO] +- org.springframework.boot:spring-boot-starter-web:jar:1.4.0.RELEASE:compile
[INFO] |  +- org.springframework.boot:spring-boot-starter:jar:1.4.0.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot:jar:1.4.0.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-autoconfigure:
jar:1.4.0.RELEASE:compile
[INFO] |  |  +- org.springframework.boot:spring-boot-starter-loggin
g:jar:1.4.0.RELEASE:compile
[INFO] |  |  |  +- ch.qos.logback:logback-classic:jar:1.1.7:compile
```



```
[INFO] |  |  |  |  \- ch.qos.logback:logback-core:jar:1.1.7:compile
[INFO] |  |  |  +- org.slf4j:jcl-over-slf4j:jar:1.7.21:compile
[INFO] |  |  |  +- org.slf4j:jul-to-slf4j:jar:1.7.21:compile
[INFO] |  |  |  \- org.slf4j:log4j-over-slf4j:jar:1.7.21:compile
[INFO] |  |  \- org.yaml:snakeyaml:jar:1.17:runtime
[INFO] |  +- org.springframework.boot:spring-boot-starter-tomcat:
jar:1.4.0.RELEASE:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-core:jar:8.5.4:compile
[INFO] |  |  +- org.apache.tomcat.embed:tomcat-embed-el:jar:8.5.4:compile
[INFO] |  |  \- org.apache.tomcat.embed:tomcat-embed-websocket:jar:8.5.4:compile
[INFO] |  +- org.hibernate:hibernate-validator:jar:5.2.4.Final:compile
[INFO] |  |  +- javax.validation:validation-api:jar:1.1.0.Final:compile
[INFO] |  |  +- org.jboss.logging:jboss-logging:jar:3.3.0.Final:compile
[INFO] |  |  \- com.fasterxml:classmate:jar:1.3.1:compile
[INFO] |  +- com.fasterxml.jackson.core:jackson-databind:jar:2.8.1:compile
[INFO] |  |  +- com.fasterxml.jackson.core:jackson-annotations:jar:2.8.1:compile
[INFO] |  |  \- com.fasterxml.jackson.core:jackson-core:jar:2.8.1:compile
[INFO] |  +- org.springframework:spring-web:jar:4.3.2.RELEASE:compile
[INFO] |  |  +- org.springframework:spring-aop:jar:4.3.2.RELEASE:compile
[INFO] |  |  +- org.springframework:spring-beans:jar:4.3.2.RELEASE:compile
[INFO] |  |  \- org.springframework:spring-context:jar:4.3.2.RELEASE:compile
[INFO] |  \- org.springframework:spring-webmvc:jar:4.3.2.RELEASE:compile
[INFO] |     \- org.springframework:spring-expression:jar:4.3.2.RELEASE:compile
[INFO] \- org.springframework.boot:spring-boot-starter-test:jar:1.4.0.RELEASE:test
[INFO]    +- org.springframework.boot:spring-boot-test:jar:1.4.0.RELEASE:test
[INFO]    +- org.springframework.boot:spring-boot-test-autoconfigure:
jar:1.4.0.RELEASE:test
[INFO]    +- com.jayway.jsonpath:json-path:jar:2.2.0:test
[INFO]    |  +- net.minidev:json-smart:jar:2.2.1:test
[INFO]    |  |  \- net.minidev:accessors-smart:jar:1.1:test
[INFO]    |  |     \- org.ow2.asm:asm:jar:5.0.3:test
[INFO]    |  \- org.slf4j:slf4j-api:jar:1.7.21:compile
[INFO]    +- junit:junit:jar:4.12:test
[INFO]    +- org.assertj:assertj-core:jar:2.5.0:test
[INFO]    +- org.mockito:mockito-core:jar:1.10.19:test
[INFO]    |  \- org.objenesis:objenesis:jar:2.1:test
[INFO]    +- org.hamcrest:hamcrest-core:jar:1.3:test
[INFO]    +- org.hamcrest:hamcrest-library:jar:1.3:test
[INFO]    +- org.skyscreamer:jsonassert:jar:1.3.0:test
[INFO]    |  \- org.json:json:jar:20140107:test
[INFO]    +- org.springframework:spring-core:jar:4.3.2.RELEASE:compile
[INFO]    \- org.springframework:spring-test:jar:4.3.2.RELEASE:test
[INFO] ------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
[INFO] Total time: 1.711 s
[INFO] Finished at: 2016-08-07T19:44:45+09:00
[INFO] Final Memory: 21M/309M
[INFO] ------------------------------------------------------------
```

「Spring」のライブラリだけでなく、

- ロギングライブラリの「SLF4J, Logback」
- JSON を扱うためのライブラリ「Jackson」
- 入力チェックを行なうためのライブラリ「Hibernate Validator」
- YAML を扱うためのライブラリ「SnakeYAML」
- ユニット・テスト用の「JUnit」「Mockito」「AssertJ」
- 組み込みアプリケーション・サーバ「Tomcat」

など、Web アプリケーションの開発に必要なライブラリが含まれていることが分かります。

[1.3.3] 「Hello World!」を出力するWebアプリケーションの作成

それでは、プログラミング入門の定番である「Hello World!」出力プログラムを「Spring Boot」で実装してみましょう。

*

先ほど雛形を作った際に出力された「src/main/java/com/example/HajibootApplication.java」を開いて、以下のコードを記述してください。

【src/main/java/com/example/HajibootApplication.java】Hello Worldの作成

```
package com.example;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication // (1)
@RestController // (2)
public class HajibootApplication {

  @GetMapping("/") // (3)
  String home() {
    return "Hello World!"; // (4)
  }

  public static void main(String[] args) {
    SpringApplication.run(HajibootApplication.class, args); // (5)
  }
}
```

プログラム解説

項番	説明
(1)	「@SpringBootApplication」アノテーションが「Spring Boot」の醍醐味。 この「アノテーション」を付けることによって、さまざまな設定が自動的に有効になり、従来の「Springアプリケーション」で必要だった設定ファイル群が不要になる。
(2)	「@RestController」アノテーションを付けることで、このクラスが「Webアプリ ケーション」においてリクエストを受け付ける「コントローラ・クラス」になること を示す。 「REST」については、「第3章「Spring Boot」による「Webアプリ開発」で説明。
(3)	「HTTP リクエスト」を GET メソッドで受けつけるためのメソッドであることを示す「@GetMapping」アノテーションを付ける。 この設定によって、このアプリケーションに「/」というパスでアクセスがあった場合 に「home メソッド」が呼ばれる。
(4)	「HTTP レスポンス」を記述。「@RestController」アノテーションが付いているクラスのメソッドで文字列を返した場合は、その文字列がそのまま「HTTP レスポンス」として出力される。
(5)	「Spring Boot」アプリケーションを実行するための処理を、「main」メソッド内に記述。 「SpringApplication.run」メソッドの「第1引数」に、「@SpringBootApplication」アノテーションをつけたクラスを指定する。

*

このアプリケーションを実行してみましょう。

「main メソッド」があることから分かるように、このクラスだけで Web アプリケーションを起動することができます。

別途「アプリケーション・サーバ」を用意してアプリケーションを「デプロイ」する必要はありません。

*

「./mvnw spring-boot:run」コマンドを実行してください[6]

【ターミナル】「./mvnw spring-boot:run」コマンドを実行

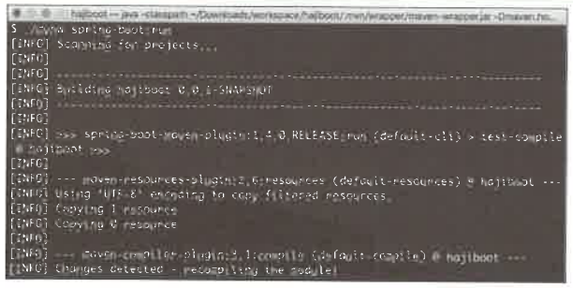
```
$ ./mvnw spring-boot:run
(途中略)
[INFO] <<< spring-boot-maven-plugin:1.4.0.RELEASE:run (default-cli) <
 test-compile @ hajiboot <<<
[INFO]
[INFO] --- spring-boot-maven-plugin:1.4.0.RELEASE:run (default-cli) @
 hajiboot ---

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.0.RELEASE)

2016-08-07 20:04:45.015  INFO 37669 --- [           main] com.example.
HajibootApplication          : Starting HajibootApplication on xxx
 with PID 37669 (/xxx/hajiboot/target/classes started by xxx in /xxx/
hajiboot)
2016-08-07 20:04:45.018  INFO 37669 --- [           main] com.example.
HajibootApplication          : No active profile set, falling back to
default profiles: default
2016-08-07 20:04:45.092  INFO 37669 --- [           main] ationCon
figEmbeddedWebApplicationContext : Refreshing org.springframework.
boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@
da28645: startup date [Sun Aug 07 20:04:45 JST 2016]; root of context
hierarchy
2016-08-07 20:04:46.238  INFO 37669 --- [           main] s.b.c.e.t.T
omcatEmbeddedServletContainer : Tomcat initialized with port(s): 8080 (http)
2016-08-07 20:04:46.248  INFO 37669 --- [           main] o.apache.ca
talina.core.StandardService   : Starting service Tomcat
2016-08-07 20:04:46.249  INFO 37669 --- [           main] org.apache.
catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.4
2016-08-07 20:04:46.349  INFO 37669 --- [ost-startStop-1] o.a.c.c.C.[
Tomcat].[localhost].[/]       : Initializing Spring embedded WebAppli
cationContext
2016-08-07 20:04:46.349  INFO 37669 --- [ost-startStop-1] o.s.web.con
text.ContextLoader            : Root WebApplicationContext: initializ
ation completed in 1262 ms
2016-08-07 20:04:46.491  INFO 37669 --- [ost-startStop-1] o.s.b.w.ser
vlet.ServletRegistrationBean  : Mapping servlet: 'dispatcherServlet'to [/]
2016-08-07 20:04:46.494  INFO 37669 --- [ost-startStop-1] o.s.b.w.ser
vlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFil
ter' to: [/*]
```

[6] 初回実行時には必要なライブラリをダウンロードする必要があるため、遅いです。二回目以降は、すぐに実行可能になります。

```
2016-08-07 20:04:46.494  INFO 37669 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2016-08-07 20:04:46.495  INFO 37669 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2016-08-07 20:04:46.495  INFO 37669 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2016-08-07 20:04:46.772  INFO 37669 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@da28645: startup date [Sun Aug 07 20:04:45 JST 2016]; root of context hierarchy
2016-08-07 20:04:46.847  INFO 37669 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/],methods=[GET]}" onto java.lang.String com.example.HajibootApplication.home()
2016-08-07 20:04:46.850  INFO 37669 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)
2016-08-07 20:04:46.851  INFO 37669 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
2016-08-07 20:04:46.883  INFO 37669 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2016-08-07 20:04:46.883  INFO 37669 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2016-08-07 20:04:46.926  INFO 37669 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2016-08-07 20:04:47.084  INFO 37669 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2016-08-07 20:04:47.156  INFO 37669 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2016-08-07 20:04:47.162  INFO 37669 --- [           main] com.example.HajibootApplication          : Started HajibootApplication in 2.583 seconds (JVM running for 5.828)
```



「./mvnw spring-boot:run」コマンドを実行

ログに出力されていますが、8080 ポートで「Tomcat」が起動しています。
「SpringApplication.run」メソッドで、「組み込みサーバ」を起動しています。

コラム Maven Wrapper

「Spring Initializr」でプロジェクトを作る場合、「mvnw」というスクリプトが付属しています。
これは「ビルド・ツール」である「Maven」の「ラッパースクリプト」です。
このスクリプトを実行すると「Maven」がダウンロードされていない場合は、「Maven」のダウンロードが行なわれるため、「Maven」をインストールしなくともすぐに「Maven」を使うことができます。

利用方法は「mvnw」と同じフォルダで、「Mac/Linux」の場合は「./mvnw ...」を、「Windows」の場合は「mvnw.bat ...」を実行するだけです。

環境変数「JAVA_HOME」に「JDK のインストール・フォルダ」を設定しておく必要があります。

実行

それでは「Web ブラウザ」で「http://localhost:8080」にアクセスしてみましょう。

「http://localhost:8080」にアクセス

「Hello World!」が出力されました。

*

アプリケーションを停止するには「Ctrl+C」を実行してください。
以下のようにアプリケーションがシャットダウンされます。

【コマンド・プロンプト】アプリケーションを停止

```
2016-08-07 20:20:08.548  INFO 38240 --- [       Thread-2] ationConfigEmbeddedWebApplicationContext : Closing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@748eeab7: startup date [Sun Aug 07 20:09:30 JST 2016]; root of context hierarchy
```

＊

ここまでの作業は、非常に簡単だったのではないでしょうか。

- 少ない依存関係の設定
- 1つのJavaクラス作成
- コマンドラインでアプリケーション実行

するだけで、「Hello World!」を出力するアプリケーションが作れました。

アプリケーションを実行する際に出力される「ログ」の「フォーマット」や、コンソール上の「ログ」に「色をつける設定」[7]なども「Spring Boot」が行なっています。
運用面で便利なちょっとした工夫がされており、簡単にアプリケーションが作れるわりには初めからクオリティは高いです。

ノート ログのフォーマットはデフォルトで、「ログ日付 ログレベル PID - [スレッド名] クラス名 : ログメッセージ」形式です。

[1.3.4] 実行可能な「jar ファイル」の作成

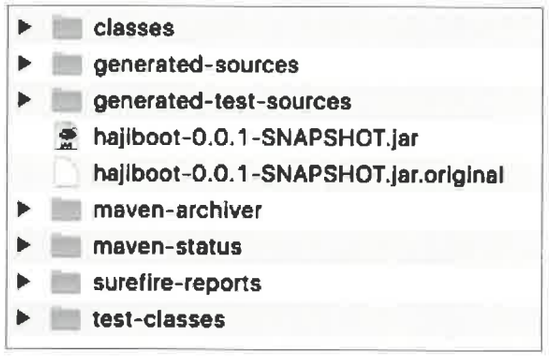
先ほどは「./mvnw spring-boot:run」コマンドでアプリケーションを起動しました。

こんどは配布や実行が可能な「jar ファイル」を作りましょう。
これも簡単です。以下のコマンドを実行してください。

【ターミナル】実行可能な「jar」の作成

```
$ ./mvnw package
```

「target」フォルダ内に「hajiboot-0.0.1-SNAPSHOT.jar」が作られたと思います。



「target」フォルダ

[7]「Windows」のコマンド・プロンプトでは色はつきません。「Mac」や「Linux」上のコンソールだと、ログがカラーリングされます。

「spring-boot-maven-plugin」によってこの「jar ファイル」は実行可能形式になっています。
さっそく実行してみましょう。

【ターミナル】「jar ファイル」の実行

```
$ java -jar target/hajiboot-0.0.1-SNAPSHOT.jar
( 途中略 )
2016-08-07 19:56:25.209  INFO 37532 --- [           main] s.b.c.e.t.T
omcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2016-08-07 19:56:25.218  INFO 37532 --- [           main] com.examp
le.HajibootApplication          : Started HajibootApplication in 3.0
seconds (JVM running for 3.439)
```

先ほどの「./mvnw spring-boot:run」と同じように「Tomcat」が「8080 番ポート」で立ち上がりました。
この「jar」さえあれば、アプリケーションが即実行可能であり、とてもポータブルです。

「HTTP」を待ち受ける「ポート番号」を変更するのも簡単です。
実行時に「--server.port= ポート番号」オプションを付けてください。

【ターミナル】「ポート番号」を変更して実行

```
$ java -jar target/hajiboot-0.0.1-SNAPSHOT.jar --server.port=8888
( 途中略 )
2016-08-07 19:57:47.393  INFO 37552 --- [           main] s.b.c.e.t.T
omcatEmbeddedServletContainer : Tomcat started on port(s): 8888 (http)
2016-08-07 19:57:47.402  INFO 37552 --- [           main] com.examp
le.HajibootApplication          : Started HajibootApplication in 2.989
seconds (JVM running for 3.435)
```

「Spring Boot」では、多くのプロパティをコマンドライン引数で設定可能[8]ですし、自分でオプションを追加するのも容易です。
このように、一度ビルドした「jar ファイル」に対して、簡単に設定変更できるような作りにすることで、アプリケーションのポータビリティを高めることができます。

はじめからこのような仕組みが用意されていることから、「ポータブルなアプリケーションを容易に作れるようにしたい」という「Spring Boot」の姿勢が伺えます。

[1.3.5] 「Spring Tool Suite」を利用した「Spring Boot」アプリケーションの開発

次に、Spring 謹製の「統合開発環境」(IDE)「Spring Tool Suite」(以降、「STS」) を利用して「Spring Boot」アプリケーションを開発します。

「STS」をインストールしていない場合は、「**[附録 .B] Spring Tool Suite のインストール**」を参照して、インストールしてください。

[8] http://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

● 作ったプロジェクトを STS にインポート

まずは、先ほど作ったプロジェクトをインポートする方法を説明します。

*

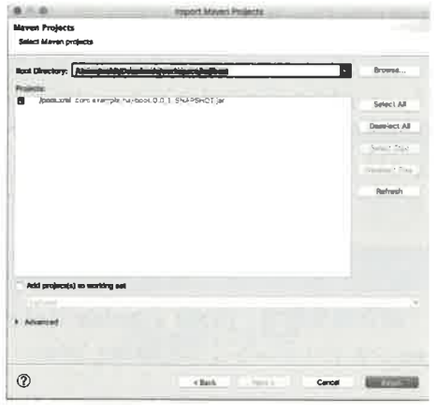
「STS」のメニューから「File」→「Import」を選択します。

「File」→「Import」メニューを選択

インポート種別として、「Maven」→「Exsisting Maven Projects」を選択します。

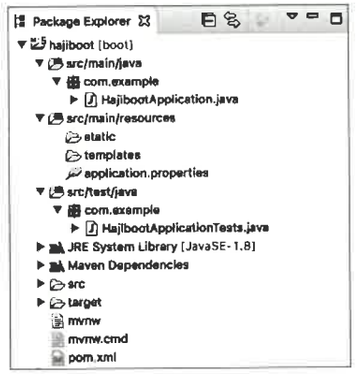
「Maven」→「Exsisting Maven Projects」を選択

「Browse」をクリックして、作ったプロジェクト（hajiboot ディレクトリ）を選択し、「Finish」をクリックします。



「Browse」をクリックして、作ったプロジェクトを選択

「Package Explorer」内に「hajiboot プロジェクト」が現われます。

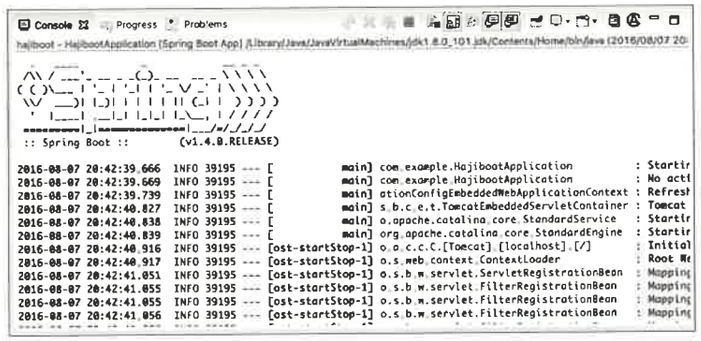


「Package Explorer」内にプロジェクトが追加される

「HajibootApplication.java」を右クリックして、「Run As」→「Spring Boot App」をクリックすると、先ほどと同様に、「組み込みサーバ」が起動します。

「Run As」→「Spring Boot App」で組み込みサーバを起動

「Console」に、「ターミナル」で実行したときと同じログが、出力されます。



「Console」画面

以降、本書で「アプリケーションを起動」、または「アプリケーションを実行」と言う場合、「./mvnw spring- boot:run コマンドを実行する」、または「Run As → Spring Boot App をクリックする」ことを指します。

*

ここで説明した「プロジェクト・セットアップ手法」は、「Spring Boot」に特化したものではなく、Mavenプロジェクトの汎用的な方法です。

*

次に、「STS」と「Spring Initializr」の連携によるプロジェクト作成方法を 説明します。

●「STS」で「Spring Boot」プロジェクトを新規作成

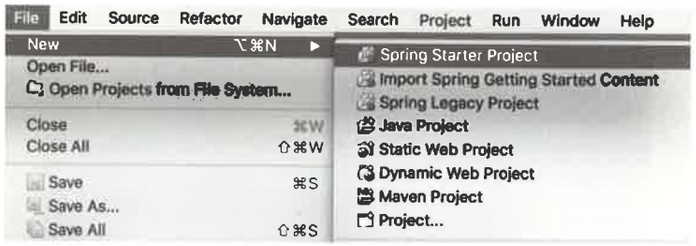
「STS」には「Spring Starter Project」という「Spring Boot」プロジェクトの雛形(ひながた)を簡単に作る仕組みが用意されています。

これは「STS」から前述の「Spring Initializr」にアクセスして雛形プロジェクトを作り、「STS」内に直接インポートする方法です。

今回は、これを使って、プロジェクトを新規作成します。

*

「STS」のメニューから、「File」→「New」→「Spring Starter Project」を選択します。

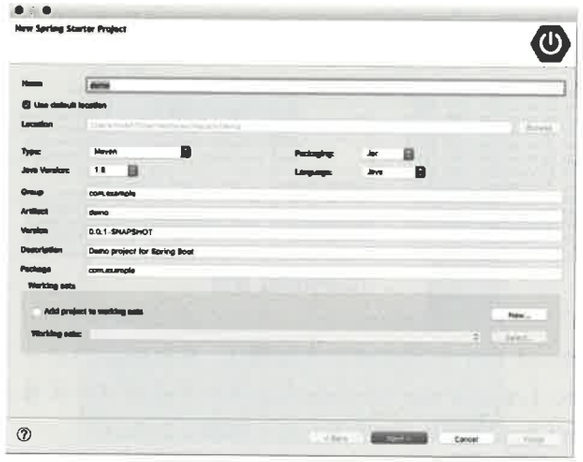


「File」→「New」→「Spring Starter Project」メニューを選択

プロジェクト情報の入力画面が表示されます。

各種入力項目はここではデフォルトのままにします。必要に応じて変更してください。

「Next」をクリックしてください。



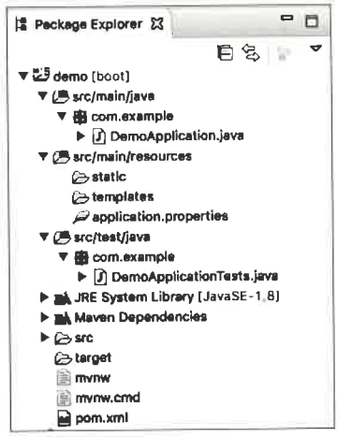
プロジェクト情報を入力

利用するプロジェクトを選択する画面が表示されます。

「Dependencies」に「Web」に入力して選択肢を絞り、「Web」にチェックを入れて、「Finish」をクリックしてください。

「Web」にチェックを入れて、「Finish」をクリック

「Package Explorer」内に「demoプロジェクト」が現われます。



「Package Explorer」内に「demoプロジェクト」が現われる

先ほどの「HajibootApplication.java」と同様に「DemoApplication.java」を編集すればいいです。

*

実際は「Spring Initializr」経由でプロジェクトを生成しているだけなので、「STS」から「Spring Starter Project」でプロジェクトを作る場合も、「Spring Initializr」でプロジェクトをダウンロードして「STS」にインポートする場合も、同じです。

> **ノート** 本書ではビルドツールとして「Maven」を使いますが、「Spring Boot」は「Maven」以外にも「Gradle」向けにもプラグインを提供しています。「Gradle」を使う例は、公式ドキュメント[9][10]を参照してください。

[10] http://docs.spring.io/spring-boot/docs/1.4.0.RELEASE/reference/html/using-boot-build-systems.html#using-boot-gradle
[11] http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-gradle

[1.3.6] 「Spring Dev Tools」でサクサク開発

ここまで「Spring Boot」を使うと、簡単にアプリケーションを実行することが 分かったと思います。

*

では、アプリケーションを変更した場合は、毎回起動し直さなくてはいけないのでしょうか。

それでは面倒臭くて、簡単にアプリケーションを実行できた効果が減少してしまいます。

そこで、「Spring Boot」と一緒に使うと効果的なツール「Spring Dev Tools」を紹介します。

*

「Spring Dev Tools」を使うことで、アプリケーションを起動したまま、Java クラス の変更を動的に反映させることができます。このような仕組みを、「Hot Reloading」と呼びます。

同様な有償ツールとしては「JRebel」がありますが、「Spring Dev Tools」は無償で利用可能です[11]。

*

アプリケーションで「Spring Dev Tools」を使うために、「pom.xml」に、以下のように修正してください。

【pom.xml】「Spring Dev Tools」の設定

```
<!-- ( 省略 ) -->
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <!-- ここから追加 -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
  </dependency>
  <!-- ここまで -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
<!-- ( 省略 ) -->
```

先ほどと同じように「Run As」→「Spring Boot App」でアプリケーションを起動します。

ログを見ると「restartedMain」というスレッド名が出力されていることが分かります。

[11]「JRebel」のほうが、汎用的に使えるようです。

これが、「Spring Dev Tools」が有効になっていることを示します。

「Spring Dev Tools」有効後のログ

```
( 略 )
2016-08-07 23:09:53.257  INFO 43023 --- [  restartedMain] s.b.c.e.t.T
omcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2016-08-07 23:09:53.263  INFO 43023 --- [  restartedMain] com.examp
le.HajibootApplication        : Started HajibootApplication in 2.595
seconds (JVM running for 3.044)
```

実 行

まずは、Java コードを変更する前に、Web ブラウザで「http://localhost:8080」にアクセスしてみましょう。

「http://localhost:8080」にアクセス

先ほどと同様に「Hello World!」が出力されました。

では、こんどは Java コード中の「Hello World」を「Hello Spring Boot」に変更してみましょう。

Java コードの変更箇所

```
@GetMapping("/")
String home() {
  return "Hello Spring Boot!"; // 出力する文字列を変更する
}
```

「STS」で「ソース・コード」を保存し、コンパイルしたら、以下のようなログが出力されたでしょうか[12]。

【ターミナル】リロード時のログ

```
2016-08-07 23:16:28.789  INFO 43118 --- [       Thread-5] ationConfigEm
beddedWebApplicationContext : Closing org.springframework.boot.context.
embedded.AnnotationConfigEmbeddedWebApplicationContext@59338c33: startup
date [Sun Aug 07 23:16:03 JST 2016]; root of context hierarchy
2016-08-07 23:16:28.791  INFO 43118 --- [       Thread-5] o.s.j.e.a.Annotat
ionMBeanExporter        : Unregistering JMX-exposed beans on shutdown
```

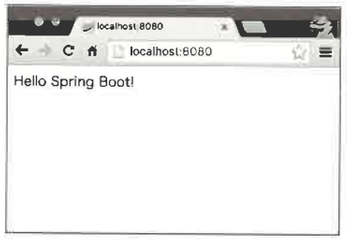
[12] ログが出力されない場合は、「target/classes/com/example/HajibootApplication.class」が更新されていません。IDE の設定でコンパイルが適切に行なわれているか確認してください。

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.0.RELEASE)

2016-08-07 23:16:28.977  INFO 43118 --- [  restartedMain] com.example.Haj
ibootApplication          : Starting HajibootApplication on xxx with PID
43118 (/xxx/workspace/hajiboot/target/classes started by makit in /xxx/
workspace/hajiboot)
(略)
2016-08-07 23:16:29.611  INFO 43118 --- [  restartedMain] s.b.c.e.t.T
omcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2016-08-07 23:16:29.613  INFO 43118 --- [  restartedMain] com.examp
le.HajibootApplication          : Started HajibootApplication in 0.67
 seconds (JVM running for 27.45)
```

*

もう一度、Web ブラウザで「http://localhost:8080」にアクセスしてください。



「http://localhost:8080」にアクセス

アプリケーションを再起動することなく、「ソース・コード」の変更が反映されました。

これで効率良く開発を進めることができるでしょう。

*

「Spring Dev Tools」は「Hot Reloading」の他に、「テンプレート・エンジン」のキャッシュ無効化も行ないます。

【3.3「Thymeleaf」を使った、「画面のある Web アプリ」の開発】で扱う「Thymeleaf」はデフォルトでテンプレートである HTML をキャッシュします。

開発中に HTML がキャッシュされると、HTML を修正するたびにアプリケーションを再起動しなくてはならず、効率が悪いです。

「Spring Dev Tools」を導入することで、再起動することなく HTML の変更を起動中のアプリケーションに反映させることができます。

また、「Spring Dev Tools」によるリロード機能は、「jar ファイル」を実行する際には無効化されるので、安心して開発中のみ本機能を利用できます。

ただし、100% リロードが成功するわけではありません。リロードがうまく行なわれない場合は、アプリケーションを再起動してください。

第2章 速習「Spring Framework」

前章で説明したように、「Spring Boot」は「Spring Framework」でアプリケーションを簡単に作るための仕組みであり、「Spring Boot」単品では、アプリケーションは作れません。

次章で「Web アプリケーション」を作りますが、まずは本章で、「DI」や「データ・アクセス」といった「Spring Framework」の基本的な要素を説明します。

2.1 「Spring Framework」による DI

「DI」とは「Dependency Injection」(依存性の注入)の略で、「Spring Framework」の根幹となる技術です。

「DI」によってクラス間の依存関係が自動で解決されます。

「インスタンス」の管理は「DI コンテナ」が行ないます。

「DI コンテナ」が「インスタンス」を生成し、その「インスタンス」に必要な「インスタンス」を設定した状態で、アプリケーションに返します。

「DI コンテナ」経由で「インスタンス」が生成されることによって、

- インスタンスの「スコープ」を制御できる
  (「シングルトンオブジェクトなのか」「毎回新規生成するのか」など)
- インスタンスの「ライフ・サイクル」をイベント制御できる
  (「インスタンス生成時」「破棄時」のイベント処理など)
- 共通的な処理を埋め込める
  (「トランザクション管理」や「ロギング処理」など)

といった副次効果を加えることができます。

また、オブジェクト間が疎結合になるため、「ユニット・テスト」がしやすくなるというメリットもあります。

「Spring Framework」による DI を理解するために、唐突ですが、「何らかの計算して、結果を表示する簡単なアプリケーション」を作りましょう。

このアプリケーションをステップ・バイ・ステップで変更していくことで、「Spring Framework」が提供する DI の機能を説明します。