Spring Boot 入門 ハンズオン 第二回

Java Doでしょう #10

スピーカー：ueno-haruki

自己紹介

- 上野　春毅
  @ueno-haruki
- 大学院生（博士後期課程）
- 仕事：大学でシステム開発
- 元：SIerのSE

本ハンズオン流れ

- 環境セットアップ&休憩　15分
- SpringBoot前半
- 休憩 10分
- SpringBoot後半
- 終了

下記URLにアクセスして
環境セットアップを行いましょう

https://goo.gl/upBqAE

今日のコンテンツ

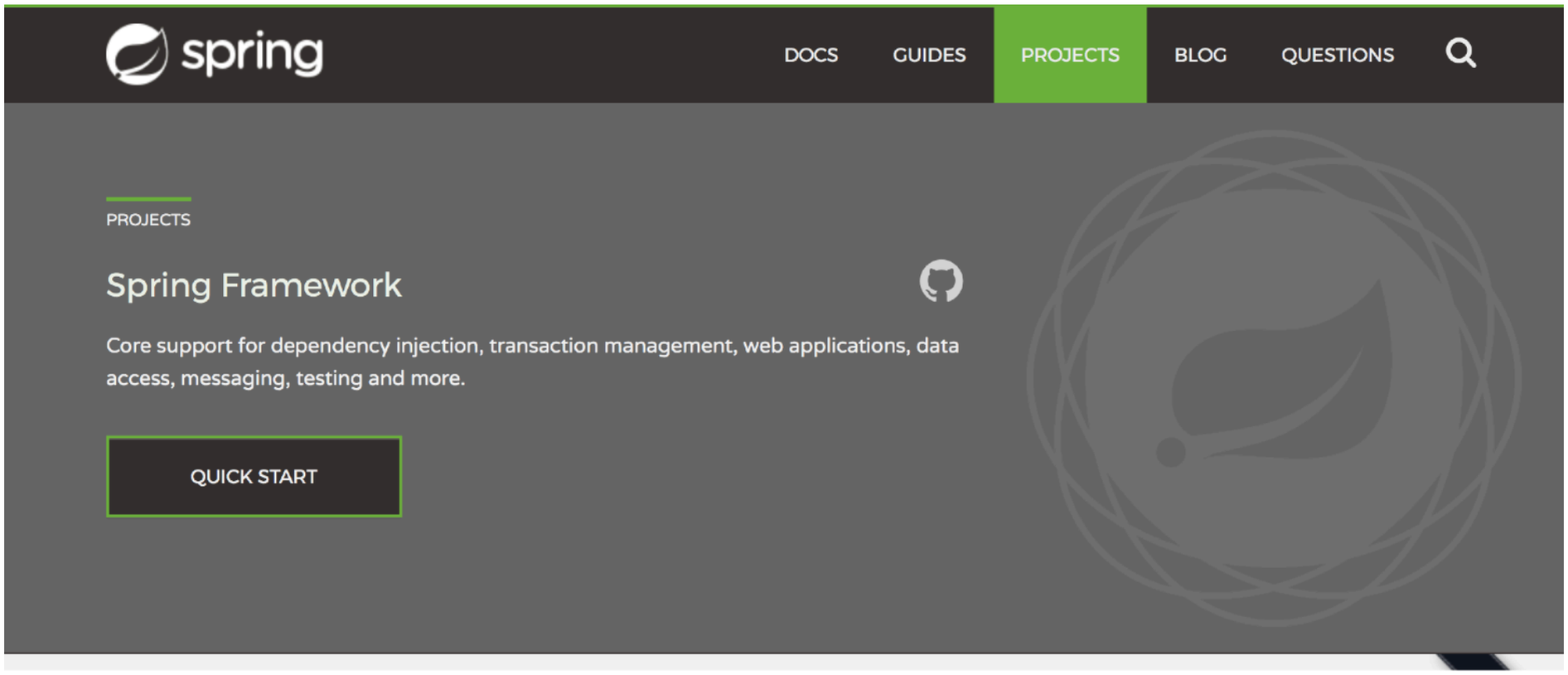
- Spring Bootってなに？
- RDBとSQLの概要
- Spring BootでSpringが提供するJDBCライブラリを使って、DBにアクセスしよう

そもそもSpringとは？

Javaで動くアプリケーションフレームワークです

WEB含むアプリケーションを作る際に、便利な機能を使って効率的に開発するためのものです

Springもその一つで、とても有力なフレームワークとして注目されています



Springが提供する便利なモジュール

たくさんあります

Spring Projects

Reactor Core	Reactor Project	Spring AMQP
Spring Batch	Spring Boot	Spring Cloud CLI
Spring Cloud Cluster	Spring Cloud Commons	Spring Cloud Config
Spring Cloud Connectors	Spring Cloud Consul	Spring Cloud Contract
Spring Cloud Data Flow	Spring Cloud Data Flow for Apache Mesos	Spring Cloud Data Flow for Apache YARN
Spring Cloud Data Flow for Cloud Foundry	Spring Cloud Data Flow for Kubernetes	Spring Cloud Function
Spring Cloud GCP	Spring Cloud Netflix	Spring Cloud Pipelines
Spring Cloud Sleuth	Spring Cloud Spinnaker	Spring Cloud Stream
Spring Cloud Stream App Starters	Spring Cloud Task	Spring Cloud Task App Starters
Spring Cloud Vault	Spring Cloud Vault	Spring Cloud Zookeeper
Spring CredHub	Spring Data Commons	Spring Data Envers
Spring Data for Apache Solr	Spring Data GemFire	Spring Data JDBC Extensions
Spring HATEOAS	Spring Integration	Spring IO Platform

…ありすぎません？

こんなにあったらどうしたらいいかよくわらない

Spring Boot

Spring bootはたくさんあるモジュールを使って
簡単にアプリケーションを作ることができるものです

設定はSpringBootが自動で行うので、
書き換えたい設定があったら書き換えるというスタイルです

RDB and SQL

SQLを使って、

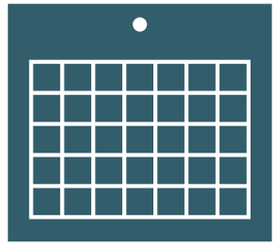
RDBに格納されるテーブルのデータを

読み書きする

SQL

RDB

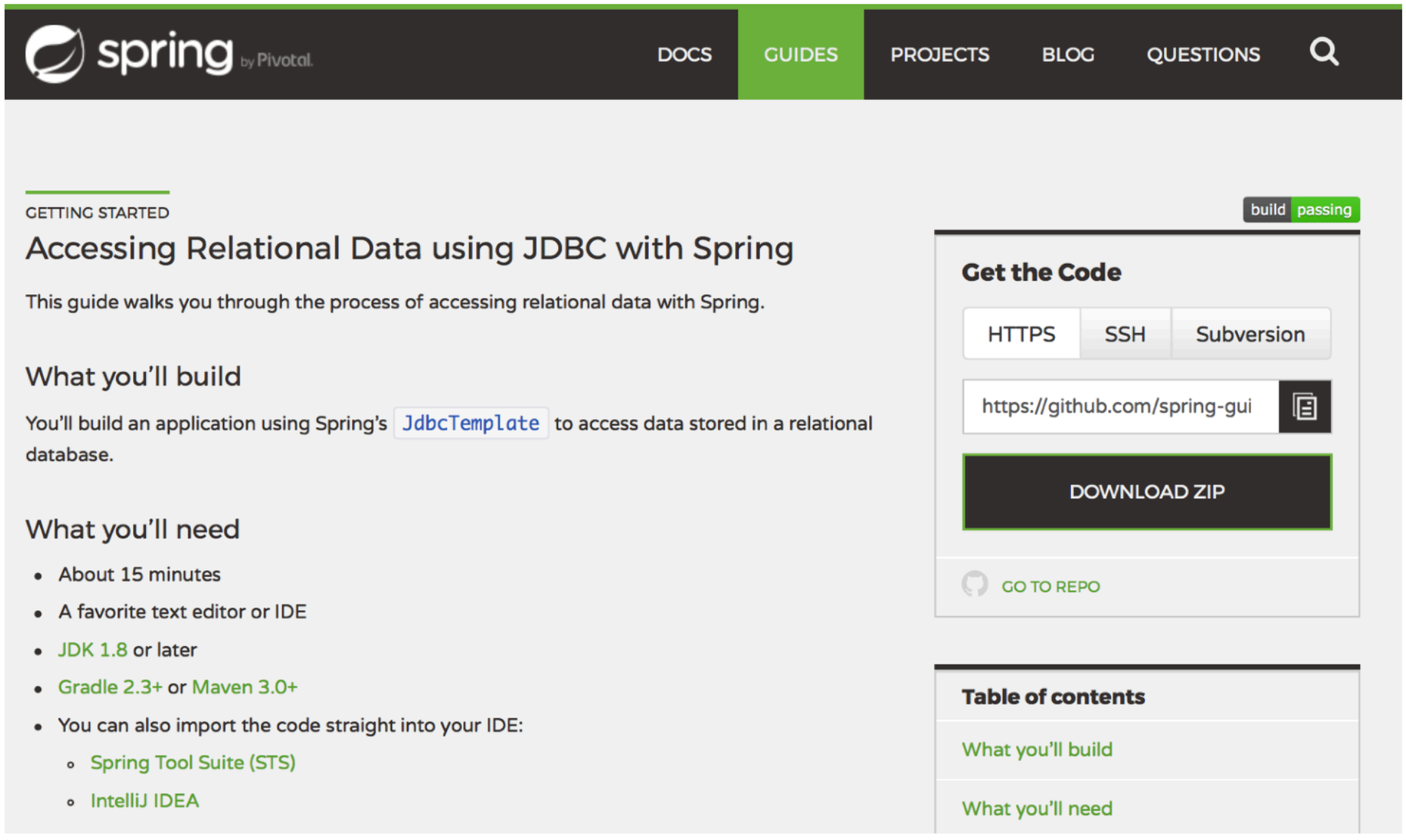
テーブル



id	first_name	last_name
1	John	Woo
2	John	Titor
3	Josh	Long

Today’s Main Contents ！ JDBC with Spring

RDB(PostgreSQL, MySQL, Oracle, etc…)にアクセスするために
Springが提供するモジュール



このモジュールを使うメリットはなんでしょう...？

そもそも、JavaでRDBにアクセスする代表的な方法として

- 素のJDBC Driverを使う
- RDB用アクセスライブラリを使う
  - SpringのJDBCライブラリ
  - Mybatis
  - Sql2o
  - etc…

素のJDBC Driver

Driverクラスのロード～コネクションの管理などを自分で行う

```
import java.sql.*;
class DBAccess {
  public static void main (String args[])
  throws SQLException, ClassNotFoundException {

    Class.forName(“org.postgresql.Driver");
    Connection conn;
    try ( conn = DriverManager.getConnection("jdbc:postgresql://127.0.0.1:5432/testdb", "test", “pass”)  ) {

      PreparedStatement stmt = conn.prepareStatement();
      ResultSet rset = stmt.executeQuery("select first_name, last_name from customer”);

      while ( rset.next() ) {
          System.out.println(rset.getInt(1) + "\t" + rset.getString(2));
      }
    catch(Exception e) {
    }
    finally() {
    }
  }
}
```

JDBCを使うかなり簡単な例

これから行うのはもっと簡素に書くことができます

Accessing Relational Data using JDBC with Spring

タイトルのページに沿って一部簡略化しながら進めます

概要

- JdbcTemplateクラスを使って
- RDBにアクセスして
- SQLを実行し
- 結果を表示する

Create a project

spring-bootとjdbcの設定を入れましょう。

今回ハンズオンでは、RDBはインストール不要なインメモリで動作するH2を使用しますので、
その設定を入れましょう。

pom.xmlを開いて、

下記のjunitの設定（今回不要）を消して、

```
<url>http://maven.apache.org</url>
```

```
</project>
```
の間に書きましょう。

juniの設定

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

```
<parent>
    <!-- spring-bootの設定 -->
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.6.RELEASE</version>
</parent>

<properties>
    <java.version>1.8</java.version>
</properties>

<dependencies>
    <dependency>
        <!-- springbootのjdbcの設定 -->
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>
    <dependency>
        <!-- H2(RDB)の設定 -->
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <!-- mavenでspringbootをサポートする設定 -->
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

pom.xmlの記述の確認

pom.xmlに書いた記述が合っているか確認するためにmvnでコンパイルしましょう。

```
$ mvn compile
```

下記のようなログが出力されるので、
[INFO] BUILD SUCCESS、
が確認できたらOKです。

spring-bootやh2のjarファイルがダウンロードされることがわかります。
（ここに記載があるのはspring-bootはすでにダウンロード済みだったため出力されていません）

休憩・閑話休題

7月に長野に行ってきました

松本城をみてきました

蕎麦が美味しかったです

個人的にとても好みでした

Create a Customer object

DBに作成するテーブルを以下としましょう。

id	first_name	last_name
1	John	Woo
2	John	Titor
3	Josh	Long

このテーブルと対応するクラスを作成します。

作る場所は
src/main/java/jp/javado/springbootです

クラス名はCustomer.javaとして作ります。

```
package jp.javado.springboot;

public class Customer {
    private long id;
    private String firstName, lastName;

    public Customer() {
    }

    public Customer(long id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
                "Customer[id=%d, firstName='%s', lastName='%s']",
                id, firstName, lastName);
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getFirstName() {
        return this.firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return this.lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Store and retrieve data

SpringBootを実行するためのクラスを作成しましょう

src/main/java/jp/javado/springboot/App.javaのクラスが雛形で用意されているので、それを書き換えて作成しましょう。

このクラスがSpringBootのアプリケーションとして最初に動くクラスであることを示すために@SpringBootApplicationを使います。

このアプリケーションはコマンドラインで動作するため、CommandLineRunnerインターフェースを実装します。

Springが提供するRDBにアクセスするためのクラスが、JdbcTemplateクラスです。 JdbcTemplateクラスを使えるようにするために、フィールド変数として書き、中身はRDBに接続するための所々の設定をSpringが自動的に行ったものを、@Autowiredをつけることで扱うことができるようになります。

@AutowiredはSpringコンテナが管理するクラスの実装をDIするためのアノテーションです。

```
package jp.javado.springboot;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.jdbc.core.JdbcTemplate;

@SpringBootApplication
public class App implements CommandLineRunner {

    private static final Logger log = LoggerFactory.getLogger(App.class);

    public static void main(String args[]) {
        SpringApplication.run(App.class, args);
    }

    @Autowired
    JdbcTemplate jdbcTemplate;

    @Override
    public void run(String... strings) throws Exception {

    }
}
```

Store and retrieve data

今回は、サンプルに従ってアプリケーション実行時にテーブルを作成しましょう。
開発時はこういったケースはあまりないとは思いますが、今回は練習用です。

run()メソッドの中に下記を書いて、アプリケーション実行時にテーブルを作成します。
なお、最初の「DROP TABLE customers IF EXSITS」は、今回は何度もアプリケーションを動かすため、
customersテーブルが存在していたら消して作り直すためにあります。

```
        log.info("Creating tables");

        jdbcTemplate.execute("DROP TABLE customers IF EXISTS");
        jdbcTemplate.execute("CREATE TABLE customers(" +
                "id SERIAL, first_name VARCHAR(255), last_name VARCHAR(255))");
```

サンプルではテーブルの設計は以下です

- idカラムは、SERIAL型（<- データ挿入時に番号が自動でふられる）
- first_nameカラムは、VARCHAR(255)型（文字列を格納する）
- last_nameカラムは、VARCHAR(255)型（文字列を格納する）

これで、実行時にH2（RDB）にさきほどのテーブルができます。 まだ中身はないです。

id	first_name	last_name

Store and retrieve data

できあがったテーブルにデータを挿入して右記となるようにするために、
プログラム中でINSERT文を実行できるようにします。

id	first_name	last_name
1	John	Woo
2	John	Titor
3	Josh	Long

最初に挿入するためのデータを用意し、そのデータをDBに一件ずつ挿入するには、
jdbcTemplate.update()メソッドを使います。さきほどに続けて以下を書きます。

```
        List<Object[]> insertDataList = new ArrayList<>();
        insertDataList.add(new Object[]{ "John", "Woo"});
        insertDataList.add(new Object[]{ "John", "Titor"});
        insertDataList.add(new Object[]{ "Josh", "Long"});

        String insertSql = "INSERT INTO customers(first_name, last_name) VALUES (?,?)";

        for (Object[] insertData : insertDataList) {
            jdbcTemplate.update(insertSql, insertData);
        }
```

Store and retrieve data

RDBのテーブルのデータを取得するSELECT文を実行するためのメソッドとして,
jdbcTemplate.query()メソッドが用意されています

query("SQL文" , "SQLの条件に設定したい値", "SQL実行結果を格納する") となっています。
"SQL実行結果を格納する"方法としてサンプルでは、 ラムダ式を使ってDBから値をrs.getString()で値を取得して、
用意していたCustomerクラスにセットするができます。
DBから値をSELECTして実行できるプログラムをさらに続けて書きましょう。

```
        log.info("Querying for customer records where first_name = 'Josh':");

        String selectSql = "SELECT id, first_name, last_name FROM customers WHERE first_name = ?";
        List<Customer> customerList = jdbcTemplate.query(selectSql, new Object[] { "Josh" },
                (rs, rowNum) -> new Customer(rs.getLong("id"), rs.getString("first_name"), rs.getString("last_name")));

        // SQL実行して取得した値を表示
        customerList.stream().forEach(customer -> log.info(customer.toString()));
```

SpringBootアプリケーションの実行

以下のコマンドを実行すると、プログラムが実行され、

- DBにテーブルを作成
- テーブルに値を挿入
- テーブルから値を取得
- データを表示

ができます。

```
$ mvn spring-boot:run
```

この中で下記の行が表示されたデータとなります。

```
INFO 19776 --- [  main] jp.javado.springboot.App : Customer[id=3, firstName='Josh', lastName='Long']
```