Spring Boot 入門 ハンズオン

Java Doでしょう #09

講師役：haruki-ueno

環境セットアップ

教科書のURL

ハンズオン[環境のセットアップ]の手順URL

下記の手順ぐらいまで順次進めてください

1. https://github.com/java-do/20170422_seminar　にブラウザでアクセスしよう

自己紹介

- ハンドル名　ueno-haruki　（＝実名）
- JavaDo運営メンバー
- 主な飲み物：コーヒー
- 最近の書いているコード
  - Spring Boot
  - Spring JDBC
  - Wicket

SpringBoot

SpringBootとは

Springフレームワークで簡単にアプリケーションを
作るための仕組み

特徴

簡単にモダンなアプリケーションを作れる
-> 代表的な依存関係はstarter系にまとまっている

複雑な設定は、自動で設定されていて意識しなくて良い
-> 変更したい設定だけ変えれば良い

組み込みサーバが同梱されている
-> tomcatのインストールが不要

本ハンズオンの位置付け

Thymeleaf
On Spring

Spring Boot

Spring MVC

Spring Security

Spring AOP

Etc…

Spring Framework (DI等のcore)

pom.xmlを編集（赤字を追加）

```
<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/maven-v4_0_0.xsd">
～省略～
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
  </parent>
  <dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>junit</groupId>
～省略～
   </dependency>
  </dependencies>
  <build>
    <plugins>
     <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
  <properties>
    <java.version>1.8</java.version>
  </properties>
</project>
```

赤字は
SpringBootの設定

SpringBootの起動クラス

jp.javado.springbootパッケージに、
JavaDoSpringApplicationクラスを作成してみましょう

```
package jp.javado.springboot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class JavaDoSpringApplication {

    public static void main(String... args) {
        SpringApplication.run(JavaDoSpringApplication.class, args);
    }
}
```

起動確認

JavaDoSpringApplicationクラスを実行してみましょう

コマンドライン　（本セッションでは, こちらで進めます）

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

IntelliJの場合

「Run/Debug Configurations」 > 「Spring Boot」
Main class: jp.javado.springboot.JavaDoSpringApplication
でRun

Eclipseの場合

起動確認

正常に起動すると以下が出ます

```
dhcp7:springboot-handson1 ueno$ mvn spring-boot:run
[INFO] Scanning for projects...
〜〜中略〜〜
[INFO] --- spring-boot-maven-plugin:1.5.2.RELEASE:run (default-cli) @ springboot-handson1 ---


  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.2.RELEASE)

2017-04-17 13:38:16.428  INFO 9666 --- [main] j.j.springboot.JavaDoSpringApplication   : Starting
JavaDoSpringApplication on dhcp7 with PID 9666 (/Users/ueno/javado/springboot-handson1/target/classes
started by ueno in /Users/ueno/javado/springboot-handson1)
〜〜中略〜〜
2017-04-17 13:38:19.008  INFO 9666 --- [main] o.s.j.e.a.AnnotationMBeanExporter        : Registering
beans for JMX exposure on startup
2017-04-17 13:38:19.086  INFO 9666 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat
started on port(s): 8080 (http)
2017-04-17 13:38:19.094  INFO 9666 --- [main] j.j.springboot.JavaDoSpringApplication   : Started
JavaDoSpringApplication in 3.043 seconds (JVM running for 7.424)
```

画面を作って見ましょう

pom.xmlに赤字の箇所を追加

```
<dependencies>
  <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Web画面用Controller

以下のIndexControllerクラスを
jp.javado.springbootパッケージに作成しましょう

```
package jp.javado.springboot;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class IndexController {

    @RequestMapping("/web/index")
    public String index() {
        return "index";
    }
}
```

templatesフォルダ作成

springboot-handson1/src/main/resourcesフォルダ配下にtemplatesフォルダを作成

もし、springboot-handson1/src/main以下にresourcesフォルダがない場合は作成しましょう

htmlの作成

src/main/resources/templates配下に
index.htmlを作成しましょう

```
<!DOCTYPE html>
<html lang="ja">
<head>
    <meta charset="UTF-8"/>
    <title>Index</title>
</head>
<body>
<h1>Hello Web page</h1>
<p>This is Thymeleaf page on Spring Boot.</p>
</body>
</html>
```

ブラウザでアクセスして表示して見ましょう

http://localhost:8080/web/index

ソースコードの解説

```
package jp.javado.springboot;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class IndexController {

    @RequestMapping("/web/index")
    public String index() {
        return "index";
    }
}
```

@Controllerをつけることで
MVCのControllerとなる

@RequestMappingをつけることで
アクセスするためのURLを記述できる

returnはテンプレート名を返却している
＝ここでは、index.htmlをブラウザに返すということに

Controller側で設定した値を表示して見ましょう

- ControllerからViewへの値を渡すには

  ModelAndViewクラスを利用する

`addObject(Object attributeValue)`
Add an attribute to the model using parameter name generation.

`setViewName(String viewName)`
Set a view name for this ModelAndView, to be resolved by the DispatcherServlet via a ViewResolver.

IndexControllerの編集

修正：赤字の箇所

```
package jp.javado.springboot;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;


@Controller
public class IndexController {

    @RequestMapping("/web/index")
    public ModelAndView index(ModelAndView modelAndView) {
        modelAndView.setViewName("index");
        modelAndView.addObject("message1", "Controllerで設定した文字列");

        return modelAndView;
    }
}
```

setViewName()で返却するテンプレート名を設定

addObject()でVIEW側(html)に渡したい値をキーバリュー形式で設定

index.htmlを編集

赤字箇所を追記

```
<!DOCTYPE html>
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Index</title>
</head>
<body>
<h1>Hello Web page</h1>
<p>This is Thymeleaf page on Spring Boot.</p>
<p th:text="${message1}">replace message</p>
</body>
</html>
```

xmlns:thでthymeleafを使えるようにする

th:textで、ModelAndViewに設定したキーのバリューに置き換わる

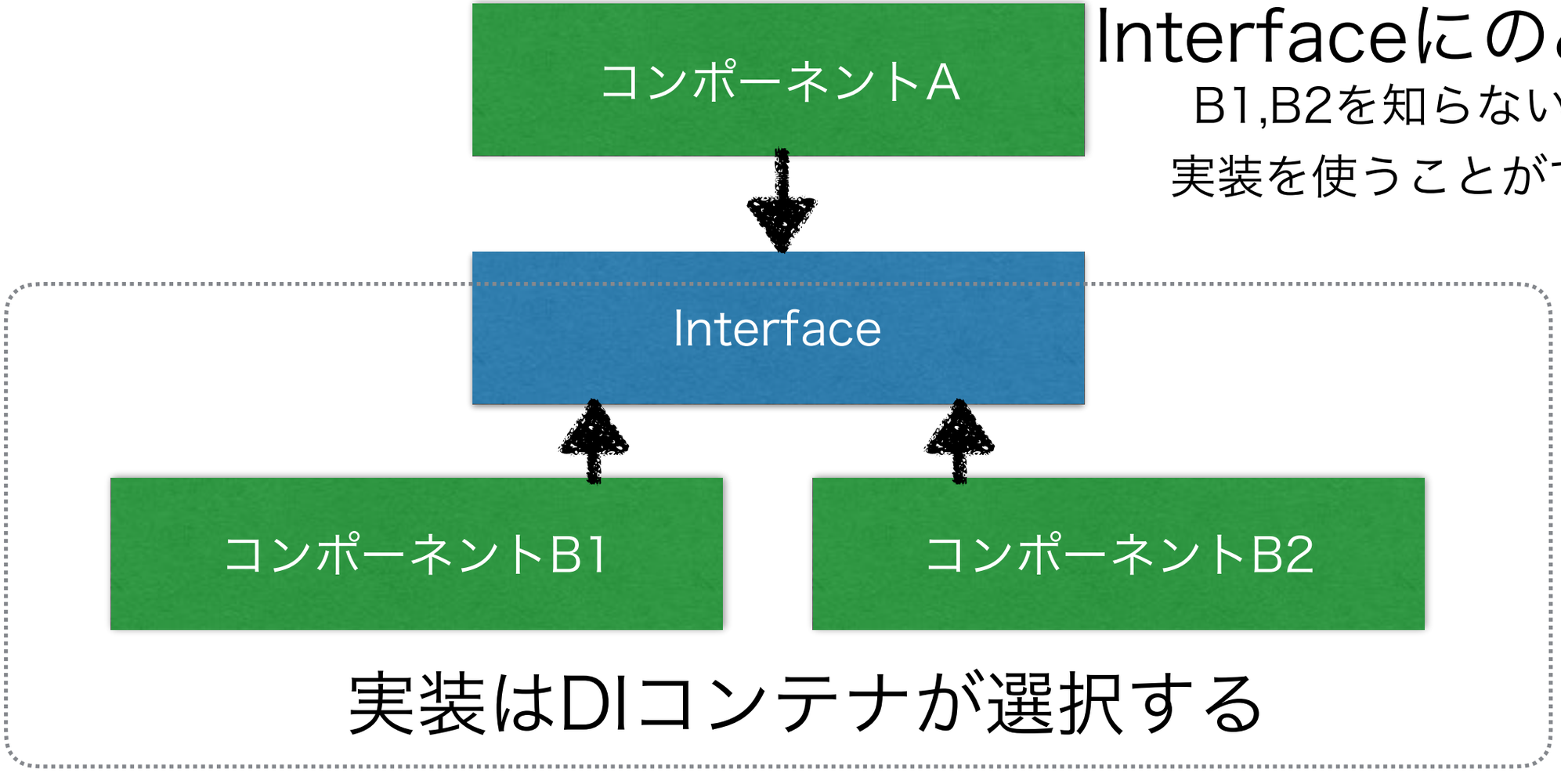
ブラウザでアクセスして表示して見ましょう

http://localhost:8080/web/index

Springといえば、DI（Dependency Injection）

実装に依存せずにInterfaceを介すことで、

コンポーネント間の依存関係を疎結合にできる



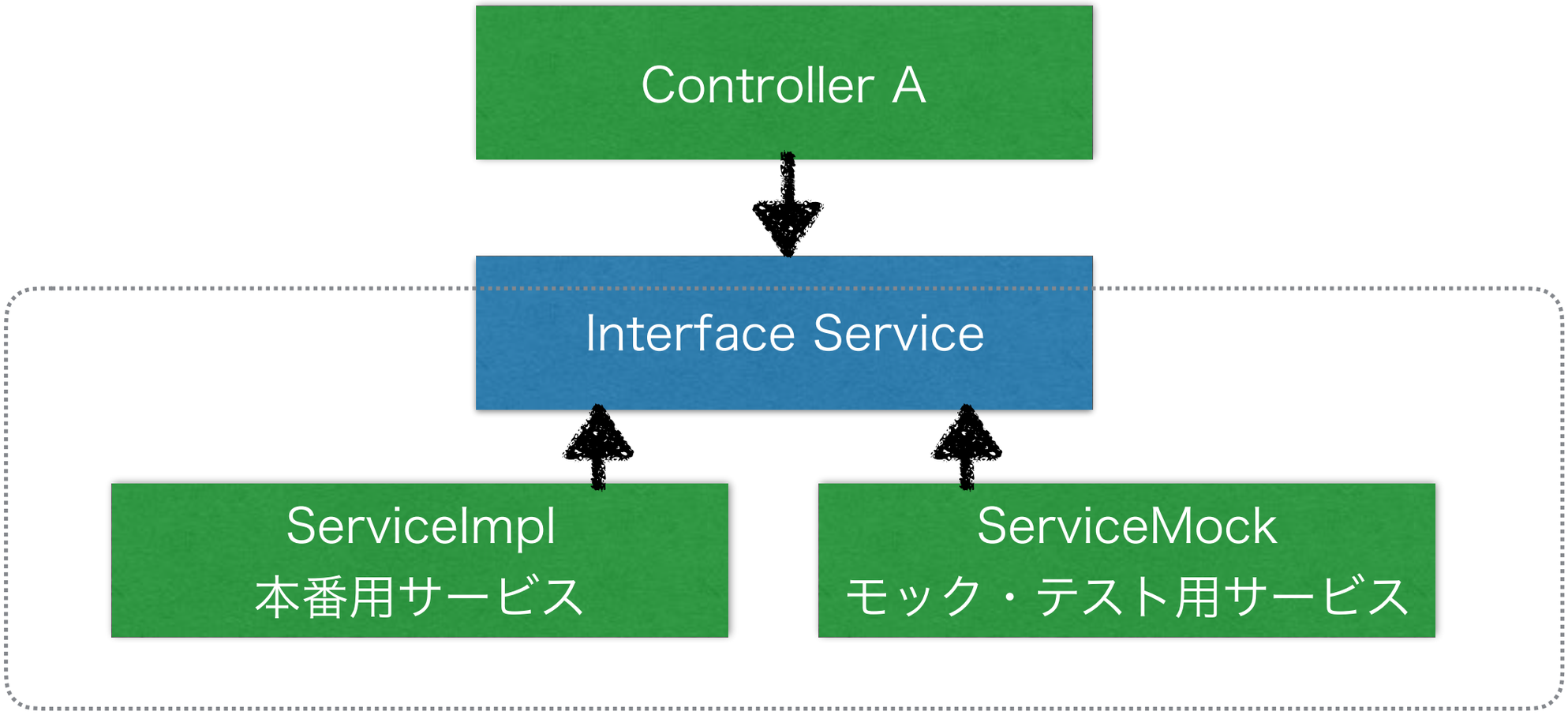
Interfaceにのみ依存

B1,B2を知らないが、

実装を使うことができる

Springといえば、DI（Dependency Injection）

例えば、Controllerクラスとサービスクラスだと



・他者が開発しているサービス実装を待っている間にモックで代用可能

・テストを行うために、テストで必要となる動作をするモックに切替可能

IndexControllerに
Serviceクラスを DIしてみましょう

SpringでDIするために、アノテーションを使います。
（古いものだとXML設定だったりしますが）

代表例

・@Service　　・・・実装のサービスクラスを表す

・@Autowired　・・・InterfaceにDIするためのもの

ISampleServiceの作成

ServiceクラスのInterfaceを作成

```
package jp.javado.springboot;


public interface ISampleService {

    String getSample(String id);

}
```

SampleServiceを作成

ISampleServiceをimplementsしたクラス

```
package jp.javado.springboot;

import org.springframework.stereotype.Service;

import java.util.HashMap;
import java.util.Map;


@Service
public class SampleService implements ISampleService {

    @Override
    public String getSample(String id) {
        Map<String, String> map = new HashMap<>();
        map.put("1", "spring");
        map.put("2", "boot");
        map.put("3", "di");
        return map.get(id);
    }
}
```

IndexControllerを編集

作成したSampleServiceをDIする

```
package jp.javado.springboot;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;


@Controller
public class IndexController {

    @Autowired
    private ISampleService sampleService;

    @RequestMapping("/web/index")
    public ModelAndView index(ModelAndView modelAndView) {
        modelAndView.setViewName("index");
        modelAndView.addObject("message1", "Controllerで設定した文字列");
        modelAndView.addObject("dimessage", sampleService.getSample("2"));
        return modelAndView;
    }
}
```

SampleServiceがDIされる

DIされたサービスを使って値を設定してみる

index.htmlを編集

赤字箇所を追記

```
<!DOCTYPE html>
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Index</title>
</head>
<body>
<h1>Hello Web page</h1>
<p>This is Thymeleaf page on Spring Boot.</p>
<p th:text="${message1}">replace message</p>
<p th:text="${dimessage}">replace message</p>
</body>
</html>
```

ブラウザでアクセスして表示して見ましょう

http://localhost:8080/web/index

ページのリンク

@{ アドレス }でページのリンクを作ることができます

<a th:href=“@{/web/list}”>List page Link

ここでは、一覧ページへのリンクを想定して
作ってみましょう

リンクを作成

index.htmlにリンクを追加

リンクページを作成

list.htmlを作成

ListControllerを作成

index.htmlを編集

```
<!DOCTYPE html>
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>Index</title>
</head>
<body>
<h1>Hello Web page</h1>
<p>This is Thymeleaf page on Spring Boot.</p>
<p th:text="${message1}">replace message</p>
<p><a th:href="@{/web/list}">List page Link</a></p>
</body>
</html>
```

赤字箇所を追記

aタグのth:href要素に @{/web/list} を書くことで、
http://localhost:8080/web/listへのリンクとなる

ListControllerの作成

IndexControllerをコピーして以下となるようにしましょう

```
package jp.javado.springboot;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;


@Controller
public class ListController {

    @RequestMapping("/web/list")
    public ModelAndView list(ModelAndView modelAndView) {
        modelAndView.setViewName("list");
        modelAndView.addObject("message1", "Controllerで設定した文字列");
        return modelAndView;
    }
}
```

list.htmlの作成

index.htmlをコピーして以下となるようにlist.htmlを作成しましょう

```
<!DOCTYPE html>
<html lang="ja" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>List</title>
</head>
<body>
<h1>List page</h1>
<p>This is Thymeleaf List page on Spring Boot.</p>
<p th:text="${message1}">replace message</p>
</body>
</html>
```

他の記法

URLのパラメータ取得

```
@RequestMapping(“/web/index/{sampleId}”)
public String index(ModelAndView modelAndView,
                    @PathVariable(“sampleId”) int sampleId)
```

リンクに値を設定

```
th:href=“@{‘/web/list/’ + ${sampleId}}”
```

みなさまの試してみたいことを試す時間にしましょう

- 例えば、以下とかいかがでしょう
  - サービスクラスを拡張・作成
    - 作成例：CafeServiceでドリンクが取得できるとか
- ページを拡張・作成
- ページ間で値の受け渡し

ここからは付録

Q. 従来の方法でTomcatサーバにデプロイするには？

85. Traditional deployment

85.1 Create a deployable war file

The first step in producing a deployable war file is to provide a `SpringBootServletInitializer` subclass and override its `configure` method. This makes use of Spring Framework's Servlet 3.0 support and allows you to configure your application when it's launched by the servlet container. Typically, you update your application's main class to extend `SpringBootServletInitializer`:

```
@SpringBootApplication
public class Application extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(Application.class);
    }

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

}
```

The next step is to update your build configuration so that your project produces a war file rather than a jar file. If you're using Maven and using `spring-boot-starter-parent` (which configures Maven's war plugin for you) all you need to do is to modify `pom.xml` to change the packaging to war:

```
<packaging>war</packaging>
```

If you're using Gradle, you need to modify `build.gradle` to apply the war plugin to the project:

```
apply plugin: 'war'
```

The final step in the process is to ensure that the embedded servlet container doesn't interfere with the servlet container to which the war file will be deployed. To do so, you need to mark the embedded servlet container dependency as provided.

```
<dependencies>
    <!-- ... -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
    </dependency>
    <!-- ... -->
</dependencies>
```

http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#howto-traditional-deployment

Q.Tomcatのバージョンを指定したいときは？

73.16.1 Use Tomcat 7.x or 8.0 with Maven

If you are using the starters and parent you can change the Tomcat version property and additionally import `tomcat-juli`. E.g. for a simple webapp or service:

```
<properties>
    <tomcat.version>7.0.59</tomcat.version>
</properties>
<dependencies>
    ...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>tomcat-juli</artifactId>
        <version>${tomcat.version}</version>
    </dependency>
    ...
</dependencies>
```

http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#howto-use-tomcat-7-maven

ご静聴いただき、

ありがとうございました