@Bean conditional registration

Spring 4, introduced a new annotation @Conditional, which is meant to be used on @Bean methods or (in case of @ComponentScan) with @Component classes, or the @Configuration classes.

The purpose of this annotation is to enable/disable target beans from being registered based on some condition. These conditions are applied at the time of the container startup.

Definition of @Conditional annotation

```
package org.springframework.context.annotation;

.....

public @interface Conditional {

  Class<? extends Condition>[] value();


}
```

Definition of Condition interface

```
package org.springframework.context.annotation;

 .....

public interface Condition {

  boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata);

}
```

@Conditional vs @Profile

The use of @Conditional and Condition is similar to selecting different profiles using @Profile but is more flexible because the conditions are specified in Java code rather than using some pre defined property flag or some system property.

Spring uses @Conditional on @Profile definition as meta annotation. That means @Conditional concept is on lower level than that of @Profile.

Spring profiles have been around since 3.1 and they were refactored in 4.0 to take advantage of this new way of applying conditions on bean registration. Here's the code source snippets of @Profile and ProfileCondition

```
.....
@Conditional(ProfileCondition.class)
public @interface Profile {
  ....
}
class ProfileCondition implements Condition {

 @Override
 public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
  if (context.getEnvironment() != null) {
   MultiValueMap<String, Object> attrs =
                        metadata.getAllAnnotationAttributes(
                                            Profile.class.getName());
  if (attrs != null) {
   for (Object value : attrs.get("value")) {
    if (context.getEnvironment().acceptsProfiles(
                                              ((String[]) value))) {
     return true;
    }
   }
   return false;
  }
 }
 return true;
 }
}
```

What is ConditionContext?

ConditionContext is the first parameter of ProfileCondition#matches method. This facade interface was introduced to work with Condition. We can query container related information via this interface. e.g. we can get instance of ConfigurableListableBeanFactory, Environment, BeanDefinitionRegistry etc.

What is AnnotatedTypeMetadata?

AnnotatedTypeMetadata helps retrieving information about specified annotation type used on underlying element i.e. the same element annotated with @Conditional: the method with @Bean or class with @Component or @Configuration.

This meta data helper can get us the attributes of annotations like @Bean, @Component, @Lazy, @Scope etc, used on the same method or class.

Example

Beans

```
public interface MyService {

}

public class MyServiceA implements MyService {

}

public class MyServiceB implements MyService {

}

public class ClientBean {

  @Autowired

  private MyService myService;


  public MyService getMyService () {

      return myService;

  }

}
```

Defining beans with condition

```
@Configuration
```

```
public class AppConfig {
  @Bean
  @Conditional(LocaleConditionUSA.class)
  public MyService myBeanA () {
      return new MyServiceA();
  }

  @Bean
  @Conditional(LocaleConditionCanada.class)
  public MyService myBeanB () {
      return new MyServiceB();
  }

  @Bean
  public ClientBean clientBean () {
      return new ClientBean();
  }
}
```

Implementations of Condition

```
import java.util.Locale;
```

```
public class LocaleConditionCanada implements Condition {


  @Override
  public boolean matches (ConditionContext context, AnnotatedTypeMetadata metadata)
{
      return Locale.getDefault()
                   .equals(Locale.CANADA);
  }
}


public class LocaleConditionUSA implements Condition {


  @Override
  public boolean matches (ConditionContext context, AnnotatedTypeMetadata metadata)
{
      return Locale.getDefault()
                   .equals(Locale.US);
  }
}
```

Main class

```
public class BeanConditionExample {


    public static void main(String[] args) {
        //In real application, System Locale would be used depending on your location.
        //Here we are setting it manually to test our example*/
        runApp(Locale.US);
        System.out.println("----------");
        runApp(Locale.CANADA);
    }

    public static void runApp(Locale locale) {
        System.out.printf("setting default locale: %s\n", locale);
        Locale.setDefault(locale);
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(
                        AppConfig.class);

        ClientBean bean = context.getBean(ClientBean.class);
        System.out.printf("Injected MyService instance in ClientBean: %s\n",
bean.getMyService()
                                                                    .getClass()
                                                                    .getSimpleName());
    }
}
```

Output

```
setting default locale: en_US
Injected MyService instance in ClientBean: MyServiceA
----------
```

```
setting default locale: en_CA

Injected MyService instance in ClientBean: MyServiceB
```

Example with ComponentScan

```
@Configuration

@ComponentScan(basePackageClasses = BeanConditionScanExample.class,

        useDefaultFilters = false,

        includeFilters = {@ComponentScan.Filter(

                type = FilterType.ASSIGNABLE_TYPE,

                value = {BeanConditionScanExample.MyClientBean.class,

                        BeanConditionScanExample.ServiceBeanImpl1.class,

                        BeanConditionScanExample.ServiceBeanImpl2.class})})


public class BeanConditionScanExample {


    public static void main(String[] args) {

        runApp(Locale.US);

        System.out.println("----------");

        runApp(Locale.CANADA);

    }


    public static void runApp(Locale locale) {
        System.out.printf("setting default locale: %s\n", locale);

        Locale.setDefault(locale);

        AnnotationConfigApplicationContext context =

                new AnnotationConfigApplicationContext(

                        BeanConditionScanExample.class);

        MyClientBean bean = context.getBean(MyClientBean.class);

        System.out.printf("Injected ServiceBean instance in MyClientBean: %s\n",
bean.getServiceBean()

                                                                             .getClass()

                                                                             .getSimpleName());

    }


    @Component

    public static class MyClientBean {

        @Autowired

        private ServiceBean serviceBean;


        public ServiceBean getServiceBean() {

            return serviceBean;

        }

    }


    public interface ServiceBean {

    }
    @Component
    @Conditional(LocaleConditionUSA.class)
    public static class ServiceBeanImpl1 implements ServiceBean {
    }

    @Component
    @Conditional(LocaleConditionCanada.class)
    public static class ServiceBeanImpl2 implements ServiceBean {
    }
}
```

Output

```
setting default locale: en_US
Injected ServiceBean instance in MyClientBean: ServiceBeanImpl1
----------
setting default locale: en_CA
Injected ServiceBean instance in MyClientBean: ServiceBeanImpl2
```