Package org.springframework.beans

Class BeanUtils

java.lang.Object
  org.springframework.beans.BeanUtils

```
public abstract class BeanUtils
extends Object
```

Static convenience methods for JavaBeans: for instantiating beans, checking bean property types, copying bean properties, etc.

Mainly for internal use within the framework, but to some degree also useful for application classes. Consider Apache Commons BeanUtils , BULL - Bean Utils Light Library , or similar third-party frameworks for more comprehensive bean utilities.

Author:

Rod Johnson, Juergen Hoeller, Rob Harrop, Sam Brannen, Sebastien Deleuze

Constructor Summary

Constructors

Constructor	Description
`BeanUtils()`	

Method Summary

All Methods Static Methods Concrete Methods Deprecated Methods

Modifier and Type	Method	Description
`static void`	`copyProperties(Object source, Object target)`	Copy the property values of the given source bean into the target bean.
`static void`	`copyProperties(Object source, Object target, Class <?> editable)`	Copy the property values of the given source bean into the given target bean, only setting properties defined in the given "editable" class (or interface).
`static void`	`copyProperties(Object source, Object target, String ... ignoreProperties)`	Copy the property values of the given source bean into the given target bean, ignoring the given "ignoreProperties".
`atic Method`	`findDeclaredMethod(Class <?> clazz, String`	Find a method with the given method name and the given

	methodName, Class <?>... paramTypes)	parameter types, declared on the given class or one of its superclasses.
static Method	findDeclaredMethodWithMinima (Class <?> clazz, String methodName)	Find a method with the given method name and minimal parameters (best case: none), declared on the given class or one of its superclasses.
static PropertyEditor	findEditorByConvention (Class <?> targetType)	Find a JavaBeans PropertyEditor following the 'Editor' suffix convention (for example, "mypackage.MyDomainClass" → "mypackage.MyDomainClassEditor").
static Method	findMethod(Class <?> clazz, String methodName, Class <?>... paramTypes)	Find a method with the given method name and the given parameter types, declared on the given class or one of its superclasses.
static Method	findMethodWithMinimalParamet (Class <?> clazz, String methodName)	Find a method with the given method name and minimal parameters (best case: none), declared on the given class or one of its superclasses.
static Method	findMethodWithMinimalParamet (Method [] methods, String methodName)	Find a method with the given method name and minimal parameters (best case: none) in the given list of methods.
static <T> Constructor <T>	findPrimaryConstructor (Class <T> clazz)	Return the primary constructor of the provided class.
static PropertyDescriptor	findPropertyForMethod (Method method)	Find a JavaBeans PropertyDescriptor for the given method, with the method either being the read method or the write method for that bean property.
static PropertyDescriptor	findPropertyForMethod (Method method, Class <?> clazz)	Find a JavaBeans PropertyDescriptor for the given method, with the method either being the read method or the write method for that bean property.
static Class <?>	findPropertyType(String propertyName, Class <?>... beanClasses)	Determine the bean property type for the given property from the given classes/interfaces, if possible.

`static String[]`	`getParameterNames(Constructor<?> ctor)`	Determine required parameter names for the given constructor, considering the JavaBeans `ConstructorProperties` annotation as well as Spring's `DefaultParameterNameDiscoverer`.
`static PropertyDescriptor`	`getPropertyDescriptor(Class<?> clazz, String propertyName)`	Retrieve the JavaBeans `PropertyDescriptors` for the given property.
`static PropertyDescriptor[]`	`getPropertyDescriptors(Class<?> clazz)`	Retrieve the JavaBeans `PropertyDescriptor`s of a given class.
`static <T> Constructor<T>`	`getResolvableConstructor(Class<T> clazz)`	Return a resolvable constructor for the provided class, either a primary or single public constructor with arguments, a single non-public constructor with arguments or simply a default constructor.
`static MethodParameter`	`getWriteMethodParameter(PropertyDescriptor pd)`	Obtain a new MethodParameter object for the write method of the specified property.
`static boolean`	`hasUniqueWriteMethod(PropertyDescriptor pd)`	Determine whether the specified property has a unique write method, i.e.
`static <T> T`	`instantiate(Class<T> clazz)`	**Deprecated.** as of Spring 5.0, following the deprecation of `Class.newInstance()` in JDK 9
`static <T> T`	`instantiateClass(Class<?> clazz, Class<T> assignableTo)`	Instantiate a class using its no-arg constructor and return the new instance as the specified assignable type.
`static <T> T`	`instantiateClass(Class<T> clazz)`	Instantiate a class using its 'primary' constructor (for Kotlin classes, potentially having default arguments declared) or its default constructor (for regular Java classes, expecting a standard no-arg setup).
`static <T> T`	`instantiateClass(Constructor<T> ctor, Object... args)`	Convenience method to instantiate a class using the given constructor.

`static boolean`	`isSimpleProperty(Class <? > type)`	Check if the given type represents a "simple" property: a simple value type or an array of simple value types.
`static boolean`	`isSimpleValueType(Class <? > type)`	Check if the given type represents a "simple" value type for bean property and data binding purposes: a primitive or primitive wrapper, an `Enum`, a `String` or other `CharSequence`, a `Number`, a `Date`, a `Temporal`, a `UUID`, a `URI`, a `URL`, a `Locale`, or a `Class`.
`static Method`	`resolveSignature(String signature, Class <? > clazz)`	Parse a method signature in the form `methodName[([arg_list])]`, where `arg_list` is an optional, comma-separated list of fully-qualified type names, and attempts to resolve that signature against the supplied `Class`.

Methods inherited from class java.lang.Object

`clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait` , `wait` , `wait`

Constructor Details

BeanUtils

```
public BeanUtils()
```

Method Details

instantiate

```
@Deprecated
public static <T> T instantiate(Class <T> clazz)
                         throws BeanInstantiationException
```

Deprecated.
as of Spring 5.0, following the deprecation of `Class.newInstance()` *in JDK 9*

onvenience method to instantiate a class using its no-arg constructor.

Parameters:

`clazz` - class to instantiate

Returns:

the new instance

Throws:

`BeanInstantiationException` - if the bean cannot be instantiated

See Also:

`Class.newInstance()`

instantiateClass

```
public static <T> T instantiateClass(Class<T> clazz)
                          throws BeanInstantiationException
```

Instantiate a class using its 'primary' constructor (for Kotlin classes, potentially having default arguments declared) or its default constructor (for regular Java classes, expecting a standard no-arg setup).

Note that this method tries to set the constructor accessible if given a non-accessible (that is, non-public) constructor.

Parameters:

`clazz` - the class to instantiate

Returns:

the new instance

Throws:

`BeanInstantiationException` - if the bean cannot be instantiated. The cause may notably indicate a `NoSuchMethodException` if no primary/default constructor was found, a `NoClassDefFoundError` or other `LinkageError` in case of an unresolvable class definition (for example, due to a missing dependency at runtime), or an exception thrown from the constructor invocation itself.

See Also:

`Constructor.newInstance(java.lang.Object...)`

instantiateClass

```
public static <T> T instantiateClass(Class<?> clazz,
                                    Class<T> assignableTo)
                          throws BeanInstantiationException
```

Instantiate a class using its no-arg constructor and return the new instance as the specified assignable type.

Useful in cases where the type of the class to instantiate (clazz) is not available, but the type desired (assignableTo) is known.

Note that this method tries to set the constructor accessible if given a non-accessible (that is, non-blic) constructor.

arameters:

`clazz` - class to instantiate

`assignableTo` - type that clazz must be assignableTo

Returns:

the new instance

Throws:

`BeanInstantiationException` - if the bean cannot be instantiated

See Also:

`Constructor.newInstance(java.lang.Object...)`

instantiateClass

```
public static <T> T instantiateClass(Constructor <T> ctor,
                                      Object ... args)
                               throws BeanInstantiationException
```

Convenience method to instantiate a class using the given constructor.

Note that this method tries to set the constructor accessible if given a non-accessible (that is, non-public) constructor, and supports Kotlin classes with optional parameters and default values.

Parameters:

`ctor` - the constructor to instantiate

`args` - the constructor arguments to apply (use `null` for an unspecified parameter, Kotlin optional parameters and Java primitive types are supported)

Returns:

the new instance

Throws:

`BeanInstantiationException` - if the bean cannot be instantiated

See Also:

`Constructor.newInstance(java.lang.Object...)`

getResolvableConstructor

```
public static <T> Constructor <T> getResolvableConstructor(Class <T> clazz)
```

Return a resolvable constructor for the provided class, either a primary or single public constructor with arguments, a single non-public constructor with arguments or simply a default constructor.

Callers have to be prepared to resolve arguments for the returned constructor's parameters, if any.

Parameters:

`clazz` - the class to check

Throws:

`IllegalStateException` - in case of no unique constructor found at all

ince:

See Also:

`findPrimaryConstructor(java.lang.Class<T>)`

findPrimaryConstructor

```
@Nullable
public static <T> Constructor<T> findPrimaryConstructor(Class<T> clazz)
```

Return the primary constructor of the provided class. For Kotlin classes, this returns the Java constructor corresponding to the Kotlin primary constructor (as defined in the Kotlin specification). For Java records, this returns the canonical constructor. Otherwise, this simply returns `null`.

Parameters:

`clazz` - the class to check

Since:

5.0

See Also:

Kotlin constructors ,
Record constructor declarations

findMethod

```
@Nullable
public static Method findMethod(Class<?> clazz,
                                String methodName,
                                Class<?>... paramTypes)
```

Find a method with the given method name and the given parameter types, declared on the given class or one of its superclasses. Prefers public methods, but will return a protected, package access, or private method too.

Checks `Class.getMethod` first, falling back to `findDeclaredMethod`. This allows to find public methods without issues even in environments with restricted Java security settings.

Parameters:

`clazz` - the class to check

`methodName` - the name of the method to find

`paramTypes` - the parameter types of the method to find

Returns:

the Method object, or `null` if not found

See Also:

`Class.getMethod(java.lang.String, java.lang.Class<?>...)` ,
`findDeclaredMethod(java.lang.Class<?>, java.lang.String, java.lang.Class<?>...)`

findDeclaredMethod

```
@Nullable
public static Method findDeclaredMethod(Class<?> clazz,
                                        String methodName,
                                        Class<?>... paramTypes)
```

Find a method with the given method name and the given parameter types, declared on the given class or one of its superclasses. Will return a public, protected, package access, or private method.

Checks `Class.getDeclaredMethod`, cascading upwards to all superclasses.

Parameters:

`clazz` - the class to check

`methodName` - the name of the method to find

`paramTypes` - the parameter types of the method to find

Returns:

the Method object, or `null` if not found

See Also:

`Class.getDeclaredMethod(java.lang.String, java.lang.Class<?>...)`

findMethodWithMinimalParameters

```
@Nullable
public static Method findMethodWithMinimalParameters(Class<?> clazz,
                                                     String methodName)
                                              throws IllegalArgumentException
```

Find a method with the given method name and minimal parameters (best case: none), declared on the given class or one of its superclasses. Prefers public methods, but will return a protected, package access, or private method too.

Checks `Class.getMethods` first, falling back to `findDeclaredMethodWithMinimalParameters`. This allows for finding public methods without issues even in environments with restricted Java security settings.

Parameters:

`clazz` - the class to check

`methodName` - the name of the method to find

Returns:

the Method object, or `null` if not found

Throws:

`IllegalArgumentException` - if methods of the given name were found but could not be resolved to a unique method with minimal parameters

See Also:

`Class.getMethods()` ,
`findDeclaredMethodWithMinimalParameters(java.lang.Class<?>, java.lang.String)`

dDeclaredMethodWithMinimalParameters

```
@Nullable
public static Method findDeclaredMethodWithMinimalParameters(Class<?> clazz,
                                                             String methodName)
                                                      throws IllegalArgumentException
```

Find a method with the given method name and minimal parameters (best case: none), declared on the given class or one of its superclasses. Will return a public, protected, package access, or private method.

Checks `Class.getDeclaredMethods`, cascading upwards to all superclasses.

Parameters:

`clazz` - the class to check

`methodName` - the name of the method to find

Returns:

the Method object, or `null` if not found

Throws:

`IllegalArgumentException` - if methods of the given name were found but could not be resolved to a unique method with minimal parameters

See Also:

`Class.getDeclaredMethods()`

findMethodWithMinimalParameters

```
@Nullable
public static Method findMethodWithMinimalParameters(Method[] methods,
                                                     String methodName)
                                              throws IllegalArgumentException
```

Find a method with the given method name and minimal parameters (best case: none) in the given list of methods.

Parameters:

`methods` - the methods to check

`methodName` - the name of the method to find

Returns:

the Method object, or `null` if not found

Throws:

`IllegalArgumentException` - if methods of the given name were found but could not be resolved to a unique method with minimal parameters

resolveSignature

```
@Nullable
public static Method resolveSignature(String signature,
                                      Class<?> clazz)
```

Parse a method signature in the form `methodName[([arg_list])]`, where `arg_list` is an optional, comma-separated list of fully-qualified type names, and attempts to resolve that signature against the supplied `Class`.

When not supplying an argument list (`methodName`) the method whose name matches and has the least number of parameters will be returned. When supplying an argument type list, only the method whose name and argument types match will be returned.

Note then that `methodName` and `methodName()` are **not** resolved in the same way. The signature `methodName` means the method called `methodName` with the least number of arguments, whereas `methodName()` means the method called `methodName` with exactly 0 arguments.

If no method can be found, then `null` is returned.

Parameters:

`signature` - the method signature as String representation

`clazz` - the class to resolve the method signature against

Returns:

the resolved Method

See Also:

`findMethod(java.lang.Class<?>, java.lang.String, java.lang.Class<?>...)`, `findMethodWithMinimalParameters(java.lang.Class<?>, java.lang.String)`

getPropertyDescriptors

```
public static PropertyDescriptor[] getPropertyDescriptors(Class<?> clazz)
                                                   throws BeansException
```

Retrieve the JavaBeans `PropertyDescriptor`s of a given class.

Parameters:

`clazz` - the Class to retrieve the PropertyDescriptors for

Returns:

an array of `PropertyDescriptors` for the given class

Throws:

`BeansException` - if PropertyDescriptor look fails

getPropertyDescriptor

```
@Nullable
public static PropertyDescriptor getPropertyDescriptor(Class<?> clazz,
                                                       String propertyName)
                                                throws BeansException
```

Retrieve the JavaBeans `PropertyDescriptors` for the given property.

Parameters:

`lazz` - the Class to retrieve the PropertyDescriptor for

`opertyName` - the name of the property

Returns:

the corresponding PropertyDescriptor, or `null` if none

Throws:

`BeansException` - if PropertyDescriptor lookup fails

findPropertyForMethod

```
@Nullable
public static PropertyDescriptor findPropertyForMethod(Method method)
                                    throws BeansException
```

Find a JavaBeans `PropertyDescriptor` for the given method, with the method either being the read method or the write method for that bean property.

Parameters:

`method` - the method to find a corresponding PropertyDescriptor for, introspecting its declaring class

Returns:

the corresponding PropertyDescriptor, or `null` if none

Throws:

`BeansException` - if PropertyDescriptor lookup fails

findPropertyForMethod

```
@Nullable
public static PropertyDescriptor findPropertyForMethod(Method method,
                                          Class<?> clazz)
                                    throws BeansException
```

Find a JavaBeans `PropertyDescriptor` for the given method, with the method either being the read method or the write method for that bean property.

Parameters:

`method` - the method to find a corresponding PropertyDescriptor for

`clazz` - the (most specific) class to introspect for descriptors

Returns:

the corresponding PropertyDescriptor, or `null` if none

Throws:

`BeansException` - if PropertyDescriptor lookup fails

Since:

3.2.13

findEditorByConvention

```
@Nullable
ublic static PropertyEditor findEditorByConvention(@Nullable
                                          Class<?> targetType)
```

Find a JavaBeans PropertyEditor following the 'Editor' suffix convention (for example, "mypackage.MyDomainClass" → "mypackage.MyDomainClassEditor").

Compatible to the standard JavaBeans convention as implemented by `PropertyEditorManager` but isolated from the latter's registered default editors for primitive types.

Parameters:

`targetType` - the type to find an editor for

Returns:

the corresponding editor, or `null` if none found

findPropertyType

```
public static Class<?> findPropertyType(String propertyName,
                                        @Nullable
                                        Class<?>... beanClasses)
```

Determine the bean property type for the given property from the given classes/interfaces, if possible.

Parameters:

`propertyName` - the name of the bean property

`beanClasses` - the classes to check against

Returns:

the property type, or `Object.class` as fallback

hasUniqueWriteMethod

```
public static boolean hasUniqueWriteMethod(PropertyDescriptor pd)
```

Determine whether the specified property has a unique write method, i.e. is writable but does not declare overloaded setter methods.

Parameters:

`pd` - the PropertyDescriptor for the property

Returns:

`true` if writable and unique, `false` otherwise

Since:

6.1.4

getWriteMethodParameter

```
public static MethodParameter getWriteMethodParameter(PropertyDescriptor pd)
```

Obtain a new MethodParameter object for the write method of the specified property.

Parameters:

`d` - the PropertyDescriptor for the property

turns:

a corresponding MethodParameter object

getParameterNames

```
public static String[] getParameterNames(Constructor<?> ctor)
```

Determine required parameter names for the given constructor, considering the JavaBeans `ConstructorProperties` annotation as well as Spring's `DefaultParameterNameDiscoverer`.

Parameters:

`ctor` - the constructor to find parameter names for

Returns:

the parameter names (matching the constructor's parameter count)

Throws:

`IllegalStateException` - if the parameter names are not resolvable

Since:

5.3

See Also:

`ConstructorProperties`, `DefaultParameterNameDiscoverer`

isSimpleProperty

```
public static boolean isSimpleProperty(Class<?> type)
```

Check if the given type represents a "simple" property: a simple value type or an array of simple value types.

See `isSimpleValueType(Class)` for the definition of *simple value type*.

Used to determine properties to check for a "simple" dependency-check.

Parameters:

`type` - the type to check

Returns:

whether the given type represents a "simple" property

See Also:

`AbstractBeanDefinition.DEPENDENCY_CHECK_SIMPLE`, `AbstractAutowireCapableBeanFactory.checkDependencies(java.lang.String, org.springframework.beans.factory.support.AbstractBeanDefinition, java.beans.PropertyDescriptor[], org.springframework.beans.PropertyValues)`, `isSimpleValueType(Class)`

isSimpleValueType

```
public static boolean isSimpleValueType(Class<?> type)
```

eck if the given type represents a "simple" value type for bean property and data binding purposes: primitive or primitive wrapper, an `Enum`, a `String` or other `CharSequence`, a `Number`, a `Date`, a

`Temporal`, a `UUID`, a `URI`, a `URL`, a `Locale`, or a `Class`.

`Void` and `void` are not considered simple value types.

As of 6.1, this method delegates to `ClassUtils.isSimpleValueType(java.lang.Class<?>)` as-is but could potentially add further rules for bean property purposes.

Parameters:

`type` - the type to check

Returns:

whether the given type represents a "simple" value type

See Also:

`isSimpleProperty(Class)`,
`ClassUtils.isSimpleValueType(Class)`

copyProperties

```
public static  void  copyProperties(Object   source,
                                    Object   target)
                            throws BeansException
```

Copy the property values of the given source bean into the target bean.

Note: The source and target classes do not have to match or even be derived from each other, as long as the properties match. Any bean properties that the source bean exposes but the target bean does not will silently be ignored.

This is just a convenience method. For more complex transfer needs, consider using a full `BeanWrapper`.

As of Spring Framework 5.3, this method honors generic type information when matching properties in the source and target objects.

The following table provides a non-exhaustive set of examples of source and target property types that can be copied as well as source and target property types that cannot be copied.

source property type	target property type	copy supported
`Integer`	`Integer`	yes
`Integer`	`Number`	yes
`List<Integer>`	`List<Integer>`	yes
`List<?>`	`List<?>`	yes
`List<Integer>`	`List<?>`	yes
`List<Integer>`	`List<? extends Number>`	yes
`String`	`Integer`	no
`Number`	`Integer`	no
`List<Integer>`	`List<Long>`	no
`List<Integer>`	`List<Number>`	no

rameters:

`ource` - the source bean

`target` - the target bean

Throws:

`BeansException` - if the copying failed

See Also:

`BeanWrapper`

copyProperties

```
public static  void  copyProperties(Object   source,
                                    Object   target,
                                    Class <?>  editable)
                             throws BeansException
```

Copy the property values of the given source bean into the given target bean, only setting properties defined in the given "editable" class (or interface).

Note: The source and target classes do not have to match or even be derived from each other, as long as the properties match. Any bean properties that the source bean exposes but the target bean does not will silently be ignored.

This is just a convenience method. For more complex transfer needs, consider using a full `BeanWrapper`.

As of Spring Framework 5.3, this method honors generic type information when matching properties in the source and target objects. See the documentation for `copyProperties(Object, Object)` for details.

Parameters:

`source` - the source bean

`target` - the target bean

`editable` - the class (or interface) to restrict property setting to

Throws:

`BeansException` - if the copying failed

See Also:

`BeanWrapper`

copyProperties

```
public static  void  copyProperties(Object   source,
                                    Object   target,
                                    String ...  ignoreProperties)
                             throws BeansException
```

Copy the property values of the given source bean into the given target bean, ignoring the given "ignoreProperties".

Note: The source and target classes do not have to match or even be derived from each other, as long the properties match. Any bean properties that the source bean exposes but the target bean does t will silently be ignored.

This is just a convenience method. For more complex transfer needs, consider using a full `BeanWrapper`.

As of Spring Framework 5.3, this method honors generic type information when matching properties in the source and target objects. See the documentation for `copyProperties(Object, Object)` for details.

Parameters:

`source` - the source bean

`target` - the target bean

`ignoreProperties` - array of property names to ignore

Throws:

`BeansException` - if the copying failed

See Also:

`BeanWrapper`