Lista Annotations Serialization

@JsonAnyGetter
@JsonGetter
@JsonPropertyOrder
@JsonRawValue
@JsonValue
@JsonRootName
@JsonSerialize

@JsonAnyGetter

Utilizzo:

consente ad un metodo getter di restituire un json a partire da una map

Esempio:

```
import java.util.HashMap;
import java.util.Map;
import com.fasterxml.jackson.annotation.JsonAnyGetter;

class Student {
   private Map<String, String> properties;
   public Student(){
      properties = new HashMap<>();
   }
   @JsonAnyGetter
   public Map<String, String> getProperties(){
      return properties;
   }
   public void add(String property, String value){
      properties.put(property, value);
   }
}
```

```
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {

       public static void main(String args[]){
             ObjectMapper mapper = new ObjectMapper();
             try{
                Student student = new Student();
                student.add("Name", "Mark");
                student.add("RollNo", "1");
                String jsonString = mapper
                   .writerWithDefaultPrettyPrinter()
                   .writeValueAsString(student);
                System.out.println(jsonString);
             }
             catch (IOException e) {
                e.printStackTrace();
             }
          }

}
```

Output:

```
{
 "RollNo" : "1",
 "Name" : "Mark"
}
```

@JsonGetter

Utilizzo:

consente di creare un json a partire dagli attributi della classe

Esempio:

```
import com.fasterxml.jackson.annotation.JsonGetter;
class Student {
         private String name;
         private int rollNo;
         public Student(String name, int rollNo){
            this.name = name;
            this.rollNo = rollNo;
         }
         @JsonGetter
         public String getStudentName(){
            return name;
         }
         public int getRollNo(){
            return rollNo;
         }
      }
```

```
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student("Mark", 1);
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "rollNo" : 1,
  "studentName" : "Mark"
}
```

@JsonProprietyOrder

Utilizzo:

permette di organizzare la sequenza degli attributi all'interno del json

Esempio:

```
import com.fasterxml.jackson.annotation.JsonPropertyOrder;

@JsonPropertyOrder({ "rollNo", "name" })
class Student {
   private String name;
   private int rollNo;
   public Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }
   public String getName(){
      return name;
   }
   public int getRollNo(){
      return rollNo;
   }
}
```

```
import java.io.IOException;

public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student("Mark", 1);
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "name" : "Mark",
  "rollNo" : 1
}
```

@JsonRawValue

Utilizzo:

consente di serializzare un testo senza sfuggire o senza alcuna decorazione

Esempio:

```
import com.fasterxml.jackson.annotation.JsonRawValue;
class Student {
         private String name;
         private int rollNo;
         private String json;
         public Student(String name, int rollNo, String json){
            this.name = name;
            this.rollNo = rollNo;
            this.json = json;
         }
         public String getName(){
            return name;
         }
         public int getRollNo(){
            return rollNo;
         }
         public String getJson(){
            return json;
         }
      }
```

```
import java.io.IOException;
public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student("Mark", 1, "{\"attr\":false}");
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "name" : "Mark",
  "rollNo" : 1,
  "json" : {"attr":false}
}
```

@JsonValue

Utilizzo:

consente di serializzare un intero oggetto utilizzando il suo unico metodo (toString)

Esempio:

```
import com.fasterxml.jackson.annotation.JsonValue;
class Student {
         private String name;
         private int rollNo;
         public Student(String name, int rollNo){
            this.name = name;
            this.rollNo = rollNo;
         }
         public String getName(){
            return name;
         }
         public int getRollNo(){
            return rollNo;
         }
         @JsonValue
         public String toString(){
            return "{ name : " + name + " }";
         }
      }
```

```
import java.io.IOException;
public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student("Mark", 1);
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
"{ name : Mark, rollNo : 1 }"
```

@JsonRootName

Utilizzo:

consente di avere un nodo radice specifico su json

Esempio:

```
import com.fasterxml.jackson.annotation.JsonRootName;
@JsonRootName(value = "student")
class Student {
   private String name;
   private int rollNo;
   public Student(String name, int rollNo){
      this.name = name;
      this.rollNo = rollNo;
   }
   public String getName(){
      return name;
   }
   public int getRollNo(){
      return rollNo;
   }
}
```

```
import java.io.IOException;
public class demo {
       public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student("Mark", 1);
               mapper.enable(SerializationFeature.WRAP_ROOT_VALUE);
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "student" : {
    "name" : "Mark",
    "rollNo" : 1
  }
}
```

@JsonSerialize

Utilizzo:

@JsonSerialize viene utilizzato per specificare il serializzatore personalizzato per eseguire il marshalling dell'oggetto json. ?

Esempio:

```
import java.util.Date;
public class Student {
       private String name;
          private int rollNo;
          @JsonSerialize(using = CustomDateSerializer.class)
          private Date dateOfBirth;
          public Student(String name, int rollNo, Date dob){
             this.name = name;
             this.rollNo = rollNo;
             this.dateOfBirth = dob;
          }
          public String getName(){
             return name;
          }
          public int getRollNo(){
             return rollNo;
          }
          public Date getDateOfBirth(){
             return dateOfBirth;
          }
}
```

```
import java.io.IOException;
public class CustomDateSerializer extends StdSerializer<Date>{
       private static final long serialVersionUID = 1L;
          private static SimpleDateFormat formatter = new
SimpleDateFormat("dd-MM-yyyy");
          public CustomDateSerializer() {
             this(null);
          }
          public CustomDateSerializer(Class<Date> t) {
             super(t);
          }
          @Override
          public void serialize(Date value,
             JsonGenerator generator, SerializerProvider arg2) throws IOException
{
             generator.writeString(formatter.format(value));
          }
}
```

```
import java.io.IOException;
public class demo {
       public static void main(String args[]) throws ParseException {
             ObjectMapper mapper = new ObjectMapper();
             SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
             try {
                Student student = new Student("Mark", 1,
dateFormat.parse("20-11-1984"));
                String jsonString = mapper
                   .writerWithDefaultPrettyPrinter()
                   .writeValueAsString(student);
                System.out.println(jsonString);
             }
             catch (IOException e) {
                e.printStackTrace();
             }
          }
}
```

<u>Output</u>:

```
{
  "name" : "Mark",
  "rollNo" : 1,
  "dateOfBirth" : "20-11-1984"
}
```

Lista Annotations Deserialization
@JsonCreator
@JsonInjection
@JsonAnySetter
@JsonSetter
@JsonDeserialize
@JsonEnumDefaultValue

@JsonCreator

Utilizzo:

Permette di trasferire i valori ottenuti dal json negli attributi di un'istanza di un oggetto

Esempio:

```java
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Student {
      public String name;
         public int rollNo;

         @JsonCreator
         public Student(@JsonProperty("theName") String name,
@JsonProperty("id") int rollNo){
            this.name = name;
            this.rollNo = rollNo;
         }
}
```

```java
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws ParseException{
            String json = "{\"id\":1,\"theName\":\"Mark\"}";
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = mapper
                  .readerFor(Student.class)
                  .readValue(json);
               System.out.println(student.rollNo +", " + student.name);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
1, Mark
```

@JsonInject

Utilizzo:

@JacksonInject viene utilizzato quando un valore di proprietà deve essere iniettato invece di essere analizzato dall'input Json. Nell'esempio seguente, stiamo inserendo un valore nell'oggetto invece di analizzare dal Json.

Esempio:

```
import com.fasterxml.jackson.annotation.JacksonInject;

public class Student {
      public String name;
         @JacksonInject
         public int rollNo;
}
```

```
import java.io.IOException;
import java.text.ParseException;
import com.fasterxml.jackson.databind.InjectableValues;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws ParseException{
            String json = "{\"name\":\"Mark\"}";
            InjectableValues injectableValues = new InjectableValues.Std()
               .addValue(int.class, 1);

            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = mapper
                  .reader(injectableValues)
                  .forType(Student.class)
                  .readValue(json);
               System.out.println(student.rollNo +", " + student.name);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
1, Mark
```

@JsonAnySetter

Utilizzo:

@JsonAnySetter consente a un metodo setter di utilizzare Map che viene quindi utilizzato per deserializzare le proprietà aggiuntive di JSON in modo simile alle altre proprietà.

Esempio:

```java
import java.util.HashMap;
import java.util.Map;
import com.fasterxml.jackson.annotation.JsonAnySetter;
public class Student {
      private Map<String, String> properties;
         public Student(){
            properties = new HashMap<>();
         }
         public Map<String, String> getProperties(){
            return properties;
         }
         @JsonAnySetter
         public void add(String property, String value){
            properties.put(property, value);
         }
}
```

```java
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            String jsonString = "{\"RollNo\" : \"1\",\"Name\" : \"Mark\"}";
            try {
               Student student =
mapper.readerFor(Student.class).readValue(jsonString);
               System.out.println(student.getProperties().get("Name"));
               System.out.println(student.getProperties().get("RollNo"));
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
Mark
1
```

@JsonSetter

Utilizzo:

@JsonSetter consente di contrassegnare un metodo specifico come metodo setter

Esempio:

```
import com.fasterxml.jackson.annotation.JsonSetter;

public class Student {
      public int rollNo;
         public String name;
         @JsonSetter("name")
         public void setTheName(String name) {
            this.name = name;
         }
}
```

```
import java.io.IOException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            String jsonString = "{\"rollNo\":1,\"name\":\"Marks\"}";

            try {
               Student student =
mapper.readerFor(Student.class).readValue(jsonString);
               System.out.println(student.name);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
Marks
```

@JsonDeserialize

Utilizzo:

@JsonDeserialize viene utilizzato per specificare un deserializzatore personalizzato per annullare il marshalling dell'oggetto json

Esempio:

```java
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;

import com.fasterxml.jackson.core.JsonParser;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.DeserializationContext;
import com.fasterxml.jackson.databind.deser.std.StdDeserializer;

public class CustomDateDeserializer extends StdDeserializer<Date> {
         private static final long serialVersionUID = 1L;
         private static SimpleDateFormat formatter = new
SimpleDateFormat("dd-MM-yyyy");
         public CustomDateDeserializer() {
            this(null);
         }
         public CustomDateDeserializer(Class<Date> t) {
            super(t);
         }
         @Override
         public Date deserialize(JsonParser parser, DeserializationContext
context)
            throws IOException, JsonProcessingException {

            String date = parser.getText();
            try {
               return formatter.parse(date);
            }
            catch (ParseException e) {
               e.printStackTrace();
            }
            return null;
         }
}
```

```java
import java.util.Date;
import com.fasterxml.jackson.databind.annotation.JsonDeserialize;
public class Student {
      public String name;
         @JsonDeserialize(using = CustomDateDeserializer.class)
         public Date dateOfBirth;
}
```

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws ParseException{
            ObjectMapper mapper = new ObjectMapper();
            String jsonString =
"{\"name\":\"Mark\",\"dateOfBirth\":\"20-12-1984\"}";
            try {
               Student student = mapper
                  .readerFor(Student.class)
                  .readValue(jsonString);
               System.out.println(student.dateOfBirth);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

<u>Output:</u>

```
Thu Dec 20 00:00:00 CET 1984
```

@JsonEnumDefaultValue

Utilizzo:

@JsonEnumDefaultValue viene utilizzato per deserializzare un valore di enumerazione sconosciuto utilizzando un valore predefinito

Esempio:

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.annotation.JsonEnumDefaultValue;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws ParseException{
      ObjectMapper mapper = new ObjectMapper();
      mapper.enable(DeserializationFeature
      .READ_UNKNOWN_ENUM_VALUES_USING_DEFAULT_VALUE);

            String jsonString = "\"abc\"";
            try {
               LETTERS value = mapper.readValue(jsonString, LETTERS.class);
               System.out.println(value);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
      }

      enum LETTERS {
         A, B, @JsonEnumDefaultValue UNKNOWN
      }
```

Output:

```
UNKNOWN
```

Lista Annotations Inclusion
@JsonIgnoreProperties

@JsonIgnore
@JsonIgnoreType
@JsonInclude
@JsonAutodetect

@JsonIgnoreProperties

Utilizzo:

@JsonIgnoreProperties viene utilizzato a livello di classe per contrassegnare una proprietà o un elenco di proprietà da ignorare

Esempio:

```
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@JsonIgnoreProperties({ "id", "systemId" })
class Student {
   public int id;
   public String systemId;
   public int rollNo;
   public String name;

   Student(int id, int rollNo, String systemId, String name){
      this.id = id;
      this.systemId = systemId;
      this.rollNo = rollNo;
      this.name = name;
   }
}
```

```
import java.io.IOException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) {
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student(1,11,"1ab","Mark");
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
   "rollNo" : 11,
   "name" : "Mark"
}
```

@JsonIgnore

Utilizzo:

@JsonIgnore viene utilizzato a livello di campo per contrassegnare una proprietà o un elenco di proprietà da ignorare.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonIgnore;
public class Student {
      public int id;
         @JsonIgnore
         public String systemId;
         public int rollNo;
         public String name;

         Student(int id, int rollNo, String systemId, String name){
            this.id = id;
            this.systemId = systemId;
            this.rollNo = rollNo;
            this.name = name;
         }
}
```

```
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try{
               Student student = new Student(1,11,"1ab","Mark");
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "id" : 1,
  "rollNo" : 11,
  "name" : "Mark"
}
```

@JsonIgnoreType

Utilizzo:

@JsonIgnoreType viene utilizzato per contrassegnare una proprietà di tipo speciale da ignorare

Esempio:

```
import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreType;
class Student {
          public int id;
          @JsonIgnore
          public String systemId;
          public int rollNo;
          public Name nameObj;

          Student(int id, int rollNo, String systemId, String name){
             this.id = id;
             this.systemId = systemId;
             this.rollNo = rollNo;
             nameObj = new Name(name);
          }

          @JsonIgnoreType
          class Name {
             public String name;
             Name(String name){
                this.name = name;
             }
          }
       }
```

```
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
       public static void main(String args[]){
             ObjectMapper mapper = new ObjectMapper();
             try {
                Student student = new Student(1,11,"1ab","Mark");
                String jsonString = mapper
                   .writerWithDefaultPrettyPrinter()
                   .writeValueAsString(student);
                System.out.println(jsonString);
             }
             catch (IOException e) {
                e.printStackTrace();
             }
          }
}
```

Output:

```
{
  "id" : 1,
  "rollNo" : 11
}
```

@JsonInclude

Utilizzo:

@JsonInclude viene utilizzato per escludere proprietà con valori null / vuoti o predefiniti

Esempio:

```
import com.fasterxml.jackson.annotation.JsonInclude;

@JsonInclude(JsonInclude.Include.NON_NULL)
class Student {
   public int id;
   public String name;

   Student(int id,String name){
      this.id = id;
      this.name = name;
   }
}
```

```
import java.io.IOException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try {
               Student student = new Student(1,null);
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "id" : 1
}
```

@JsonAutoDetect

Utilizzo:

@JsonAutoDetect può essere utilizzato per includere proprietà che non sarebbero altrimenti accessibili

Esempio:

```
import com.fasterxml.jackson.annotation.JsonAutoDetect;

@JsonAutoDetect(fieldVisibility = JsonAutoDetect.Visibility.ANY)
class Student {
   private int id;
   private String name;

   Student(int id,String name) {
      this.id = id;
      this.name = name;
   }
}
```

```
import java.io.IOException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]){
            ObjectMapper mapper = new ObjectMapper();
            try{
               Student student = new Student(1,"Mark");
               String jsonString = mapper
                  .writerWithDefaultPrettyPrinter()
                  .writeValueAsString(student);
               System.out.println(jsonString);
            }
            catch (IOException e) {
               e.printStackTrace();
            }
         }
}
```

Output:

```
{
  "id" : 1,
  "name" : "Mark"
}
```

Lista Annotation Type Handling
@JsonTypeInfo
@JsonSubTypes
@JsonTypeName

@JsonTypeInfo

Utilizzo:

@JsonTypeInfo viene utilizzato per indicare i dettagli delle informazioni sul tipo che devono essere incluse nella serializzazione e nella deserializzazione

Esempio:

```
import java.io.IOException;
import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonTypeInfo.As;
import com.fasterxml.jackson.annotation.JsonTypeName;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws IOException {
            Shape shape = new demo.Circle("CustomCircle", 1);
            String result = new ObjectMapper()
               .writerWithDefaultPrettyPrinter()
               .writeValueAsString(shape);
            System.out.println(result);
            String json = "{\"name\":\"CustomCircle\",\"radius\":1.0,
\"type\":\"circle\"}";
            Circle circle = new
ObjectMapper().readerFor(Shape.class).readValue(json);
            System.out.println(circle.name);
         }
         @JsonTypeInfo(use = JsonTypeInfo.Id.NAME,
            include = As.PROPERTY, property = "type") @JsonSubTypes({

            @JsonSubTypes.Type(value = Square.class, name = "square"),
            @JsonSubTypes.Type(value = Circle.class, name = "circle")
         })
         static class Shape {
            public String name;
            Shape(String name){
               this.name = name;
            }
         }
         @JsonTypeName("square")
         static class Square extends Shape {
            public double length;
            Square(){
               this(null,0.0);
            }
            Square(String name, double length){
               super(name);
               this.length = length;
            }
         }
```

```
          @JsonTypeName("circle")
          static class Circle extends Shape {
             public double radius;
             Circle(){
                this(null,0.0);
             }
             Circle(String name, double radius) {
                super(name);
                this.radius = radius;
             }
          }
       }
```

Output:

```
{
   "type" : "circle",
   "name" : "CustomCircle",
   "radius" : 1.0
}
CustomCircle
```

@JsonSubTypes

Utilizzo:

@JsonSubTypes is used to indicate subtypes of types annotated.

Esempio:

```
import java.io.IOException;

import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonTypeInfo.As;
import com.fasterxml.jackson.annotation.JsonTypeName;
import com.fasterxml.databind.ObjectMapper;

public class demo {
       public static void main(String args[]) throws IOException{
            Shape shape = new demo.Circle("CustomCircle", 1);
            String result = new ObjectMapper()
               .writerWithDefaultPrettyPrinter()
               .writeValueAsString(shape);
            System.out.println(result);
            String json = "{\"name\":\"CustomCircle\",\"radius\":1.0,
\"type\":\"circle\"}";
            Circle circle = new
ObjectMapper().readerFor(Shape.class).readValue(json);
            System.out.println(circle.name);
         }
         @JsonTypeInfo(use = JsonTypeInfo.Id.NAME,
            include = As.PROPERTY, property = "type") @JsonSubTypes({

            @JsonSubTypes.Type(value = Square.class, name = "square"),
            @JsonSubTypes.Type(value = Circle.class, name = "circle")
         })
         static class Shape {
            public String name;
            Shape(String name) {
               this.name = name;
            }
         }
         @JsonTypeName("square")
         static class Square extends Shape {
            public double length;
            Square(){
               this(null,0.0);
            }
            Square(String name, double length){
               super(name);
```

```
                this.length = length;
             }
          }

          @JsonTypeName("circle")
          static class Circle extends Shape {
             public double radius;
             Circle(){
                this(null,0.0);
             }
             Circle(String name, double radius){
                super(name);
                this.radius = radius;
             }
          }
}
```

Output:

```
{
   "type" : "circle",
   "name" : "CustomCircle",
   "radius" : 1.0
}
CustomCircle
```

@JsonTypeName

Utilizzo:

@JsonTypeName viene utilizzato per impostare i nomi dei tipi da utilizzare per la classe annotata

Esempio:

```
import java.io.IOException;
import com.fasterxml.jackson.annotation.JsonSubTypes;
import com.fasterxml.jackson.annotation.JsonTypeInfo;
import com.fasterxml.jackson.annotation.JsonTypeInfo.As;
import com.fasterxml.jackson.annotation.JsonTypeName;
import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws IOException {
            Shape shape = new demo.Circle("CustomCircle", 1);
            String result = new ObjectMapper()
               .writerWithDefaultPrettyPrinter()
               .writeValueAsString(shape);
            System.out.println(result);
            String json = "{\"name\":\"CustomCircle\",\"radius\":1.0,
\"type\":\"circle\"}";
            Circle circle = new
ObjectMapper().readerFor(Shape.class).readValue(json);
            System.out.println(circle.name);
         }
         @JsonTypeInfo(use = JsonTypeInfo.Id.NAME,
            include = As.PROPERTY, property = "type") @JsonSubTypes({

            @JsonSubTypes.Type(value = Square.class, name = "square"),
            @JsonSubTypes.Type(value = Circle.class, name = "circle")
         })
         static class Shape {
            public String name;
            Shape(String name){
               this.name = name;
            }
         }
         @JsonTypeName("square")
         static class Square extends Shape {
            public double length;
            Square(){
               this(null,0.0);
            }
            Square(String name, double length){
               super(name);
               this.length = length;
            }
         }
```

```
          @JsonTypeName("circle")
          static class Circle extends Shape {
             public double radius;
             Circle(){
                this(null,0.0);
             }
             Circle(String name, double radius){
                super(name);
                this.radius = radius;
             }
          }
}
```

Output:

```
{
  "type" : "circle",
  "name" : "CustomCircle",
  "radius" : 1.0
}
CustomCircle
```

Lista Annotation General

@JsonProperty
@JsonFormat
@JsonUnwrapped
@JsonView
@JsonManagedReference
@JsonBackReference
@JsonIdentityInfo
@JsonFilter

@JsonProperty

Utilizzo:

@JsonProperty viene utilizzato per contrassegnare un metodo getter / setter non standard da utilizzare rispetto alla proprietà json.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonProperty;

public class Student {
      private int id;
         Student(){}
         Student(int id){
            this.id = id;
         }
         @JsonProperty("id")
         public int getTheId() {
            return id;
         }
         @JsonProperty("id")
         public void setTheId(int id) {
            this.id = id;
         }
}
```

```
import java.io.IOException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
      public static void main(String args[]) throws IOException {
            ObjectMapper mapper = new ObjectMapper();
            String json = "{\"id\" : 1}";
            Student student =
mapper.readerFor(Student.class).readValue(json);
            System.out.println(student.getTheId());
         }
}
```

Output:

```
1
```

@JsonFormat

Utilizzo:

@JsonFormat viene utilizzato per specificare il formato durante la serializzazione o la deserializzazione. Viene utilizzato principalmente con i campi Data.

Esempio:

```
import java.util.Date;
import com.fasterxml.jackson.annotation.JsonFormat;
public class Student {
      public int id;
         @JsonFormat(shape = JsonFormat.Shape.STRING, pattern =
"dd-MM-yyyy")
         public Date birthDate;
         Student(int id, Date birthDate){
            this.id = id;
            this.birthDate = birthDate;
         }
}
```

```
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
      public static void main(String args[]) throws IOException,
ParseException {
            ObjectMapper mapper = new ObjectMapper();
            SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("dd-MM-yyyy");
            Date date = simpleDateFormat.parse("20-12-1984");

            Student student = new Student(1, date);
            String jsonString = mapper
               .writerWithDefaultPrettyPrinter()
               .writeValueAsString(student);
            System.out.println(jsonString);
         }
}
```

Output:

```
{
  "id" : 1,
  "birthDate" : "19-12-1984"
}
```

@JsonUnwrapped

Utilizzo:

@JsonUnwrapped viene utilizzato per scartare i valori degli oggetti durante la serializzazione o la deserializzazione.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonUnwrapped;
public class Student {
       public int id;
          @JsonUnwrapped
          public Name name;
          Student(int id, Name name){
             this.id = id;
             this.name = name;
          }
          static class Name {
             public String first;
             public String last;
          }
}
```

```
import java.io.IOException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
        public static void main(String args[]) throws IOException,
ParseException{
             ObjectMapper mapper = new ObjectMapper();
             SimpleDateFormat simpleDateFormat = new
SimpleDateFormat("dd-MM-yyyy");
             Date date = simpleDateFormat.parse("20-12-1984");
             Student.Name name = new Student.Name();
             name.first = "Jane";
             name.last = "Doe";
             Student student = new Student(1, name);
             String jsonString = mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(student);
             System.out.println(jsonString);
          }
}
```

Output:

```
{
  "id" : 1,
  "first" : "Jane",
  "last" : "Doe"
}
```

@JsonView

Utilizzo:

@JsonView viene utilizzato per controllare i valori da serializzare o meno.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonView;
public class Student {
       @JsonView(Views.Public.class)
          public int id;
          @JsonView(Views.Public.class)
          public String name;
          @JsonView(Views.Internal.class)
          public int age;

          Student(int id, String name, int age) {
             this.id = id;
             this.name = name;
             this.age = age;
          }
}
```

```
public class Views {
       static class Public {}
          static class Internal extends Public {}
}
```

```
import java.io.IOException;
import java.text.ParseException;
import com.fasterxml.jackson.databind.ObjectMapper;
public class demo {
       public static void main(String args[]) throws IOException, ParseException
{
             ObjectMapper mapper = new ObjectMapper();
             Student student = new Student(1, "Mark", 12);
             String jsonString = mapper
                .writerWithDefaultPrettyPrinter()
                .withView(Views.Public.class)
                .writeValueAsString(student);
             System.out.println(jsonString);
          }
}
```

Output:

```
{
  "id" : 1,
  "name" : "Mark"
}
```

@JsonManagedReference

Utilizzo:

@JsonManagedReferences e @JsonBackReferences vengono utilizzati per visualizzare oggetti con relazione padre figlio. @JsonManagedReferences viene utilizzato per fare riferimento all'oggetto padre e @JsonBackReferences viene utilizzato per contrassegnare gli oggetti figlio.

Esempio:

```java
import com.fasterxml.jackson.annotation.JsonManagedReference;

public class Book {
       public int id;
          public String name;

          Book(int id, String name, Student owner){
             this.id = id;
             this.name = name;
             this.owner = owner;
          }
          @JsonManagedReference
          public Student owner;
}
```

```java
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonBackReference;

public class Student {
       public int rollNo;
          public String name;

          @JsonBackReference
          public List<Book> books;

          Student(int rollNo, String name){
             this.rollNo = rollNo;
             this.name = name;
             this.books = new ArrayList<Book>();
          }
          public void addBook(Book book){
             books.add(book);
          }
}
```

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
	public static void main(String args[]) throws IOException, ParseException
{
		ObjectMapper mapper = new ObjectMapper();
		Student student = new Student(1, "Mark");
		Book book1 = new Book(1,"Learn HTML", student);
		Book book2 = new Book(1,"Learn JAVA", student);

		student.addBook(book1);
		student.addBook(book2);

		String jsonString = mapper
		   .writerWithDefaultPrettyPrinter()
		   .writeValueAsString(book1);
		System.out.println(jsonString);
	   }
}
```

Output:

```
{
  "id" : 1,
  "name" : "Learn HTML",
  "owner" : {
    "rollNo" : 1,
    "name" : "Mark"
  }
}
```

@JsonBackReference

Utilizzo:

@JsonManagedReferences e @JsonBackReferences vengono utilizzati per visualizzare oggetti con relazione padre figlio. @JsonManagedReferences viene utilizzato per fare riferimento all'oggetto padre e @JsonBackReferences viene utilizzato per contrassegnare gli oggetti figlio.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonManagedReference;

public class Book {
       public int id;
          public String name;

          Book(int id, String name, Student owner) {
             this.id = id;
             this.name = name;
             this.owner = owner;
          }

          @JsonManagedReference
          public Student owner;
}
```

```
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonBackReference;

public class Student {
       public int rollNo;
          public String name;

          @JsonBackReference
          public List<Book> books;

          Student(int rollNo, String name){
             this.rollNo = rollNo;
             this.name = name;
             this.books = new ArrayList<Book>();
          }
          public void addBook(Book book){
             books.add(book);
          }
}
```

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
       public static void main(String args[]) throws IOException, ParseException
{
             ObjectMapper mapper = new ObjectMapper();
             Student student = new Student(1, "Mark");
             Book book1 = new Book(1,"Learn HTML", student);
             Book book2 = new Book(1,"Learn JAVA", student);

             student.addBook(book1);
             student.addBook(book2);

             String jsonString = mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(book1);
             System.out.println(jsonString);
          }
}
```

Output:

```
{
  "id" : 1,
  "name" : "Learn HTML",
  "owner" : {
    "rollNo" : 1,
    "name" : "Mark"
  }
}
```

@JsonIdentityInfo

Utilizzo:

@JsonIdentityInfo viene utilizzato quando gli oggetti hanno una relazione padre figlio. @JsonIdentityInfo viene utilizzato per indicare che l'identità dell'oggetto verrà utilizzata durante la serializzazione / deserializzazione.

Esempio:

```java
import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
@JsonIdentityInfo(
                generator = ObjectIdGenerators.PropertyGenerator.class,
                property = "id")
             class Book{
                public int id;
                public String name;

                Book(int id, String name, Student owner){
                   this.id = id;
                   this.name = name;
                   this.owner = owner;
                }
                public Student owner;
             }
```

```java
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.annotation.JsonIdentityInfo;
import com.fasterxml.jackson.annotation.ObjectIdGenerators;
@JsonIdentityInfo(
                generator = ObjectIdGenerators.PropertyGenerator.class,
                property = "id")
             class Student {
                public int id;
                public int rollNo;
                public String name;
                public List<Book> books;

                Student(int id, int rollNo, String name){
                   this.id = id;
                   this.rollNo = rollNo;
                   this.name = name;
                   this.books = new ArrayList<Book>();
                }
                public void addBook(Book book){
                   books.add(book);
                }
             }
```

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;

public class demo {
       public static void main(String args[]) throws IOException, ParseException{
             ObjectMapper mapper = new ObjectMapper();
             Student student = new Student(1,13, "Mark");
             Book book1 = new Book(1,"Learn HTML", student);
             Book book2 = new Book(2,"Learn JAVA", student);

             student.addBook(book1);
             student.addBook(book2);

             String jsonString = mapper
                .writerWithDefaultPrettyPrinter()
                .writeValueAsString(book1);
             System.out.println(jsonString);
          }
}
```

Output:

```
{
  "id" : 1,
  "name" : "Learn HTML",
  "owner" : {
    "id" : 1,
    "rollNo" : 13,
    "name" : "Mark",
    "books" : [ 1, {
      "id" : 2,
      "name" : "Learn JAVA",
      "owner" : 1
    } ]
  }
}
```

@JsonFilter

Utilizzo:

@JsonFilter viene utilizzato per applicare il filtro durante la serializzazione / deserializzazione, ad esempio quali proprietà devono essere utilizzate o meno.

Esempio:

```
import com.fasterxml.jackson.annotation.JsonFilter;

@JsonFilter("nameFilter")
class Student {
   public int id;
   public int rollNo;
   public String name;

   Student(int id, int rollNo, String name) {
      this.id = id;
      this.rollNo = rollNo;
      this.name = name;
   }
}
```

```
import java.io.IOException;
import java.text.ParseException;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.ser.FilterProvider;
import com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter;
import com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;

public class demo {
       public static void main(String args[]) throws IOException, ParseException
{
             ObjectMapper mapper = new ObjectMapper();
             Student student = new Student(1,13, "Mark");

             FilterProvider filters = new SimpleFilterProvider() .addFilter(
                "nameFilter",
SimpleBeanPropertyFilter.filterOutAllExcept("name"));

             String jsonString = mapper.writer(filters)
                .withDefaultPrettyPrinter()
                .writeValueAsString(student);
             System.out.println(jsonString);
          }
}
```

Output:

```
{
  "name" : "Mark"
}
```

documentazione : https://www.tutorialspoint.com/jackson_annotations