

Skillbox

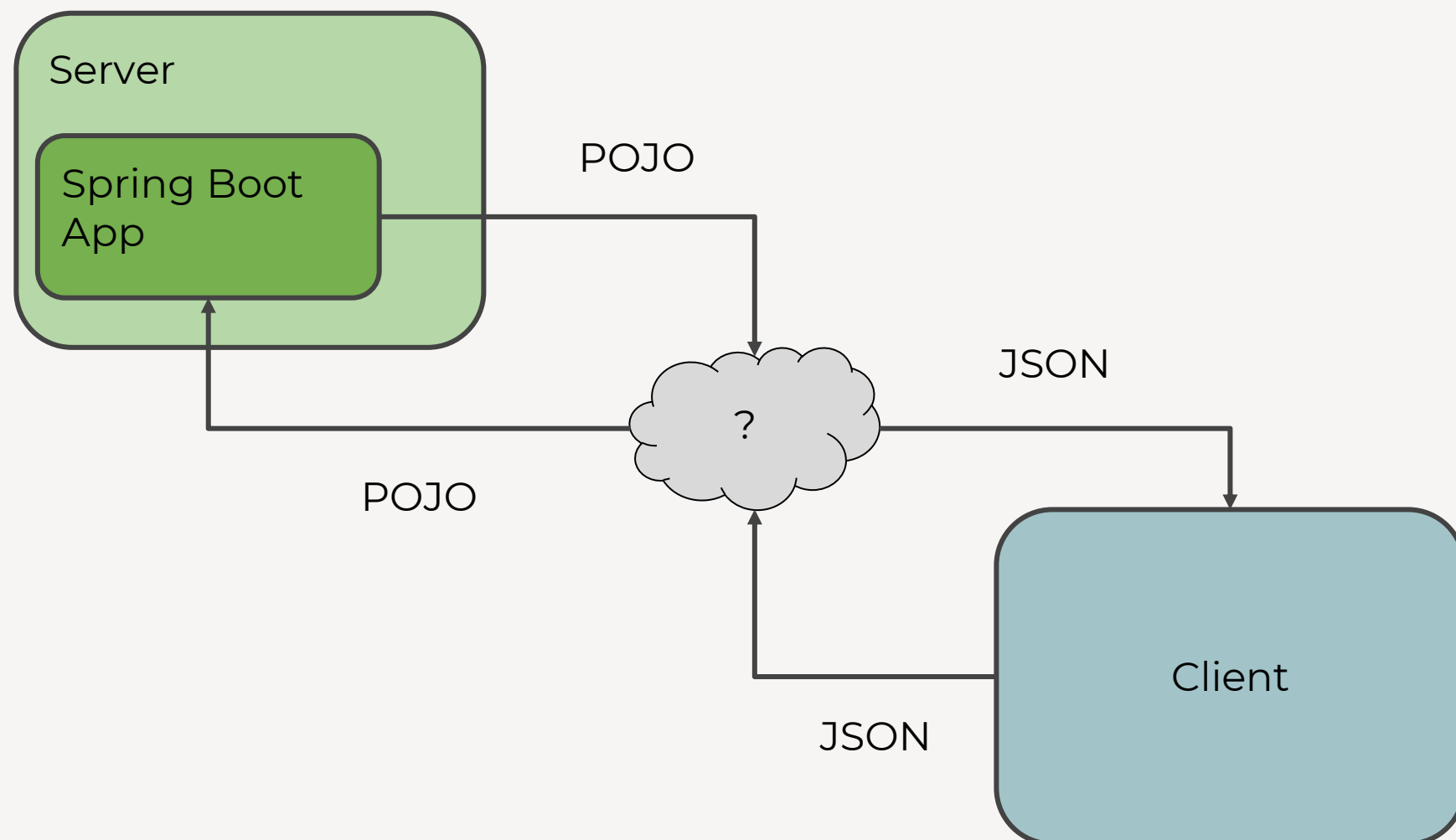
# Создание REST API

**Кирилл Кошаев**

Java-разработчик в «Газпром информ»

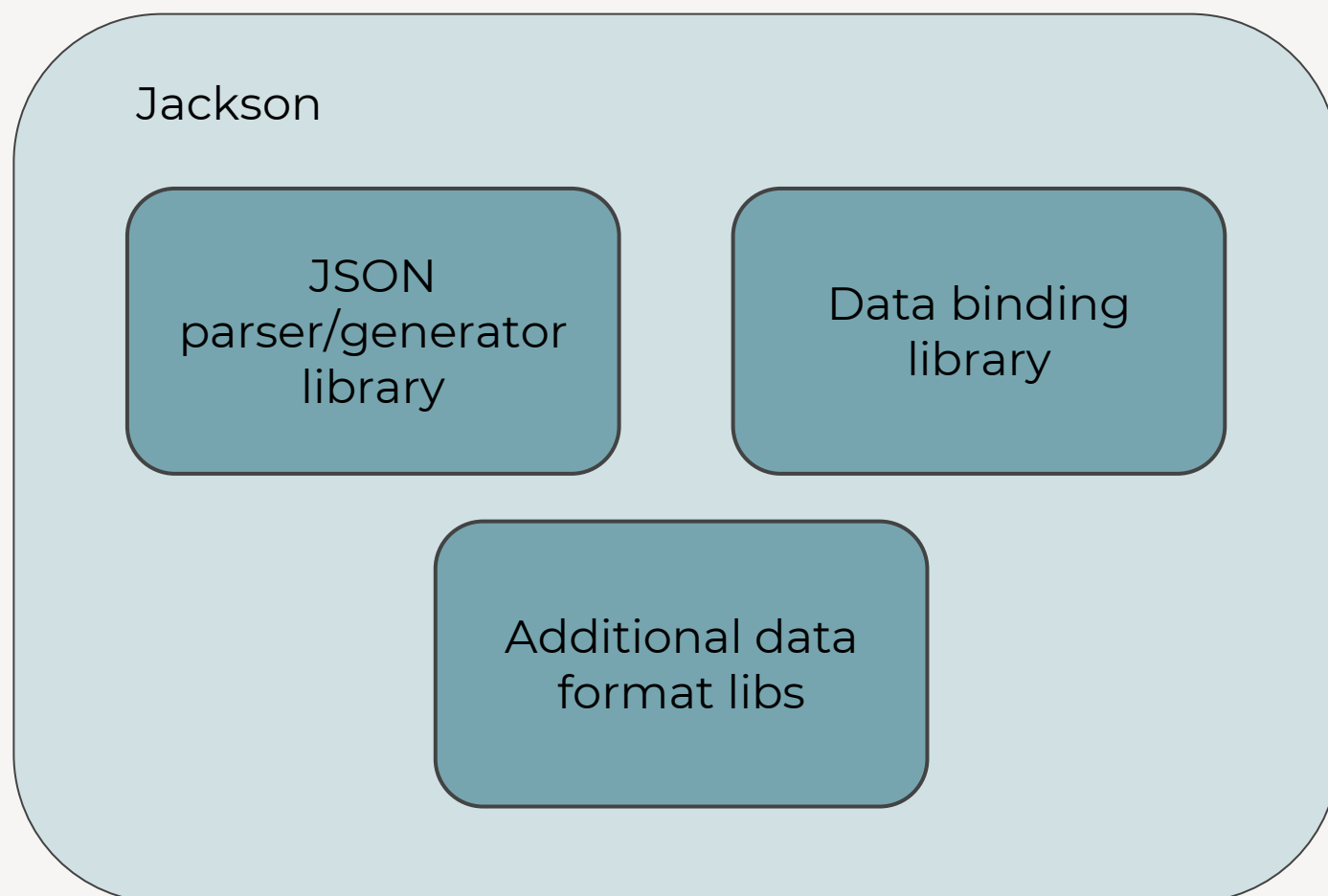
# POJO ↔ JSON

Каким образом происходит конвертация из одного формата в другой без единой строчки кода с нашей стороны?



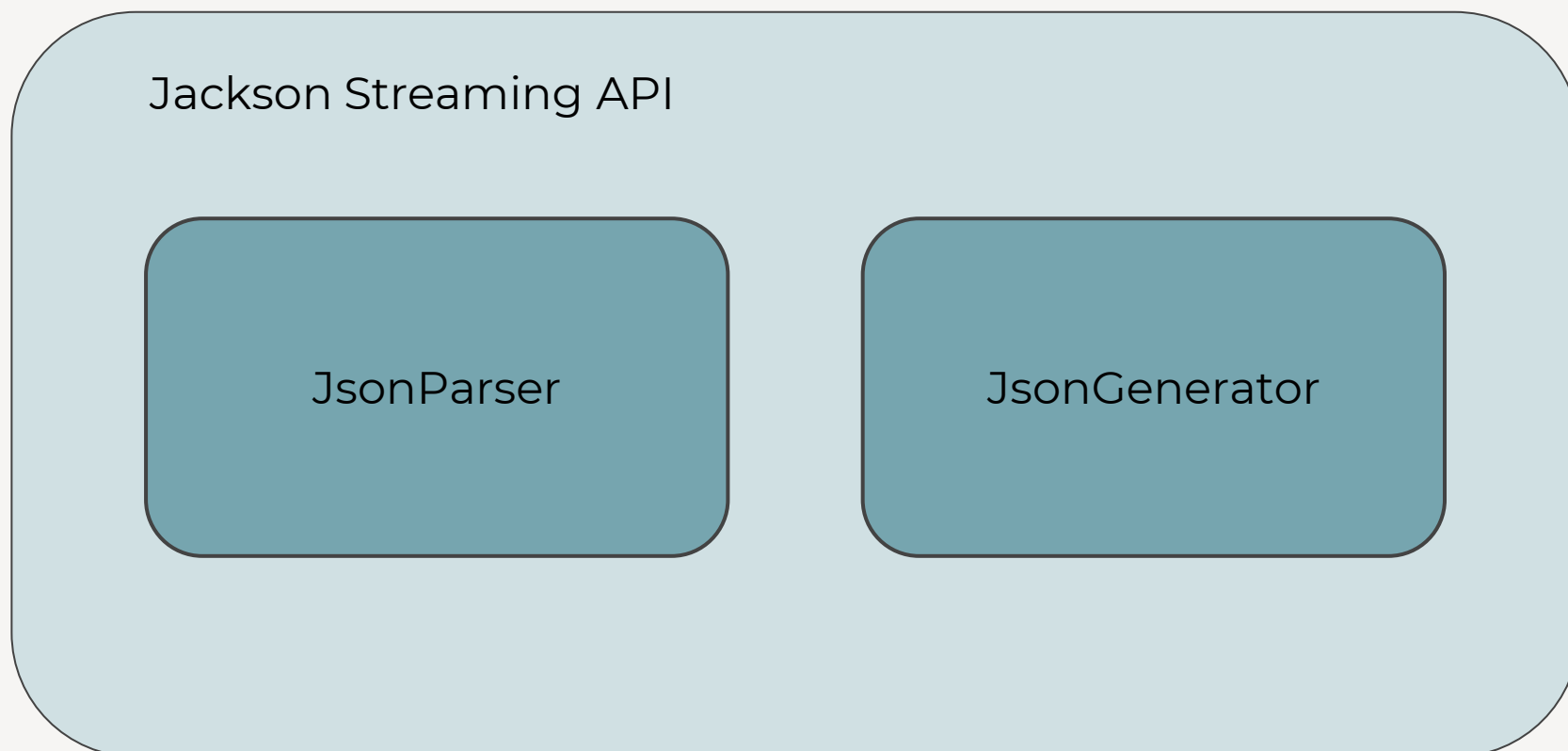
# Jackson

Jackson представляет собой набор инструментов обработки данных не только для Java, но и платформы JVM в целом



# Streaming API

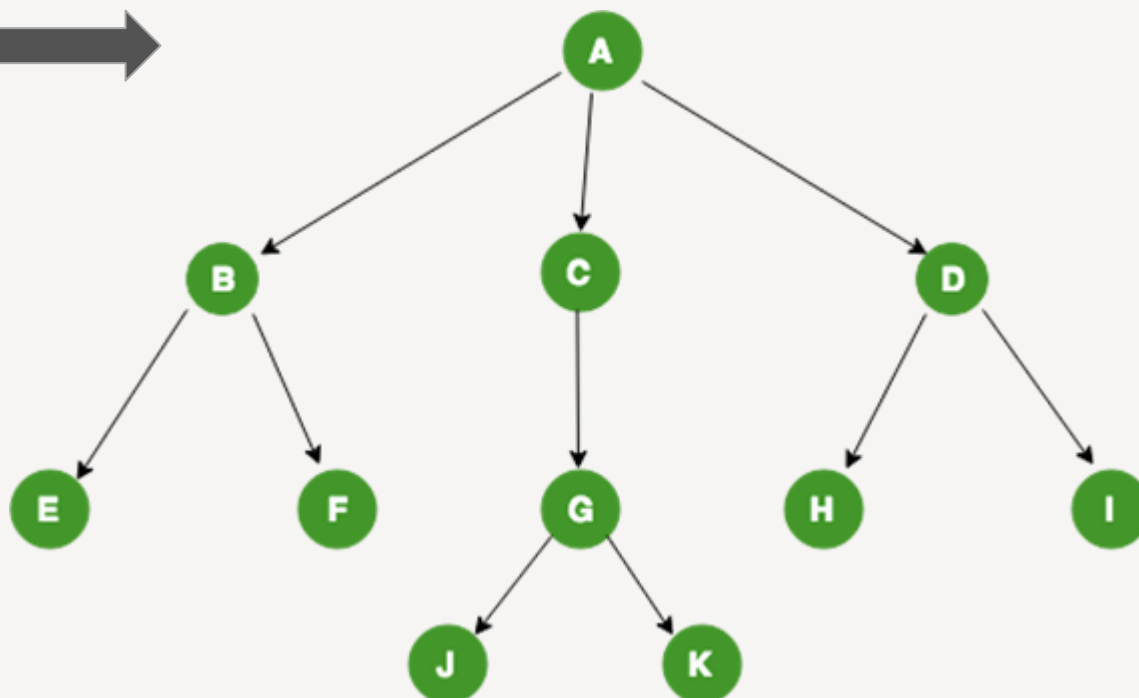
Streaming API заключается в чтении и записи содержимого JSON файлов как отдельных событий. Объект `JsonParser` читает данные, тогда как объект `JsonGenerator` записывает данные



# Tree Model

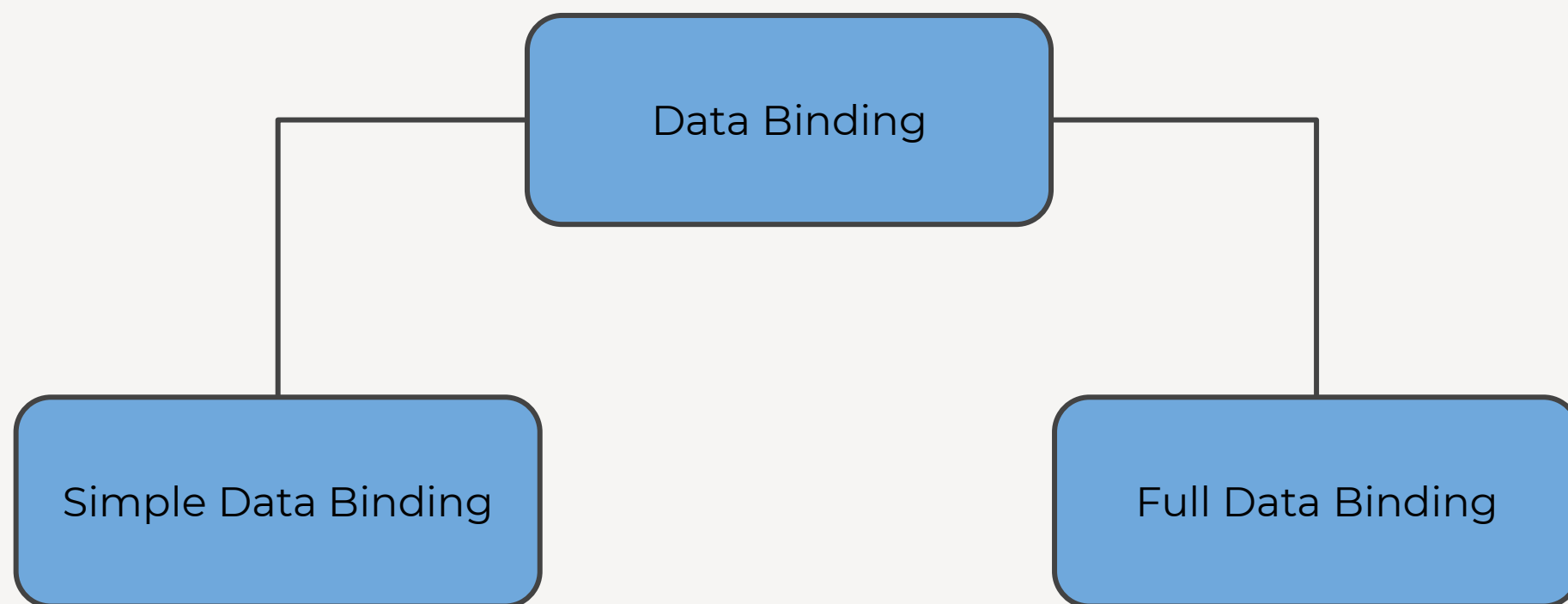
С помощью объекта ObjectMapper создается дерево узлов JsonNode и далее, вызывая различные методы дерева, можно манипулировать загруженной структурой JSON

**{JSON}**



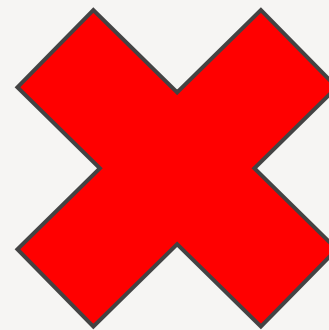
# Data Binding

Преобразует JSON в POJO — обычный java объект и обратно с помощью средства доступа к свойствам — ObjectMapper или аннотаций



# @JsonAnyGetter

```
{  
  "name": "BeanName",  
  "properties": {  
    "attr2": "value2",  
    "attr1": "value1"  
  }  
}
```



@JsonAnyGetter

```
{  
  "name": "My bean",  
  "attr2": "val2",  
  "attr1": "val1"  
}
```



@JsonAnyGetter

# @JsonGetter

```
public class MyBean {
    private int id;
    private String name;

    // свойство name существует
    @JsonGetter("name")
    public String getTheName() {
        return name;
    }

    // название метода без префикса get...
    @JsonGetter
    public String testMethod1() {
        return "testMethod1";
    }

    // название метода с префиксом get...
    @JsonGetter
    public String getTestMethod2() {
        return "getTestMethod2";
    }

    // // свойство myProperty не существует
    @JsonGetter("myProperty")
    public String testMethod3() {
        return "testMethod3";
    }
}
```



# @JsonPropertyOrder

```
@JsonPropertyOrder({ "name", "id" })  
public class MyBean {  
    public int id;  
    public String name;  
}
```

Порядок свойств в POJO

Порядок свойств  
для сериализации

```
{  
    "name": "My bean",  
    "id": 1  
}
```

JSON-результат

# @JsonRawValue

```
public class RawBean {  
    public String name;  
  
    @JsonRawValue  
    public String json;  
}
```

JSON-строка

Аннотация

```
{  
    "name": "My bean",  
    "json": {  
        "attr": false  
    }  
}
```

JSON-результат

# @JsonRootName

```
@JsonRootName(value = "user")
public class User {
    public int id;
    public String name;
}
```

POJO

Имя для корневого  
элемента

```
{
  "User": {
    "id": 1,
    "name": "John"
  }
}
```

JSON-результат, вложенный  
в корневой элемент


# @JsonCreator

```
{  
    "id":1,  
    "theName":"My bean"  
}
```



Имя поля 'name'  
в десериализуемом JSON

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




Аннотация

# @JacksonInject

```
public class BeanWithInject {  
    @JacksonInject  
    public int id;  
  
    public String name;  
}
```


POJO с @JacksonInject  
аннотацией



```
String json = "{ \"name\": \"My bean\" }";
```

```
InjectableValues inject = new InjectableValues.Std()  
    .addValue(int.class, 1);  
BeanWithInject bean = new ObjectMapper().reader(inject)  
    .forType(BeanWithInject.class)  
    .readValue(json);
```

Вставка значения inject-свойства  
на этапе десериализации



# @JsonAnySetter

# @JsonSetter

Аннотации @JsonAnySetter и @JsonSetter работают так же, как и их getter коллеги при сериализации, но в обратную сторону и при десериализации

```
public class MyBean {  
    public int id;  
    private String name;  
  
    @JsonSetter("name")  
    public void setTheName(String name) {  
        this.name = name;  
    }  
}
```

Имя метода не распознается парсером. Аннотация @JsonSetter указывает имя свойства в явном виде

```
public class SomeBean {  
    public String name;  
    private Map<String, String> properties;  
  
    @JsonAnySetter  
    public void add(String key, String value) {  
        properties.put(key, value);  
    }  
}
```

Аннотация указывает на то, что при десериализации свойства должны складываться в map-коллекцию

# @JsonAlias

При помощи аннотации @JsonAlias можно задать несколько различных имен десериализуемых свойств

```
public class AliasBean {  
    @JsonAlias({ "fName", "f_name" })  
    private String firstName;  
    private String lastName;  
}
```

# **@JsonIgnoreProperties**

## **@JsonIgnore**

## **@JsonIgnoreType**

## **@JsonProperty**

```
@JsonIgnoreProperties({"name"})
public class BeanWithIgnoreAnnotations {
    @JsonIgnore
    public int id;

    @JsonProperty("name")
    public String name;

    @JsonIgnoreType
    public static class Sub {
        public String subName;
        public String subId;
    }
}
```



**Спасибо  
за внимание!**