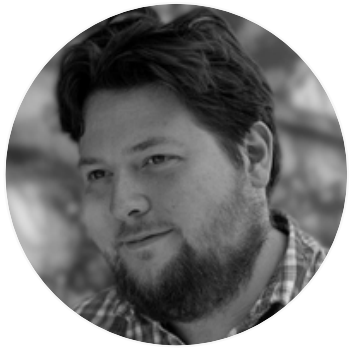
Consuming JSON with a Binding API



Richard Warburton

@richardwarburto www.monotonic.co.uk

The Binding Problem

```
public class LoanApplication
{
  private String name;

  public String getName()
  {
    return name;
  }

…
```

- Plain Old Java Object (POJO)
- Commonly used domain object

- Encodes the same structure as JSON

- Why duplicate it?

Demo

Binding the Bank Loan document

Basic API usage + conventions

Demo

Integrating the Binding API into a servlet

Approving/Denying loan application

Advanced Binding

```
@JsonCreator
public ImmutableLoanApplication(
    @JsonProperty("name") final String name,

…
```

Immutable Pojos

`@JsonCreator` – **tell Jackson to use this constructor**

`@JsonProperty` – **which property the parameters binds to**

```
@JsonProperty("name")
public String getApplicantName()
```

```
@JsonProperty("name")
private String applicantName;
```

Rename Properties

`@JsonProperty` **on a field**

Ignore Properties

```
@JsonIgnore
public void setJobs(final Map<String, Job> jobs)
{
    this.jobs = jobs;
}
```

Transform Properties

```
@JsonProperty("jobs")
public void setJobsJson(final List<Job> jobs)
{
  this.jobs = jobs.stream()
    .collect(toMap(Job::getTitle, job -> job));
}
```

Custom Deserializers

```
LocalDateDeserializer deser =
  new LocalDateDeserializer(ISO_LOCAL_DATE);

SimpleModule module = new SimpleModule()
   .addDeserializer(LocalDate.class, deser);

ObjectMapper mapper = new ObjectMapper()
    .registerModule(module);
```

Demo

Time for the Awkward Squad!

- Immutable Pojos
- Changing properties
  - Hiding/renaming
- Custom serializers

Conclusions

Evaluation of Binding API

Pros	Cons
Simple	Can be slower
Avoids duplication / boilerplate	Less flexible
Easy to map to domain objects	- eg: different API versions

Summary

Binding is the simplest and most common approach

Not always best

- Limited flexibility
- Worst performance