Serde

Florian "Florob" Zeitz

2023-05-10

Serde

- framework for **ser**ializing and **de**serializing Rust data structures
- data structures that know how to (de-)serialize themselves
- data formats that know how to (de-)serialize things
- `trait`s describing these concepts
- data model to describe the interaction

Traits

- four *main* traits
- `Serialize` and `Deserialize` for data structures
  - usually `#[derive]`d, not manually implemented
- `Serializer` and `Deserializer` for data formats

Data Model

- API by which data structures and data formats interact
- "Serde's type system"
- simplified form of Rust's type system
- 29 types
  - 14 primitives: `bool`, `char`, number types
  - string, byte char
  - option
  - seq
  - various struct, tuple and unit types

derive()

- `Serialize` and `Deserialize` are typically derived
- attributes can specify special behavior
  - enum representation
  - renaming
  - custom (de-)serialization

Basic derive()

```
#[derive(Serialize, Deserialize)]
struct Timestamp {
    seconds: u32,
    nanos: u64,
}
```

Example JSON output:

```
{"seconds":5,"nanos":64}
```

Renaming all fields

```
#[derive(Serialize, Deserialize)]
#[serde(rename_all = "camelCase")]
struct Circle {
    center_point: (u16, u16),
    outer_diameter: u16,
}
```

Example JSON output:

```
{"centerPoint":[0,0],"outerDiameter":23}
```

Renaming individual fields

```
#[derive(Serialize, Deserialize)]
struct Circle {
    #[serde(rename = "center")]
    center_point: (u16, u16),
    #[serde(rename = "diameter")]
    outer_diameter: u16,
}
```

Example JSON output:

```
{"center":[0,0],"diameter":23}
```

Renaming all variants

```
#[derive(Serialize, Deserialize)]
#[serde(rename_all = "SCREAMING_SNAKE_CASE")]
enum Alignment {
    ChaoticNeutral,
    LawfulEvil,
    NeutralGood
}
```

Example JSON output:

```
"LAWFUL_EVIL"
```

Custom serialization for fields

```
fn serialize_hex<S>(v: &u64, s: S) -> Result<S::Ok, S::Error>
where S: Serializer {
  s.serialize_str(&format!("{:016x}", v))
}

#[derive(Serialize, Deserialize)]
struct Entity {
    #[serde(serialize_with = "serialize_hex")]
    entity_id: u64,
}
```

Example JSON output:

```
{"entity_id":"c01dedfffec00ffe"}
```

Custom deserialization for fields

```
fn deserialize_hex<'a, D>(input: D) -> Result<u64, D::Error>
where D: serde::Deserializer<'a> {
    #[derive(Deserialize)]
    struct StringWrap(String);

    let s = StringWrap::deserialize(input)?;
    Ok(u64::from_str_radix(&s.0, 16).unwrap())
}

#[derive(Serialize, Deserialize)]
struct Entity {
    #[serde(deserialize_with = "deserialize_hex")]
    entity_id: u64,
}
```

Data formats

Crates for various formats are available:

serde_json JSON

postcard `#![no_std]` focused (de-)serializer

ciborium Concise Binary Object Representation (CBOR)

rmp_serde MessagePack

Usage examples

Crates per convention expose:

- `Seralizer` and `Deserializer` types
- `Error` and `Result` types
- `to_…()` functions for supported serialization targets
- `from_…()` functions for supported deserialization sources

```
serde_json::to_string(&value);
serde_json::from_str(r#"{"center":[0,0],"d":23}"#);

rmp_serde::to_vec(&value);
rmp_serde::from_slice(&[…]);
```

- https://serde.rs
- https://crates.io/keywords/serde