Interpreter Pattern

버티 프레이시(Bertie Prayc)가 우리 서버에서 중요한 데이터를 훔쳐서 비트토렌트(BitTorrent) 시스템을 통해 공유하고 있다. 버티의 가짜 계정을 만들어 그의 평판을 떨어뜨려야 한다.

페드로: 비트토렌트 시스템은 .torrent 파일에 기반하고 있어서 Bencode 인코더를 만들어야 해요.

이브: 네, 그렇다면 먼저 Bencode 포맷에 대해 알아야겠네요.

다음은 Bencode 인코딩 규칙이다.

- 2개의 데이터 타입이 지원된다.
  - 정수 N은 i<N>e로 인코딩된다. (예) 42 = i42e
  - 문자열 S는 <length>:<contents>로 인코딩된다. (예) hello = 5:hello
- 2개의 컨데이너가 지원된다.
  - 리스트는 l<contents>e로 인코딩된다. (예) [1, “Bye”] = li1e3:Byee
  - 맵은 d<contents>e로 인코딩된다. (예) {“R” 2, “D” 2} = d1:Ri2e1:Di2ee
    - 키는 단순히 문자열이고, 값에는 모든 Bencode 요소가 허용된다.

페드로: 쉬워 보이네요

이브: 그럴 수도 있지만, 값들은 중첩될 수 있다는 점을 고려해야 해요. 예를 들면 리스트 안의 리스트 같은.

페드로: 물론이죠. bencode 인코딩에는 해석자 패턴을 사용할 수 있을 것 같네요.

이브: 한번 해 보시죠.

페드로: 모든 bencode 요소들을 위한 인터페이스부터 시작해 보죠.

```
interface BencodeElement {
  String interpret();
}
```

페드로: 그리고 나서 각각의 데이터 타입과 데이터 컨데이너에서 위의 인터페이스를 구현합니다.

```
class IntegerElement implements BencodeElement {
  private int value;

  public IntegerElement(int value) {
    this.value = value;
  }

  @Override
  public String interpret() {
    return "i" + value + "e";
  }
}
```

```
class StringElement implements BencodeElement {
  private String value;

  StringElement(String value) {
    this.value = value;
  }

  @Override
  public String interpret() {
    return value.length() + ":" + value;
  }
}

class ListElement implements BencodeElement {
  private List<? extends BencodeElement> list;

  ListElement(List<? extends BencodeElement> list) {
    this.list = list;
  }

  @Override
  public String interpret() {
    String content = "";
    for (BencodeElement e : list) {
      content += e.interpret();
    }
    return "l" + content + "e";
  }
}

class DictionaryElement implements BencodeElement {
  private Map<StringElement, BencodeElement> map;

  DictionaryElement(Map<StringElement, BencodeElement> map) {
    this.map = map;
  }

  @Override
  public String interpret() {
    String content = "";
    for (Map.Entry<StringElement, BencodeElement> kv : map.entrySet()) {
      content += kv.getKey().interpret() + kv.getValue().interpret();
    }
    return "d" + content + "e";
  }
}
```

페드로: 마지막으로 bencode 인코딩 문자열을 만들 수 있어요.

```
// discredit user
Map<StringElement, BencodeElement> mainStructure = new
HashMap<StringElement, BencodeElement>();
// our victim
mainStructure.put(new StringElement("user"), new StringElement("Bertie"));
// just downloads files
mainStructure.put(new StringElement("number_of_downloaded_torrents"), new
IntegerElement(623));
// and nothing uploads
mainStructure.put(new StringElement("number_of_uploaded_torrents"), new
IntegerElement(0));
// and nothing donates
mainStructure.put(new StringElement("donation_in_dollars"), new
IntegerElement(0));
// prefer dirty categories
mainStructure.put(new StringElement("preffered_categories"),
                      new ListElement(Arrays.asList(
                          new StringElement("porn"),
                          new StringElement("murder"),
                          new StringElement("scala"),
                          new StringElement("pokemons")
                      )));
BencodeElement top = new DictionaryElement(mainStructure);

// let's totally discredit him
String bencodedString = top.interpret();
BitTorrent.send(bencodedString);
```

이브: 재미있네요, 그런데 코드량이 너무 많네요!

페드로: 기능이 많다 보니 읽기 어려워졌어요.

이브: 코드가 곧 데이터(Code is Data)라는 말을 들어 보신 적 있을 거예요. 그래서 클로저에서는 훨씬 수월해지죠.

```
;; multimethod to handle bencode structure
(defmulti interpret class)

;; implementation of bencode handler for each type
(defmethod interpret java.lang.Long [n]
  (str "i" n "e"))

(defmethod interpret java.lang.String [s]
  (str (count s) ":" s))

(defmethod interpret clojure.lang.PersistentVector [v]
  (str "l"
       (apply str (map interpret v))
       "e"))
```

```
(defmethod interpret clojure.lang.PersistentArrayMap [m]
  (str "d"
       (apply str (map (fn [[k v]]
                         (str (interpret k)
                              (interpret v))) m))
       "e"))

;; usage
(interpret {"user" "Bertie"
            "number_of_downloaded_torrents" 623
            "number_of_uploaded_torrent" 0
            "donation_in_dollars" 0
            "preffered_categories" ["porn"
                                    "murder"
                                    "scala"
                                    "pokemons"]})
```

이브: 데이터를 정의하는 것이 얼마나 쉬운지 보이세요?

페드로: 그러네요.interpret은 bencode 자료형 당 단순히 하나의 함수네요. 별도의 클래스가 아니라.

이브: 맞아요, 해석자 패턴은 단순히 트리를 처리하는 함수들일 뿐인거죠.

관련 문서

- Clojure Design Patterns