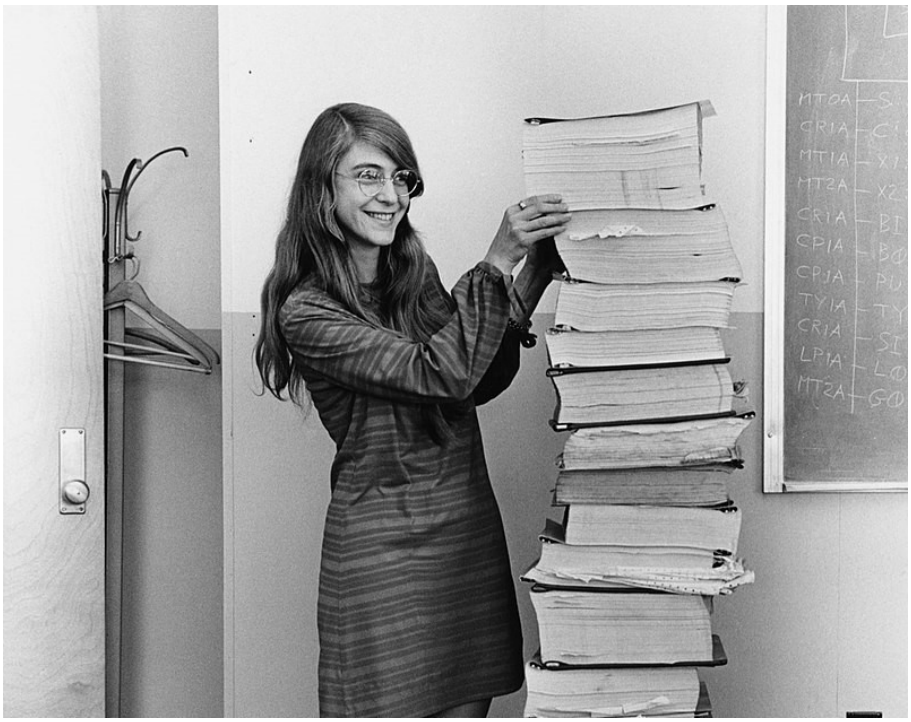
JiiVal

Java Validation Engine



"With a preventative paradigm,
most errors aren’t allowed into a system in the first place,
just by the way the system is defined.
With such an approach, the more reliable the system,
the higher the productivity in its lifecycle."

(Margaret Hamilton)

JiiVal: una panoramica

JiiVal è un engine di validazione per oggetti di input complessi, il cui scopo è ridurre il codice boilerplate (sezioni di codice ripetute diverse volte all'interno del software con poche variazioni) che si deve scrivere in linguaggio Java quando occorre validare input complessi.
Con JiiVal si può validare qualsiasi tipo di oggetto Java, sia che venga usato all'interno dell'architettura del progetto, sia che si tratti di un oggetto generato da un parsing Json eseguito mediante Jackson o librerie simili.

Il problema della validazione: storia

L'allunaggio del 1969 è stato possibile nonostante un errore potenzialmente critico, poiché il software era stato progettato tenendone conto.
La storia dell'informatica corrisponde alla storia di un problema che la attraversa in tutta la sua lunghezza: quello della validazione degli input.
Lo scienziato statunitense Margaret Hamilton, direttrice e supervisore dello sviluppo software per i programmi Apollo e Skylab presso la NASA, ha formalizzato il concetto secondo cui la maggior parte dei problemi dell'informatica arriva da input sbagliati.
"Gli errori" scrive la Hamilton, che su questa teoria ha poi creato la propria azienda *"non solo ci dicono come costruire sistemi privi di essi ma anche ci forniscono, in modo del tutto inaspettato, un paradigma per il futuro"*.
Questo punta di vista risulta evidente anche nell'informatica contemporanea in quanto, se è possibile controllare in modo efficace l'input, in modo particolare se dipendente da fonti esterne al sistema, è possibile prevenire la maggior parte dei problemi del software.

Il problema della validazione: anatomia

JiiVal nasce dall'esigenza di risolvere una parte consistente dei problemi derivanti dal processo di validazione.
Vi sono infatti input che possono essere definiti come semplici, per i quali non si riscontrano particolari problemi, mentre altri sono più complessi sia come struttura che come requisiti. È in quest'ultimo caso che la validazione e le regole implicite a questa iniziano a divenire problematiche per lo sviluppatore e, ad oggi, il problema viene ancora gestito con strutture obsolete e poco pratiche.
Nel formato XML, per esempio, esiste un concetto denominato DTD (Document Type Definition) in cui è possibile settare un insieme di regole che un file .xml deve rispettare per essere valido. In questo senso XML è un formato sul quale è possibile definire a monte un set di regole al fine di determinare la sua validità, ma la complessità del formato DTD ne scoraggia spesso l'uso se non per controlli elementari che non coprono tutte le casistiche che sarebbero necessarie.
In JSON e in altri formati di markup più moderni e leggeri questa possibilità non è nemmeno presente, in quanto la correttezza dei dati viene sacrificata in nome della semplicità sintattica.

A questi problemi si aggiunge l'esperienza pratica: JSON è lo standard de facto per i servizi REST, ma spesso le specifiche di input dei servizi risultano complesse, difficili e non separabili: servono molti dati, regolati da molte regole, e serve tutto insieme. Questi problemi si verificano spesso nella fase di sviluppo relativa ai requisiti e alle specifiche, sopratutto quando si lavora a progetti di integrazione la cui base di partenza prevede regole ereditate da sistemi legacy. Questo porta a strutture JSON complesse, di dimensioni considerevoli e che presentano delle interdipendenze tra i campi riguardanti l'environment, altri dati presenti nel Json o su una fonte dati esterna, tutti necessari al completamento di una transazione.

Dalla teoria alla pratica

Il problema della validazione degli input, finché confinato nei formalismi accademici, risulta marginale ma, nella pratica dell'informatica applicata, determina situazioni di criticità dettate dall'applicazione operativa dei fattori di un sistema.
Input e output che ci si trova a gestire nel mondo pratico, fuori dalle realtà istituzionali, sono maggiormente densi e complessi.
Supponiamo di dovere effettuare un cambio di offerta su un contratto di un'utenza nella cornice di un sistema enterprise per multi-utility: il suddetto contratto ha come requisito primario l'essere esistente, poi si deve verificare la coerenza dell'offerta col tipo di utenza (pubblica, privata…), controllare che tutte le informazioni necessarie siano presenti in funzione del tipo di utenza o richiedente eccetera.
Al momento non esiste ad oggi una libreria specifica che permetta di superare questo genere di difficoltà e il supporto di quelle esistenti è troppo elementare. Una delle librerie più utilizzate come Jackson, per esempio, fornisce un unico tipo di validazione specificabile tramite annotation: Null/Not Null incondizionato.
Per risolvere nella pratica un sistema di validazioni complesse esiste una sola soluzione: scrivere Business Logic per controllare questo genere di informazioni, e che tipicamente prevede una serie di costrutti if-else spesso annidati che, non di rado, divengono il punto debole di molti software.
Questi costrutti nella peggiore delle ipotesi vengono sparsi nel codice della logica di business, in alternativa vengono raggruppati in un'unica classe di validazione che sposta il problema senza risolverlo alla radice: in ogni caso le classi risultano poco mantenibili, facilitano l'apparizione di bug e all'aumento della loro complessità strutturale corrisponde un numero più alto di errori nel sistema.

Una soluzione per domarli tutti (gli input)

JiiVal è un engine nato per risolvere i problemi illustrati nei paragrafi precedenti.
Una libreria leggera e snella che permette di definire direttamente sugli oggetti Java un set di regole che devono essere rispettate affinché quell'oggetto sia considerato valido e processabile.
In sostanza, l'engine si ispira alla validazione elementare annotation-based di altre librerie come Jackson, estendendone i concetti con quello che manca, ovvero con il fatto che è possibile stabilire direttamente negli oggetti di mapping tutte le regole di cui il sistema necessita.

La libreria utilizza un set limitato di annotation, due necessarie per identificare i campi che il validatore deve considerare nelle logiche di controllo (`@ValidatorId` e `@ValidatorField`), e due per stabilire le vere e proprie regole di validazione (`@Required` e `@MustBe`). Tramite queste annotation e un semplice linguaggio condizionale si è in grado di controllare la maggior parte dei requisiti di input direttamente nelle classi di mapping.

Required e Must Be: requisiti condizionali

La libreria di validazione JiiVal non si limita alla validazione incodizionata (es. “questo attributo deve sempre essere presente”), che spesso sono di scarsa utilità pratica, ma permette di specificare all’interno delle annotation Required e MustBe le condizioni affinché l’annotation stessa sia attiva in un determinato contesto.
Con l’annotation Required si indica che un campo è richiesto (quindi non può essere null) a seconda di una serie di condizioni specifiche.
Required verifica solo che un valore sia presente: se un campo è presente, Required è inattivo in quanto il campo è già valorizzato.
Se però il campo non è presente, allora Required verifica le condizioni: se non c’è nessuna condizione (quindi assertion), significa che il campo è richiesto a prescindere (quindi tale campo non può essere null), altrimenti viene verificato il rispetto delle condizioni specificate all’interno dell’annotation (in logica *and*)
MustBe è invece una funzione diversa: indica che il campo deve essere compreso in un certo range di valori, incondizionatamente o se si verificano determinate condizioni (sempre in logica *and*).
Entrambe le annotation sono ripetibili su uno stesso attributo: in questo caso vengono considerate con logica *or*.

Coverage della libreria

JiiVal rappresenta il giusto compromesso funzionale tra la complessità delle dichiarazioni necessarie alla libreria e la copertura di una percentuale dei casi d’uso più alta possibile. JiiVal è in grado, attraverso le sue annotation standard, di coprire la maggior parte delle situazioni di validazione normalmente affrontate nella pratica; per sopperire ai restanti requisiti difficilmente risolvibili è possibile creare oggetti custom, possibilità prevista dalla libreria tramite classi estensibili e specificabili nelle annotation. In questo caso, la gestione viene demandata al programmatore e il risultato della validazione viene integrato nel comportamento standard della libreria.

Lo scopo è fornire ai programmatori un engine di facile utilizzo, estendibile sui casi non contemplati in modo rapido, senza gravare la libreria di complessità superflua nella maggior parte dei casi.

Spostare o risolvere il problema?

Si potrebbe pensare, di primo acchito, che la libreria JiiVal non risolve il problema della validazione ma si limita a spostarlo sugli oggetti. A volte, però, lo spostamento del problema nella giusta posizione e col giusto approccio coincide con la sua risoluzione.

Eseguire il mapping fra le specifiche di progetto e gli oggetti Java, ovvero mappare la struttura di input in classi POJO è una pratica normalissima ed eseguita a prescindere; permettere di sintetizzare le specifiche di validazione, spesso organizzate in forma tabellare, rilevate durante la fase di analisi dei prerequisiti del software direttamente in quelle classi significa avere un singolo punto di intervento e un mapping diretto tra specifiche di progetto e codice sviluppato.
In JiiVal, la barriera tra analista e programmatore viene meno in quanto ciò che viene rilevato dal primo viene direttamente determinato negli oggetti dal secondo.
Quindi un'evoluzione del processo di mapping che permette di centralizzare in un unico posto gli oggetti e le loro logiche di validazione.
Il team di sviluppo si trova a dovere cambiare un parametro per una specifica differente di progetto?
Nel metodo tradizionale di validazione, è necessario controllare e aggiornare tutti i servizi riguardanti quel parametro, magari cercando all'interno del codice tutti i punti in cui uno specifico campo viene usato e controllato; con JiiVal, si va invece nell'oggetto investito dal cambio di specifica, si modifica la condizione e questa sarà subito valida per tutto il codice.
Quindi una centralizzazione sugli oggetti di business che consente maggiore praticità di sviluppo e un notevole risparmio di tempo.
Mediante l'engine di JiiVal si va a creare un'alta corrispondenza tra l'analisi del software e le logiche di programmazione.
Se normalmente in caso di programma scritto da zero, il passaggio tra analista e programmatore prevede numerose lavorazioni intermedie, con JiiVal intanto che si procede a mappare le classi, vengono anche compilate le validazioni di queste.
Mentre si mappano gli oggetti e le condizioni rilevati dall'analista, si va a mappare direttamente anche le regole di validazione, peraltro con una sintassi molto più compatta rispetto al codice Java standard.
Quindi JiiVal non solo risolve il problema della validazione ma va anche a impattare in modo notevole sulla sequenza lavorativa tra analista e programmatore.
Il risultato è che qualsiasi implementazione o modifica successiva è più veloce e robusta, consentendo di avere un codice più facile da scrivere e mantenere.

JiiVal è un progetto pensato per coniugare un'alta qualità del software e il risparmio di risorse, tenendo sullo sfondo l'importanza della risoluzione dei problemi storici dell'ingegneria informatica.