Lingua: italiano

07/11/2022

Lorenzo Lotti

JCSP

(JSON Chat System Protocol)

v1.0

[JCSP/1.0]

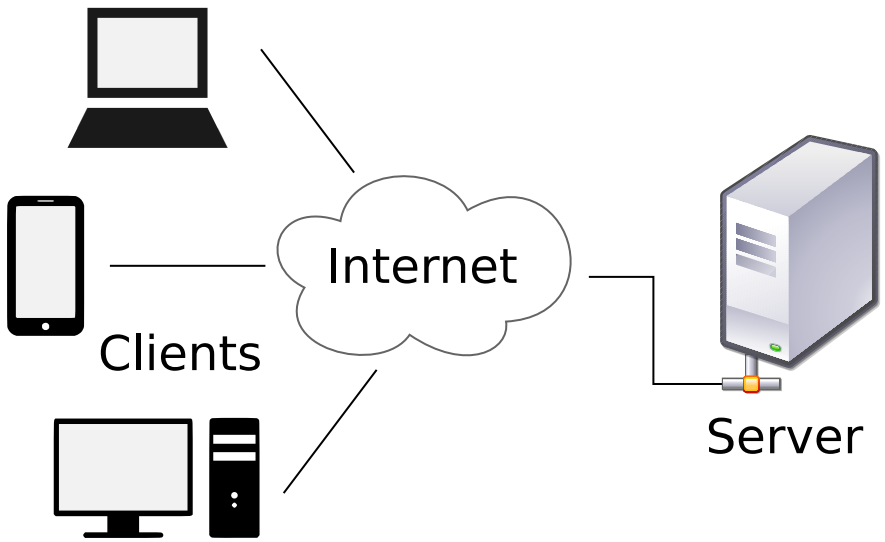
Documentazione:

Architettura fondamentale

Introduzione

Il JCSP (JSON Chat System Protocol) è un protocollo open-standard di **livello applicazione** (ISO/OSI) che si occupa della trasmissione e gestione d'informazioni in un sistema di chat centralizzato **client-server.**

JCSP sfrutta il formato JSON per la serializzazione della maggior parte dei dati, metadati e/o testo e utilizza, a livello trasporto, il protocollo TCP per la trasmissione.

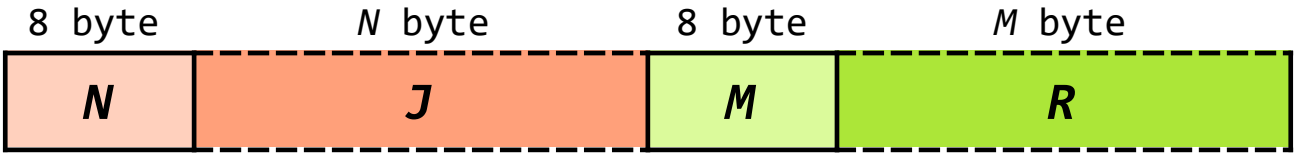


Livello trasporto

A livello trasporto JCSP utilizza **TCP** (Transmission Control Protocol) per la comunicazione tra client e server.

I dati vengono imbustati e spediti nel socket nella seguente configurazione:

- **8 byte** della lunghezza dei dati JSON (***N***)
- ***N* byte** di dati JSON (***J***)
- **8 byte** della lunghezza dei dati raw (***M***)
- ***M* byte** di dati raw (***R***)



Questa configurazione prende il nome di **DPDU** (Doubled Protocol Data Unit)

I dati raw (campo R) possono contenere qualsiasi tipo di dato aggiuntivo che il client può interpretare a piacimento.
Ad esempio si potrebbe implementare un client JCSP con il supporto per l'invio di immagini.

Il server non può e non deve alterare questo campo, ma unicamente e obbligatoriamente inoltrarlo.

Payload JSON

Il campo J del DPDU contiene la stringa JSON (JavaScript Object Notation), con **codifica UTF-16**, che porta con se le informazioni chiave della chat.

JSON è un formato open-standard per la rappresentazione testuale (human-readable) di strutture di dati. La sua sintassi è basata su quella degli oggetti JavaScript, il che rende ogni JSON uno script JS valido.

Benché sia fortemente consigliato, le stringhe JSON del campo J non devono essere necessariamente minimizzate, è sufficiente che seguino lo standard **ECMA-404**.

Relazione client-server

Tra client e server intercorre una **relazione *1:X*** dove *1* è il numero di server ed *X* il numero di client. Ciò significa che un server JCSP deve necessariamente supportare connessioni da parte di più client. Il server deve **gestire i client in maniera asincrona e concorrente** per garantire il corretto funzionamento dell'intero sistema.

Stati client

I client posso trovarsi in due distinstinti stati nei confronti del sistema:

- Stato di "escluso" (**stato iniziale**)
- Stato di "incluso"

Ogni client in stato di "incluso" ha associato un ID di tipo **UUID** (Universally Unique Identifier).

Schemi JSON

Introduzione

In questa sezione sono elencati gli schemi JSON validi per il campo J del DPDU. Saranno quindi definiti i messaggi che client e server si scambiano e i loro nomi.

Per ogni schema, se necessario, saranno definiti dei **vincoli sugli attributi che devranno essere validati obbligatoriamente dal server**.

Ogni messaggio, a seconda del suo schema, può avere degli effetti sul sistema. Le caratteristiche che definiscono questi effetti sono:

- *Direzione*: client(s) $\rightleftarrows$ server
- Vincolo stato client: escluso/incluso
- *Necessità di risposta*: si/no
- *Trigger*: causa dell'invio di uno schema
- *Effetto diretto*:
  effetto diretto specificato che avviene sul destinatario.

Schema "schema-error"

Direzione messaggio: server → client

Risposta necessaria: no

Trigger: schema non valido

```
{
  "schema": "schema-error"
}
```

Questo messaggio speciale viene inviato dal server al client quando un messaggio ricevuto ha uno schema non valido o inesistente.

Questo schema prevale su qualsiasi altro schema di risposta.

Schema "state-error"

Direzione messaggio: server → client

Risposta necessaria: no

Trigger:
stato del client / ID non valido

```
{
  "schema": "state-error"
}
```

Questo messaggio speciale viene inviato dal server al client quando quest'ultimo invia un messaggio in uno stato in cui non è consentito o se il messaggio contiene un UUID inesistente sul server.

Questo schema prevale su qualsiasi altro schema di risposta tranne schema-error.

Schema "join"

Direzione messaggio: client → server

Vincolo stato client: escluso

Risposta necessaria: si

Effetto diretto:

Il server **include il client con il suo nome utente** e **genera un UUID** ad esso associato.

```
{
  "schema": "join",
  "username": "<username>"
}
```

Dove *<username>* corrisponde al nome utente che il client vuole utilizzare. Questo attributo è valido solamente se:

- il nome utente è **univoco sul server**
- la lunghezza è **minimo 2 caratteri**
- la lunghezza è **massimo 16 caratteri**
- i caratteri sono compresi nell'intervallo **]0x20, 0x7E]**

Schema "join-error"

Direzione messaggio: server → client

Vincolo stato client: escluso

Risposta necessaria: no

Trigger:
messaggio con schema "join" non valido

```
{
  "schema": "join-error",
  "error": "<error-type>"
}
```

Dove *<error-type>* corrisponde al **tipo di errore riscontrato con il nome utente.**
I valori di questo attributo possono unicamente essere:

- `uniqueness` : errore di univocità
- `length` : lunghezza non valida
- `interval` : intervallo non valido

[p.10 per maggiori dettagli sulla convalida del nome utente]

Schema "join-ok"

Direzione messaggio: server → client

Vincolo stato client: escluso

Risposta necessaria: no

Trigger:
messaggio con schema "join" valido

Effetto diretto:
Il client **salva il proprio ID** (UUID) e passa allo stato di "**incluso**"

```
{
  "schema": "join-ok",
  "id": "<uuid>"
}
```

Dove *<uuid>* corrisponde all'**UUID associato al client** che ha fatto la richiesta join.

[p.6 per maggiori dettagli sugli UUID]

Schema "list"

Direzione messaggio: client → server

Vincolo stato client: incluso

Risposta necessaria: si

```
{
  "schema": "list",
  "id": "<uuid>"
}
```

Dove *<uuid>* corrisponde all'**ID del client.**

Lo scopo dei messaggi con questo schema è quello di ottenere dal server la lista di tutti i client connessi.

Schema "list-update"

Direzione messaggio: server → client(s)

Vincolo stato client: incluso

Risposta necessaria: no

(Trigger):
messaggio con schema "list"

```
{
  "schema": "list-update",

  "usernames":
  [
    "<username1>",
    "<username2>",
    "<username3>",
    ...,
    "<usernameN>"
  ]
}
```

Dove l'attributo *"usernames"* è un array JSON di stringhe contenente **gli username di ciascun client escluso il richiedente.**

Questo schema può essere inviato dal server in risposta a una richiesta list o

in un momento definito dalla sua implementazione.

È consigliato implementare il server in modo che invii questo schema a ogni client ogni volta che un nuovo utente viene incluso.

Schema "send"

Direzione messaggio: client → server

Vincolo stato client: incluso

Risposta necessaria: si

Effetto diretto:

Il server **inoltra il contenuto del messaggio ai destinatari** specificati (vedere schema "notification": p.21).

```
{
  "schema": "send",
  "id": "<uuid>",
  "text": "<message-content>",

  "usernames":
  [
    "<username1>",
    "<username2>",
    "<username3>",
    ...,
    "<usernameN>"
  ]
}
```

Dove *<uuid>* corrisponde all'**ID del client**, *<message-content>* corrisponde al contenuto

del messaggio testuale e l’attributo *"usernames"* è un array JSON di stringhe contenente **gli username di ciascun destinatario.**

<message-content> deve necessariamente contenere **almeno un carattere visibile** (spaziature escluse).

L’attributo *"usernames"* deve rispettare i seguenti vincoli:

- Contenere **almeno uno username**
- Contenere solo username **esistenti**
- **Non contenere lo username del mittente**

Schema "send-error"

Direzione messaggio: server → client

Vincolo stato client: incluso

Risposta necessaria: no

Trigger:
messaggio con schema "send" non valido

```
{
  "schema": "send-error",
  "error": "<error-type>",

  "usernames":
  [
    "<username1>",
    "<username2>",
    "<username3>",
    ...,
    "<usernameN>"
  ]
}
```

Dove *<error-type>* corrisponde al **tipo di errore riscontrato con il contenuto del messaggio o con gli username dei destinatari**, e l'attributo *"usernames"* è un array JSON di stringhe contenente i

destinatari non validi. I valori di *<error-type>* possono unicamente essere:

- `invisible` : il testo non è valido
- `still` : non ci sono destinatari
- `target` : i destinatari non sono validi

(nel caso di un tipo di errore invisible/still l'attributo *"usernames"* si presenterà come un array vuoto)

[p.17 per maggiori dettagli sulla convalida di uno schema "send"]

Schema "send-ok"

Direzione messaggio: server → client

Vincolo stato client: incluso

Risposta necessaria: no

Trigger:
messaggio con schema "send" valido

```
{
  "schema": "send-ok"
}
```

(La richiesta send è andata a buon fine)

Schema "notification"

Direzione messaggio: server → client(s)

Vincolo stato client: incluso

Risposta necessaria: no

Trigger:
messaggio con schema "send" valido

```
{
  "schema": "notification",
  "sender": "<username>",
  "text": "<message-content>"
}
```

Dove *<username>* è il **nome utente del mittente** (ottenuto dal server convertendo l'UUID ricevuto) e *<message-content>* è il **contenuto del messaggio.**

Questo schema viene inviato solo a ogniuno dei destinatari definiti nella richiesta send tramite l'attributo *"usernames"* (vedere p.16).

Schema "check"

Direzione messaggio: server → client

Vincolo stato client: incluso

Risposta necessaria: si

```
{
  "schema": "check"
}
```

Questo messaggio speciale può essere inviato ai client per controllare se sono ancora connessi a livello trasporto.
Nel caso in cui l'invio non avvenga o non si ottenga alcuna risposta si può ritentare; sta all'implementazione del server JCSP decidere dopo quanti tentativi escludere il client.

È importante che il server si assicuri che tutti i client in stato di "incluso" siano effettivamente connessi a livello trasporto per garantire il corretto funzionamento del sistema.

Schema "hello"

Direzione messaggio: client → server

Vincolo stato client: incluso

Risposta necessaria: no

(Trigger):
messaggio con schema "check"

```
{
  "schema": "hello",
  "id": "<uuid>"
}
```

Dove *<uuid>* corrisponde all'**ID del client.**

Questo schema può essere inviato dal client in risposta a una richiesta check o in un momento definito dalla sua implementazione.

Questo messaggio speciale consente al server di capire se un client in stato di "incluso" è effettivamente connesso a livello trasporto.

Schema "exit"

Direzione messaggio: client → server

Vincolo stato client: incluso

Risposta necessaria: no

Effetto diretto:

Il server **esclude il client (eliminando nome utente e UUID) e lo disconnette a livello trasporto.**

```
{
  "schema": "exit",
  "id": "<uuid>"
}
```

Dove *<uuid>* corrisponde all'**ID del client che vuole escludersi.**