# How-To: install a Joseki SPARQL endpoint for VIVO

This page contains out-of-date information. If you are interested running a SPARQL endpoint with VIVO version 1.5 or 1.6, refer to the Setting up a VIVO SPARQL Endpoint page.

## Introduction

This wiki page entitled *How to install a Joseki SPARQL endpoint for VIVO* was adapted from the text based HOWTO on the VIVO Sourceforge downloads section. It is meant to serve as a starting point for setting up your SPARQL endpoint for VIVO using the Joseki software that is part of the Jena project. The example details installing a Joseki SPARQL endpoint using the VIVO Virtual BOX virtual appliance. The most important part of the example comes from showing how to setup the Joskei config file called Joseki-config.ttl to hook up to your VIVO triple store. Also, this example instruction set assumes you are installing on Debian Linux or a Debian variant like Ubuntu. If you have VIVO running on another system you will have to follow the Joseki install instructions for your system and then use the sample Joseki-config.ttl provided. WARNING!- If you don't want anonymous updates via your spark-ly new SPARQL endpoint then please make sure you follow instructions for adding a filter in step 8 below. Comments on improving this page are very welcome. Good luck and good SPARQL-ing.

### Update

Note that the Apache Jena project is close to a 1.0 release of Fuseki – new 1.0 and 1.01 snapshots

Fuseki configuration for VIVO shared by NickV, 9/9/13:

```
# Licensed under the terms of http://www.apache.org/licenses/LICENSE-2.0


@prefix :        <#> .
@prefix fuseki:  <http://jena.apache.org/fuseki#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix tdb:     <http://jena.hpl.hp.com/2008/tdb#> .
@prefix ja:      <http://jena.hpl.hp.com/2005/11/Assembler#> .
@prefix jumble:   <http://rootdev.net/vocab/jumble#> .
@prefix sdb: <http://jena.hpl.hp.com/2007/sdb#> .


[] rdf:type fuseki:Server ;
# Timeout - server-wide default: milliseconds.

  # Format 1: "1000" -- 1 second timeout
  # Format 2: "10000,60000" -- 10s timeout to first result, then 60s timeout to for rest of query.
# See java doc for ARQ.queryTimeout

  ja:context [ ja:cxtName "arq:queryTimeout" ;  ja:cxtValue "10000,60000" ] ;


  fuseki:services (
<#service_VIVO_read_only>

  ) .


# SDB
[] ja:loadClass "net.rootdev.fusekisdbconnect.SDBConnect" .
jumble:SDBConnect rdfs:subClassOf ja:RDFDataset .


<#service_VIVO_read_only> rdf:type fuseki:Service ;
rdfs:label                      "TUe VIVO Service (R)" ;

   fuseki:name                     "VIVO" ;
   fuseki:serviceQuery             "query" ;
fuseki:serviceQuery             "sparql" ;

   #fuseki:serviceUpdate            "update" ;
   #fuseki:serviceUpload            "upload" ;
#fuseki:serviceReadWriteGraphStore      "data" ;

   # A separate read-only graph store endpoint:
   #fuseki:serviceReadGraphStore       "get" ;
```

```
fuseki:dataset           <#ufvivo_dataset_read> ;

   #fuseki:dataset           <#VIVOStore> ;
   .
<#ufvivo_dataset_read> rdf:type      sdb:DatasetStore ;
sdb:store <#VIVOStore>

.
<#VIVOStore> rdf:type jumble:SDBConnect;
rdfs:label                  "TUE VIVO SDB Store";

   sdb:layout                  "layout2";
   jumble:defaultUnionGraph "true" ;
sdb:engine                  "InnoDB";

   sdb:connection
   [ rdf:type sdb:SDBConnection;
sdb:sdbHost "localhost";

       sdb:sdbType "MySQL";
       sdb:sdbName "vivo";
sdb:sdbUser "private";

       sdb:sdbPassword "private";
       sdb:driver "com.mysql.jdbc.Driver";
]

     .
```

# VIVO Help

- mailing list: vivo-dev-all
- list archive: List Archive
- main wiki: Wiki Home
- Live IRC chat room: #VIVO

## How to install a Joskei SPARQL endpoint for VIVO

## Step 1: Install the latest VIVO Virtual Appliance ( if using VM for testing)

Download and set up a VIVO virtual appliance from:

```
       http://sourceforge.net/projects/vivo/files/VIVO%20Virtual%20Appliances/
```

Note: all the shell commands are assuming you are on the console of the virtual appliance, but they should work on any Debian system. Virtual Machine Install Documentation is available at:

http://vivoweb.org/support/user-guide/installation

After installation and configuration, start the virtual appliance and login to the command line interface. The username is vitro. The password is vitro123.

## Step 2 Install Joseki (debian)

Change directories

```
vitro@vivo:/usr/local$ cd /usr/local
```

Download Joseki from:

```
vitro@vivo:/usr/local$ sudo wget http://sourceforge.net/projects/joseki/files/Joseki-SPARQL/Joseki-3.4.2/joseki-3.4.2.zip/download
```

Install unzip, needed to unpack the Joseki ZIP file

```
vitro@vivo:/usr/local$ sudo apt-get install unzip
```

Unzip Joseki in usr/ocal

```
vitro@vivo:/usr/local$ sudo unzip joseki-3.4.2.zip
```

Make the script in bin executable

```
vitro@vivo:/usr/local/Joseki-3.4.2$ cd Joseki-3.4.2
```

vitro@vivo:/usr/local/Joseki-3.4.2$ sudo chmod u+x bin/*

## Step 3. Configure Joseki

See below for #Configuration for VIVO version 1.3.

Backup the config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo cp joseki-config.ttl joseki-config.default
```

Make the new Joseki config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo touch joseki-config-vivo.ttl
```

Edit the config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo nano joseki-config-vivo.ttl
```

First, we declare some prefixes, then some basic information about this file. Because the configuration file is RDF, order of the sections does not matter to the server but it does help a human reading the file. Copy and paste the following config file into the joseki-config.ttl:

COPY THE FOLLOWING TEXT INTO YOUR CONFIG

```
## -------------------------------------------------------------
# Title: VIVO Joseki example config
# Author(s): cpb@ufl.edu, drspeedo@ufl.edu
# ModDate: 20100805
# ShortDescR: This is an example joseki config to access vivo
#             data using the joseki service.
# Warranty: Use at your own risk.
## -------------------------------------------------------------
# This file is written in N3 / Turtle

@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .

@prefix module: <http://joseki.org/2003/06/module#> .
@prefix joseki: <http://joseki.org/2005/06/configuration#> .
@prefix ql:     <http://jena.hpl.hp.com/2003/07/query/> .
@prefix ja:     <http://jena.hpl.hp.com/2005/11/Assembler#> .

## -------------------------------------------------------------
## This file is written in N3 / Turtle
## It is an RDF graph - order of RDF triples does not matter
## to the machine but it does help people who need to edit this file.
## -------------------------------------------------------------
## Note: adding rdfs:label to blank nodes will cause Joseki
## to print that in log messages.
## -------------------------------------------------------------
## About this configuration
<> rdfs:label "Joseki Configuration File" .
```

```
## ------------------------------------------------------------
## About this server
[]  rdf:type joseki:Server ;
 joseki:serverDebug   "true" ;
# Example of some initialization code.
  joseki:initialization
  [ module:implementation
   [ module:className <java:org.joseki.util.ServiceInitSimple> ;
    rdfs:label "Example initializer" ; ]
  ] ;
  .
## ------------------------------------------------------------
## Services
## Services are the points that request are sent to.
## serviceRef that will be used to match requests to services,
## not some resource URI for the description.

## Note that the service reference and the routing of incoming
## requests by URI as defined by web.xml have to align.

# Service 1
# General purpose SPARQL processor, no dataset, expects the
# request to specify the dataset (either by parameters in the
# protocol request or in the query itself).

 []
  rdf:type        joseki:Service ;
  rdfs:label        "service point";
  joseki:serviceRef    "sparql" ;
  joseki:processor    joseki:ProcessorSPARQL;
  joseki:dataset    <#ds1> ;
  .

## ------------------------------------------------------------
# Service 2 - SPARQL processor only handling a given dataset
 []
  rdf:type            joseki:Service ;
  rdfs:label          "SPARQL on the VIVO model" ;
  joseki:serviceRef   "VIVOService" ;
  # web.xml just route this name to Joseki
  # dataset part
  joseki:dataset      <#ds1> ;
  # Service part.
  # This processor will not allow either the protocol,
  # nor the query, to specify the datset.
  joseki:processor    joseki:ProcessorSPARQL_FixedDS ;
  .
## ------------------------------------------------------------
## Datasets
 <#ds1>   rdf:type ja:RDFDataset ;
  ja:defaultGraph    <#modelDB1> ;
  rdfs:label "Dataset 1" ;
  ja:namedGraph
  [ ja:graphName <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> ;
  ja:graph          <#modelDB1> ] ;
  .
## ------------------------------------------------------------
# This is very very bad. Configure MySQL security correctly
## ------------------------------------------------------------
## ------------------------------------------------------------
 <#modelDB1> rdf:type ja:RDBModel ;
 ja:connection
 [
  ja:dbType "MySQL" ;
  ja:dbURL          <jdbc:mysql://localhost/vitro> ;
  ja:dbUser         "root" ;
  ja:dbPassword     "vitro123" ;
  ja:dbClass        "com.mysql.jdbc.Driver" ;
  ] ;
  ja:reificationMode    ja:minimal ;
## Minimal means fastpath is possible.
```

```
   ja:modelName "http://vitro.mannlib.cornell.edu/default/vitro-kb-2"
   .

 ## ------------------------------------------------------------
 ## Processors

  joseki:ProcessorSPARQL
  rdfs:label "General SPARQL processor" ;
  rdf:type joseki:Processor ;
  module:implementation joseki:ImplSPARQL ;

  # Parameters - this processor processes FROM/FROM NAMED
  joseki:allowExplicitDataset       "true"^^xsd:boolean ;
  joseki:allowWebLoading            "true"^^xsd:boolean ;
  ## And has no locking policy (it loads data each time).
  ## The default is mutex (one request at a time)
  joseki:lockingPolicy               joseki:lockingPolicyNone ;
  .

  joseki:ProcessorSPARQL_FixedDS
    rdfs:label "SPARQL processor for fixed datasets" ;
    rdf:type joseki:Processor ;
    module:implementation joseki:ImplSPARQL ;

 # This processor does not accept queries with FROM/FROM NAMED
   joseki:allowExplicitDataset       "false"^^xsd:boolean ;
   joseki:allowWebLoading            "false"^^xsd:boolean ;
   joseki:lockingPolicy              joseki:lockingPolicyMRSW ;
   .

 # WARNING!!!! If you don't want people to update see wiki
 # WARNING!!!! See how to add filter to block update proc.
   joseki:ProcessorSPARQLUpdate
   rdfs:label "SPARQL Update processor" ;
   rdf:type joseki:Processor ;
   module:implementation joseki:ImplSPARQLUpdate ;
   joseki:lockingPolicy              joseki:lockingPolicyMRSW ;
   .

   joseki:ImplSPARQL
    rdf:type   joseki:ServiceImpl ;
    module:className
    <java:org.joseki.processors.SPARQL>
    .

   joseki:ImplSPARQLUpdate
    rdf:type   joseki:ServiceImpl ;
    module:className
    <java:org.joseki.processors.SPARQLUpdate>
    .

 # Local Variables:
 # tab-width: 4
 # indent-tabs-mode: nil
 # End:
```

## Configuration for VIVO version 1.3

Backup the config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo cp joseki-config.ttl joseki-config.default
```

Make the new Joseki config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo touch joseki-config-vivo.ttl
```

Edit the config file

```
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo nano joseki-config-vivo.ttl
```

Copy and paste the following to the newly created Joseki VIVO config file

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .

@prefix module: <http://joseki.org/2003/06/module#> .
@prefix joseki: <http://joseki.org/2005/06/configuration#> .
@prefix ja:     <http://jena.hpl.hp.com/2005/11/Assembler#> .
@prefix sdb:      <http://jena.hpl.hp.com/2007/sdb#> .

<> rdfs:label "Joseki Configuration File - SDB example" .
# Stripped down to support one service that exposes an
# SDB store as a SPARQL endpoint for query and one for
# one of the graphs in an SDB store.

[]  rdf:type joseki:Server .

## ---------------------------------------------------------------
## Services

# Service publishes the whole of the SDB store - this is the usual way to use SDB.
<#service1>
    rdf:type            joseki:Service ;
    rdfs:label          "SPARQL-SDB" ;
    joseki:serviceRef   "sparql" ;      # web.xml must route this name to Joseki
    joseki:dataset      <#sdb-part> ;
    joseki:processor    joseki:ProcessorSPARQL_FixedDS ;
    .

#to support connections using Sesame's HTTPRepository
<#service2>
    rdf:type            joseki:Service ;
    rdfs:label          "SPARQL-SDB" ;
    joseki:serviceRef   "sparql/repositories/" ;      # web.xml must route this name to Joseki
    joseki:dataset      <#sdb-part> ;
    joseki:processor    joseki:ProcessorSPARQL_FixedDS ;
    .

# Service that publishes part of an SDB store.
## <#service2>
##     rdf:type            joseki:Service ;
##     rdfs:label          "SPARQL-SDB (alt)" ;
##     joseki:serviceRef   "sparql/graph" ;   # web.xml must route this name to Joseki
##     joseki:dataset      <#sdb-part> ;
##     joseki:processor    joseki:ProcessorSPARQL_FixedDS ;
##     .

## SPARQL/Update.
## Creation of named graph from web requests is not supported.

<#serviceUpdate>
    rdf:type            joseki:Service ;
    rdfs:label          "SPARQL/Update" ;
    joseki:serviceRef   "update/service" ;
    # dataset part
    joseki:dataset      <#sdb>;     # Same as service1
    joseki:processor    joseki:ProcessorSPARQLUpdate
    .

## ---------------------------------------------------------------
## Datasets

## See also SDB documentation -- http://jena.hpl.hp.com/wiki/SDB
## Special declarations to cause SDB to be used.

## Initialize SDB.
```

```
## Tell the system that sdb:DatasetStore is an implementation of ja:RDFDataset .
## Tell the system that sdb:Model is an implementation of ja:RDFDataset .

[] ja:loadClass "com.hp.hpl.jena.sdb.SDB" .
sdb:DatasetStore  rdfs:subClassOf  ja:RDFDataset .
sdb:Model rdfs:subClassOf  ja:Model .

<#sdb> rdf:type sdb:DatasetStore ;
    ## Number of concurrent connections allowed to this dataset.
    joseki:poolSize     64 ;
    sdb:store <#store> .

<#store> rdf:type sdb:Store  ;
    rdfs:label "SDB" ;
    sdb:layout         "layout2" ;
    sdb:connection
    [ rdf:type sdb:SDBConnection ;
##       sdb:sdbType        "postgresql" ;
##       sdb:sdbHost        "localhost" ;
##       sdb:sdbName        "SDB" ;

     # Using Apache Derby
     sdb:sdbHost        "databasehost.my.edu" ;
     sdb:sdbType        "MySQL" ;
     sdb:sdbName        "databasename" ;
     sdb:sdbUser        "databaseuser" ;
     sdb:sdbPassword    "databasepassword" ;
     sdb:driver "com.mysql.jdbc.Driver" ;
    ]
    .

# Pick one graph out of the SDB store.
# Do not assemble the whole of the store this way - it is less efficient for that.
<#sdb-part> rdf:type ja:RDFDataset ;
    # If ja:namedGraph is used here, there is no correspondence
    # with the name in the SDB store.
    ja:defaultGraph <#sdb-one-graph> ;
    .

<#sdb-one-graph> a sdb:Model ;
    sdb:dataset <#sdb> ;
    # Uncomment to pick out a named graph from the store.
    # If no "sdb:namedGraph" appearsm the store's default graph is used.
    # is used as default graph of the dataset publically visible.
    #sdb:graphName <http://example/aNamedGraph> ;
    # Or even the merge of all named graphs
    sdb:graphName <urn:x-arq:UnionGraph> ;
    .

## ---------------------------------------------------------------
## Processors

joseki:ProcessorSPARQL_FixedDS
    rdfs:label "SPARQL processor for fixed datasets" ;
    rdf:type joseki:Processor ;
    module:implementation
    [   rdf:type   joseki:ServiceImpl ;
        module:className <java:org.joseki.processors.SPARQL>
    ] ;

    # This processor does not accept queries with FROM/FROM NAMED
    joseki:allowExplicitDataset       "false"^^xsd:boolean ;
    joseki:allowWebLoading            "false"^^xsd:boolean ;
    # The database is safe for MRSW (multiple-reader, single-writer).
    # joseki:lockingPolicy              joseki:lockingPolicyMRSW ;
    # recommended locking policy when not using SDB connection pooling
    # joseki:lockingPolicy joseki:lockingPolicyMutex ;
    joseki:lockingPolicy joseki:lockingPolicyNone ;
    .

#joseki:ProcessorSPARQLUpdate
```

```
#    rdfs:label "SPARQL Update processor" ;
#    rdf:type joseki:Processor ;
#    module:implementation
#    [   rdf:type   joseki:ServiceImpl ;
#        module:className <java:org.joseki.processors.SPARQLUpdate>
#    ] ;
    # joseki:lockingPolicy             joseki:lockingPolicyMRSW ;
    # recommended locking policy when not using SDB connection pooling
    # joseki:lockingPolicy joseki:lockingPolicyMutex ;
#    joseki:lockingPolicy joseki:lockingPolicyNone ;
#    .

# Local Variables:
# tab-width: 4
# indent-tabs-mode: nil
# End:
```

Modify the configuration for connection to the database:

```
     sdb:sdbHost        "databasehost.my.edu" ;
     sdb:sdbType        "MySQL" ;
     sdb:sdbName        "databasename" ;
     sdb:sdbUser        "databaseuser" ;
     sdb:sdbPassword    "databasepassword" ;
     sdb:driver "com.mysql.jdbc.Driver" ;
```

Save and exit using:
ctrl-x

Replace the default config with the new VIVO config
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo cp joseki-config-vivo.ttl joseki-config.ttl

Change directory into lib
vitro@vivo:/usr/local/Joseki-3.4.2$ cd lib

## 4. Install the MySQL Connector

Download the mysql connector

```
wget http://dev.mysql.com/get/Downloads/Connector-J/mysql-connector-java-5.0.8.zip/from/http://mysql.mirrors.hoobly.com/
```

Unzip the connector

```
mv index.html mysql-connector-java-5.0.8.zip
unzip mysql-connector-java-5.0.8.zip
```

Move the jar file up into the lib directory

```
cp mysql-connector-java-5.0.8/mysql-connector-java-5.0.8-bin.jar /usr/local/joseki/lib/Download the mysql connector
```

## 5. Configure Networking on Local Machine

Shut down the virtual appliance
vitro@vivo:/usr/local/Joseki-3.4.2/lib$ sudo shutdown -h now

You may need to open ports for 2020 in your own environment. In our example VM system you need to run the following comands to allow access from your computer to the virtual machine guest. This is simliar to the commands for opening port 8080 in the VIVO VM setup instructions. THERE ARE NO LINEBREAKS IN THESE COMMANDS.
Mylocalmachine$ VBoxManage setextradata "<NAME OF YOUR VM HERE IF YOU CHANGED IT>" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/sparql/GuestPort" 2020
Mylocalmachine$VBoxManage setextradata "<NAME OF YOUR VM HERE IF YOU CHANGED IT>" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/sparql/HostPort" 2020
Mylocalmachine$VBoxManage setextradata "<NAME OF YOUR VM HERE IF YOU CHANGED IT>" "VBoxInternal/Devices/pcnet/0/LUN#0/Config/sparql/Protocol" TCP

## 6. Start Joseki

Restart the virtual appliance and login at the command prompt
Change directory to the Joseki root
vitro@vivo:/$ cd /usr/local/Joseki-3.4.2

Start the Joseki server to make sure its working ( set JOSEKIROOT NOW )
vitro@vivo:/usr/local/Joseki-3.4.2$ sudo JOSEKIROOT=/usr/local/Joseki-3.4.2 ./bin/rdfserver

Go to your local web browser and enter the following URL:
http://localhost:2020

### Sample SPARQL

If everything is working correctly, you will see the SPARQL generic home page for Joseki. You can now execute a sample query by entering the following into the form:

```
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?facultyname ?collegeName
WHERE
  {
    ?a rdf:type core:Faculty .
    ?a rdfs:label ?facultyname .
    ?a ?pred ?c .
    ?c rdf:type core:College .
    ?c rdfs:label ?collegeName
  }
ORDER BY ASC(?facultyname)
LIMIT 10
```

## 7. Startup script

Sample Startup script file called "joskei" in (for debian): /etc/init.d/
Needs to be run as root if you are going to start manually. init written
by nskaggs@ufl.edu.

```
#!/bin/bash
#/etc/init.d/joseki

PIDFILE=/var/run/joseki.pid
LOGFILE=/var/log/joseki.log

NAME=joseki

# Carry out specific functions when asked to by the system
case "$1" in
       start)
               if [ ! -f $PIDFILE ]; then
                       echo "Starting $NAME"
                       export JOSEKIROOT=/usr/local/joseki
                       cd /usr/local/joseki
                       /usr/local/joseki/bin/rdfserver &> $LOGFILE & echo $! > $PIDFILE
               else
                       echo "$NAME already running"
               fi
               ;;
       stop)
             echo "Stopping $NAME"
             start-stop-daemon --stop --pidfile $PIDFILE
             rm $PIDFILE
               ;;
       restart)
               echo "Restarting $NME"
               $0 stop && sleep 5
               $0 start
               ;;
       *)
               echo "Usage: /etc/init.d/joseki {start|stop|restart}"
               exit 1
               ;;
esac

exit 0
```

## 8. Add Endpoint Update Security filter

( disables anonymous access to the Joseki Update Service)

Thanks to: http://www.mediawiki.org/wiki/User:Alfredas/JosekiSecurity

---

In order to restrict access to the Joseki update service and make it accessible only from localhost:

- add this to the top of web.xml under joseki/WEB-INF

```
 <!-- Access Filter -->
 <filter>
     <filter-name>AccessFilter</filter-name>
     <filter-class>nl.tudelft.tbm.servletaccessfilter.ServletAccessFilter</filter-class>
 </filter>

 <filter-mapping>
   <filter-name>AccessFilter</filter-name>
   <url-pattern>/update/service/*</url-pattern>
 </filter-mapping>
```

- download jar and put it in the servlet engine's classpath
- restart server

```
package nl.tudelft.tbm.servletaccessfilter;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;

public class ServletAccessFilter implements Filter {

  @Override
  public void destroy() {
  }

  @Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException,
ServletException {
    if (request.getRemoteAddr().equals("0:0:0:0:0:0:0:1") || request.getRemoteAddr().equals("127.0.0.1")) {
      chain.doFilter(request, response);
    } else {
      HttpServletResponse httpr = (HttpServletResponse) response;
      httpr.sendError(HttpServletResponse.SC_FORBIDDEN);
    }

  }

  @Override
  public void init(FilterConfig config) throws ServletException {
  }

}
```